Sieci neuronowe
SNR

Andrzej Pacut

Instytut Automatyki i Informatyki Stosowanej
Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Semestr zimowy 2021/2022

prof. dr hab. inż. Andrzej Pacut

- Instytut Automatyki i Informatyki Stosowanej
  Wydział Elektroniki i Technik Informacyjnych
  Zespół Biometrii i Uczenia Maszynowego, kierownik Zespołu
  pok. 522
  andrzej.pacut@pw.edu.pl (proszę nie używać innych moich adresów)
  zbum.iapw.edu.pl

- wykład: czwartki 10-12 - obecnie wyłącznie w formie zdalnej.

  materiały wykładowe: na serwerze Studia i na stronie Zespołu

- **konsultacje**:
  czwartki 12:15-13:00 na platformie MS Teams
  inne terminy: e-mail

Materiały pomocnicze:

- Andrzej Pacut, Sieci Neuronowe - Materiały do wykładu 2021-22.

  Zakres Materiałów nie obejmuje całej treści wykładu.
  Materiały ulegają modyfikacjom w trakcie wykładu.

Oznaczenia:

- materiały wykładu

⊕ materiały wykładu - dodatkowe (obowiązkowe)

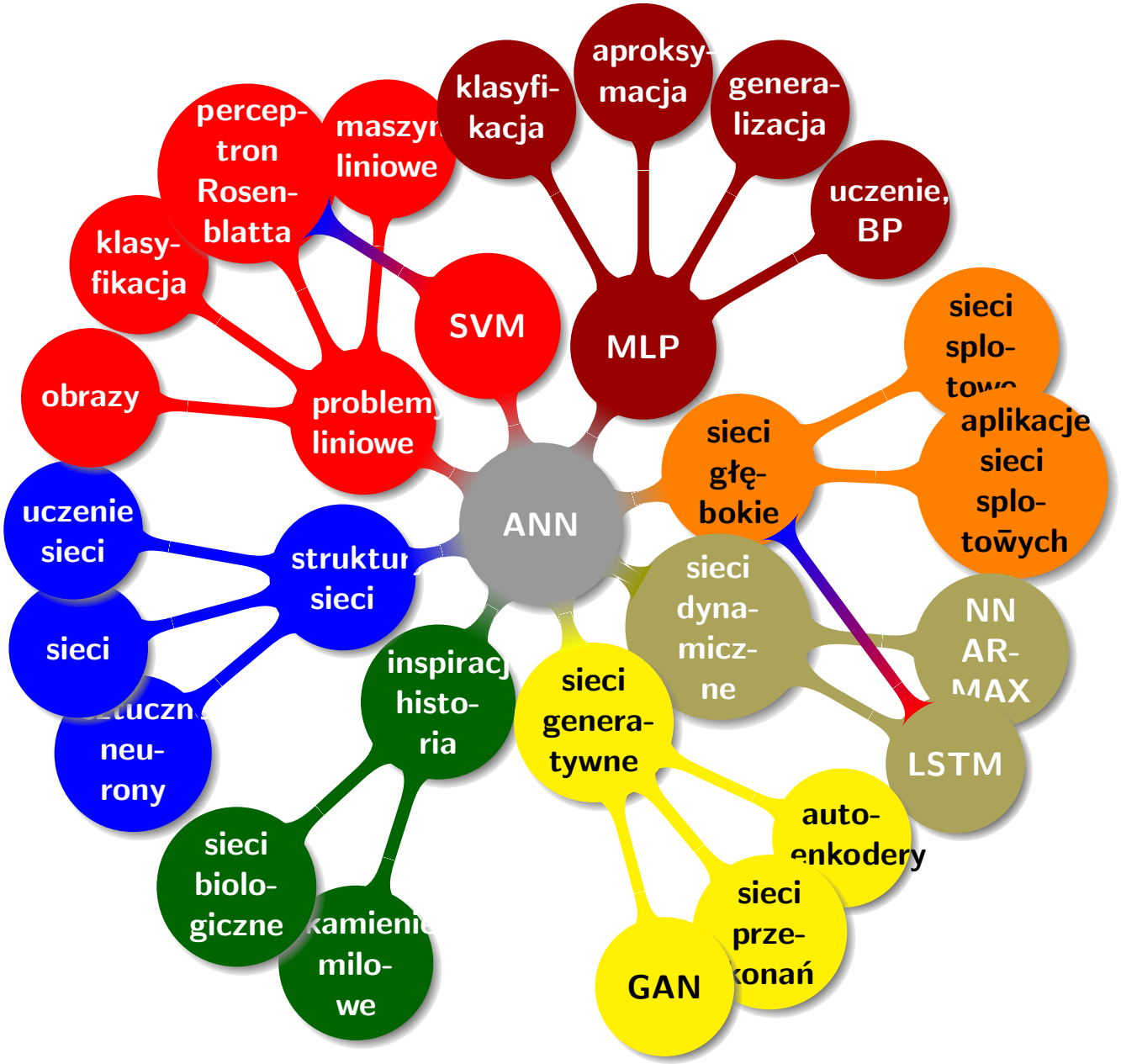
Ⓡ powtórzenia materiału

Ⓔ przykłady

⊛ materiały nadobowiązkowe

Części wykładu

I: Wstęp do sieci neuronowych
II: Sieci perceptronowe a klasyfikacja
III: Sieci perceptronowe a aproksymacja
IV: Głębokie sieci neuronowe


ANN
MLP
aproksy-
macja
klasyfi-
kacja
genera-
lizacja
uczenie,
BP
SVM
maszyn
liniowe
percep-
tron
Rosen-
blatta
klasy-
fikacja
obrazy
problem
liniowe
sieci
splo-
tow
aplikacje
sieci
splo-
towych
sieci
głę-
bokie
sieci
dyna-
micz-
ne
NN
AR-
MAX
LSTM
uczenie
sieci
struktur
sieci
sieci
neu-
rony
inspirac
histo-
ria
sieci
genera-
tywne
auto-
enkodery
sieci
biolo-
giczne
kamieni
milo-
we
sieci
prze-
konań
GAN

Część I: Wstęp do sieci neuronowych

Temat 1. Neurony i sieci - naturalne i sztuczne

Temat 2. Naturalne systemy neuronowe

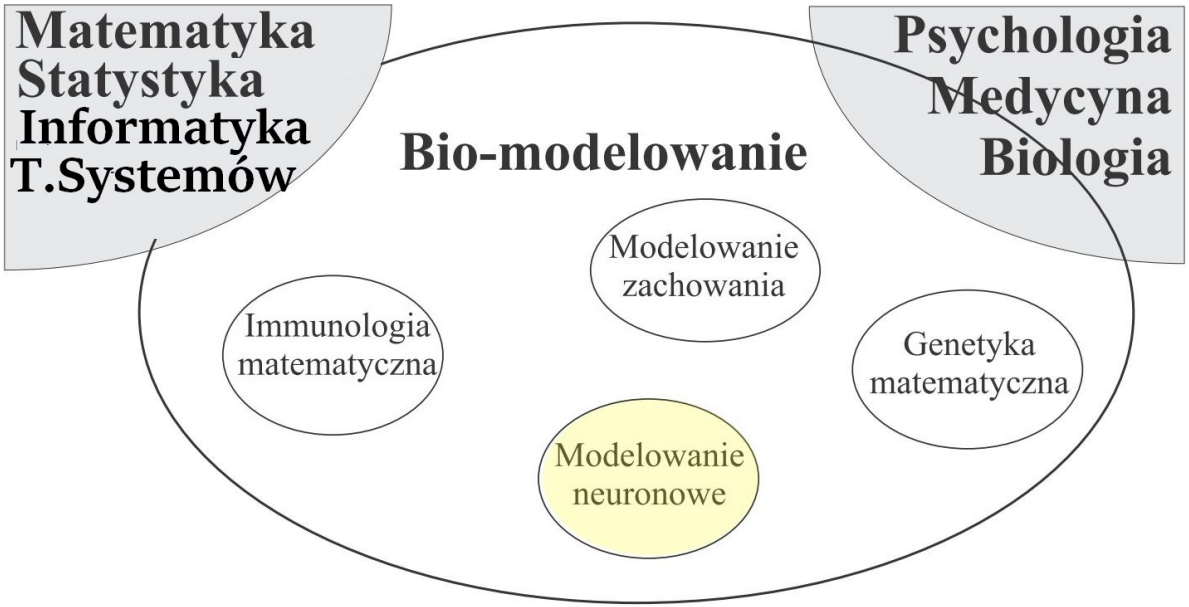
Temat 3. Sztuczne sieci neuronowe - przegląd

Temat 1: Neurony i sieci - naturalne i sztuczne

1.1 Modelowanie neuronowe
1.2 Obszar sieci neuronowych
1.3 Kamienie milowe rozwoju sieci neuronowych

1.1 Modelowanie neuronowe
1.2 Obszar sieci neuronowych
1.3 Kamienie milowe rozwoju sieci neuronowych

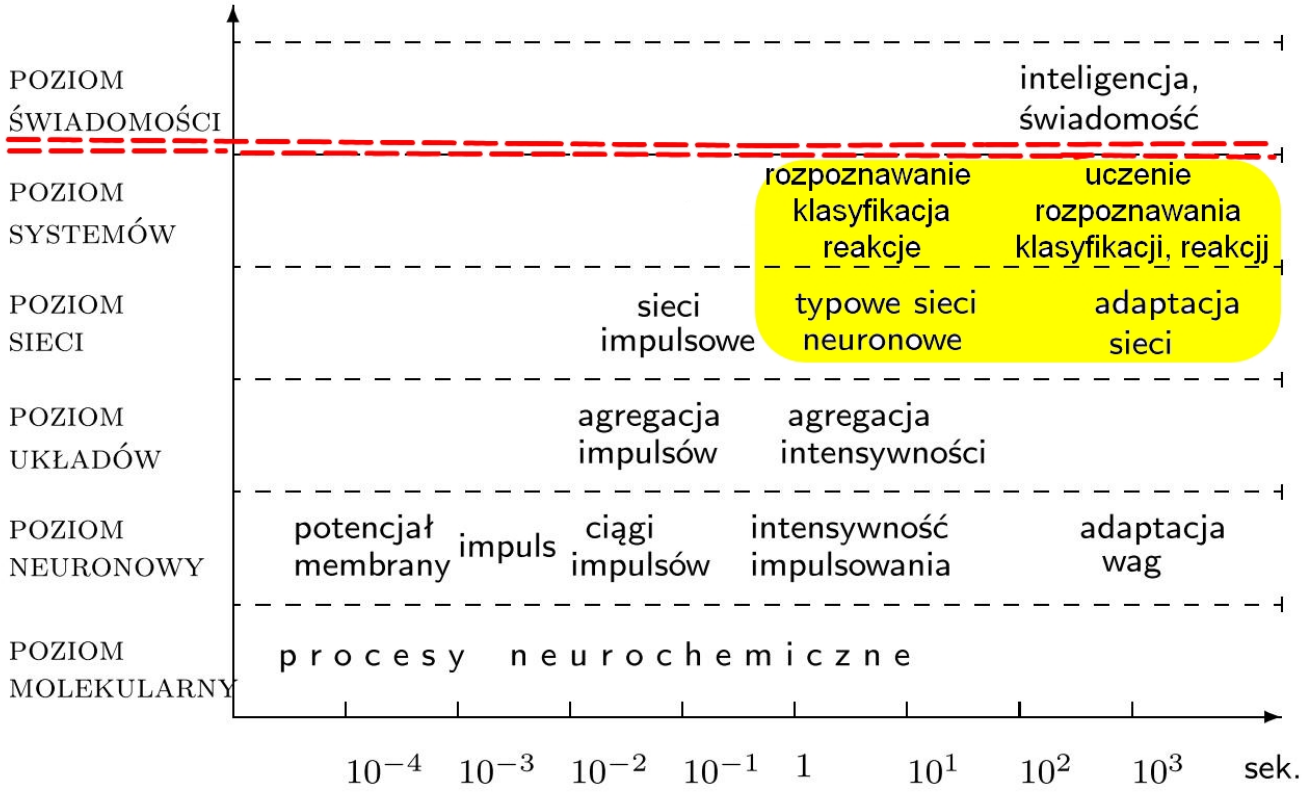
Bio-modelowanie



- *bio-modelowanie* (biomatematyka): modelowanie matematyczne budowy i działania żywych organizmów i ich części
- gałęzie bio-modelowania: modelowanie neuronowe, modele narządów, immunologia matematyczna, genetyka matematyczna, etc.
- neuronauka (neurobiologia matematyczna), (neuroinformatyka), w tym *modelowanie mózgu* i *modelowanie neuronów*
- cele: *poznawcze*: jak jest zbudowane, jak działa w naturze
- zastosowania: biologia, medycyna, farmacja, psychologia

Neurobiologia matematyczna

Skale modeli neuronowych



Neurobiologia matematyczna

Mózg

https://www.dreamstime.com/royalty-free-stock-photo-male-human-head-skull-brain-ghost-effect-side-view

- detekcja bodźców
- lokalizacja
- pamięć
- porównywanie, klasyfikacja, rozpoznawanie
  - obrazów
  - pisma, utworów
  - mowy, mówcy, utworów
- adaptacja do środowiska
- uczenie
- prognozowanie
- regulacja
- wnioskowanie (dedukcja)
- inteligencja
- świadomość

Neurobiologia matematyczna

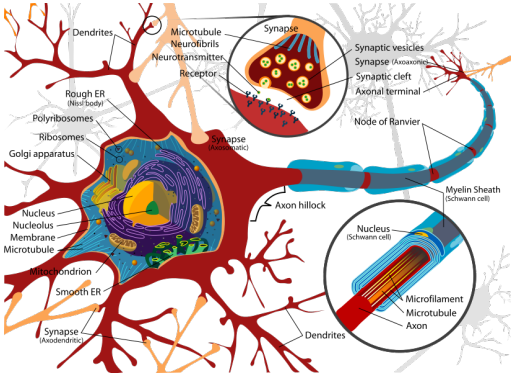
Aspekty modelowania mózgu

https://www.dreamstime.com/royalty-free-stock-photo-male-human-head-skull-brain-ghost-effect-side-view;

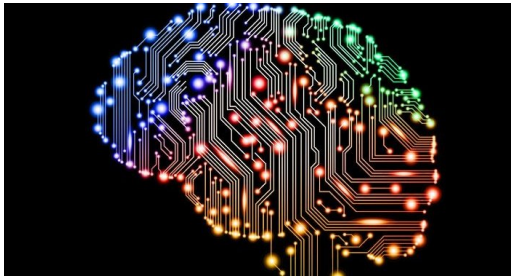
http://extrawall.net/?option=wallpaper&id=244; http:/commons.wikimedia.org/wiki/User:LadyofHats/gallery2#/media/File:Complete_neuron_cell_diagram_en.svg; Yann Le Cun, Deep Learning. Should we copy the brain to build intelligent machines

modelowanie elementów, połączeń i funkcji w różnych skalach czasu i przestrzeni

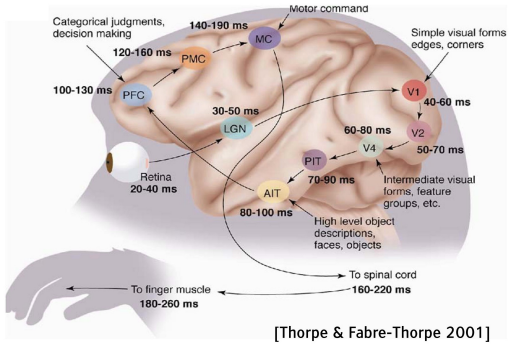
elementy $\Longrightarrow$



połączenia i sygnały $\Longrightarrow$

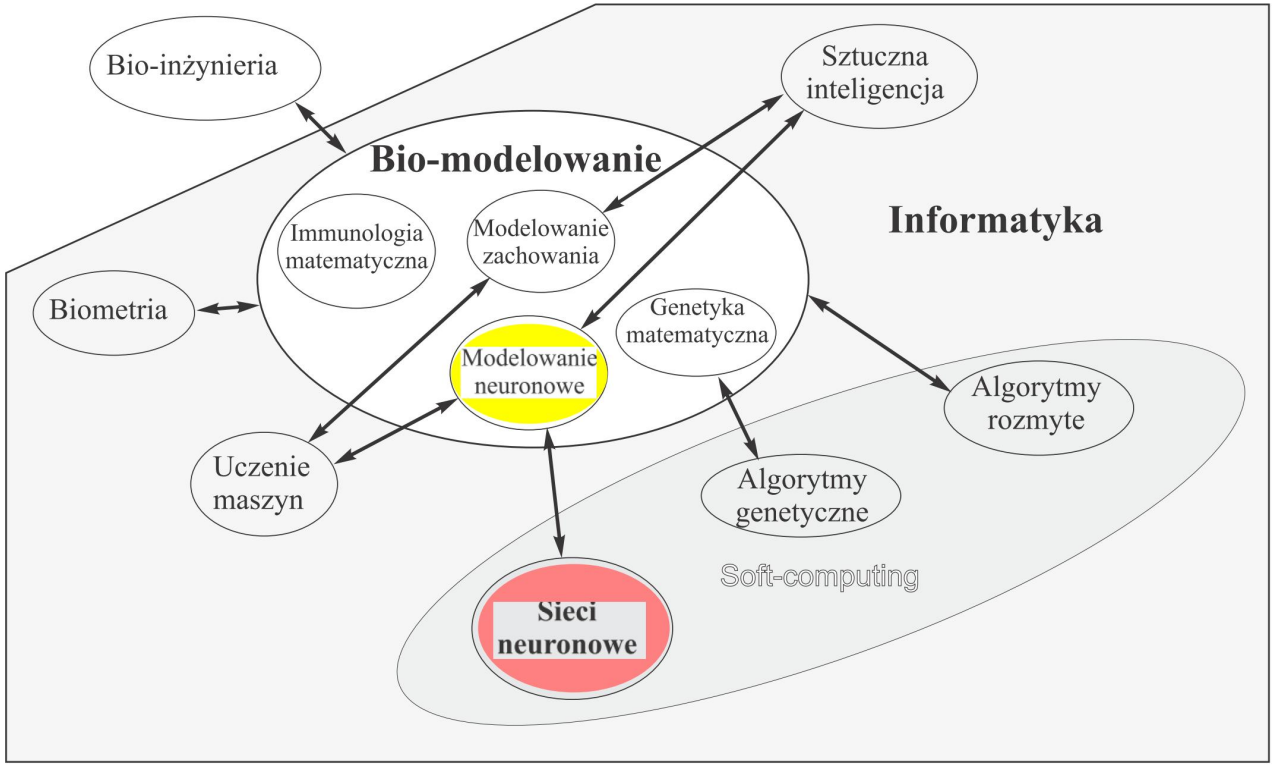


funkcje $\Longrightarrow$



[Thorpe & Fabre-Thorpe 2001]

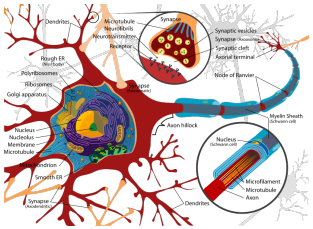
Inspiracje biologiczne w informatyce

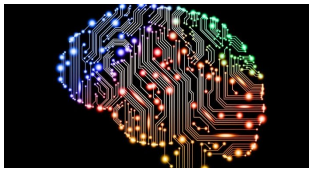


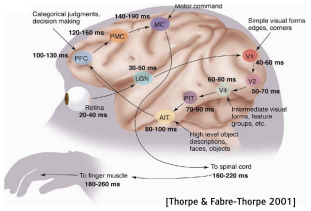
- cele: *specyficzne dla obszaru zastosowań* (*nie-biologiczne*)
- modyfikacje modeli biologicznych; *rozwiązania nie kopiują biologii*
- potrzeba ponawiania inspiracji biologicznych

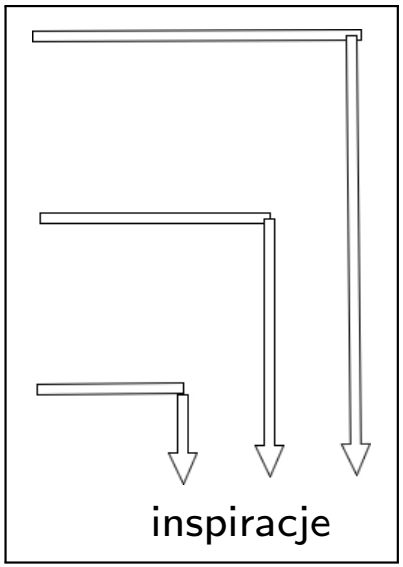
Inspiracje neurobiologiczne w informatyce

Sztuczne sieci neuronowe

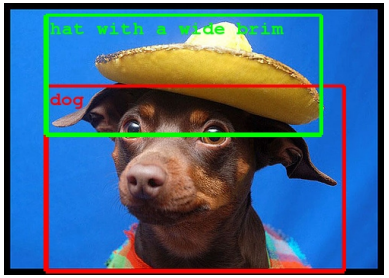


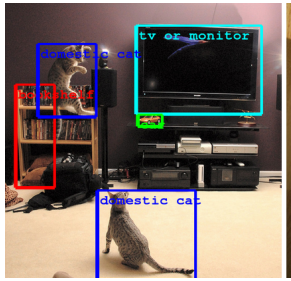


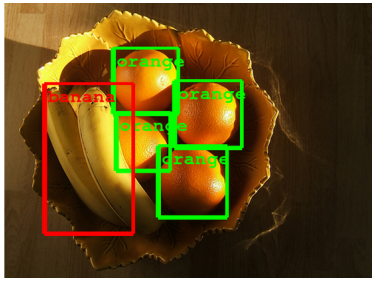




- detekcja bodźców •
- porównywanie • pamięć •
- klasyfikacja, rozpoznawanie bodźców :
- obrazów, podpisu •
- dźwięków, mowy, mówcy •
- detekcja • lokalizacja •
- adaptacja • uczenie •
- prognozowanie • regulacja •
- inteligencja • świadomość







https://www.dreamstime.com/royalty-free-stock-photo-male-human-head-skull-brain-ghost- effect-side-view,
http://extrawall.net/?option=wallpaper&id=244,
Y. Le Cun (2016). *Deep Learning*. URL: //indico.cern.ch/event/510372/attachments/1245509/1840815/lecun-20160324-cern.pdf, *Building a deeper understanding of images* (2014). Google. URL: research.googleblog.com/2014/09/building-deeper-understanding-of-images.html

1.1 Modelowanie neuronowe
1.2 Obszar sieci neuronowych
1.3 Kamienie milowe rozwoju sieci neuronowych

Definicje sieci neuronowych

Random House Webster's College Dictionary, 2010 (2.) Model obliczeniowy do symulacji zachowania biologicznych sieci neuronowych, w tym procesów uczenia, przetwarzania języka, rozwiązywania problemów, w celu automatycznego („self directed") przetwarzania informacji.

Collins English Dictionary (2014) W informatyce, program lub system inspirowany budową mózgu zaprojektowany do imitowania funkcjonalności ludzkiego mózgu, w szczególności procesów uczenia.

American Heritage Dictionary (2016) Urządzenie lub program w którym wiele połączonych elementów przetwarza informację równolegle, ucząc się z przeszłych wzorców; używane często do modelowania mózgu.

Wikipedia (2018) (ang.) Systemy obliczeniowe inspirowane przez biologiczne systemy neuronowe mózgu, które „uczą się" wykonywania zadań na podstawie przykładów, bez tworzenia reguł specyficznych dla zadania.

Merriam-Webster (2018) Architektura komputerowa, w której wiele procesorów jest połączonych w sposób podobny do połączeń między neuronami w mózgu i która potrafi się uczyć w procesie prób i błędów.

Sieci neuronowe - charakteryzacja przez własności

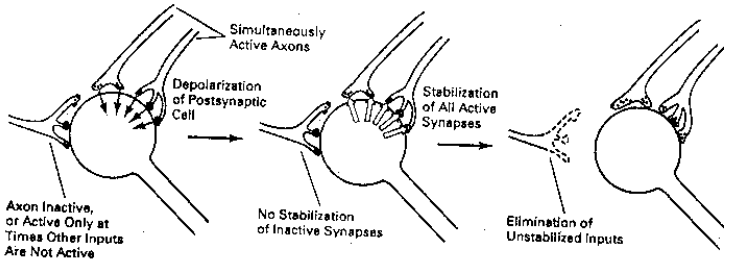
- **inspiracje biologiczne**: budowa i działanie inspirowane strukturą i działaniem naturalnych sieci neuronowych
- **modularność**: sieć procesorów (neuronów, jednostek obliczeniowych) o podobnej strukturze
- **przetwarzanie numeryczne** (nie symboliczne)
- **połączenia**: przetwarzanie jest określone przez struktura połączeń procesorów a nie przez konkretne procesory
- **obliczenia równoległe**
- **nieliniowość**
- **adaptacja do zmian środowiska**: uczenie na przykładach, trening
- **generalizacja**: mogą ekstrapolować dane stosowane przy treningu
- **nadmiar parametrów**
- **tolerancja na uszkodzenia**: nie ma potrzeby znajdowania i usuwania wadliwych procesorów, stopniowa degradacja jakości

1.1 Modelowanie neuronowe
1.2 Obszar sieci neuronowych
1.3 Kamienie milowe rozwoju sieci neuronowych

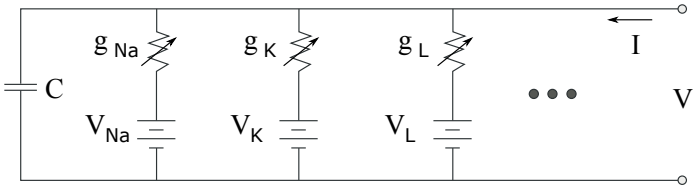
Kamienie milowe rozwoju sieci neuronowych (1)

↗ **pierwsze sztuczne neurony**

- **1943**: **model neuronu Threshold Logic Unit** $y = \mathbf{1}(\boldsymbol{w}^\top \boldsymbol{u} - d)$
  Warren S. **McCulloch** (neurobiolog, U. Chicago), Walter **Pitts** (logik)
  W. S. McCulloch and W. H. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biophysics* 5, pp. 115–133. Reprinted in **AnderRosen88**
- **1949**: **uczenie**, wzrost siły powiązań na skutek stymulacji (*reguła Hebba*)
  Donald **Hebb** (psycholog, McGill U., Montreal)
  D. O. Hebb (1949). *The organization of behavior*. New York: J. Wiley and Sons



efekty stymulacji - reguła Hebba
https://www.benbest.com/science/anatmind/anatmd4.contents



model dynamiczny neuronu
https://commons.wikimedia.org/wiki/File:Hodgkin-huxley-circuit.svg

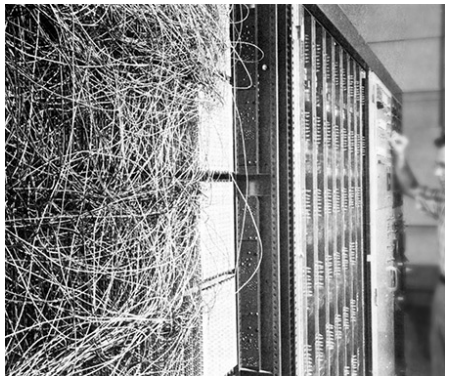
- **1952**: **model dynamiczny neuronu**, Alan Lloyd **Hodgkin** i Andrew Fielding **Huxley** (U. of Cambridge), Nagroda Nobla w Fizjologii i Medycynie 1963

Kamienie milowe rozwoju sieci neuronowych (2)

↗ **pierwsze sieci**

- **1956**: **sztuczna inteligencja**, konferencja w Dartmouth College (Hanover, New Hampshire) - McCarthy, Minsky, Shannon, Solomon, Newell,...
- **1958**: **perceptron**, Frank **Rosenblatt** (Cornell U., Ithaca, NY)
  F. Rosenblatt (1958). "The perceptron: a probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6, pp. 386–408. Reprinted in **AnderRosen88**, Ch. 8, pp. 89-114, 1988

Frank Rosenblatt



Rosenblatt's Perceptron

Marvin Minsky
www.conversationagent.com/2016/01/value-of-negative-knowledge.html

- **1960**: **Adaline, LMS**, Bernard **Widrow** i Marcian 'Ted' **Hoff** (Stanford Univ.)
  B. Widrow and M. E. Hoff (1960). "Adaptive switching circuits". In: *1960 IRE WESCON Convention Record*. Vol. 4. New York, pp. 96–104 (Stanford U., Stanford, CA)
- **1969**: **perceptron** ↓↓↓, Marvin **Minsky**, Seymour **Papert** (MIT, Cambridge, MA)
  M. L. Minsky and S. A. Papert (1969). *Perceptrons*. Cambridge, Massachusetts: MIT Press. Also available: Expanded Edition, MIT Press, Cambridge, Ma., 1988

Kamienie milowe rozwoju sieci neuronowych (3)

↘ **"dekada spokoju"**

- **1972**: **liniowa pamięć asocjacyjna**: **Kohonnen** (Helsinki U. of Tech.), **Anderson** (Brown U., Providence, RI)

  T. Kohonen (1972). "Correlation Matrix Memories". In: *IEEE Transactions on Systems, Man, and Cybernetics* C-21, pp. 353–359 ,

  J. A. Anderson (1972). "A Simple Neural Network Generating an Interactive Memory". In: *Mathematical Biosciences* 14, pp. 197–220

- **1974**: **propagacja wsteczna**, Paul **Werbos** (Harvard U., Cambridge, MA)

  P. J. Werbos (1974). "Beyond regression: New tools for prediction and analysis in the behavioral sciences". Ph.D. Thesis. Cambridge, MA: Harvard University

- **1980** **Neocognitron** - pierwowzór głębokich sieci splotowych, Kunihiko **Fukushima** (Osaka Univ.)

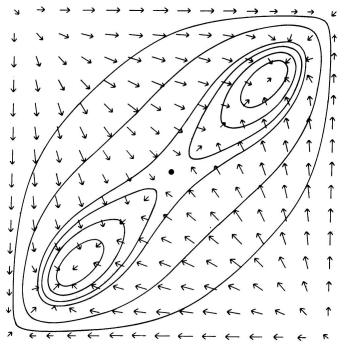
  K. Fukushima (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36, pp. 193–202

Kamienie milowe rozwoju sieci neuronowych (4)

ponowny rozwój

- **1982**: **pamięć a stabilność sieci**, John **Hopfield** (Caltech)

  J. J. Hopfield (1982). "Neural networks and physical systems with emergent collective computational abilities". In: *Proc. of the National Academy of Sciences of the USA* 79, pp. 2554–2558



Hopfield (1984)

- **1983**: **stabilność sieci neuronowych**, *tworzenie wzorców*, Michael A. **Cohen**, Stephen **Grossberg** (Boston U.)

  M. A. Cohen and S. Grossberg (1983). "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13, pp. 815–826. Takze: http://cns-web.bu.edu/Profiles/Grossberg

- **1984 Maszyny Boltzmanna**, **Hinton, Sejnowski, Ackley**, Carnegie-Mellon U., Pittsburgh, Pensylwania

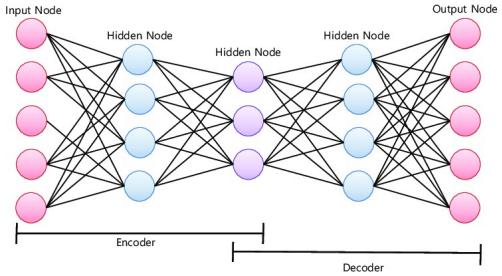
  G. Hinton, T. Sejnowski, and D. Ackley (1984). *Boltzmann machines: Constraint satisfaction networks that learn*. Tech. rep. TR-CMU-CS-84-119. Carnegie-Mellon University, Dept. of Computer Science

Kamienie milowe rozwoju sieci neuronowych (5)

ponowny rozwój

- **1986**: **uczenie sieci z użyciem propagacji wstecznej**, David **Rumelhart**, Geoffrey **Hinton** i Ronald **Williams** (PDP group, MIT, Cambridge, MA)

  D. E. Rumelhart, G. E. Hinton, and R. J. Williams (1986b). "Learning internal representations by error propagation". In: *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*. Ed. by D. E. Rumelhart and J. McClelland. Vol. 1. Cambridge, MA: MIT Press. Chap. 8



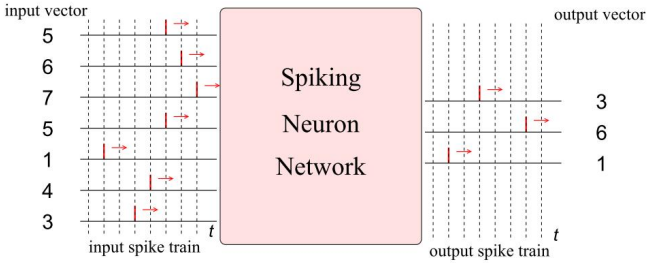
Nam et al. (2018)

- **1986**: **autoenkodery - uczenie nienadzorowane**, David **Rumelhart**, Geoffrey **Hinton** i Ronald **Williams**

  D. E. Rumelhart, G. E. Hinton, and R. J. Williams (1986a). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536

- **1987**: **sieci impulsowe**, Wolfgang **Maass**, (Tech. Univ. Grass, Austria)

  W. Maass (Dec. 1997). "Networks of spiking neurons: The third generation of neural network models". In: *Neural Networks* 10.9, pp. 1659–1671. URL: www.igi.tugraz.at/maass/psfiles/85b.pdf



Paugam-Moisy and Bohte (2012)

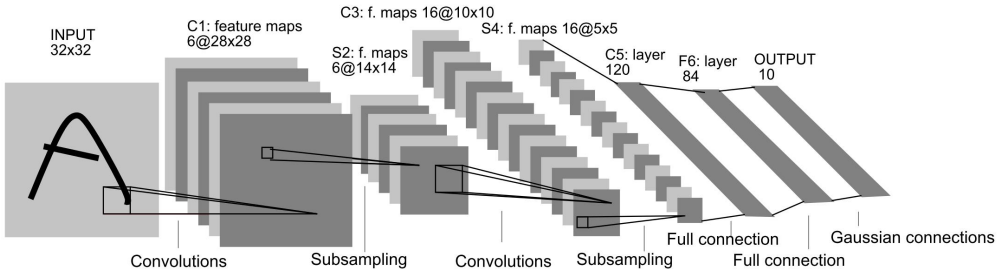
Kamienie milowe rozwoju sieci neuronowych (6)

rozwój

- **1998**: **maszyny wektorów podpierających** SVM, Vladimir N. **Vapnik** (Inst. Sterowania, Moskwa, Adaptive Systems Research Dept., AT&T Bell Labs, Holmdel, New Jersey) i Alexey Ya. **Chervonenkis** (Rosyjska Akademia Nauk i Uniw. Londyn)

  V. Vapnik (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag; V. Vapnik (1998). *Statistical Learning Theory*. Wiley
- **1998**: **głębokie sieci splotowe** - początki, Yann **LeCun**, AT&T Bell Labs., Holmdel, New Jersey; później Facebook AI Research, New York

  Y. LeCun et al. (Nov. 1998). "Gradient-Based Learning applied to Document Recognition". In: *Proceedings of the IEEE*, pp. 1–46



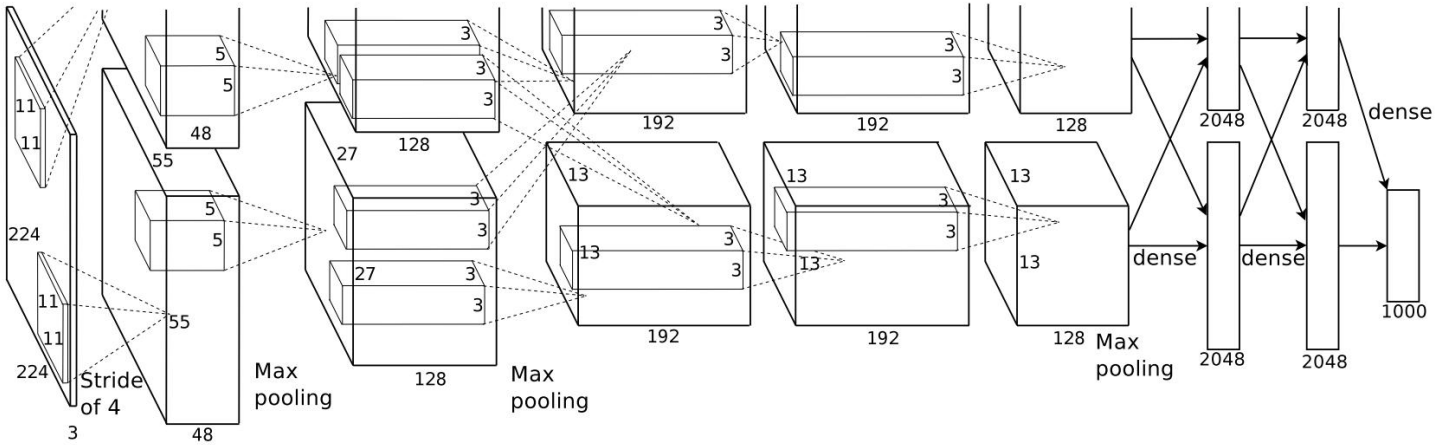
- **2002**: **zawężone maszyny Boltzmanna** RBM, Geoffrey **Hinton** (Univ. of Toronto, Google)

  G. Hinton (2002). "Training products of experts by minimizing contrastive divergence". In: *Neural Computation* 14, pp. 1771–1800
- **2006**: **głębokie sieci przekonań** (ang. *Deep Belief Network*), Geoffrey **Hinton** i inni

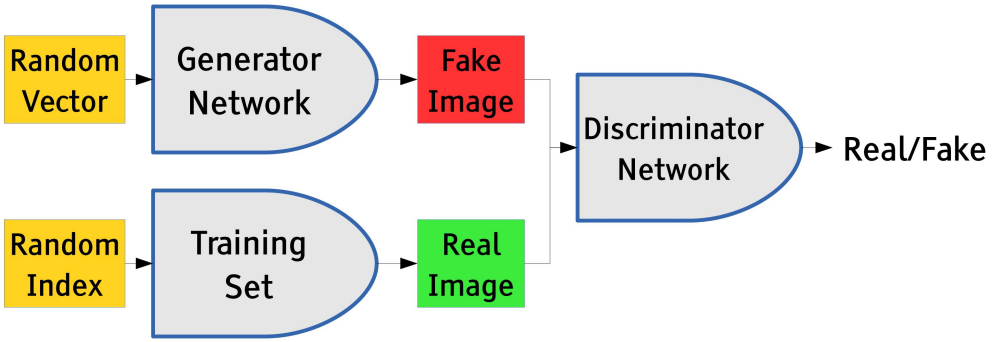
Kamienie milowe rozwoju sieci neuronowych (7)

gwałtowny rozwój

- **2012**: **głębokie sieci splotowe - klasyfikacja obrazów**, Alex **Krizhevsky**, Ilya **Sutskever** i Geoffrey **Hinton** (Google, Mountain View, California)



- **2014**: **generacyjne sieci antagonistyczne** (ang. *generative adversarial networks*) **GAN**, Ian Goodfellow et al. (Stanford Univ., Google Research)



Zastosowania sieci neuronowych

zagadnienie	sieci	dane uczące
klasyfikacja obrazów na podstawie ich cech	perceptronowe	pary (wzorzec 1D, klasa)
	SVM	-
klasyfikacja obrazów	CNN, DCNN	pary (obraz 2D/3D, klasa), pary (tensor, klasa)
klasyfikacja nienadzorowana	głębokie autoenkodery	obrazy 2D/3D
aproksymacja funkcji	perceptronowe	pary (argument, wartość funkcji)
optymalizacja kombinatoryczna	Hopfielda	–
pamięci asocjacyjne	Hopfielda	–
prognozowanie	NNARMAX, sieci LSTM	dane historyczne
klasyfikacja tekstu	sieci LSTM, DCNN	pary (tekst, klasa)
rozpoznawanie mowy/mówcy	DCNN, sieci LSTM	np. pary (fragment mowy, mówca)
GAN		obrazy

CNN, DCNN (ang. *deep convolution networks*) - głębokie sieci splotowe
SVM (ang. *support vector machine*) - maszyna wektorów podpierających
LSTM (ang. *long-short term memory*)- pamięć długo- i krótkoterminowa
NNARMAX (ang. *neural networks ARMAX*) - neuronowa aproksymacja ARMAX
GAN (ang. *generative adversarial networks*): generatywne sieci antagonistyczne

Zastosowania sieci neuronowych

Dziedziny zastosowań

- telekomunikacja (aproksymacja)
- sterowanie i robotyka (aproksymacja)
- systemy ekspertowe, systemy decyzyjne (optymalizacja, sterowanie)
- biometria (modelowanie, klasyfikacja)
- ekonomia (prognozowanie, modelowanie)
- medycyna (klasyfikacja)
- wojskowość (klasyfikacja, identyfikacja)

Literatura

Materiały źródłowe

Anderson, J. A. (1972). "A Simple Neural Network Generating an Interactive Memory". In: *Mathematical Biosciences* 14, pp. 197–220.

Cohen, M. A. and S. Grossberg (1983). "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13, pp. 815–826. Takze: http://cns-web.bu.edu/Profiles/Grossberg.

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2, pp. 303–314.

Fukushima, K. (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36, pp. 193–202.

Hebb, D. O. (1949). *The organization of behavior*. New York: J. Wiley and Sons.

Hinton, G., T. Sejnowski, and D. Ackley (1984). *Boltzmann machines: Constraint satisfaction networks that learn*. Tech. rep. TR-CMU-CS-84-119. Carnegie-Mellon University, Dept. of Computer Science.

Hinton, G. E., S. Osindero, and Y. Teh (2006). "A fast learning algorithm for deep belief net". In: *Neural Computation* 18, pp. 1527–1554.

Hinton, G. (2002). "Training products of experts by minimizing contrastive divergence". In: *Neural Computation* 14, pp. 1771–1800.

Literatura (cd.)

Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective computational abilities". In: *Proc. of the National Academy of Sciences of the USA* 79, pp. 2554–2558.

— (May 1984). "Neurons with graded response have collective computational properties like those of two-state neurons". In: *Proc. of the National Acad. of Sciences of the USA* 81, pp. 3088–3092.

Kohonen, T. (1972). "Correlation Matrix Memories". In: *IEEE Transactions on Systems, Man, and Cybernetics* C-21, pp. 353–359.

LeCun, Y. et al. (Nov. 1998). "Gradient-Based Learning applied to Document Recognition". In: *Proceedings of the IEEE*, pp. 1–46.

Maass, W. (Dec. 1997). "Networks of spiking neurons: The third generation of neural network models". In: *Neural Networks* 10.9, pp. 1659–1671. URL: www.igi.tugraz.at/maass/psfiles/85b.pdf.

McCulloch, W. S. and W. H. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biophysics* 5, pp. 115–133. Reprinted in **AnderRosen88**.

Minsky, M. L. and S. A. Papert (1969). *Perceptrons*. Cambridge, Massachusetts: MIT Press. Also available: Expanded Edition, MIT Press, Cambridge, Ma., 1988.

Rosenblatt, F. (1958). "The perceptron: a probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6, pp. 386–408. Reprinted in **AnderRosen88**, Ch. 8, pp. 89-114, 1988.

Literatura (cd.)

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986b). "Learning internal representations by error propagation". In: *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*. Ed. by D. E. Rumelhart and J. McClelland. Vol. 1. Cambridge, MA: MIT Press. Chap. 8.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag.

— (1998). *Statistical Learning Theory*. Wiley.

Werbos, P. J. (1974). "Beyond regression: New tools for prediction and analysis in the behavioral sciences". Ph.D. Thesis. Cambridge, MA: Harvard University.

Widrow, B. and M. E. Hoff (1960). "Adaptive switching circuits". In: *1960 IRE WESCON Convention Record*. Vol. 4. New York, pp. 96–104.

Materiały pomocnicze

Building a deeper understanding of images (2014). Google. URL: research.googleblog.com/2014/09/building-deeper-understanding-of-images.html.

Goodfellow, I. J. et al. (2014). "Generative Adversarial Nets". In: *Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014)*, pp. 2672–2680.

Hagan, M. T. et al. (2014). *Neural Network Design*. 2nd ed. URL: http://hagan.okstate.edu/NNDesign.pdf.

Hajek, M. (2005). *Neural Networks*.

Literatura (cd.)

Le Cun, Y. (2016). *Deep Learning*. URL: //indico.cern.ch/event/510372/attachments/1245509/1840815/lecun-20160324-cern.pdf.

Nam, S. et al. (Jan. 2018). "Forged Signature Distinction Using Convolutional Neural Network for Feature Extraction". In: *Applied Sciences* 8, p. 153. DOI: 10.3390/app8020153.

Paugam-Moisy, H. and S. Bohte (2012). "Computing with Spiking Neuron Networks". In: *Handbook of Natural Computing*. Ed. by G. Rozenberg, T. H. W. Bäck, and J. N. Kok. Heidelberg: Springer Verlag. URL: pdfs.semanticscholar.org/8997/95d8e3c07976f62149eb79612e06231d6aee.pdf.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986a). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536.

Sarle, W. S., ed. (1997). *Neural Network FAQ, part 1 of 7: Introduction*. comp.ai.neural-nets, ftp://ftp.sas.com/pub/neural/FAQ.html.

Temat 2: Naturalne systemy neuronowe

2.1 Elementy systemu nerwowego
2.2 Działanie neuronu
2.3 Biologiczne sieci neuronowe

2.1 Elementy systemu nerwowego
2.2 Działanie neuronu
2.3 Biologiczne sieci neuronowe

⊕ Biologiczne systemy neuronowe

Podstawowe elementy budowlane

- *neurony*
  - rozróżnianie bodźców, pamięć, przetwarzanie sygnałów, rozpoznawanie, wnioskowanie, sterowanie, uczenie, świadomość ?
  - w mózgu ludzkim $\sim$ 86 mld. neuronów (w tym kora mózgu: $\sim$ 16 mld. neuronów) (Herculano-Houzel and Lent (2005), Azevedo et al. (2009))
  - ogólna liczba synaps u osób dorosłych: 10^{14} (dzieci 10^{15})
  - średnia liczba synaps neuronu: ok. 7000
- *komórki glejowe*;
  - wymiana jonów, tlenu, składników odżywczych, metabolitów; osłona aksonów, poza-neuronowy układ sterowania?
  - w mózgu ludzkim ~ 85 mld.
- *substancja (macierz) pozakomórkowa* (ECM) - materiał w przestrzeni pozakomórkowej, złożony z włókien i płynu tkankowego
- *płyn mózgowo-rdzeniowy*
- *krew*
  - dostarczanie tlenu

⊕ Neurony biologiczne

- *ciało komórki*

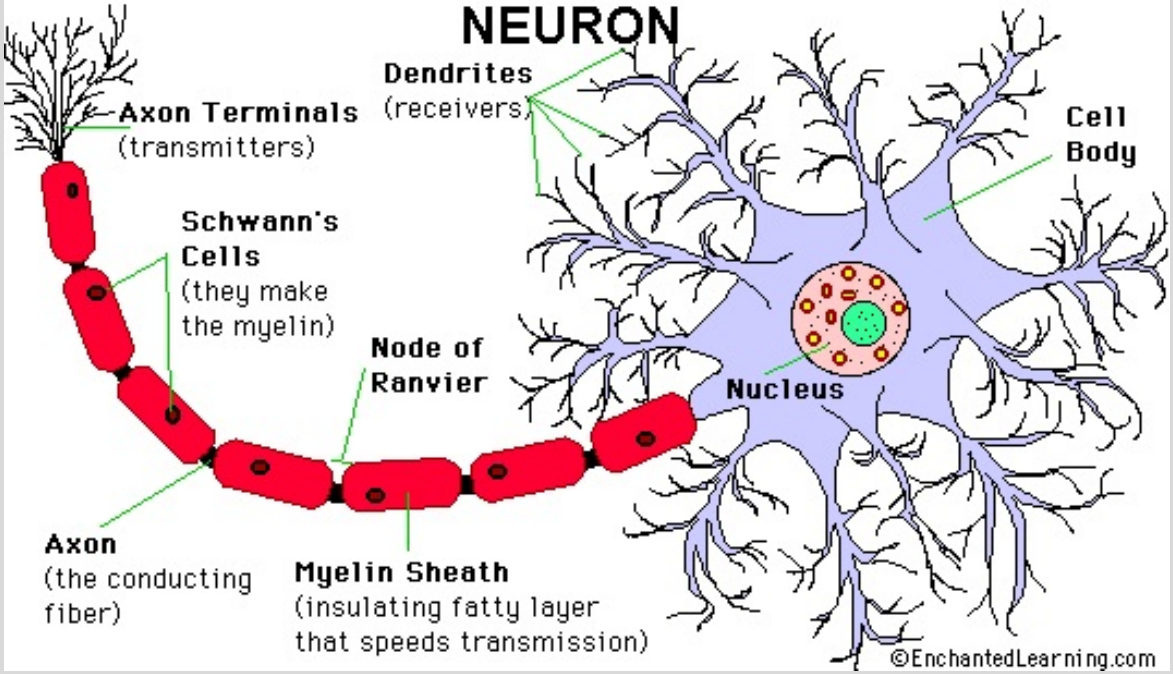
  $\varnothing$: 5μ m (komórki granularne) do 100 μm (neurony ruchowe)

  zawiera *jądro*, *kwas rybonukleinowy RNA*, struktury produkujące proteiny

- *błona komórkowa*

  grubość 5 nm, oddziela cytoplazmę od płynu zewnętrznego

  kontroluje przepływ cząstek chemicznych

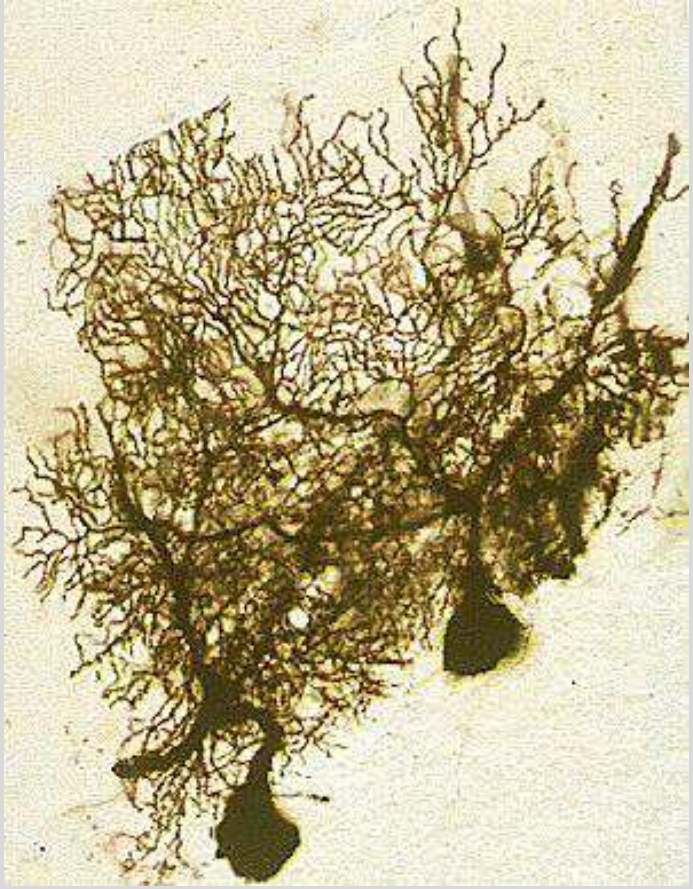


Jeananda Col, Enchanted Learning, `https://www.enchantedlearning.com/subjects/anatomy/brain/Neuron.shtmlneuron`, 2001

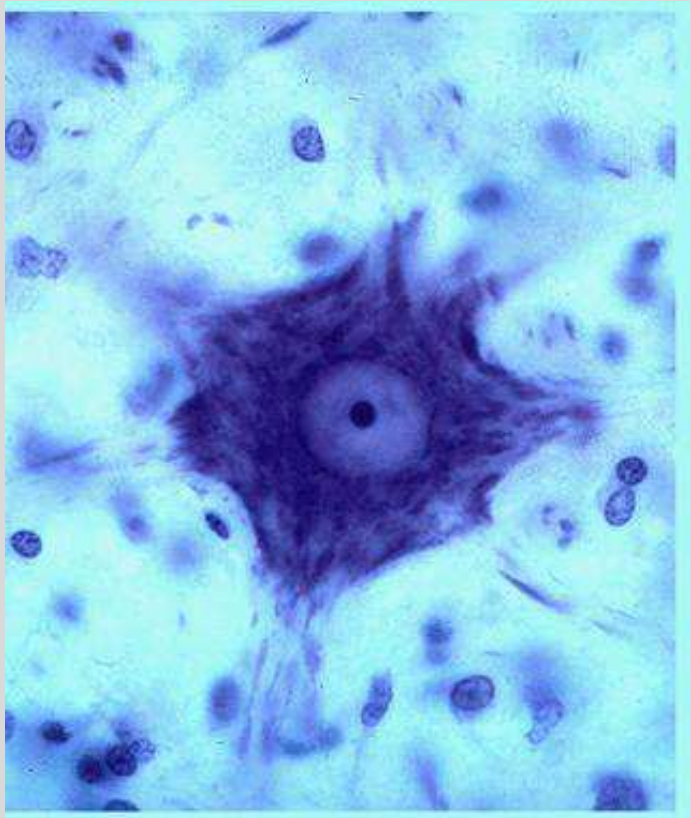
- *dendryty*
  włókna wejściowe, długość do 0.1 mm,
- *akson*
  pojedyncze włókno wyjściowe, długość 1 mm do 1 m, rozgałęzia się do synaps innych neuronów,
- *synapsy*
  punkty kontaktu, od kilku do 90 tys. (komórki Purkiniego) na akson
  człowiek: do $\approx$ biliarda (10^{15}) synaps

⊕ Neurony biologiczne

Zdjęcia neuronów



Komórki Purkiniego
(kora móżdżku)

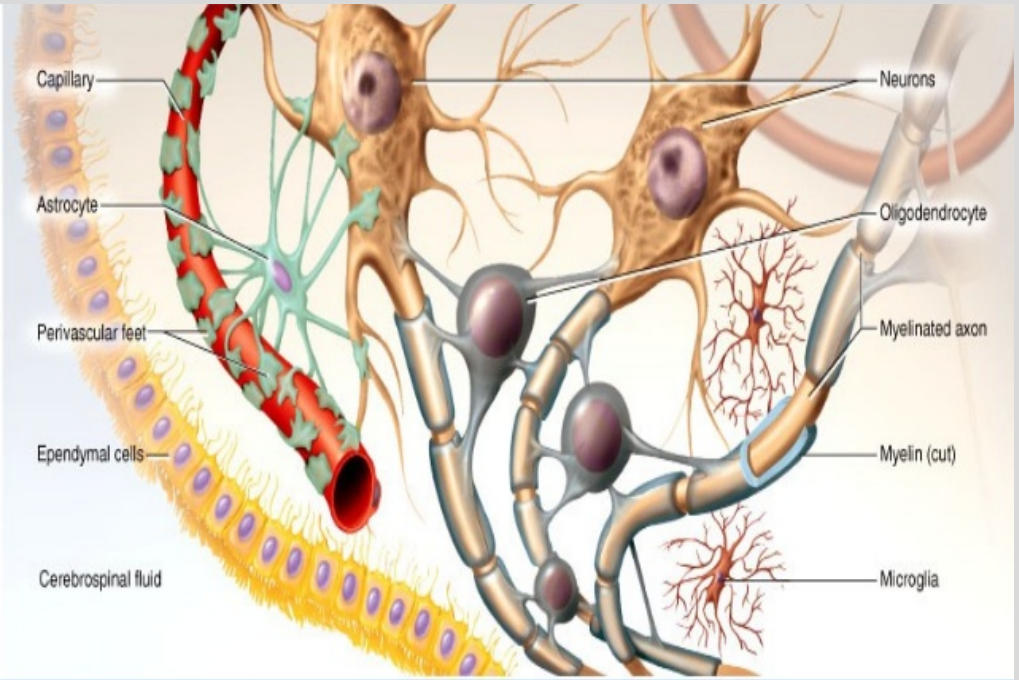


neuron ruchowy
(rdzeń kręgowy)
$\varnothing \approx 100\mu$

⊕ Elementy budowy biologicznych systemów neuronowych

Komórki glejowe

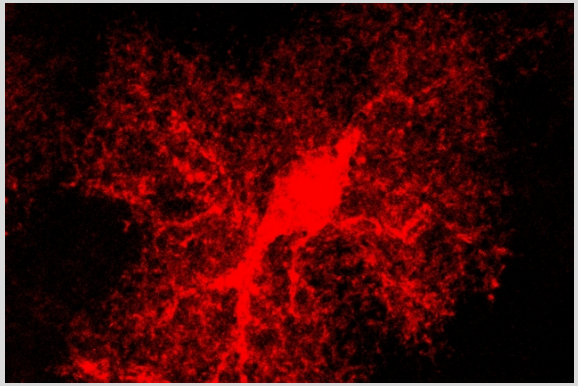
- mózg: liczba komórek glejowych ≈ liczba neuronów



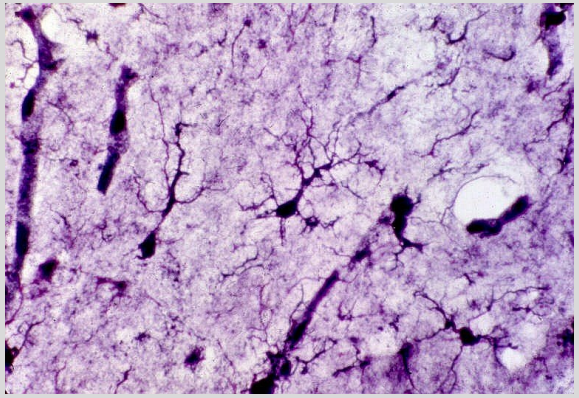
Anatomy and Physiological aspects of glial cells, Role in Health and Disease Health & Medicine, Technology 2014, https://www.slideshare.net/drrahulkumarsingh/ glial-cells-neurobiology-and-clinical-aspects

- *astrocyty*: bariera krew - mózg, regulują środowisko chemiczne, usuwanie neurotransmiterów, **analogowa regulacja impulsowania**
- *oligodendrocyty*: owijają się wokół aksonów tworząc *mielinę*, izolującą akson
- *mikroglej*: usuwanie patogenów i obumarłych neuronów, nieprzytwierdzone do struktur mózgu
- *komórki ependymalne*: wyścielaja komory mózgu i kanał rdzenia kręgowego

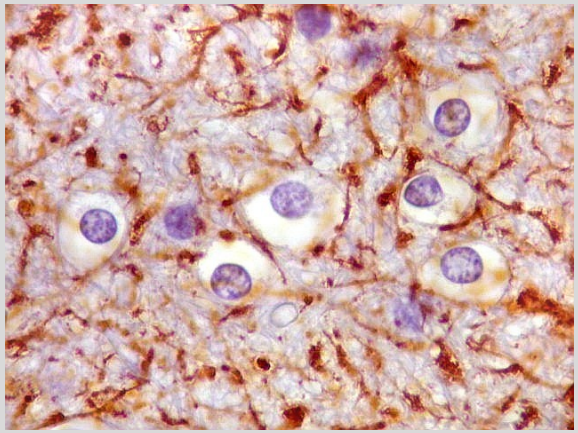
⊕ Komórki glejowe



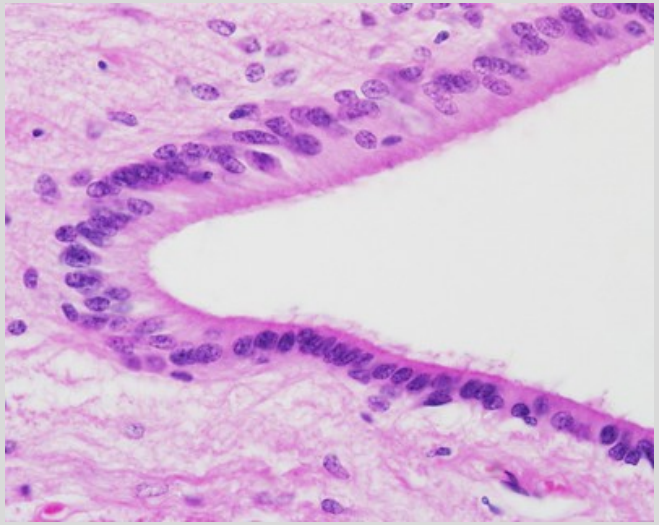
astrocyty
D. Orenstein (2018). "Neuroscientists give "invisible" cells a new look". In: *MIT News*



mikroglej
L. S. Queiroz and R. A. Paes (2019). *Neuroimaging website of Dept. of Pathology and Radiology*. Tech. rep. Campinas, Brasil: State University of Campinas, School of Medicine (FCM-UNICAMP)



oligodendrocyty
L. S. Queiroz and R. A. Paes (2019). *Neuroimaging website of Dept. of Pathology and Radiology*. Tech. rep. Campinas, Brasil: State University of Campinas, School of Medicine (FCM-UNICAMP)



komórki ependymalne
ependymal cells by Michelle L, flickr
www.flickr.com/photos/mluon753/8467081756/in/

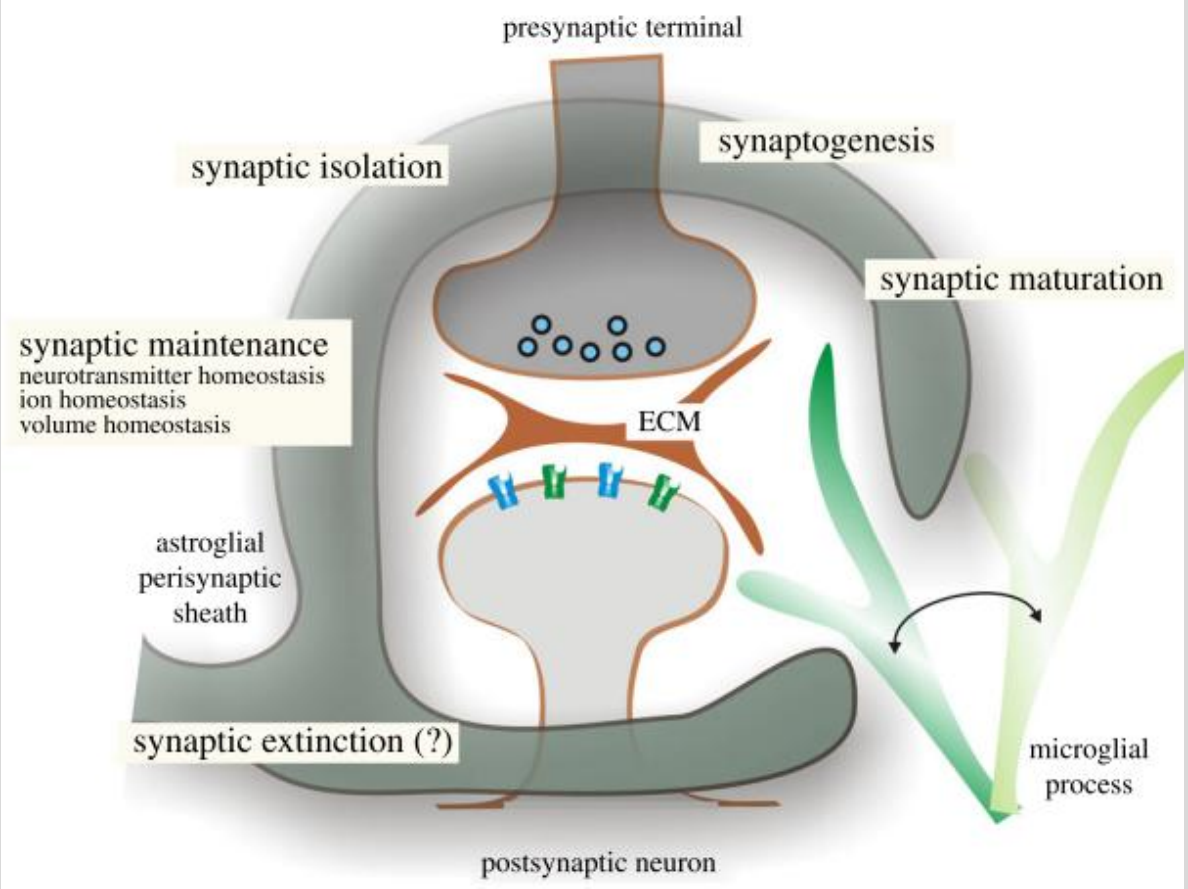
⊕ Hipoteza: Zintegrowana sieć neuronowo-glejowa

M. Nedergaard and A. Verkhratsky (July 2012). "Artifact versus reality–how astrocytes contribute to synaptic events". In: *Glia* 60.7, pp. 1013–1023
M. Sur and M. C. Flanders (2018). *Astrocyte-Neuron interactions in visual cortex circuits*. grant NIH. Picower Institute for Learning and Memory, MIT
D. Orenstein (2018). "Neuroscientists give "invisible" cells a new look". In: *MIT News* VerkhNeder14

- neurony: sygnały binarne - kod impulsowania (ang. *spiking code*) (potencjał czynnościowy neuronów)
- astrocyty: sygnały analogowe - kod wapniowy (ang. *calcium code*) (sprzężenia zwrotne i sprzężenia proste)
  - uwalnianie neurotransmiterów, dyfuzja neurotransmiterów przez płyn pozakomórkowy
  - dyfuzja wtórnych posłańców i metabolitów przez synapsy elektryczne
  - sygnalizacja wapniowa Ca^{++}: synapsy elektryczne między astrocytami wyzwalają cząstki wtórnego posłańca (IP3), IP3 aktywują kanały wapniowe neuronów i uwalniają cząstki Ca^{++}do wnętrza komórki; w rezultacie fala Ca^{++}propaguje z komórki do komórki i stabilizuje poziom wapnia w przestrzeni międzykomórkowej
- *kołyska astroglejowa* (ang. *astroglial cradle*) uczenie - plastyczność synaps
  - izoluje synapsy od sygnałów pozasynaptycznych
  - tłumi przesłuchy pomiędzy synapsami
  - pozwala na długoterminową plastyczność kontaktów synaptycznych (zaburzenia: epilepsja, schizofrenia, zaburzenia koncentracji)

⊕ Hipoteza: Zintegrowana sieć neuronowo-glejowa

Kołyska astroglejowa

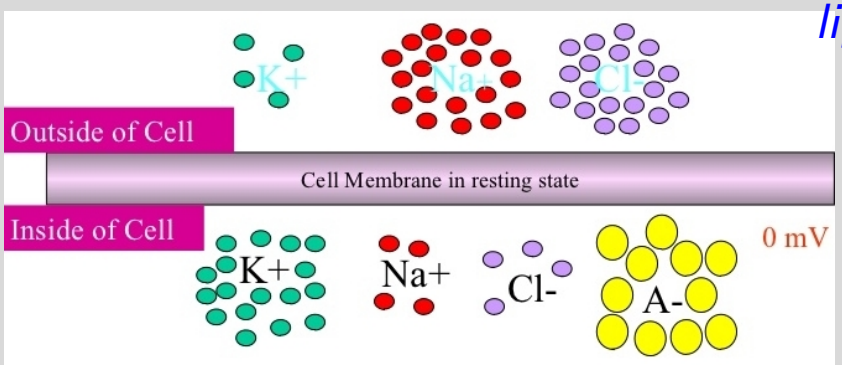


Kołyska astroglejowa

2.1 Elementy systemu nerwowego
2.2 Działanie neuronu
2.3 Biologiczne sieci neuronowe

⊕ Błona komórkowa

Koncentracja jonów



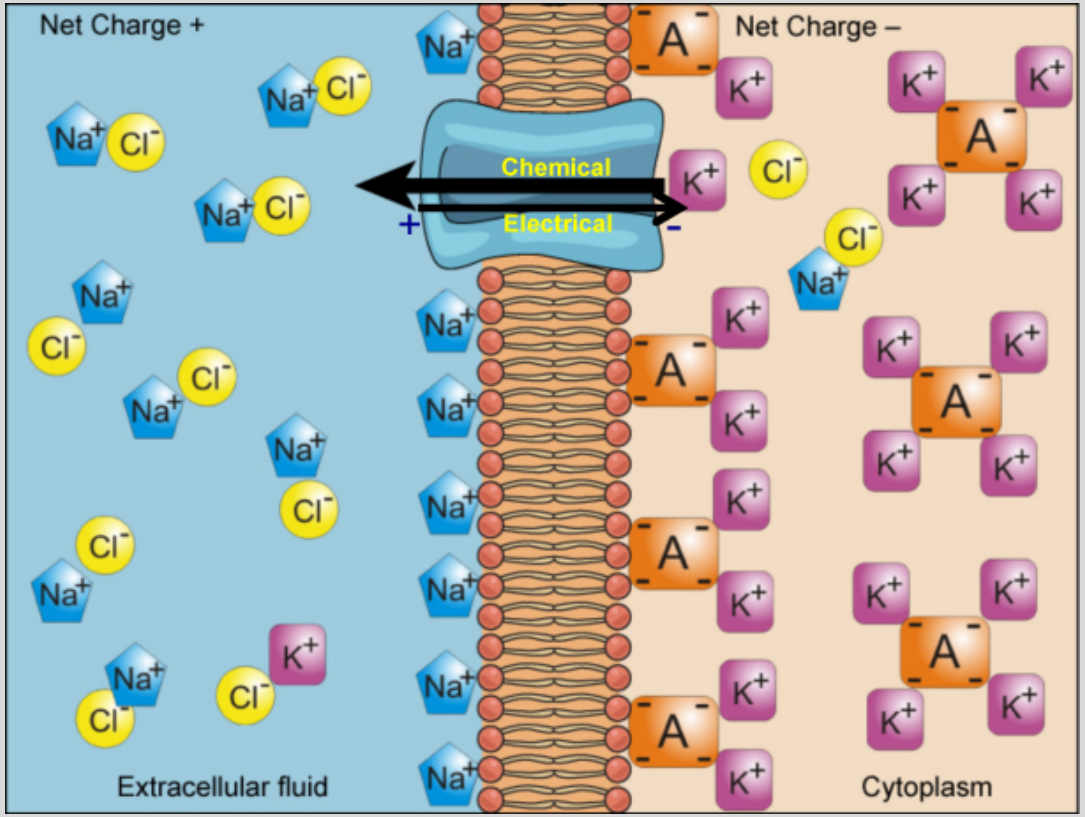
S. Resnick (2012). *Neurophysiology, Brain Anatomy and Physiology*. URL: https://www.slideshare.net/jonijutt/brain-anatomy-physiology

- *błona komórkowa*: *podwójna warstwa lipidowa* (ang. *lipid bilayer*)
  - separuje aniony organiczne A^- i kationy potasu K^+ *wewnątrz komórki*
  - separuje jony sodu Na^+, wapnia Ca^{++} i chloru Cl^- *na zewnątrz komórki*
  - przepuszcza wodę (osmoza) i małe molekuły niezjonizowane (O_2, CO_2)
  - *nieprzepuszczalna* dla jonów (K^+, Na^+, Ca^{++}, Cl^-, HCO_3), małych molekuł rozpuszczalnych (glukoza) i dużych molekuł (proteiny, RNA)

jony	symbol	wewn. mmol/litr	zewn. mmol/litr
proteiny	A^-	29	0
kationy potasu	K^+	140	5
kationy sodu	Na^+	12	145
aniony chloru	Cl^-	13	110
kationy wapnia	Ca^{++}	0.0001	1.5
aniony wodorowęglanu	HCO_3^-	8	27
płyn fizjologiczny	NaCl		9g/l

⊕ Potencjał synaptyczny

- potencjał membrany komórkowej: *potencjał synaptyczny*
- *potencjał spoczynkowy* -55 do -100 mV wzgl. płynu międzykomórkowego
- przepływ jonów przez membranę powoduje zmiany potencjału synaptycznego



S. Telleen (n.d.). *Forces That Move Ions and Molecules*. image by "Synaptitude"/Wikimedia Commons. OpenStax. URL: https://cnx.org/contents/RKGtQxyN@2/Forces-That-Move-Ions-and-Mole

⊛ Potencjał równowagi

- potencjał równowagi: efekt
  - przepływu jonów w kierunku mniejszych koncentracji
  - wzrostu różnicy potencjałów w następstwie tego ruchu
- jeśli błona jest przepuszczalna tylko dla cząstek jednego jonu potencjał równowagi określony przez *równanie Nernsta*
- uogólnienie dla kilku jonów (jednowartościowych): równanie *Goldmana-Hodgina-Katza*

⊛ Potencjał równowagi

⊛ Potencjał równowagi jonu: Równanie Nernsta

Potencjał równowagi jonu o koncentracjach c^{out} i c^{in} po obu stronach membrany

$$E = RT/(zF) \ln c^{\text{out}}/c^{\text{in}}$$

R stała gazowa $= 8.314472\,[J^{\circ}K^{-1}\text{mol}^{-1}]$

T temperatura $[^{\circ}K]$ $(^{\circ}C + 273.15)$

z wartościowość jonu $(\pm 1, \pm 2, \ldots)$

F stała Faradaya $= 96485\,[C/\text{mol}]$

Ⓔ Przykład

Dla $T = 37\,^{\circ}C$:

$$\begin{aligned} E_{\text{K}^+} &= 26.727 \ln(5/100) &&= \text{-80 mV} \\ E_{\text{Na}^+} &= 26.727 \ln(150/15) &&= \text{62 mV} \\ E_{\text{Cl}^-} &= 26.727 \ln(13/150 &&= \text{-65 mV} \\ E_{\text{Ca}^{++}} &= 13.363 \ln(2/0.0002) &&= \text{246 mV} \end{aligned}$$

⊛ Potencjał równowagi jonów: równanie Goldmana-Hodgina-Katza

Potencjał równowagi kationów $\mathcal{K}$ i anionów $\mathcal{A}$ po obu stronach membrany

$$E = RT/F \ln \frac{\sum_{i \in \mathcal{A}} p_i c_i^{\text{in}} + \sum_{i \in \mathcal{K}} p_i c_i^{\text{out}}}{\sum_{i \in \mathcal{A}} p_i c_i^{\text{out}} + \sum_{i \in \mathcal{K}} p_i c_i^{\text{in}}}$$

⊛ Kanały jonowe pasywne

https://med.libretexts.org/Bookshelves/Anatomy_and_Physiology/Book%3A_Anatomy_and_Physiology_(Boundless)/10%3A_Overview_of_the_Nervous_System/10.5%3A_Neurophysiology/10.5B%3A_Ion_Channels Medicine LibreText, Book: Anatomy and Physiology (Boundless), Aug 14, 2020, https://courses.lumenlearning.com/boundless-ap/

- *kanały niebramkowane* (ang. *leakage channels*) - przepuszczalność stała dla określonych jonów
- *kanały bramkowane napięciem* (ang. *voltage-sensitive*) otwierane/zamykane różnicą potencjałów po obu stronach membrany
  - Ⓔ przykład: kanały sodowe Na^+; próg reakcji; $100 - 200/\mu$, 7M jonów na ms. na kanał - udział w generacji i propagacji potencjału czynnościowego
- *kanały bramkowane ligandami* (ang. *ligand-gated channels*) (*bramkowane molekularnie*) otwierane/zamykane w odpowiedzi na molekuły *cząstki sygnalizacyjnej* liganda (np neurotransmitera) wiążące się z proteinami kanału, (zewnętrzną lub wewnętrzną), która sama nie jest przenoszona
  - ⊛ *pierwsi posłańcy* (ang. *first messengers*): zewnątrzkomórkowe molekuły, wywołujące zmiany fizjologiczne w błonie komórkowej
  - ⊛ *wtórni posłańcy* (ang. *second messengers*): wewnątrzkomórkowe substancje uwalniane w odpowiedzi na pierwszych posłańców
- kanały *bramkowane dźwiękiem* (w komórkach włosowatych ucha wewnętrznego), *światłem*, *temperaturą*, *naciskiem*, etc.

⊛ Aktywny transport molekuł

- *oddychanie komórkowe*: cząstki pokarmowe (np. glukoza) są utleniane (do dwutlenku węgla i wody,) a energia gromadzona w postaci ATP (adenosine triphosphate)
- ⊛ *ATP* dostarcza energii komórce przekształcając się w wyniku hydrolizy (absorpcji wody) do *ADP* (adenosine diphosphate) i grupy fosforanowej (ang. *phosphate*)
- *pompy jonowe*: proteiny membrany używające zewnętrzną energię komórkową (ang. *cellular energy*) (ATP) do aktywnego transportu jonów przeciwnie do "gradientu koncentracji"
- *bezpośredni transport aktywny*: proteiny błony komórkowej (*transportery*) łączą się z ATP i wykorzystują energię hydrolizy powodując przepływ jonów i małych molekuł w kierunku przeciwnym do ich koncentracji
  - *pompy wapniowo-potasowe* Na^+/K^+: pompują jony wapnia Na^+na zewnątrz a jony potasu K^+do wnętrza komórki: zużywają do 1/3 energii komórki

⊛ Kanały jonowe

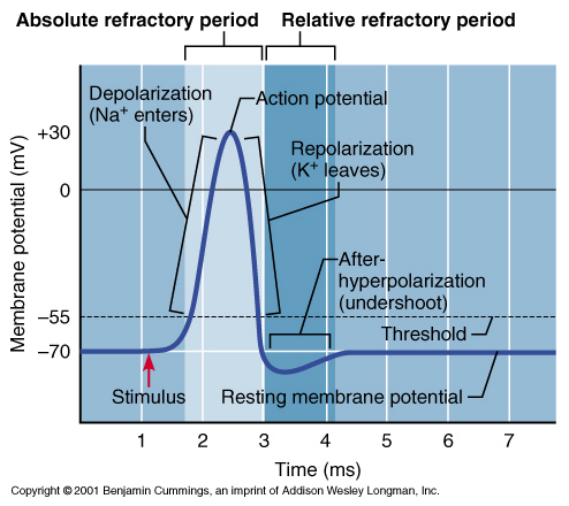
Zmiany potencjału synaptycznego

Michael D. Mann, Ph.D., The Nervous System In Action, 1997-2017, http://michaeldmann.net
OpenStax, Biology. OpenStax CNX

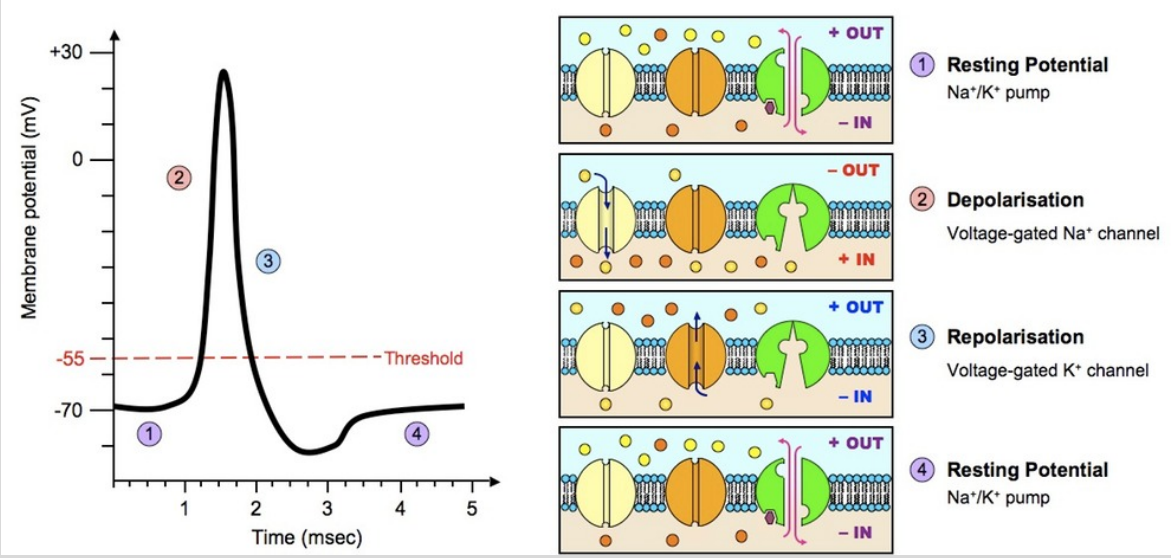
- cząstki neurotransmitera łączą się (ang. *bind*) z receptorami kanałów jonowych (Na^+, Cl^-, K^+, Ca^{++}) *bramkowanych ligandami* (ang. *ligand-gated*) takimi jak neurotransmitery
- wywołuje to
  - depolaryzację ↗ potencjału membrany — potencjał pobudzający *excitatory postsynaptic potential* EPSP — *synapsy pobudzające* (ang. *excitatory*)
    - Ⓔ przykład: neurotransmiter ACh (acetylocholina) otwiera kanały sodowe (bramkowane ligandami); dopływ Na^+wytwarza *potencjał pobudzający* (EPSP)
  - lub polaryzację ↘ potencjału membrany — potencjał hamujący *inhibitory postsynaptic potential* IPSP — *synapsy hamujące* (ang. *inhibitory*)
    - Ⓔ przykład: neurotransmiter GABA wiążąc się z receptorami $GABA_A$ (1 ms.) otwierając kanały chlorkowe (dopływ Cl^-) i z receptorami $GABA_B$ otwierając kanały potasowe (wypływ K^+), co wytwarza *potencjał hamujący* (IPSP)
- przekroczenie progu wywołuje potencjał czynnościowy w komórce postsynaptycznej
- neurotransmiter jest usuwany z przestrzeni synaptycznej poprzez rozpad enzymatyczny (ACh) lub aktywny transport do aksonu presynaptycznego

⊕ Potencjał czynnościowy

- bodziec depolaryzuje ↗ membranę
- integracja depolaryzacji aksonu na *wzgórku aksonu* (ang. *axon hillock*) (niski próg)
- po osiągnięciu *potencjału progowego* otwierają się kanały sodowe sterowane napięciem – **dopływ jonów sodowych** Na^+ dodatkowo depolaryzuje membranę ↗
- zamykają się kanały sodowe
- otwierają się sterowane napięciem kanały potasowe – **wypływ jonów potasowych** K^+ ↘ - faza depolaryzacji
- *hiperpolaryzacja* (polaryzacja błony poniżej wartości spoczynkowej) ↘, neuron nie impulsuje (okres odpoczynku)
- zamykają się kanały potasowe
- *pompy sodowo-potasowe* usuwają jony Na^+ i dostarczają jony K^+ przywracając potencjał spoczynkowy



https://legacy.owensboro.kctcs.edu



⊕ Dodatkowe uwagi

- ze względu na upływność jonów (ang. *ion leakage*), neurony mogą wytwarzać impuls nawet bez stymulacji
- niektóre dendryty mogą wytwarzać impulsy, niektóre aksony nie impulsują
- zakłada się, że informacja jest przenoszona przez
  - impulsy (ang. *spikes*)
  - wzorce w postaci serii impulsów (ang. *trains of impulses*)
- całkowity czas sygnału pojedynczego impulsu: 0.2 - 5 msec, częstość impulsowania u człowieka średnio 100 Hz (niektóre neurony nawet do 1000 Hz)
- prędkość transmisji sygnału 0.9 -90 mm/msec.

⊛ Model Hodgkina-Huxleya, 1952

- prąd membrany

$$C_m \frac{\mathrm{d}u(t)}{\mathrm{d}t} = i_{\mathsf{K}} + i_{\mathsf{Na}} + i_{\mathsf{L}}$$

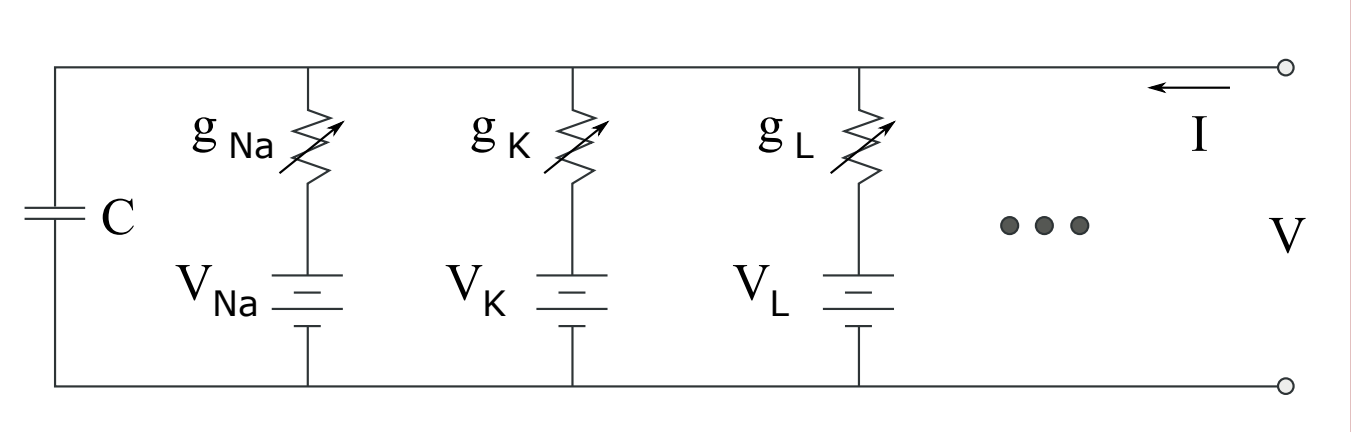
gdzie

$$i_{\mathsf{Na}} = (U_{\mathsf{Na}} - u(t))\, g_{\mathsf{Na}} \qquad \text{kanał sodowy}$$

$$i_{\mathsf{K}} = (U_{\mathsf{K}} - u(t))\, g_{\mathsf{K}} \qquad \text{kanał potasowy}$$

$$i_{\mathsf{L}} = -u(t)\, \bar{g}_{\mathsf{L}} \qquad \text{upływność}$$

a potencjały równowagi U_{Na} and U_{K} dane są równaniami Nernsta



http://www.vce.bioninja.com.au/

⊛ Model Hodgkina-Huxleya, 1952 (2)

- stany otwarcia m, n lub zamknięcia h bramek sterowanych napięciem zmieniają się zgodnie z niezależnymi procesami Markowa

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha_x(V)(1-x) - \beta_x(V)x, \qquad x \in \{m, n, h\}$$

  gdzie $1/\alpha$, $1/\beta$ są stałymi czasowymi
- przewodności kanałów

$$g_{\mathsf{K}} = n^4(t)\,\bar{g}_{\mathsf{K}}$$
$$g_{\mathsf{Na}} = m^3(t)\,h(t)\,\bar{g}_{\mathsf{Na}}$$

  tzn. kanał potasowy jest otwarty, gdy cztery bramki h są w stanie 1, a kanał sodowy jest otwarty gdy trzy bramki n i bramka h są w stanie 1,
- dla aktywacji m, n, intensywności $\alpha_x(u)$ and $\beta_x(u)$ są nieliniowymi funkcjami napięcia

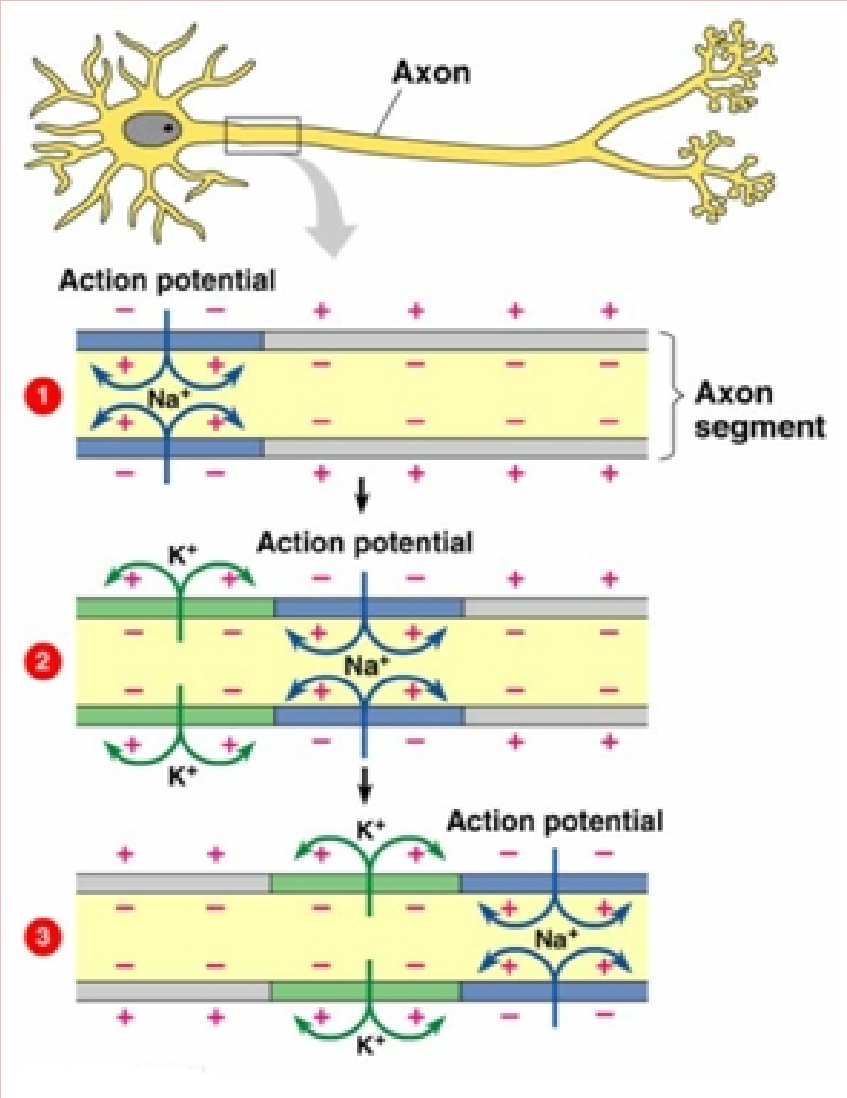
$$\alpha_x = \frac{-L_x}{-1-\exp(-L_x)}, \qquad \beta_x = c_x \exp(-M_x), \qquad x \in \{m, n\}$$

  gdzie L_x, M_x są liniowymi funkcjami u o dodatnich współczynnikach, $c_x > 0$, a dla inaktywacji role α and β są zamienione

⊛ Transmisja sygnału wzdłuż aksonu do synapsy

Transmisja ciągła

- akson niemielinowany
- zmiana potencjału, powodując przepływ jonów, przemieszcza się przez membranę aksonu depolaryzując sąsiednie miejsca and powodując "wędrówkę" procesów polaryzacji i depolaryzacji
- ze względu na depolaryzację, przemieszczanie się w przeciwnym kierunku nie jest możliwe



ibbiology wikifoundry

⊛ Transmisja sygnału wzdłuż aksonu do synapsy

Transmisja skokowa

- *mielina* (ang. *myelin sheath*) tworzona jest przez komórki glejowe (mózg) i komórki Schwanna (włókna obwodowe)
- mielina izoluje akson znacznie silniej tłumiąc sygnał zwiększając szybkość sygnału (z 0.5 m/s. dla aksonów niemielinowanych do 130 m/s dla aksonów mielinowanych)
- potencjał czynnościowy jest wzbudzany tylko w *węzłach Ranviera* (ang. *Ranvier nodes*): dopływ jonów sodu Na^+ wywołuje depolaryzację wystarczającą do przekroczenia progu w następnym węźle

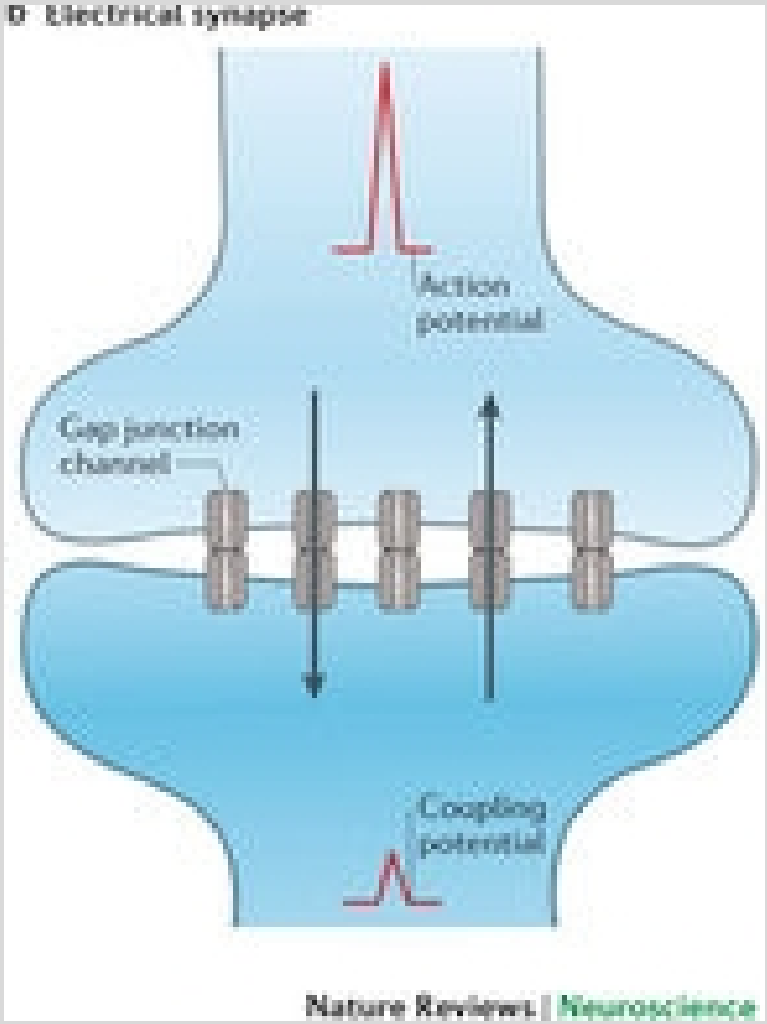
⊛ Synapsy

- komunikacja między neuronami odbywa się głównie za pośrednictwem struktur zwanych *synapsami*
- synapsy łączą **akson z dendrytem** lub **akson z aksonem**, ale możliwe są także inne kombinacje
- neuron (i jego struktury) wysyłający sygnał nazywamy *presynaptycznym*, a neuron odbierający sygnał - *postsynaptycznym*
- w zależności od sposobu przekazywania informacji rozróżniamy synapsy chemiczne i synapsy elektryczne
- synapsy chemiczne są znacznie (10^3) razy wolniejsze niż elektryczne, ale pozwalają na wzmacnianie sygnału

⊕ Synapsy

Synapsy elektryczne

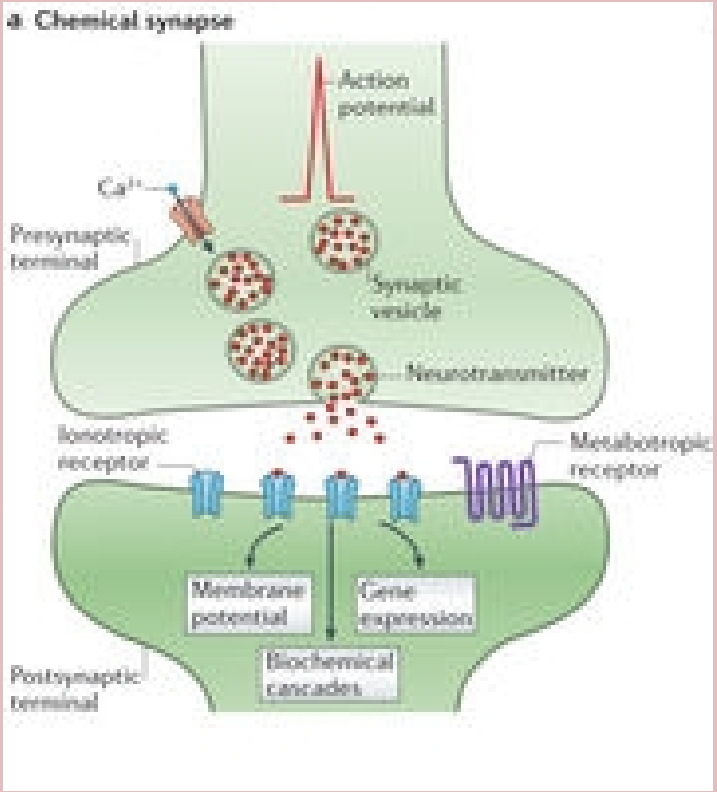
- *synapsa elektryczna* umożliwia przepływ jonów bezpośrednio do komórki postsynaptycznej, poprzez szczeliny proteinowe (ang. *gap junction proteins*) w przylegających (ang. *fused*) (2-4 nm) membranach komórek



⊛ Synapsy

Synapsy chemiczne

- *synapsa chemiczna*: *kolbka synaptyczna* (ang. *synaptic knob*) w aksonie - *szczelina synaptyczna* 15–200 nm - *kolec dendrytyczny* (ang. *dendritic spine*)
- kolbki wypełnione są *pęcherzykami (ang. synaptic vesicles)* ϕ 50–200 *nm* zawierającymi cząstki *neurotransmitera*
- *prawo Dale'a*: neuron produkuje tylko jeden rodzaj neurotransmitera ale może mieć receptory różnych neurotransmiterów
- *synapsa chemiczna* uwalnia cząstki *neurotransmitera* z neuronu presynaptycznego, które przepływają przez *szczelinę synaptyczną* (ok. 20 nm) i pobudzają receptory w błonie postsynapstycznej, umożliwiając przepływ jonów do komórki postsynapstycznej



Alberto E. Pereda, Electrical synapses and their functional interactions with chemical synapses, Nature Reviews Neuroscience 15, 250–263 (2014)

⊛ Synapsy

Potencjały presynaptyczne

Michael D. Mann, Ph.D., The Nervous System In Action, 1997-2017, http://michaeldmann.net; http://cnx.org/contents/185cbf87-c72e-48f5-b51e-f14f21b5eabd@10.117, OpenStax, Biology. OpenStax CNX.

- *synapsa* jest połączeniem zakończenia aksonu *neuronu presynaptycznego* tworzącego *kolbkę synaptyczną* (ang. *synaptic knob*) z dendrytem neuronu *postsynaptycznego* (na mięśniach: złącza neuromuskularne)
- kolbka synaptyczna jest wypełniona pęcherzykami zawierającymi neurotransmitter
- *presynaptyczny potencjał czynnościowy* dochodząc do zakończenia aksonu (*kolbki synaptycznej* (ang. *synaptic knob*)) otwiera kanały sodowe bramkowane napięciem
- potencjał czynnościowy na kolbce synaptycznej otwiera kanały wapniowe Ca^{++}
- jony wapnia Ca^{++} uaktywniają ruch *pęcherzyków synaptycznych* (ang. *synaptic vehicles*) i wywołuje rozpad niektórych pęcherzyków
- ich neurotransmiter jest uwalniany do *szczeliny synaptycznej* (ang. *synaptic cleft*)

⊕ Synapsy

Połączenia neuronów

Synapsy pomiędzy dwoma aksonami (czerwone) i komórką neuronową (żółta). Kolorowany obraz mikroskopu skanującego powiększenie 10000 razy
Wywiad z Rodolfo Llinásem. "The electric brain," *NOVA Web site*, http://www.pbs.org/wgbh/nova/mind/electric.html (accessed Dec. 1,2008).

- liczba synaps ok. $10^{11} - 10^{12}$
- *gęstość* synaps: $750 \cdot 10^6/\text{mm}^3$
- *zbieżność* 8000 synaps /neuron
  - do ponad 100 000 (komórki Purkiniego)
- *rozbieżność*: 1000 synaps dla aksonu
  - komórka granularna - sygnał do ok. 300 komórek Purkiniego
  - 200 synaps na 1 mm długości aksonu

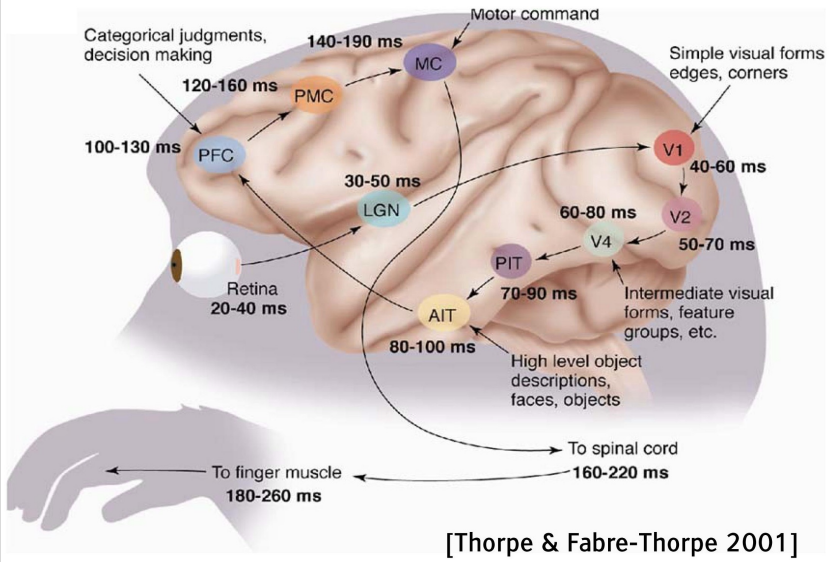
2.1 Elementy systemu nerwowego
2.2 Działanie neuronu
2.3 Biologiczne sieci neuronowe

⊕ Naturalne systemy neuronowe

Połączenia neuronów

- połączenia pomiędzy
  - wyróżnionymi komórkami
  - grupami komórek
  - niespecyficzne
- *sprzężenia zwrotne*

  przykład: kora móżdżku ⇒ jądra podstawowe ⇒ talamus ⇒ kora móżdżku
- rodzaje przewodnictwa
  - substancja szara: ciała neuronów, dendryty, synapsy; kora mózgu i móżdżku, jądra mózgu, rdzeń kręgosłupa, niemielinowane aksony
  - *substancja biała*: mielinowane aksony



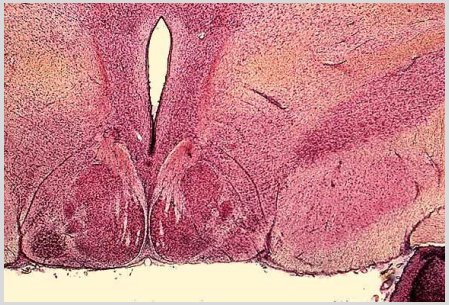
[Thorpe & Fabre-Thorpe 2001]

Y. Le Cun (2016). *Deep Learning*. URL: //indico.cern.ch/event/510372/attachments/1245509/1840815/lecun-20160324-cern.pdf intelligent machines

⊕ Naturalne systemy neuronowe

Topologia połączeń

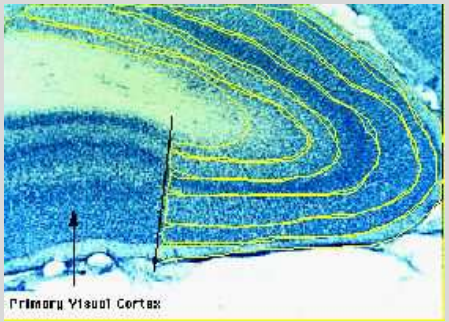
- *jądra*



jądro ciała suteczkowatego w podwzgórzu

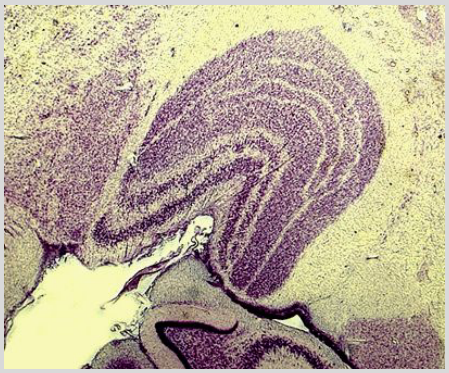
- *warstwy*

  kora mózgu (6 warstw)
  kora móżdżku (3 warstwy)



kora potyliczna : pierwszorzędowa kora wzrokowa (left, 7 layers) drugorzędowa kora wzrokowa (right, 6 layers)

- *pofałdowane*, *nałożone*



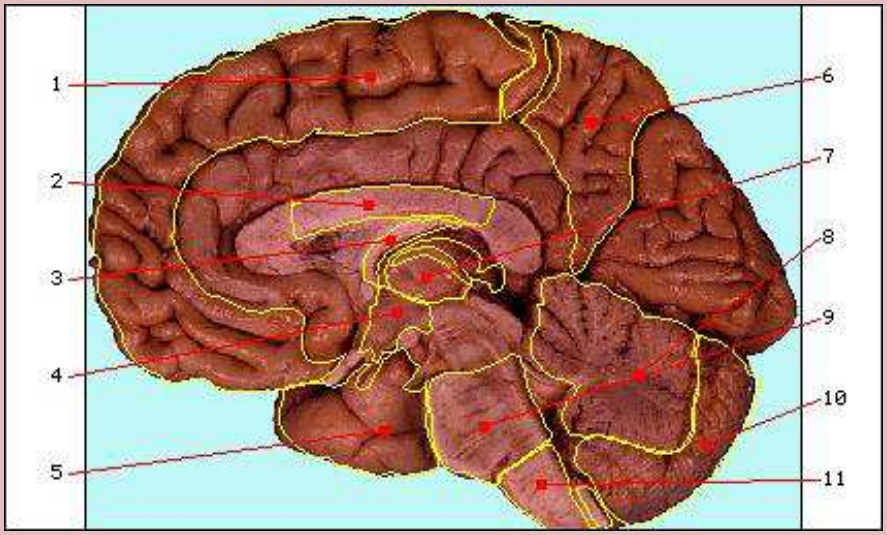
Lateral Geniculate Nucleus (we wzgórzu): jądro utworzone z 6 pofałdowanych warstw, warstwy 1, 4, 6 i 2, 3, 5 otrzymują informację z jednego oka; warstwy 1 i 2 przetwarzają położenie i ruch, warstwy 3-6 wspomagają ostrości widzenia i przetwarzają kolor

wszystkie zdjęcia za zezwoleniem: Digital Anatomist Project at the University of Washington,

⊛ Naturalne systemy neuronowe

Mózg: Przekrój pionowy

prawa półkula mózgu



$90 \cdot 10^9$ neurons, 0.2 m^2, *bruzdy*, *płaty*

fotografia za zgodą Digital Anatomist Project
at the University of Washington

1 *płat czołowy*, kora ruchowa M1
mowa, przetwarzanie informacji, uwaga

2 *ciało modzelowate*

3 *sklepienie*

4 *podwzgórze*

5 *płat skroniowy, kora słuchowa A1*

6 *płat ciemieniowy, kora somato-sensoryczna S1*, skóra, pozycja

7 *wzgórze*

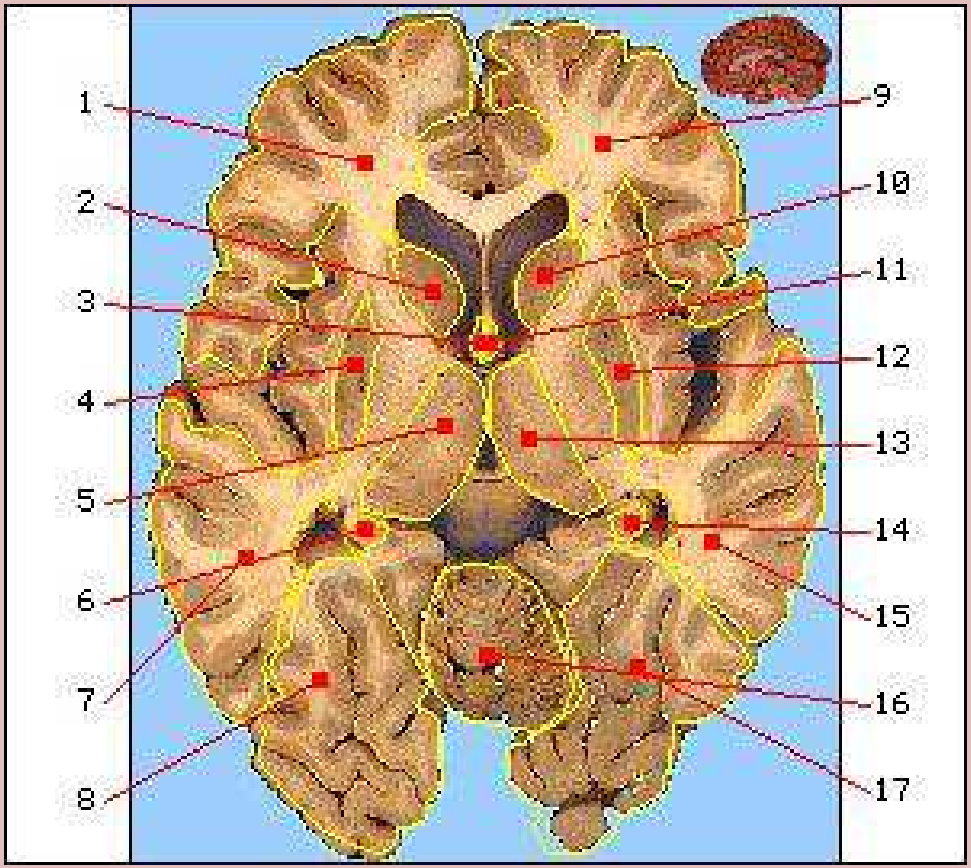
8,10 *móżdżek*

9 *most*

11 *rdzeń przedłużony*

⊛ Naturalne systemy neuronowe

Mózg: przekrój poziomy



fotografia: za zgodą Digital Anatomist Project at the University of Washington

1,9 *płat czołowy*: *kora ruchowa* M1 mowa, przetwarzanie informacji, uwaga

2,10 *jądro ogoniaste*

3,11 *sklepienie*

4,12 *skorupa*

5,13 *wzgórze*

6,14 *hipokamp*: pamięć asocjacyjna, pamięć krótkoterminowa, orientacja 3D, emocje (*układ limbiczny*)

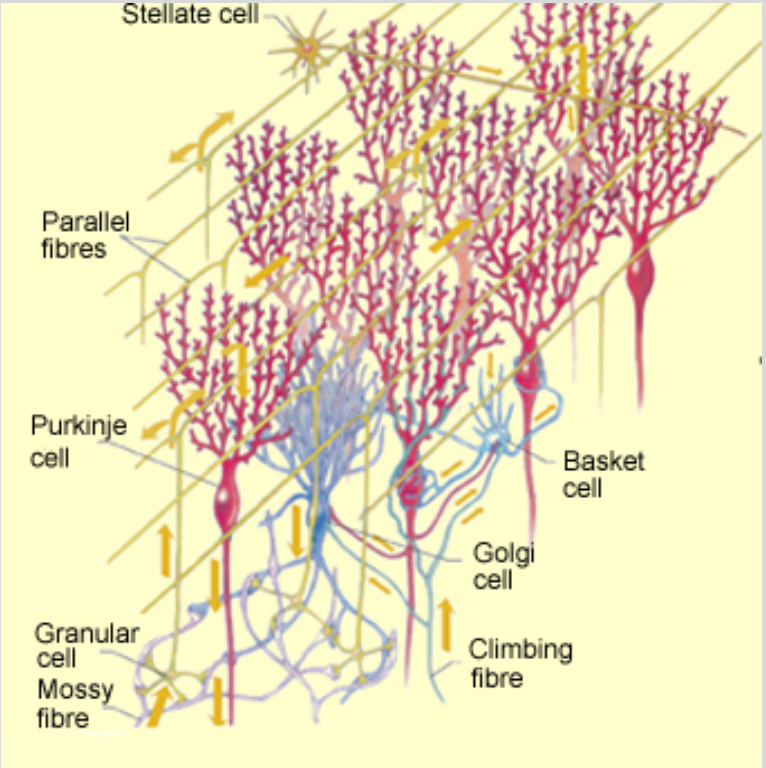
7,15 *płat skroniowy* kora słuchowa A1

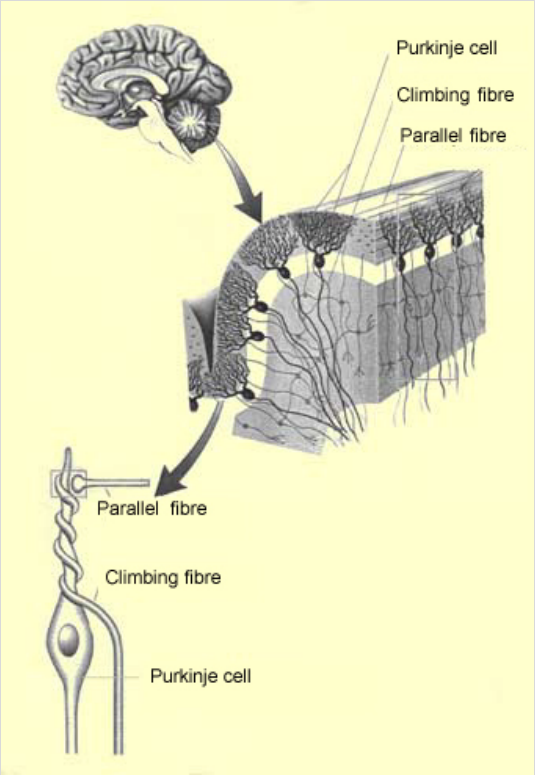
16 *móżdżek*

8,17 *płat potyliczny*, kora wzrokowa V1

⊕ Naturalne systemy neuronowe

Móżdżek





włókna równoległe i włókna pnące

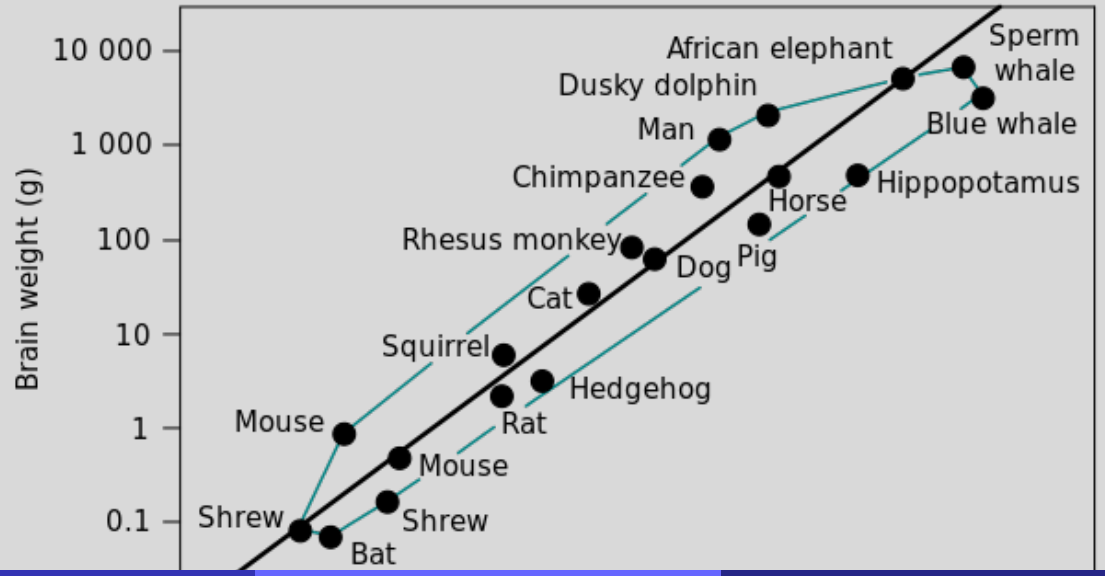
fotografia: za zgodą Digital Anatomist Project at the University of Washington

- struktura połączeń móżdżku
  - włókno mszyste (ang. *mossy fiber*) do ok 500 komórek granularnych
  - komórka granularna do ok. 300 komórek Purkiniego;
  - komórka Purkiniego od ponad 200000 komórek granularnych
  - łącznie ok 10^{14} synaps

⊕ Elementy budowy biologicznych systemów neuronowych

Neurony w mózgu

- liczba neuronów: w mózgu ludzkim $\sim$ 86 mld. neuronów (Herculano-Houzel and Lent (2005), Azevedo et al. (2009)) (w tym kora mózgu: $\sim$ 16 mld. neuronów) móżdżek: 69 mld. neuronów
- liczba innych komórek: w mózgu ludzkim ~ 85 mld.
- masa mózgu: 1508 g (w tym kora 1233 g) móżdżek: 154g
- objętość mózgu 1400 cm3,
- powierzchnia kory mózgu: 2400 cm^2



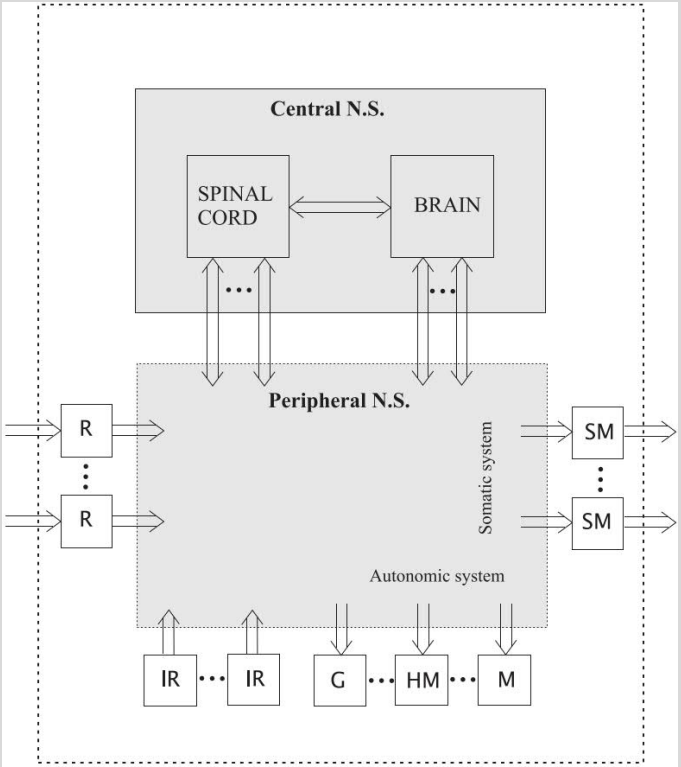
⊕ System nerwowy

System nerwowy jako układ sterowania

uwaga • rozróżnianie bodźców • pamięć • rozpoznawanie • kojarzenie • porównywanie • komunikacja • dedukcja • uczenie • inteligencja • świadomość

R *receptory:*
środowiska odległego
(oczy, uszy)
środowiska bliskiego
(receptory skóry)
pozycji
(receptory w labiryncie, przeguby, ścięgna mięśnie)

IR *receptory wewnętrzne*



SM mięśnie prążkowane

G gruczoły

HM mięśnie serca

M mięśnie gładkie

⊕ Sterowanie

- receptory: neurony przetwarzające sygnały fizyczne w elektryczne
  - *telereceptors*: rozpoznawanie środowiska odległego (uszy, oczy)
  - *exteroceptors*: rozpoznawanie środowiska bliskiego (receptory skóry)
  - *proprioceptors*: określanie położenia ciała w środowisku (labirynt, receptory mięśni, ścięgien, przegubów)
  - *interoceptors*: rozpoznawanie stanu organów wewnętrznych
- receptory: podział tradycyjny - *zmysły*: smak, węch, słuch, wzrok, dotyk
  - podziały bardziej szczegółowe, np. dotyk: lekki, lekki ucisk, silny ucisk, wibracja, temperatura, ból, etc.
  - inne zmysły: *ultra-fiolet* (pszczoły), *podczerwień* (węże), *fale EM 50–5000 Hz* (ryba elektryczna), *fale EM 0.1–20 Hz* (rekiny), *przemieszczenie wody* (wodne kręgowce), *pole magnetyczne* (gołębie)
- koncepcje układów sterowania
  - system hormonalny (ang. *endocrine*) (hormonalny) poprzez wydzielanie hormonów do krwi przez gruczoły hormonalne i przenoszenie ich do odpowiednich komórek
  - system nerwowy komunikujący CNS z mięśniami i gruczołami za pomocą sygnałów elektrycznych (o naturze chemicznej) - szybszy, bardziej zlokalizowany
  - zintegrowana sieć neuronowo-glejowa

Literatura

Materiały źródłowe

Hebb, D. O. (1949). *The organization of behavior*. New York: J. Wiley and Sons.

Nedergaard, M. and A. Verkhratsky (July 2012). "Artifact versus reality–how astrocytes contribute to synaptic events". In: *Glia* 60.7, pp. 1013–1023.

Orenstein, D. (2018). "Neuroscientists give "invisible" cells a new look". In: *MIT News*.

Queiroz, L. S. and R. A. Paes (2019). *Neuroimaging website of Dept. of Pathology and Radiology*. Tech. rep. Campinas, Brasil: State University of Campinas, School of Medicine (FCM-UNICAMP).

Sur, M. and M. C. Flanders (2018). *Astrocyte-Neuron interactions in visual cortex circuits*. grant NIH. Picower Institute for Learning and Memory, MIT.

Telleen, S. (n.d.). *Forces That Move Ions and Molecules*. image by "Synaptitude"/Wikimedia Commons. OpenStax. URL: https://cnx.org/contents/RKGtQxyN@2/Forces-That-Move-Ions-and-Mole.

Materiały pomocnicze

Arbib, M. A., ed. (1998). *The handbook of Brain Theory and Neural Networks*. Cambridge, Massachusetts: MIT Press.

Azevedo, F. A. et al. (2009). "Equal Numbers of Neuronal and Nonneuronal Cells Make the Human Brain an Isometrically Scaled-Up Primate Brain". In: *The Journal of Comparative Neurology* 513, pp. 532–541.

Literatura (cd.)

Churchland, P. S. (1989). *Neurophylosophy. Toward a United Science of the Mind Brain*. Cambridge, Massachusetts: A Bradford Book. The MIT Press.

Gołąb, B. and W. W. Traczyk (1986). *Anatomia i fizjologia człowieka*. Warszawa: PZWL.

Herculano-Houzel, S. and R. Lent (2005). "Isotropic fractionator: a simple, rapid method for the quantification of total cell and neuron numbers in the brain". In: *J. Neurosci.* 25.

Le Cun, Y. (2016). *Deep Learning*. URL: //indico.cern.ch/event/510372/attachments/1245509/1840815/lecun-20160324-cern.pdf.

Resnick, S. (2012). *Neurophysiology, Brain Anatomy and Physiology*. URL: https://www.slideshare.net/jonijutt/brain-anatomy-physiology.

Schüz, A. (n.d.). "Neuroanatomy in Computational Perspective". In: *Arbib 1998*, pp. 622–626.

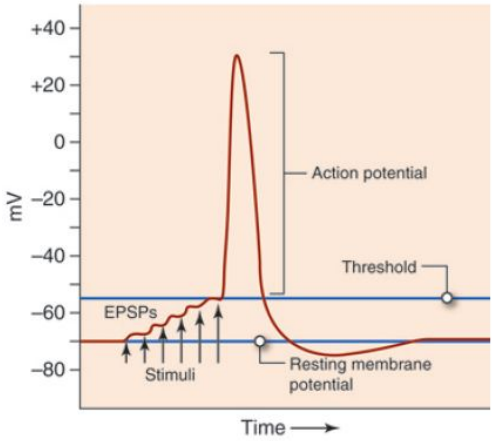
Temat 3: Sztuczne sieci neuronowe - przegląd

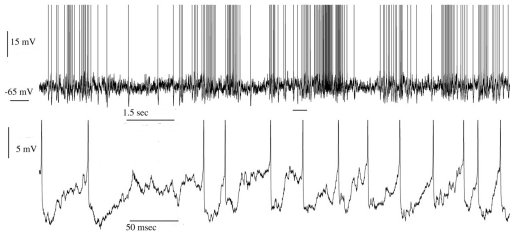
3.1 Sygnały w sztucznych sieciach neuronowych
3.2 Sztuczny neuron
3.3 Warstwy
3.4 Konstrukcja sztucznych sieci neuronowych
3.5 Uczenie sieci
3.6 Sieci statyczne a sieci dynamiczne
3.7 Tablice wielowymiarowe i tensory
3.8 ® Splot 2D

3.1 Sygnały w sztucznych sieciach neuronowych
3.2 Sztuczny neuron
3.3 Warstwy
3.4 Konstrukcja sztucznych sieci neuronowych
3.5 Uczenie sieci
3.6 Sieci statyczne a sieci dynamiczne
3.7 Tablice wielowymiarowe i tensory
3.8 ® Splot 2D

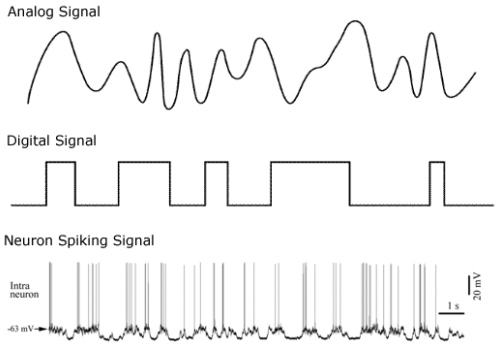
Inspiracje biologiczne

Sygnały w sieciach naturalnych: struktura czasowa





D. Chen, E.E. Fetz, "Characteristic Membrane Potential Trajectories in Primate Sensorimotor Cortex Neurons Recorded In Vivo", Journal of Neurophysiology vol. 94, 2005, 2713-2735

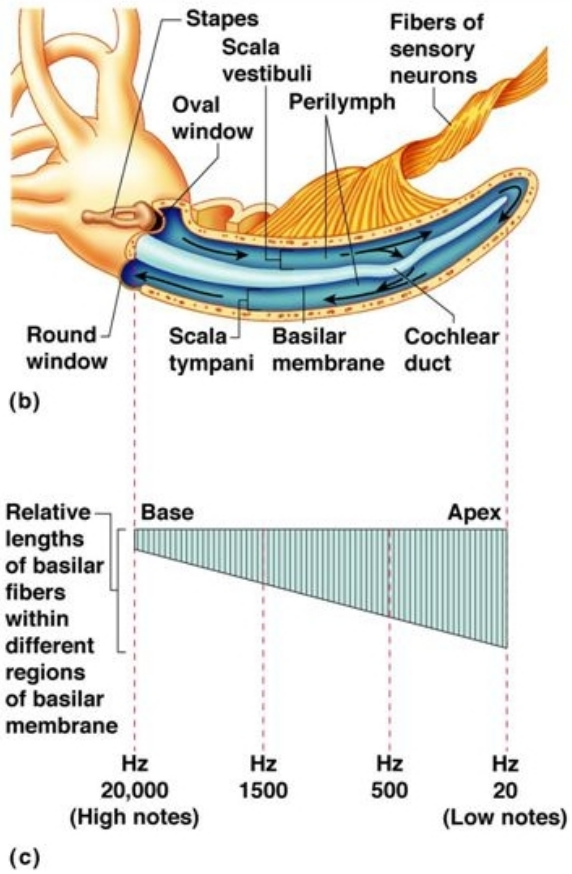


Wersje sygnału neuronu sztucznego inspirowane przez różne skale czasu obserwacji błony komórkowej neuronu i liczności obserwowanych neuronów

- sygnał analogowy: potencjał błony komórkowej
- sygnał binarny: aktywność neuronu (0-1)
- sygnał impulsowy: odległości między impulsami neuronu
- sygnał analogowy: intensywność impulsowania neuronu lub średnia intensywność impulsowania zbioru neuronów

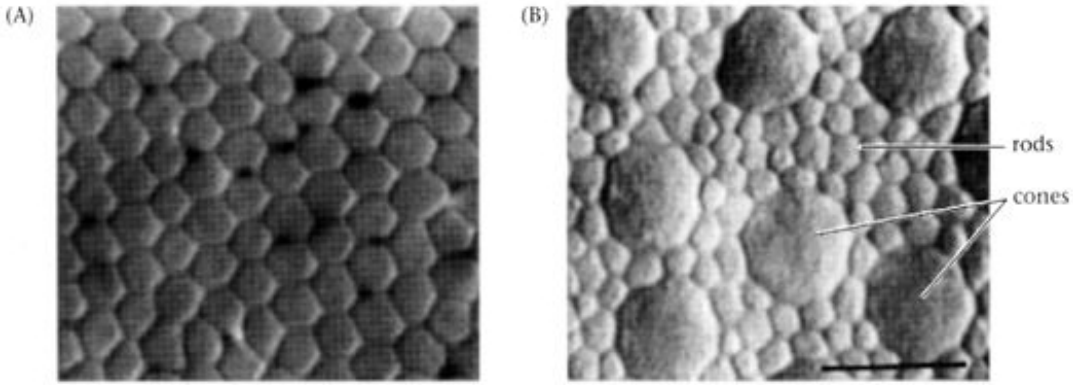
Inspiracje biologiczne

Sygnały w sieciach naturalnych: struktura przestrzenna (1)

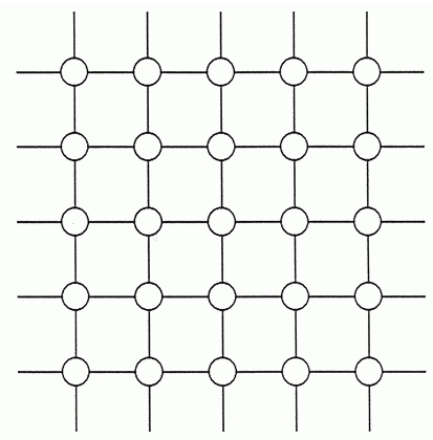


mechanizm słuchu
Elaine N. Marieb, Essentials of human anatomy and physiology,
`http://slideplayer.com/slide/7601597/25`

$\boldsymbol{u} \in \mathbb{R}^p$
tablica 1D, wektor



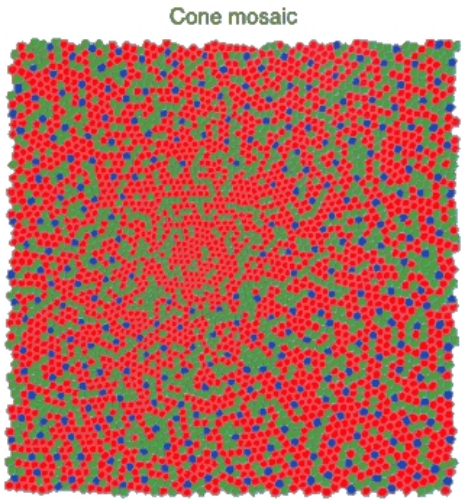
fotoreceptory siatkówki: A) czopki (ang. *cones*) w centrum, B) czopki i pręciki (ang. *rods*) poza centrum
Christine A. Curcio e.a. *Human photoreceptor topography*. J. Comp. Neurology, Feb. 1990, vol. 292 n.4, pp. 497-523



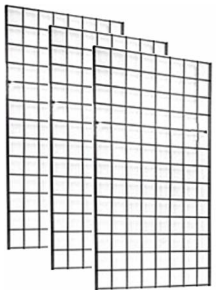
$\boldsymbol{u} \in \mathbb{R}^{p' \times p''}$
tablica 2D, macierz

Inspiracje biologiczne

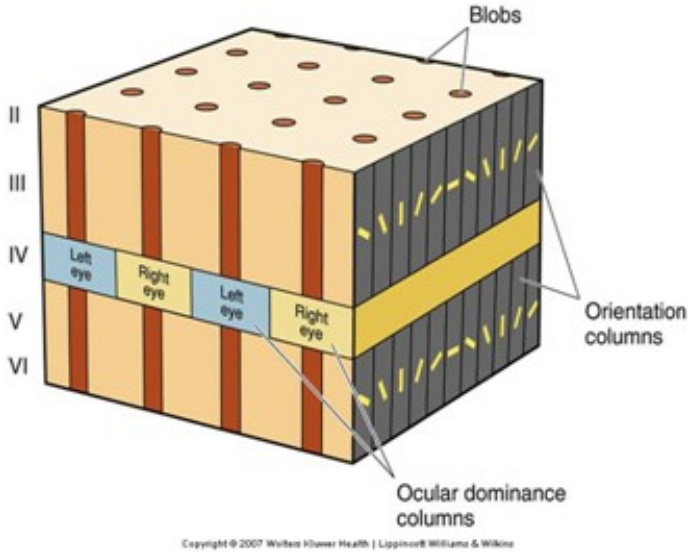
Sygnały w sieciach naturalnych: struktura przestrzenna (2)



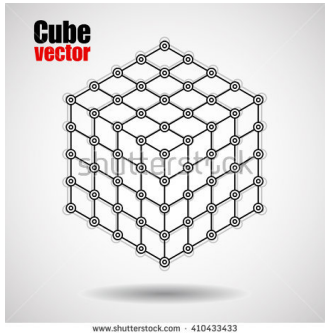
rozkład czopków R,G,B w pobliżu centrum siatkówki
Ethan D. Montag, JIMG 774 Vision and Psychophysics
`https://www.cis.rit.edu/people/faculty/montag/vandplite/pages/chap_9/ch9p1.html`



$\boldsymbol{u} \in \mathbb{R}^{p' \times p'' \times 3}$
tablica 3D



moduły kory mózgu: struktura kolumnowa obszaru V1 kory
Lippincott, Williams and Wilkins, Neuroscience: Exploring the Brain, Wolters Kluwer Health 2007, Chapter 10: The Central Visual System.
http://slideplayer.com/slide/5776936/



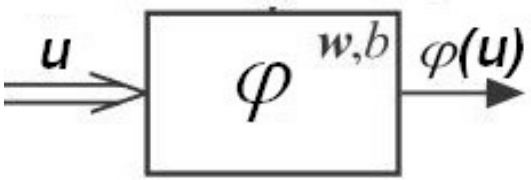
$\boldsymbol{u} \in \mathbb{R}^{p' \times p'' \times p'''}$
tablica 3D

Sygnały w sztucznych sieciach neuronowych

- typowa sztuczna sieć neuronowa ma postać tzw. *warstw neuronów*, które są połączone szeregowo
- stosuje się szereg rodzajów warstw, od których zależą własności sieci
- sygnały w sieciach neuronowych zwykle do jednej z trzech klas:
  - skalarów, będących np. sygnałami wyjściowymi neuronów
  - wektorów, będących np. sygnałami wyjściowymi tzw. warstw perceptronowych, lub sygnałami wejściowymi (zwykle wektorami cech)
  - tablic rzędu 2 (macierze) lub rzędu 3 reprezentujących obrazy lub przekształcone obrazy - najczęściej jako wejścia i wyjścia tzw. warstw splotowych
- wszystkie powyższe klasy sygnałów obejmuje się wspólną nazwą *tensorów*

3.1 Sygnały w sztucznych sieciach neuronowych
3.2 Sztuczny neuron
3.3 Warstwy
3.4 Konstrukcja sztucznych sieci neuronowych
3.5 Uczenie sieci
3.6 Sieci statyczne a sieci dynamiczne
3.7 Tablice wielowymiarowe i tensory
3.8 ® Splot 2D

Struktura neuronu



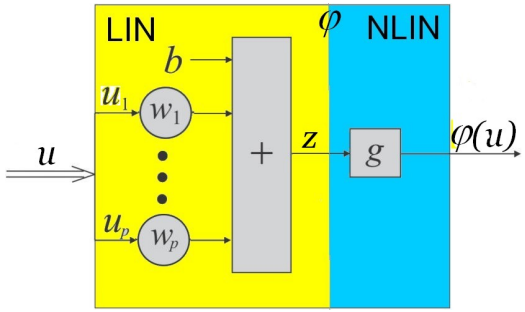
φ: neuron jako transformacja sygnałów,
$\mathcal{R}$ pole recepcyjne, $\boldsymbol{u} \in \mathcal{R}$: wejście,
$\boldsymbol{w}, b$: parametry neuronu

- źródła sygnału wejściowego neuronu (inne neurony lub sygnały wejściowe sieci) nazywamy *polem recepcyjnym* $\mathcal{R}$ tego neuronu
- *sygnał wejściowy* neuronu zapisujemy w postaci tablicy
  - tablicy 1D (np. wektor cech obrazu lub obraz wektoryzowany)
  - tablicy 2D (np. macierz stopniach szarości obrazu, dla tzw. sieci splotowych)
  - 3D - tablicy 3D (np. obraz RGB dla sieci splotowych)
- wektoryzacja tabeli
  - tablicę można wektoryzować, tzn. zapisać postaci wektora (w dowolny ustalony sposób)
  - zaleta: unifikacja własności
  - wada: wektoryzacja gubi związki przestrzenne między elementami obrazu (topologię obrazu)
  - wada: trudne wykrywanie i operacje na kierunkach, skupieniach, wzorcach
- *sygnał wyjściowy* neuronu jest skalarny

Równanie neuronu

- standardowe równanie neuronu

$$y = \varphi(\boldsymbol{u}) = g\Big(\overbrace{b + \underbrace{\sum_{u_i \in R} w_i u_i}_{\substack{\text{suma ważona wejść} \\ \text{biologia: działanie synapsy}}}}^{\substack{\text{biologia: potencjał czynnościowy} \\ \text{część afiniczna } z}}\Big)$$



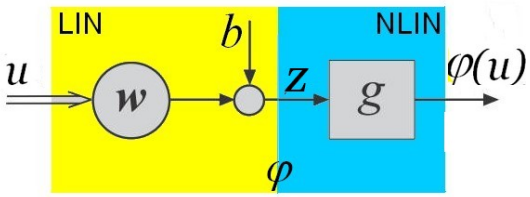
$\boldsymbol{u} \in \mathcal{R}$: sygnały w polu recepcyjnym,
b: *obciążenie*, $\boldsymbol{w}$: *wagi*,
z: (skalarny) sygnał liniowy,
g: *funkcja aktywacji*

  - część afiniczna z ("filtr liniowy"): suma *obciążenia* oraz sumy ważonej wejść
  - część nieliniowa: *funkcja aktywacji* części afinicznej $y = g(z)$

- nieliniowość neuronów jest ich niezbędną cechą umożliwiającą aproksymacje i klasyfikacje przez sieci neuronowe
- *(multi)indeksy* $\boldsymbol{i}$ zależą od struktury tablicy wejściowej, np.
  - i dla wektorów lub tablic wektoryzowanych
  - x, y dla macierzy (np. obrazu 2D w skali szarości)
  - x, y, d dla warstwy 3D (np. obrazu RGB lub zbioru obrazów 2D)
  - $i_1, \ldots, i_n$ dla warstwy nD

Równanie neuronu

Parametry



schemat wektorowy neuronu

- wektorowe równanie neuronu

$$\varphi(\boldsymbol{u}) = g(b + \boldsymbol{w}^{\mathsf{T}}\boldsymbol{u}_R)$$

gdzie $\boldsymbol{u}_R$ jest podwektorem wektora $\boldsymbol{u}$ o elementach z pola recepcyjnego

- parametry: wagi $\boldsymbol{w}$ i obciążenie b podlegają zwykle procesowi uczenia
- przeciwieństwo obciążenia $d = -b$ interpretowane jest jako *próg stymulacji*, zatem neuron oblicza funkcję aktywacji ponadprogowej kombinacji liniowej wejść i wag
- *obciążenie* włączane do wektorowych równań neuronu jako waga dodatkowego stałego (np. jednostkowego) wejścia

$$g(\boldsymbol{u}) = g\Big(\sum_{i=0}^{u} \boldsymbol{w}_i \boldsymbol{u}_i\Big)$$

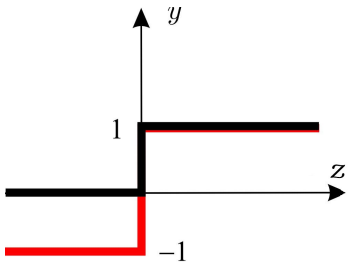
gdzie $u_0 = 1$, $w_0 = b$

- *hiperparametry*: elementy strukturalne dopasowywane po zakończeniu uczenia wag, lub w ogóle nie modyfikowane
  - funkcja aktywacji g
  - pole recepcyjne $\mathcal{R}$

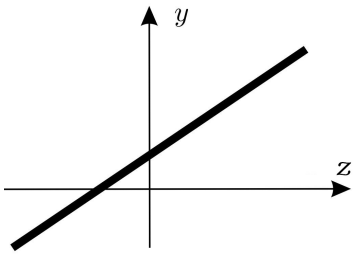
Typowe neurony

Funkcje aktywacji

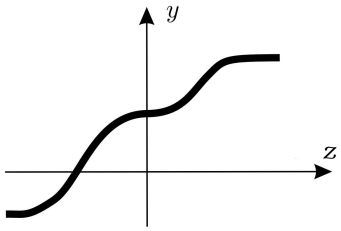
- w perceptronie Rosenblatta a obecnie w SVM stosowane są *twarde nieliniowości*: (funkcja znaku lub funkcja skoku)
- w sieciach Adaline stosowano aktywacje liniowe
- w aproksymatorach perceptronowych stosuje się zwykle *sigmoidy*
- w klasyfikatorach perceptronowych stosuje się zwykle *sigmoidy* i funkcje softmax
- w sieciach splotowych stosuje się zwykle funkcje *prostujące ReLU* (ang. *rectified linear units*) (Nair and Hinton 2010) i sigmoidy oraz *softmax*



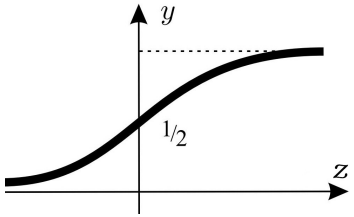
funkcja skoku, funkcja znaku



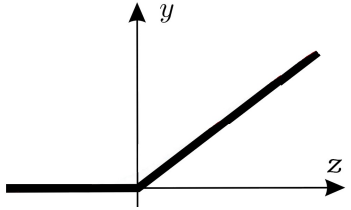
funkcja liniowa



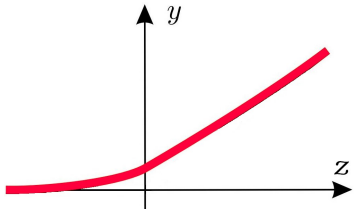
funkcja sigmoidalna
$g(-\infty) = a < g(\infty) = b$, niemalejąca



(standardowy) sigmoid
$g(z) = \frac{1}{1+\exp(-\sigma z)}$



"funkcja prostująca" ReLU
$g(z) = \begin{cases} 0 & z < 0 \\ z & z \geqslant 0 \end{cases}$

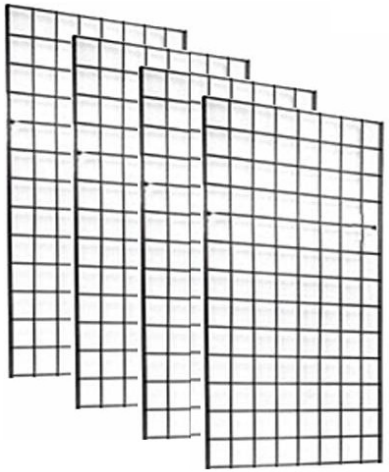


softplus (*soft ReLU*)
$\ln(1 + \mathrm{e}^z)$

3.1 Sygnały w sztucznych sieciach neuronowych
3.2 Sztuczny neuron
3.3 Warstwy
3.4 Konstrukcja sztucznych sieci neuronowych
3.5 Uczenie sieci
3.6 Sieci statyczne a sieci dynamiczne
3.7 Tablice wielowymiarowe i tensory
3.8 ® Splot 2D

Warstwy

- *warstwa*: tablica (1D, 3D, ...) neuronów φ wykorzystujących te same wejścia (z poprzedniej warstwy lub warstwy wejściowej)
  - każdy neuron może mieć inne pole recepcyjne w ramach tablicy wejść
  - sygnał wyjściowy warstwy $\varphi(\boldsymbol{u})$ jest tablicą o takim samym rozmiarze jak tablica neuronów (gdyż każdy neuron ma jedno wyjście)



- na ogół zakłada się, że nie ma połączeń pomiędzy neuronami warstwy (choć neurony te **są zależne** ze względu na proces uczenia)
- podstawowe typy warstw używanych w sieciach perceptronowych i głębokich sieci splotowych
  - warstwy perceptronowe: liniowe i nieliniowe warstwy 1D
  - warstwy liniowe 3D o połączeniach lokalnych
  - warstwy liniowe 3D o połączeniach lokalnych i współdzielonych wagach: warstwy splotowe
  - warstwy nieliniowe
  - warstwy agregujące (ang. *pooling layers*)
  - warstwy normalizujące
  - warstwa softmax

Typy warstw

Warstwy liniowe 1D

- pola recepcyjne $\mathcal{R}_i$ są zwykle identyczne: *połączenia kompletne* ze wszystkimi sygnałami poprzedniej warstwy
- *warstwa liniowa* o połączeniach kompletnych realizuje *przekształcenie afiniczne* ("liniowe")

$$y_i = \varphi(\boldsymbol{u}) = b_i + \sum_j w_{ij} u_j$$

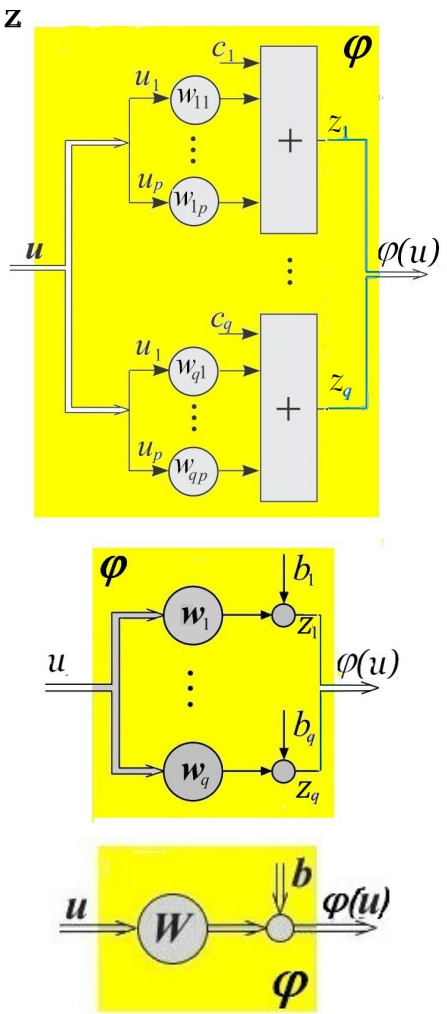
- notacja wektorowo-macierzowa

$$\boldsymbol{y} = \varphi(\boldsymbol{u}) = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{u}$$

gdzie

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_p \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_q \end{bmatrix}$$

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^\top \\ \vdots \\ \boldsymbol{w}_q^\top \end{bmatrix} = \underbrace{\begin{bmatrix} w_{1,1} & \cdots & w_{1,p} \\ \vdots & \vdots & \vdots \\ w_{q,1} & \cdots & w_{q,p} \end{bmatrix}}_{\text{wejścia}} \Bigg\} \text{neurony}$$



wizualizacje liniowej warstwy perceprtonowej

Typy warstw

Warstwy liniowe 3D

- liniowe warstwy 3D złożone sa z wielu (D) *kanałów* (lub: *pól aktywacji*)
  - każdy kanał (d) warstwy ma jednakowy rozmiar (maksymalnie $n_X \times n_Y$, w zależności od uwzględniania warunków brzegowych) i otrzymuje sygnały ze wszystkich D' kanałów poprzedniej warstwy (nie ma połączeń z kanałami bieżącej warstwy)
  - każdy neuron kanału d wykorzystuje tylko sygnały ze swojego pola recepcyjnego o rozmiarze $\leqslant \delta x \times \delta y$ i oblicza ich funkcję afiniczną ("liniową")

⊕ wyjście każdego kanału (o indeksie d) można zapisać w postaci sumy działań na **wszystkich** kanałach poprzedniej warstwy

$$y_{xyd} = b_{xyd} + \sum_{d'} \sum_{(x'y') \in \mathcal{R}_{xyd}} w_{xyd,x'y'd'} u_{x'y'd'}$$

- xyd są indeksami neuronu xy w kanale d
- wejście warstwy jest tablicą $\mathbf{U}' = u_{x'y'd'}$ przedstawianą jako d' kanałów $u_{x'y'}$; podobnie wyjście warstwy jest tablicą $\mathbf{y} = y_{xyd}$
- wagi neuronów tworzą tablicę $\mathbf{w} = w_{xyd,x'y'd'}$, obciążenia tablicę $\mathbf{b} = b_{xyd}$ (tensory rzędu 6 i 3)

Typy warstw

Liniowe warstwy splotowe 3D, współdzielenie wag

- *współdzielenie wag* (ang. *weight sharing*) w głębokich sieciach splotowych 3D: założenie, że wagi, obciążenie i kształt pola recepcyjnego są identyczne dla każdego neuronu w ramach **tego samego kanału**
- w rezultacie, znacznie ($n_X n_Y$ krotnie zmniejsza się liczba wag i obciążeń warstwy

⊕ warstwa ze współdzieleniem wag

$$y_{xyd} = b_d + \sum_{d'} \sum_{(x',y') \in \mathcal{R}_{xy}} w_{x'y'd'd} u_{x',y',d'}$$

zatem $\mathbf{W}$ jest tensorem rzędu 4, $\mathbf{b}$ tensorem rzędu 1

⊕ indeksy pola recepcyjnego dla kanału wygodnie indeksować względem centrum (x, y) tego pola; wyjście warstwy zależy od **splotu** wag i wejść (stąd nazwa warstwy)

$$y_{xyd} = b_d + \sum_{d'} \sum_{(\Delta x',\Delta y') \in \mathcal{R}} w_{\Delta x',\Delta y',d',d} u_{x+\Delta x',y+\Delta y',d'}$$

Typy warstw

Warstwy nieliniowe

- typowa warstwa nieliniowa: złożenie warstwy liniowej i tablicy funkcji aktywacji; w wyniku: tablica o rozmiarze warstwy liniowej
- *(nieliniowa) warstwa perceptronowa* realizuje funkcje aktywacji wyjść warstwy liniowej

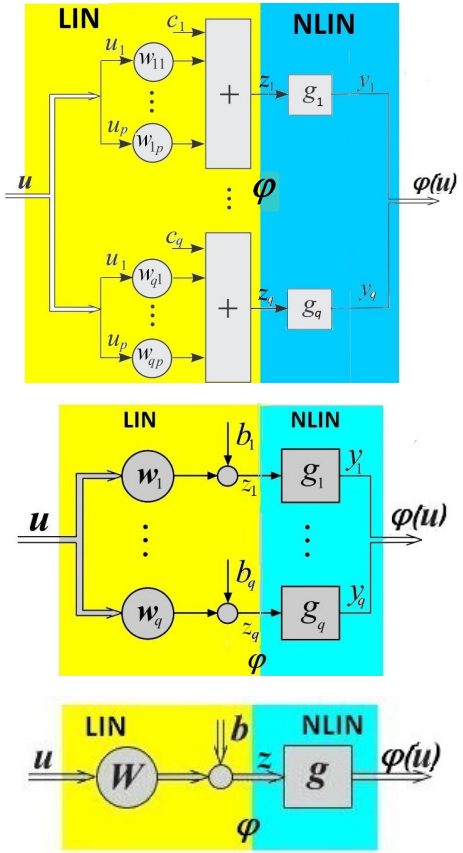
$$y_i = \mathbf{g}\Big(\boldsymbol{b}_i + \sum_j \boldsymbol{W}_{ij}\boldsymbol{u}_j\Big)$$

tzn. w notacji wektorowej dla połączeń kompletnych

$$\boldsymbol{y} = \mathbf{g}(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{u})$$

- *(nieliniowa) warstwa splotowa 3D*: funkcja aktywacji wyjść splotowej warstwy liniowej

$$y_{xyd} = \mathbf{g}\Big(b_d + \sum_{d'} \sum_{(\Delta x', \Delta y') \in \mathcal{R}} w_{\Delta x', \Delta y', d', d} u_{x+\Delta x', y+\Delta y', d'}\Big)$$

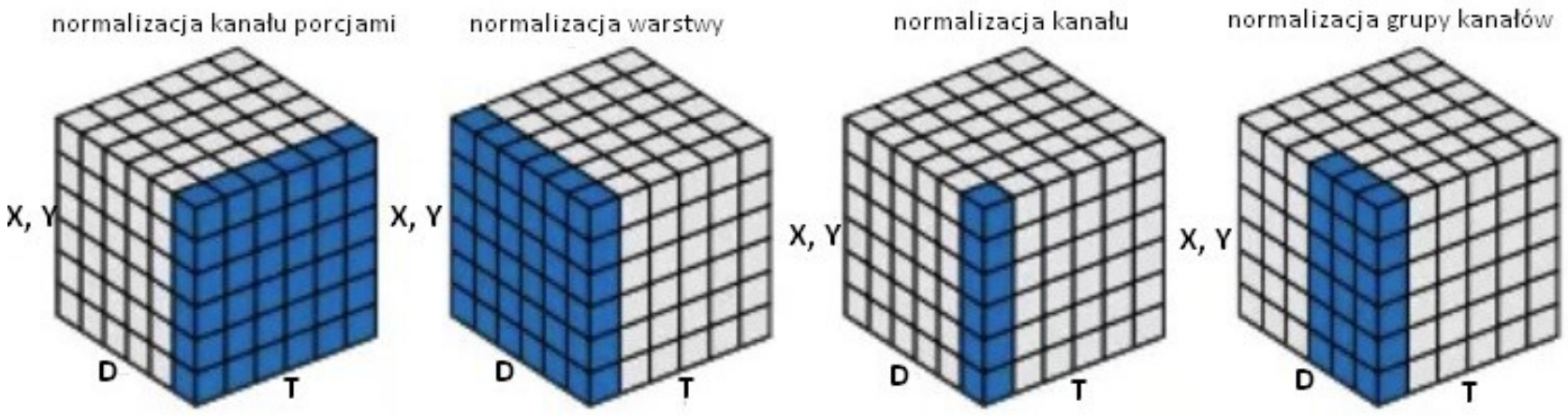


wizualizacje nieliniowej warstwy perceptronowej

Typy warstw

Warstwy skalujące

- warstwa skalująca normalizuje sygnały w ramach różnych obszarów i zbiorów danych
- główne podejścia
  - normalizacja *kanału porcjami danych* T (ang. *batch normalization*): normalizacja $T \times X \times Y$ dla zadanego kanału d
  - normalizacja *warstwy*: normalizacja $X \times Y \times D$ dla zadanej chwili t
  - normalizacja *kanału*: normalizacja $X \times Y$ dla każdej chwili t i każdego kanału d



Yuxin Wu Kaiming He, Group Normalization, 2018, `https://arxiv.org/pdf/1803.08494.pdf`

Typy warstw

Warstwy skalujące - Normalizacja statystyczna

- oblicza średnią i wariancję empiryczną porcji danych $\{\boldsymbol{x}(1), \ldots, \boldsymbol{x}(N)\}$ (wektorów, obrazów), tzn.

$$\overline{\boldsymbol{x}} = \frac{1}{N} \sum_{t=1}^{N} \boldsymbol{x}(t), \qquad s^2 = \frac{1}{N} \sum_{t=1}^{N} \|\boldsymbol{x}(t) - \overline{\boldsymbol{x}}\|^2$$

lub (jeden przebieg sumowania)

$$\begin{bmatrix} \overline{\boldsymbol{x}} \\ \hline \|\boldsymbol{x}\|^2 \end{bmatrix} = \frac{1}{N} \sum_{t=1}^{N} \begin{bmatrix} \boldsymbol{x}(t) \\ \|\boldsymbol{x}(t)\|^2 \end{bmatrix}, \quad s^2 = \overline{\|\boldsymbol{x}\|^2} - \|\overline{\boldsymbol{x}}\|^2$$

- normalizacja danych

$$\boldsymbol{x}(t)^o = \frac{1}{s + \epsilon} (\boldsymbol{x}(t) - \overline{\boldsymbol{x}})$$

- wielkości $\overline{\boldsymbol{x}}$, s można dodatkowo adjustować podczas uczenia
- warstwa normalizująca wstawiana jest zwykle między warstwę liniową a nieliniowość
- zastosowanie normalizacji kanałów porcjami w sieci BN-Inception (Google, Inception ver. 2, 2015) o rekordowo niskim błędzie klasyfikacji 4.8% (człowiek – 5.1%) (Ioffe and Szegedy 2015)

Typy warstw

Warstwy agregujące (podpróbkujące)

- *warstwa podpróbkująca* lub (ang. *subsampling layer*) lub *agregująca* (ang. *pooling layer*) 3D: każdy kanał tworzony poprzez *agregację* w *obszarze agregacji* A odpowiadającego mu kanału poprzedniej warstwy
- agregacja może polegać np. na obliczeniu
  - wartości maksymalnej (ang. *max-pooling*) w ramach obszaru agregacji kanału

$$y_i = \max_{u \in A_i} u$$

  - średniej ważonej (ang. *average pooling*)

$$y_i = \overline{\boldsymbol{u}}_i = \frac{1}{\|A_i\|} \sum_{u \in A_i} u$$

- brak parametrów decyzyjnych; hiperparametry definiują obszar agregacji: najczęściej 2×2 lub 3×3
- efektem działania warstwy agregującej jest
  - podpróbkowanie obrazów (zmniejszenie częstotliwości próbkowania w ramach kanału)
  - zmniejszenie wrażliwości wyjść kanału na niewielkie przesunięcia obrazu
  - uśrednianie szumów

Typy warstw

Warstwa softmax

- *softmax*: funkcja $\boldsymbol{s} : \mathbb{R}^m \mapsto [0,1]^m$ o składowych

$$s_i(u_1,..,u_m) \stackrel{\text{def}}{=} s_i(\boldsymbol{u}) \stackrel{\text{def}}{=} \frac{\exp u_i}{\sum_{k=1}^m \exp u_k}$$

- funkcja softmax przekształca dowolne wejścia $u_1,..,u_m$ w wyjścia $y_1,..,y_m$, które
  - **zachowują porządek**, tzn

$$u_i < u_j \quad \Rightarrow \quad y_i < y_j$$

  - mają własności m-punktowego rozkładu prawdopodobieństwa

$$\sum_{i=1}^m y_i = 1 \qquad \text{i} \qquad 0 \leqslant y_i \leqslant 1, \quad i = 1,..,m$$

- warstwa wyjściowa softmax w klasyfikatorach, pozwala na traktowanie wyjść jako rozkład prawdopodobieństwa

⊕ Typy warstw

Softmax - związek z maksimum

- argumenty funkcji softmax skaluje się wprowadzając *temperaturę* $T > 0$

$$\boldsymbol{s}^T(\boldsymbol{u}) \stackrel{\text{def}}{=} \boldsymbol{s}(\boldsymbol{u}/T)$$

- $T \searrow$ zaostrza różnice „prawdopodobieństw" - stosowane dla wiarygodnych wyników
- dla $T \approx 0$: softmax skupia się wokół argumentów ekstremalnych (aproksymuje funkcję $\arg\max$)

$$s^T(\boldsymbol{u}) \xrightarrow[T\to 0]{} \begin{cases} 1/M & \text{jeśli } s_i(\boldsymbol{u}) = \max_j s_j(\boldsymbol{u}) \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

  gdzie M to krotność ekstremum globalnego $\max_j s_j(\boldsymbol{u})$ (warstwa *WTA* (ang. *winner takes all*))
- $T \nearrow$ wygładza „rozkład"; stosowane dla mało wiarygodnych wyników
- dla $T \to \infty$: rozkład równomierny

Ⓔ przykłady

$$\boldsymbol{s}^T(1,1,1) = (1/3, 1/3, 1/3)$$

$$\boldsymbol{s}^T(1,-3,1) \xrightarrow[T\to 0]{} (0.5, 0, 0.5)$$

$$\boldsymbol{s}^T(-3,1,-3) \xrightarrow[T\to 0]{} (0, 1, 0)$$

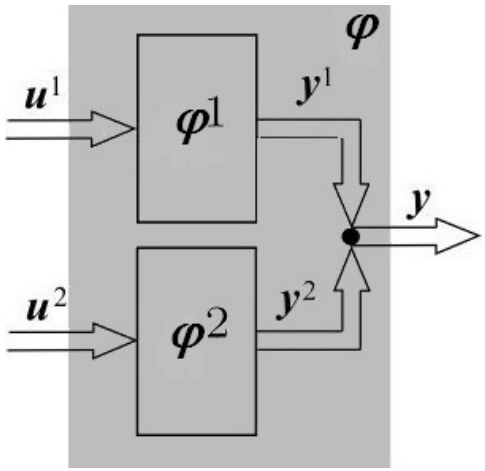
3.1 Sygnały w sztucznych sieciach neuronowych
3.2 Sztuczny neuron
3.3 Warstwy
3.4 Konstrukcja sztucznych sieci neuronowych
3.5 Uczenie sieci
3.6 Sieci statyczne a sieci dynamiczne
3.7 Tablice wielowymiarowe i tensory
3.8 ® Splot 2D

Sztuczne sieci neuronowe

Budowa sieci z warstw

połączenie równoległe warstw tworzy się poprzez scalenie (konkatenację) ich wyjść

$$\boldsymbol{y} = (\boldsymbol{y}^1, \boldsymbol{y}^2) \text{ tzn. } \varphi = (\varphi^1, \varphi^2)$$

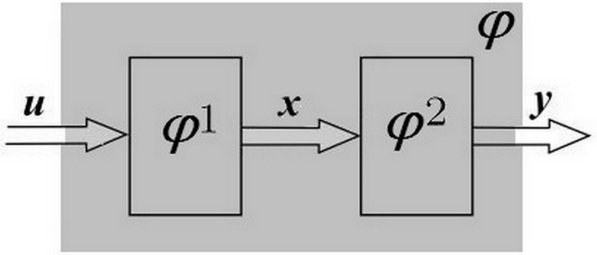


- wynik równoważny warstwie o większe liczbie wejść

połączenie szeregowe warstw jest złożeniem przekształceń w kolejnych warstwach

$$\varphi(\boldsymbol{u}) = \varphi^2\left(\varphi^1(\boldsymbol{u})\right)$$

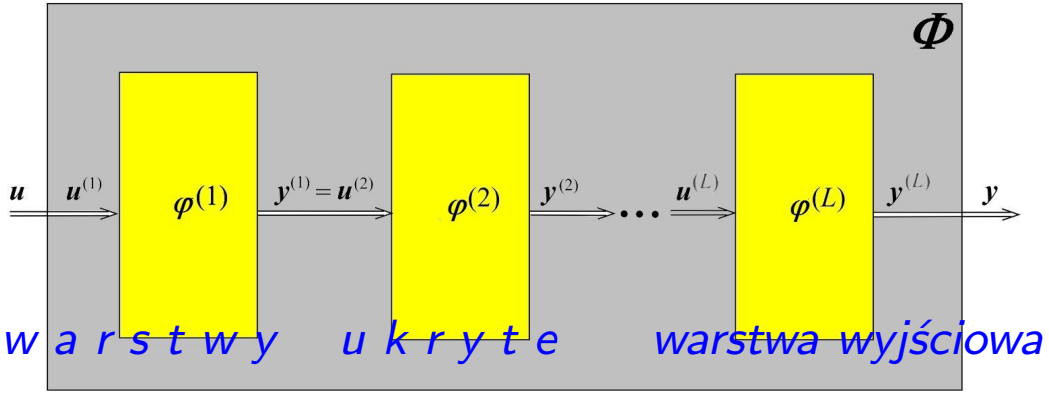
$$\text{tzn.} \quad \varphi = \varphi^2 \circ \varphi^1$$



- w przypadku nieliniowym jest strukturą bardziej złożoną niż pojedyncza warstwa

Sztuczne sieci neuronowe

Sieć wielowarstwowa



- *sieć wielowarstwowa* ϕ jest połączeniem szeregowym warstw

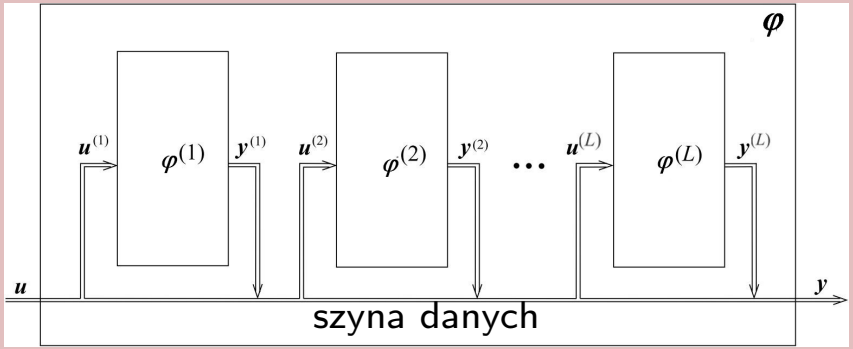
$$\boldsymbol{u}^{(1)} = \boldsymbol{u}, \quad \boldsymbol{u}^{(2)} = \boldsymbol{y}^{(1)}, \quad \boldsymbol{u}^{(L)} = \boldsymbol{y}^{(L-1)}, \quad \boldsymbol{u}^{(L)} = \boldsymbol{y}$$

- tzn. przekształcenie sygnału jest *złożeniem przekształceń w warstwach*

$$\phi(\boldsymbol{u}) = \boldsymbol{\varphi}^{(L)}\left(\boldsymbol{\varphi}^{L-1} \ldots \left(\boldsymbol{\varphi}^{(1)}(\boldsymbol{u})\right)\right.$$
$$\phi = \boldsymbol{\varphi}^{(L)} \circ \ldots \circ \boldsymbol{\varphi}^{(1)}$$

⊛ Sztuczne sieci neuronowe

Sieci warstwowe bez sprzężeń



- w *sieci warstwowej bez sprzężeń* ϕ każda warstwa otrzymuje sygnały wyjściowe wszystkich poprzednich warstw i sygnał wejściowy sieci

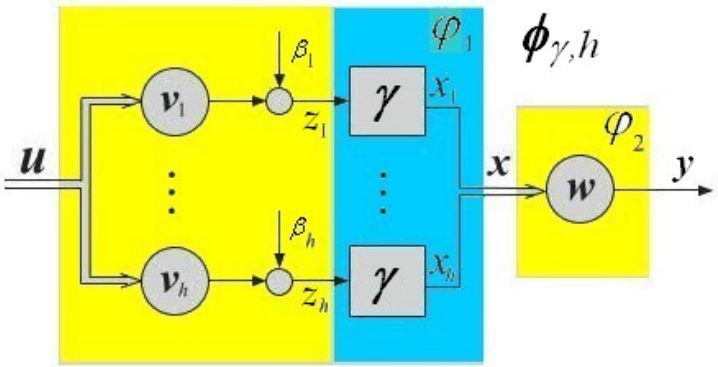
$$\boldsymbol{u}^{(1)} = \boldsymbol{u}, \quad \boldsymbol{u}^{(2)} = (\boldsymbol{u}, \boldsymbol{y}^{(1)}), \quad \ldots \quad \boldsymbol{u}^{(k)} = (\boldsymbol{u}^{(k-1)}, \boldsymbol{y}^{(k-1)}), \quad k = 2, \ldots, L$$

⊛ przekształcenie sygnału w sieci warstwowej bez sprzężeń ma postać $\boldsymbol{y} = \phi(\boldsymbol{u}) = \left(\boldsymbol{u}, \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(L)}\right)$, gdzie

$$\begin{aligned}
\boldsymbol{y}^{(0)} &= \boldsymbol{u} \\
\boldsymbol{y}^{(1)} &= \boldsymbol{\varphi}^{(1)}(\boldsymbol{y}^{(0)}) \\
&\ldots \\
\boldsymbol{y}^{(k)} &= \boldsymbol{\varphi}^{(k)}(\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \ldots \boldsymbol{y}^{(k-1)}), \quad k = 1, \ldots, L
\end{aligned}$$

⊛ każda sieć bez sprzężeń jest równoważna pewnemu perceptronowi wielowarstwowemu

Podstawowa sieć dwuwarstwowa



podstawowa sieć dwuwarstwowa

- *podstawowa sieć dwuwarstwowa*
  $\phi_{\gamma,h}$: dwuwarstwowy perceptron

$$\phi_{\gamma,h}(\boldsymbol{u}) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\gamma}(\beta + \boldsymbol{V}\boldsymbol{u})$$
$$= \sum_{i=1}^{h} w_i \gamma_i(\beta + \boldsymbol{V}\boldsymbol{u})$$

którego

- warstwa ukryta o h neuronach ma identyczne funkcje aktywacji γ
- warstwa wyjściowa jest liniowa o zerowym obciążeniu

Podstawowa sieć dwuwarstwowa

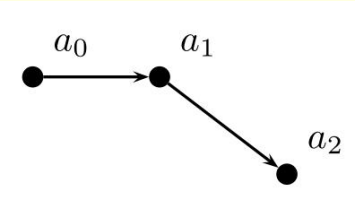
Uniwersalny aproksymator

- zadana klasa funkcji, np.
  - funkcji o wartościach $\mathbb{R}$ określonej na $\mathbb{R}$ (np. aproksymacja funkcji czasu)
  - funkcji o wartościach $\{0, 1\}$ określonej na kostce $\{0, 1\}^k$ (np. aproksymacja funkcji boolowskiej k zmiennych)
  - funkcji o wartościach $\mathbb{R}$ określonej na $\mathbb{R}^2$ (np. aproksymacja obrazów)
  - funkcji o wartościach $1, \ldots, k$ określonej na $X \times Y$ (klasyfikacja obrazu w skali szarości na k klas)
  - funkcji o wartościach $1, \ldots, k$ określonej na $X \times Y \times \{1, 2, 3\}$ (klasyfikacja obrazu kolorowego)
- dla zadanej klasy funkcji i zadanej dokładności, w rodzinie podstawowych sieci dwuwarstwowych o *odpowiednio dużej warstwie ukrytej* istnieje sieć *dowolnie dokładnie* aproksymująca tę funkcję, tzn. rodzina ta jest uniwersalnym aproksymatorem (dowolnie bliska?, zadanej klasy? odpowiednio dużej?...)
- tę samą własność ma dowolny nadzbiór tej rodziny

® Graf skierowany

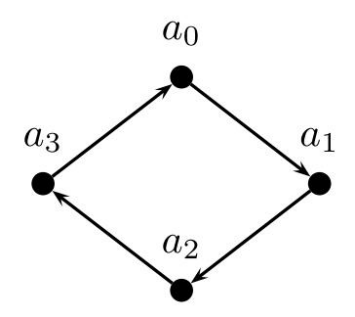
- *graf skierowany* (*digraph*) (W, K)
  W - zbiór *wierzchołków*
  K - zbiór *krawędzi skierowanych*, podzbiór $W \times W$
- *ścieżka*: podzbiór wierzchołków $\{a_0, \ldots, a_m\}$ such that (a_{i-1}, a_i), $i = 1, \ldots, m$ są krawędziami

  (a_0= początek ścieżki, a_m - koniec ścieżki)



- *cykl*: ścieżka zaczynająca się i kończąca w tym samym wierzchołku,

  tzn. podzbiór wierzchołków $\{a_0, \ldots, a_m\}$ taki, że (a_{i-1}, a_i), $i = 1, \ldots, m$, i (a_m, a_0) są wierzchołkami



Ⓡ Acykliczny graf skierowany

- *acykliczny graf skierowany*: graf bez cykli
- wierzchołki można uporządkować tak, że jeśli $x_i \prec x_j$ to $i < j$ (*porządek topologiczny*)
- dla grafów skierowanych acyklicznych, jeśli istnieje **krawędź** z X do Y, to

  X jest *rodzicem* Y

  $\mathcal{P}Y =$ zbiór rodziców Y

  Y jest *dzieckiem* X;

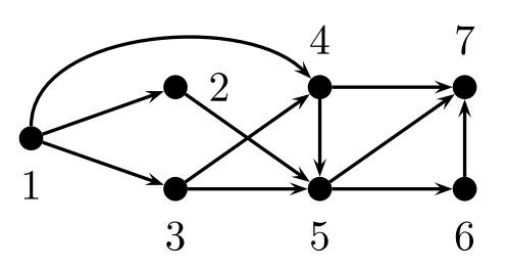
  $\mathcal{C}X =$ zbiór dzieci X
- jeśli istnieje **ścieżka** from X do Y to

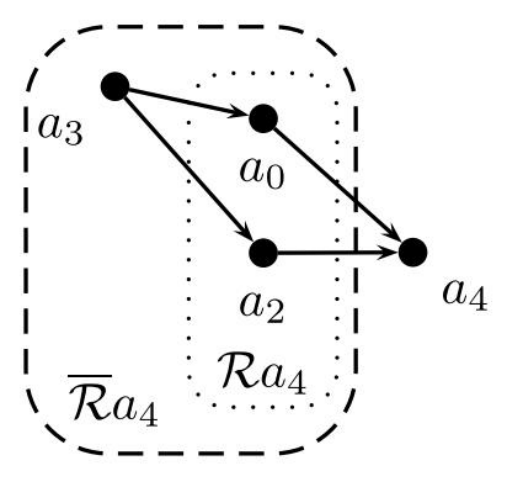
  X *poprzedza* Y ($X \prec Y$),
  X jest *przodkiem* Y

  $\mathcal{A}X =$ zbiór przodków X

  Y jest *potomkiem* X

  $\mathcal{D}X =$ zbiór potomków X



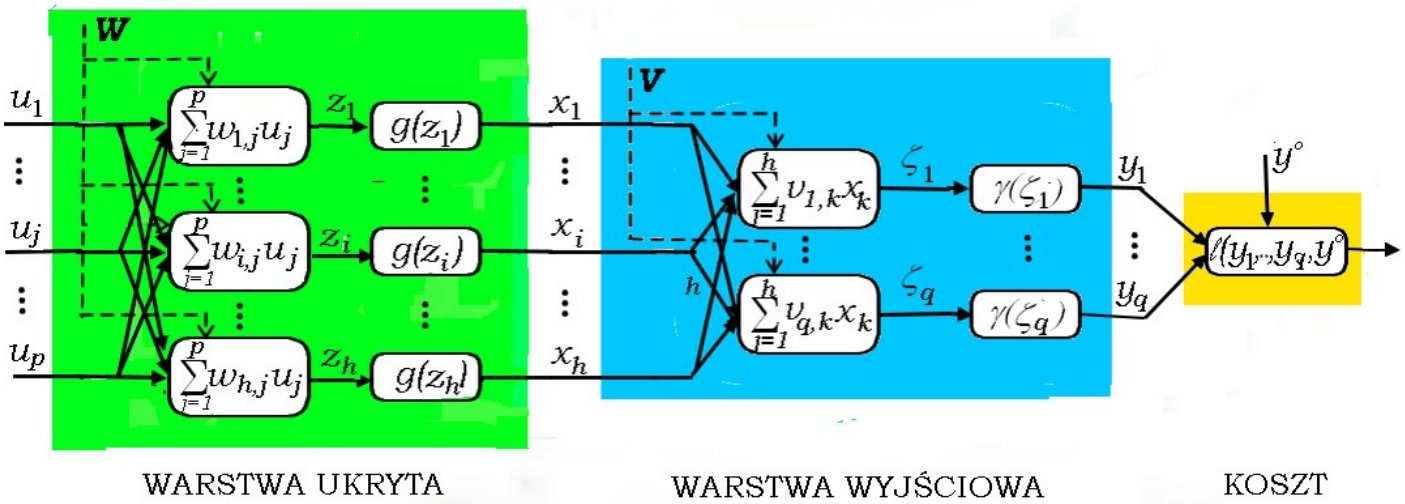


Sztuczne sieci neuronowe jako grafy

Graf sieci neuronowej

Dowolną sieć bez sprzężeń można przedstawić w postaci *acyklicznego grafu skierowanego* (W, K), którego

- wierzchołki W reprezentują funkcje
- krawędzie K reprezentują wartości argumentów i wartości funkcji
  - krawędzie wychodzące $\leftrightarrow$ wartości funkcji
  - krawędzie przychodzące $\leftrightarrow$ argumenty funkcji
  - wierzchołki bez rodziców $\leftrightarrow$ wejścia sieci u_i
  - wierzchołki bez dzieci $\leftrightarrow$ wyjścia sieci y_j lub tzw. koszt sieci



Przykłady sztucznych sieci neuronowych

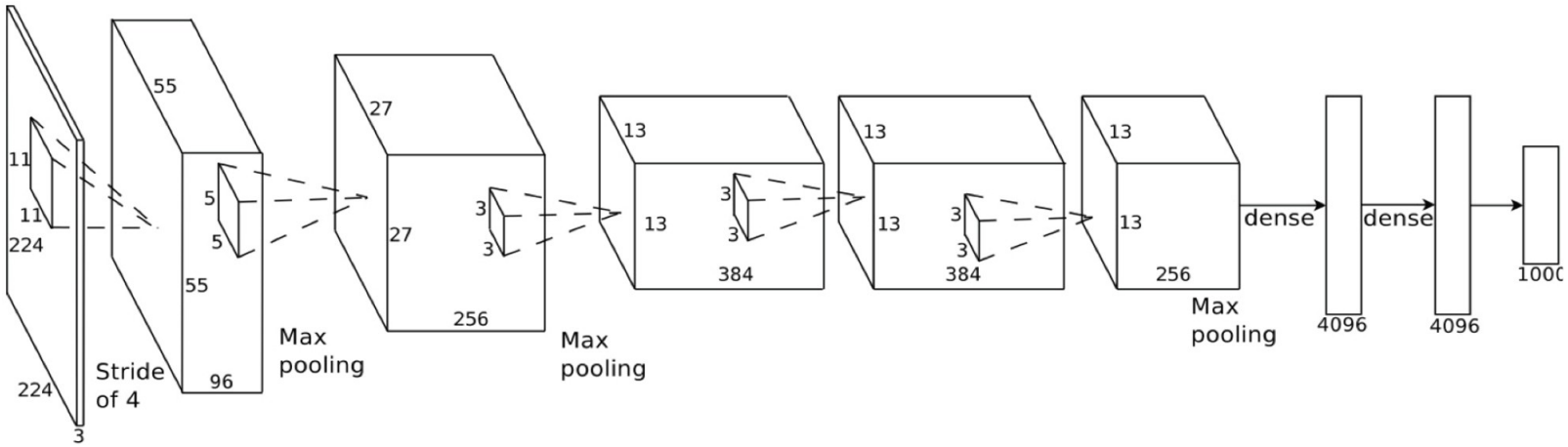
Sieci głębokie

- *sieć głęboka*: sieć wielowarstwowa o więcej (zwykle znacznie więcej) niż dwu warstwach
  - sieci splotowe
  - głębokie autoenkodery
  - głębokie sieci przekonań
  - generacyjne sieci antagonistyczne (GAN)
  - autoenkodery wariacyjne
  - ...

Przykłady sztucznych sieci neuronowych

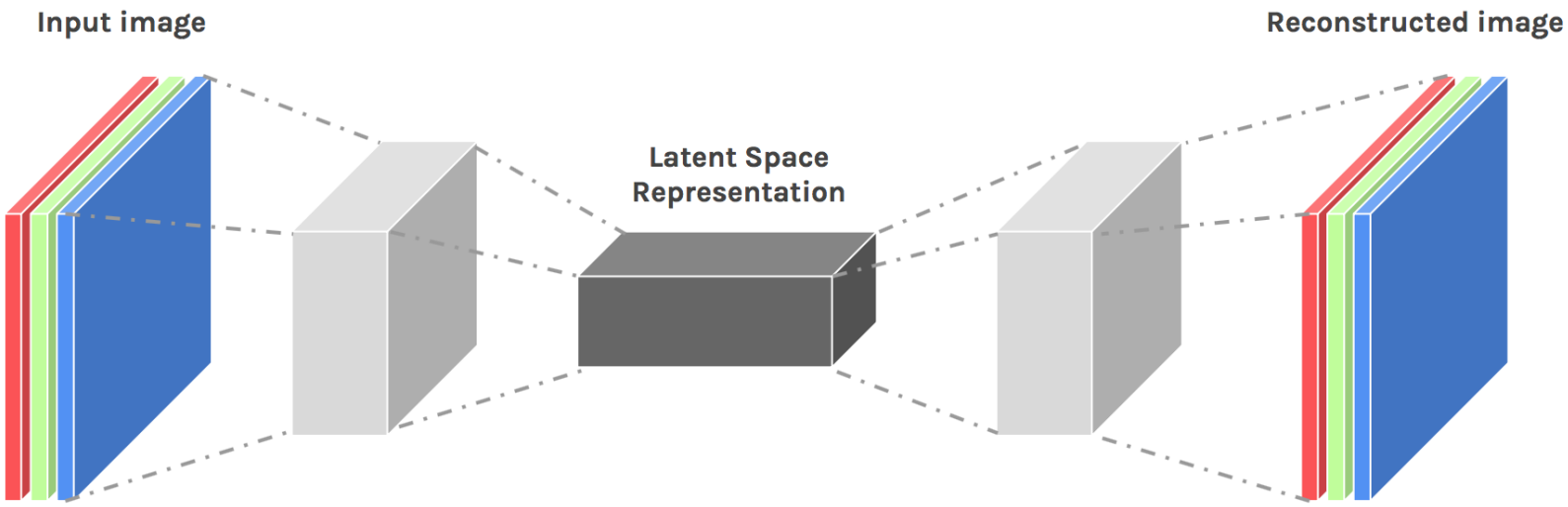
Głębokie sieci splotowe

- *sieć splotowa*: sieć wielowarstwowa (zwykle głęboka) o połączeniach lokalnych lub kompletnych, zwykle zgodnie z następującym schematem
  - wejście - tablica 2D lub 3D
  - *warstwa splotowa* 3D złożona z wielu *map aktywacji* 2D;
    - *głębokość warstwy* = liczba map aktywacji
    - *mapa aktywacji*: wynik nieliniowej filtracji poprzedniej warstwy
      każdy jej piksel jest *funkcją aktywacji* kombinacji liniowej jego *pola recepcyjnego* we wszystkich mapach poprzedniej warstwy i *wag*
  - *warstwy agregujące* (ang. *pooling*) (podpróbkowanie) - opcjonalne (3D)
  - *warstwy normalizujące* - opcjonalne (3D)
  - *warstwy perceptronowe* - połączenia kompletne (gęste) (1D)
  - warstwa wyjściowa - dla klasyfikacji: warstwa softmax (1D)



Przykłady sztucznych sieci neuronowych

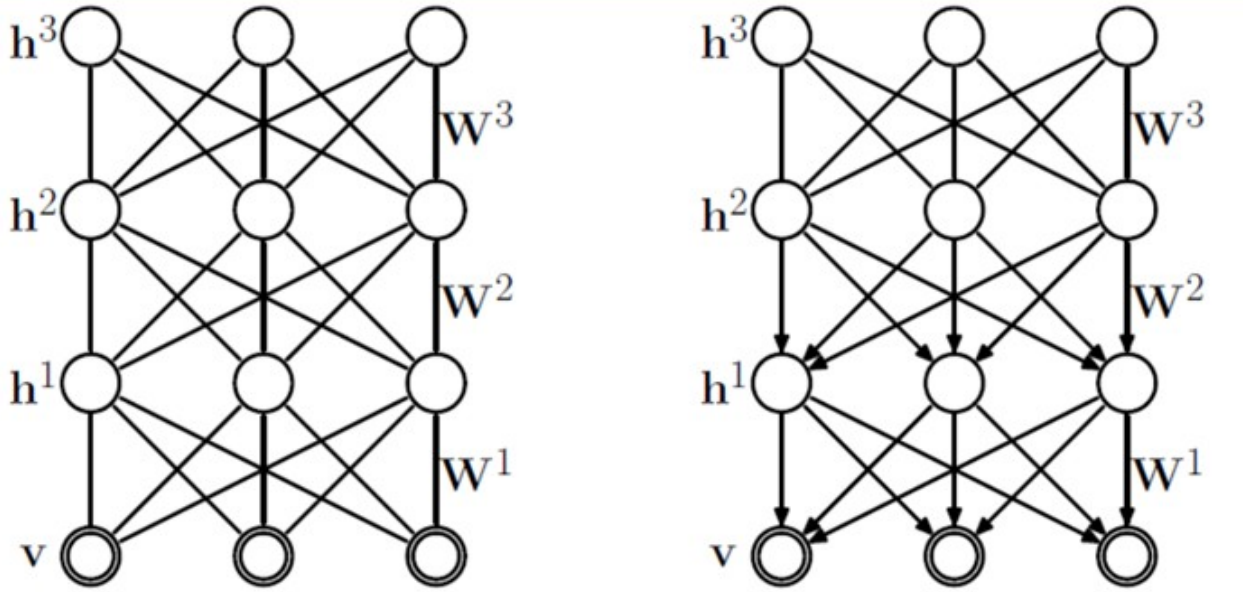
Głębokie autoenkodery



Julien Despois, 2017. Autoencoders - Deep Learning bits.
https://hackernoon.com/autoencoders-deep-learning-bits-1-11731e200694

Przykłady sztucznych sieci neuronowych

Głębokie sieci Boltzmanna i głębokie sieci przekonań



Deep Boltzmann Machine Deep Belief Network

3.1 Sygnały w sztucznych sieciach neuronowych
3.2 Sztuczny neuron
3.3 Warstwy
3.4 Konstrukcja sztucznych sieci neuronowych
3.5 Uczenie sieci
3.6 Sieci statyczne a sieci dynamiczne
3.7 Tablice wielowymiarowe i tensory
3.8 ® Splot 2D

Uczenie sieci

- *uczenie*, *trening*, *adaptacja*: proces zmian wag (i obciążeń) sieci na podstawie obserwacji środowiska, zmierzający do tego, by sieć neuronowa *działała w pożądany sposób*, np. realizowała pożądaną funkcję wejść
- *środowisko* poznawane jest poprzez obserwację sygnałów oddziałujących na sieć i pewnych sygnałów dodatkowych, np. *pożądanego wyjścia* sieci

(E) przykłady wyjść pożądanych

- dla klasyfikacji: wektor $\boldsymbol{y}^o$ rzeczywistej przynależności obrazu wejściowego do poszczególnych klas $\mathcal{C}_k$: $$y_k^o = \begin{cases} 1 & \text{jeśli } \boldsymbol{u} \in \mathcal{C}_k \\ 0 & \text{poza tym} \end{cases}$$
- dla aproksymacji funkcji f: wartość funkcji aproksymowanej $\boldsymbol{y}^o = f(\boldsymbol{u})$

- *funkcja strat* ocenia koszt niezgodności pomiędzy rzeczywistym a pożądanym działaniem sieci
- *algorytm uczenia* wag i obciążeń sieci zwykle minimalizuje funkcję strat względem tych parametrów

Uczenie sieci

Typowy mechanizm uczenia

- *para ucząca*: (wartość wejścia sieci, pożądana wartość wyjścia sieci)
- *zbiór uczący*: zbiór *par uczących* $\{(\boldsymbol{u}_n, \boldsymbol{y}_n^o), i = 1, \ldots, N\}$
- w kolejnych krokach *treningu* $t = 1, 2, \ldots, T$, ze zbioru trenującego losowane są *przykłady trenujące* $\big(\boldsymbol{u}(t), \boldsymbol{y}^o(t)\big)$ tworząc *ciąg uczący*
- zwykle elementy zbioru trenującego będziemy indeksować (indeksy od 1 do N), a elementom ciągu trenującego przypiszemy argumenty t (od 1 do T)
- *uczenie*: rekurencyjna *korekcja* parametrów (wag, obciążeń) $\boldsymbol{\vartheta}$, np.

$$\boldsymbol{\vartheta}(t+1) = \underbrace{\boldsymbol{A}}_{\text{algorytm uczenia}}\Big(\boldsymbol{\vartheta}(t),\ \underbrace{\boldsymbol{y}(t)}_{\text{wyjście rzeczywiste sieci}},\ \underbrace{\boldsymbol{y}^o(t)}_{\text{wyjście pożądane sieci}}\Big)$$

$$\boldsymbol{y}(t) = \boldsymbol{\phi}\big(\boldsymbol{u}(t)\big) - \text{ wyjście sieci}$$

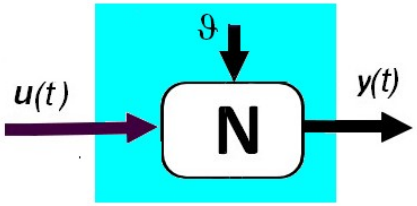
- np. dla sieci perceptronowej

$$\boldsymbol{\vartheta}(t+1) = \boldsymbol{\vartheta}(t) - \gamma\, \underbrace{\nabla_{\boldsymbol{\vartheta}}\, \ell}_{\text{gradient funkcji strat } \ell}\Big(\underbrace{\boldsymbol{y}^o(t) - \boldsymbol{y}(t)}_{\text{odchyłka wyjścia}}\Big)$$

3.1 Sygnały w sztucznych sieciach neuronowych
3.2 Sztuczny neuron
3.3 Warstwy
3.4 Konstrukcja sztucznych sieci neuronowych
3.5 Uczenie sieci
3.6 Sieci statyczne a sieci dynamiczne
3.7 Tablice wielowymiarowe i tensory
3.8 ® Splot 2D

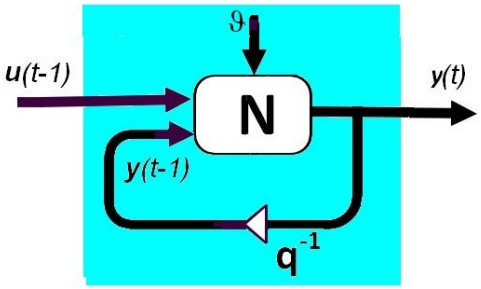
Sieci statyczne a sieci dynamiczne

- *sieci statyczne* $\boldsymbol{y}(t) = \boldsymbol{\phi}\big(\boldsymbol{u}(t); \boldsymbol{\vartheta}\big)$
  $\boldsymbol{\vartheta}$ - wektor parametrów sieci (wagi i obciążenia dla wszystkich warstw)



- *sieci dynamiczne*

$$\boldsymbol{y}(t) = \boldsymbol{\phi}\big(\boldsymbol{u}(t-1), \ldots, \boldsymbol{u}(0); \boldsymbol{\vartheta}\big)$$



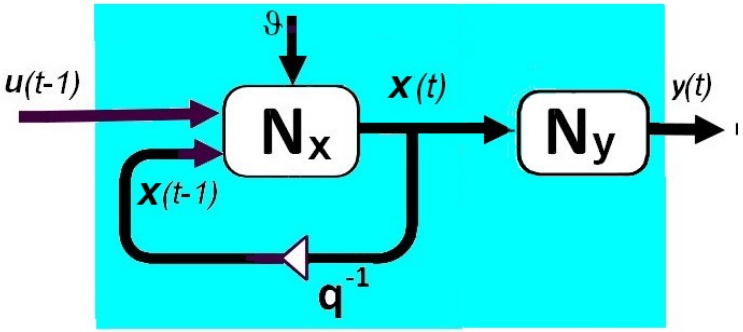
  - sieć rekurencyjna

$$\boldsymbol{y}(t) = \boldsymbol{\phi}\big(\boldsymbol{y}(t-1), \boldsymbol{u}(t-1); \boldsymbol{\vartheta}\big)$$

  - ogólniej: sieć stanu i sieć wyjścia

$$\boldsymbol{x}(t) = \boldsymbol{\phi}_{\boldsymbol{x}}\big(\boldsymbol{x}(t-1), \boldsymbol{u}(t-1); \boldsymbol{\vartheta}_{\boldsymbol{x}}\big)$$

$$\boldsymbol{y}(t) = \boldsymbol{\phi}_{\boldsymbol{y}}\big(\boldsymbol{x}(t-1); \boldsymbol{\vartheta}_{\boldsymbol{y}}\big)$$



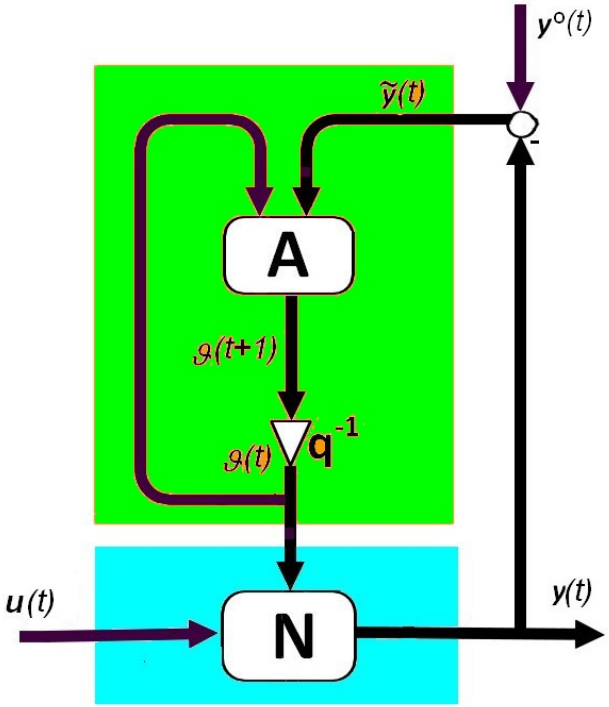
Sieci statyczne z uczeniem

- sieć statyczna z uczeniem

$$\boldsymbol{y}(t) = \boldsymbol{\phi}\Big(\boldsymbol{u}(t);\ \boldsymbol{\vartheta}(t)\Big)$$

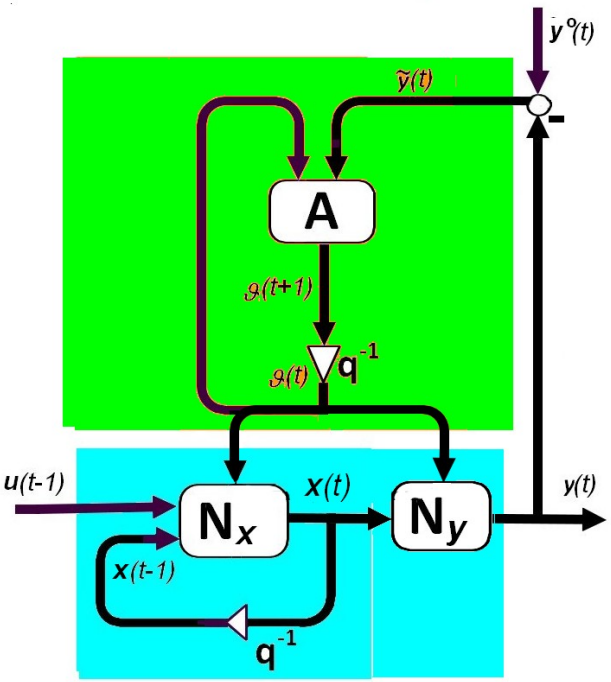
$$\boldsymbol{\vartheta}(t+1) = \boldsymbol{A}\Big(\boldsymbol{\vartheta}(t), \boldsymbol{y}^o(t), \boldsymbol{y}(t)\Big)$$

staje się układem dynamicznym



Sieci dynamiczne z uczeniem

- sieć dynamiczna z uczeniem
- neuron dynamiczny
  - estymuje właściwe **chwilowe wartości** wyjścia sieci (jak najbliższe wartościom pożądanym)
  - analizuje **mechanizm zmian** relacji wejście-pożądane wyjście
- proces uczenia musi być "oddzielony" od procesu estymacji bieżących wartości
  - *szybka dynamika*: mechanizm określania bieżącego wyjścia sieci na podstawie sieciowego modelu bieżącej relacji wejście - wyjście pożądane (niezwiązany z uczeniem)
  - *wolna dynamika*: mechanizm (uczenia), tzn. modyfikacji modelu mechanizmu wejście - wyjście pożądane



3.1 Sygnały w sztucznych sieciach neuronowych
3.2 Sztuczny neuron
3.3 Warstwy
3.4 Konstrukcja sztucznych sieci neuronowych
3.5 Uczenie sieci
3.6 Sieci statyczne a sieci dynamiczne
3.7 Tablice wielowymiarowe i tensory
3.8 ® Splot 2D

⊕ Tensory

Tensory w fizyce

Materiały pomocnicze: Brannon (2004), Clarke (2011), Dullemond and Peeters (2010), Feng (2017), Hackl and Goodarzi (2010), Kolecki (2002), Ostrowska-Maciejewska (2007), Porat (2014), and Sochi (2016)

- prawa fizyki powinny być wyrażane w sposób niezależny od transformacji współrzędnych, tzn.powinny być **prawdziwe w dowolnym dopuszczalnym układzie współrzędnych**
  - współrzędne ortogonalne prostokątne (Kartezjańskie) (x, y, z)
  - współrzędne ortogonalne cylindryczne $(x, r, y$
  - współrzędne ortogonalne sferyczne $(1, \rho, \rho \sin \phi)$
  - współrzędne krzywoliniowe
  - geometria nieeuklidesowa
- skalar: wielkość niezależna od układu współrzędnych

Ⓔ przykłady skalarów: temperatura, długość drogi

- tensor jest funkcją przyporządkowująca wektorom wielkości skalarne (wymaga uściślenia: jakie wektory? jaka funkcja?)
- wartość tensora dla jego argumentów jest skalarem, a więc **nie zależy od doboru układu współrzędnych**
- prawa fizyki wyrażane w kategoriach tensorów nie zależą od układu współrzędnych

⊛ Zmiana bazy

Kontrawektory

- $\mathcal{V}$ - przestrzeń wektorowa skończenie wymiarowa, tzn. dla założonej bazy $\boldsymbol{E} = [\boldsymbol{e}_1, .., \boldsymbol{e}_n]$ dowolny wektor $\vec{\boldsymbol{v}} \in \mathcal{V}$ można zapisać jako

$$\vec{\boldsymbol{v}} = \sum_j v_j \boldsymbol{e}_j = \boldsymbol{E}\boldsymbol{v}$$

gdzie $\boldsymbol{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$, przy czym v_j są *współrzędnymi* wektora $\vec{\boldsymbol{v}}$ w bazie $\boldsymbol{E}$

- liniowa zmiana bazy: $\boldsymbol{e}'_i = \boldsymbol{S}\boldsymbol{e}_i,\ i = 1, \ldots, n$, tzn. $\boldsymbol{E}' = \boldsymbol{E}\boldsymbol{S}$ gdzie $\boldsymbol{S}$ jest (dodatnio określoną) *macierzą transformacji bazy*
- transformacja bazy nie zmienia wektora, zatem współczynniki wektora ulegną zmianie

Ⓔ przykład: jeśli każdy wektor bazowy a_i zmieni się a_i-krotnie ($a_i > 0$) to

$$\vec{\boldsymbol{v}} = \sum_i \overline{\boldsymbol{e}}_i \overline{v}_i = \sum_i a_i \boldsymbol{e}_i \overline{v}_i \qquad \text{tzn.} \qquad \overline{v}_i = \frac{1}{a_i} v_i$$

czyli każda ze współrzędnych **zmieni** się $1/a_i$ krotnie

- w ogólnym przypadku $\vec{\boldsymbol{v}} = \boldsymbol{E}'\boldsymbol{v}' = \boldsymbol{E}\boldsymbol{S}\boldsymbol{v}' = \boldsymbol{E}\boldsymbol{v}$ zatem współczynniki wektora ulegają zmianie przeciwnej niż baza

$$\boldsymbol{E}' = \boldsymbol{E}\boldsymbol{S} \qquad \Rightarrow \qquad \boldsymbol{v}' = \boldsymbol{S}^{-1}\boldsymbol{v}$$

- *wektory kontrawariantne* (tzn. o współrzędnych zmieniającymi się przeciwnie)

⊛ Zmiana bazy

Kowektory

- funkcje liniowe wektorów o wartościach rzeczywistych $f : V \mapsto \mathbb{R}$

$$f(\boldsymbol{v}) = \boldsymbol{y}^{\mathsf{T}}\boldsymbol{v}$$

są wyznaczane przez *wektory dualne* $\boldsymbol{y} = \begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix}$

- zbiór $\mathcal{V}^*$ tych funkcji jest przestrzenią wektorową (tzw. *przestrzeń dualna*)

Ⓔ przykłady negatywne (nieliniowe): $g(\boldsymbol{v}) = \boldsymbol{v}^{\mathsf{T}}\boldsymbol{v}$, $g(\boldsymbol{v}) = \det(\boldsymbol{v}\boldsymbol{v}^{\mathsf{T}})$

- każdą funkcję f można zapisać w postaci wykorzystującej *bazę dualną* $\boldsymbol{V} = [\boldsymbol{v}_1 \cdots \boldsymbol{v}_n]$, gdzie $\boldsymbol{v}_i = v_i\boldsymbol{\epsilon}_i$, $\boldsymbol{\epsilon}_i$ jest i-tym wersorem
- liniowa transformacja bazy pierwotnej $\boldsymbol{E}' = \boldsymbol{E}\boldsymbol{S}$ pociąga za sobą transformację bazy dualnej $\boldsymbol{V}' = S^{-1}\boldsymbol{V}$
- w rezultacie współrzędne wektorów dualnych zmieniają się **zgodnie ze zmianami bazy** - *wektory kowariantne, kowektory* (ang. *co-vary*)

⊛ Tensory

Tensor jako funkcja wektorów

- *tensor* rzędu n: funkcja multiliniowa (liniowa względem każdego argumentu) o $n = p + q$ argumentach wektorowych $\boldsymbol{v}_i \in \mathcal{V} = \mathbb{R}^m$ i wartościach rzeczywistych

$$y = f(\boldsymbol{v}_1, .., \boldsymbol{v}_p, \boldsymbol{v}_1^*, .., \boldsymbol{v}_q^*), \qquad y \in \mathbb{R}, \qquad \boldsymbol{v}_i \in \mathcal{V}, \quad \boldsymbol{v}_j^* \in \mathcal{V}^*$$

⊛ Tensory

Tensor jako tablica w zadanej bazie

- tensor - jako funkcja multiliniowa - może być zdefiniowany poprzez jej wartości dla wszystkich m wektorów bazowych w $\mathbb{R}^m$ dla każdego spośród n argumentów wektorowych, tzn. poprzez znajomość m^n liczb rzeczywistych
- tensor może być zatem reprezentowany przez parę $(\mathbf{F}, \boldsymbol{B})$, gdzie
  - $\mathbf{F}$ jest tablicą o rozmiarze $\underbrace{m \times \cdots \times m}_{n \text{ razy}}$
  - $\boldsymbol{B} = \{\boldsymbol{e}_1, .., \boldsymbol{e}_m\}$ jest dowolnie wybraną bazą wektorową w $\mathcal{V}$

  przy czym zmiana bazy powoduje odpowiednią zmianę tablicy $\mathbf{F}$

⊛ Tensory

Tensory w informatyce

- jeżeli w zastosowaniu nie dokonujemy zmiany bazy, to
  - nie jest istotne rozróżnienie między wektorami i kowektorami
  - tensor może być określony wprost jako tablica wielowymiarowa $\underbrace{m \times \cdots \times m}_{n \text{ razy}}$
- taką interpretację stosujemy dla sieci neuronowych; własności tensora (zależności od bazy) definiowanego w matematyce są nieistotne

⊕ Tensory

Różne rozumienie tensora

- porównanie definicji

funkcja multiliniowa $\mathbb{R}^n \mapsto \mathbb{R}$	para (tablica, baza)	tablica
definicje równoważne		znaczne uproszczenie
fizyka, matematyka		sieci neuronowe
niezależny od układu współrzędnych	tablica zależna od bazy	(ustalona) baza euklidesowa
argumenty: wektory i kowektory		nie rozróżniane
	działania zachowują multiliniowość	dowolne działania

- pojęcie matematyczne a jego komputerowa reprezentacja

	pojęcie	reprezentacja
wektor	element przestrzeni liniowej	tablica 1D
macierz	operacja liniowa w $\mathbb{R}^n$	tablica 2D
tensor	obiekt niezmienniczy wzgl. transf. liniowej	tablica nD

Tensor jako tablica

Notacja

- w zastosowaniach informatycznych tensor rzędu n zdefiniujemy jako *tablicę* (ang. *array*) indeksowana przez n indeksów

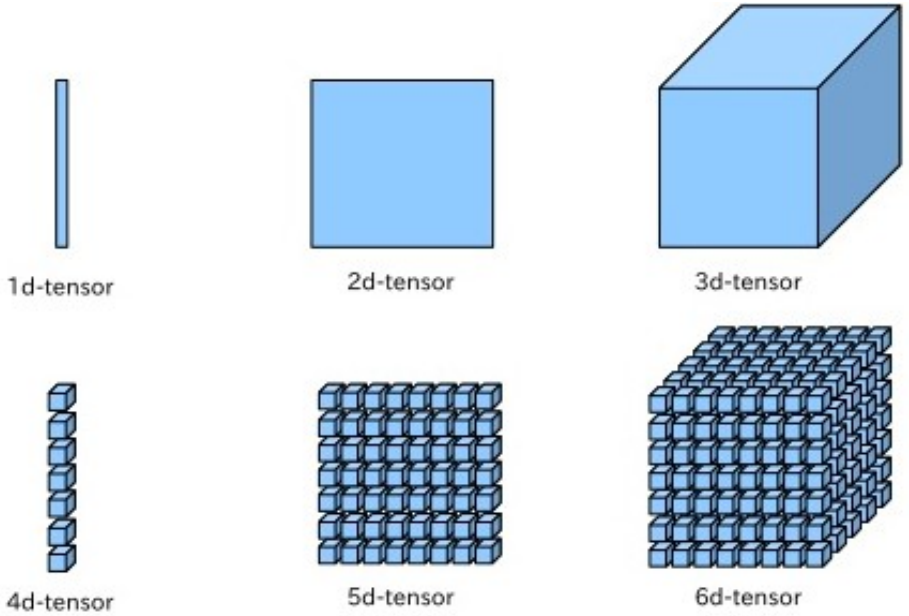
$$\mathbf{A} = \mathbf{A}_{i_1,..,i_n} = \{a_{i_1,..,i_n} : i_1 = 1,..,i_{1_{\max}}, \ldots, i_n = 1,..,i_{n_{\max}}\}$$

  - n: *rząd* tensora (ang. *rank, order, degree*)
  - $i_1, \ldots, i_n$: *indeksy* tensora
  - *rozmiar (kształt) tablicy* $i_{1\max} \times \cdots \times i_{n\max}$
- *notacja indeksowa* tensorów
  - tensory oznaczamy zwykle dużymi literami pogrubionymi, często z *indeksami*, na ogół nie oddzielanych przecinkami
  - indeksy należy rozumieć jako etykiety którym **nie należy** przypisywać konkretnych wartości liczbowych
  - wartości liczbowe elementów tensora oznaczamy na ogół literami cienkimi

Tensor jako tablica

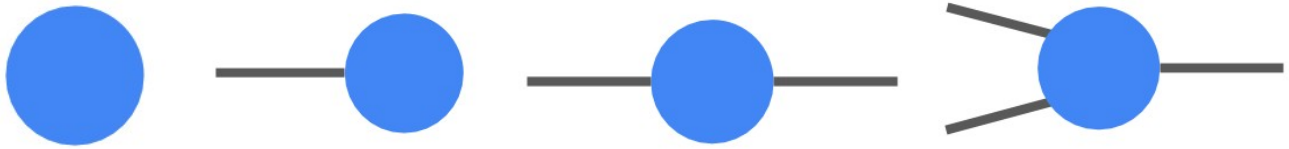
Interpretacje graficzne

- tensory rzędu 0 utożsamiane są ze skalarami, rzędu 1 - z wektorami, rzędu 2 - z macierzami (tożsamości te nie są usprawiedliwione w teorii tensorów)



Knoldus 2012

- diagramy Penrose'a



diagramy Penrose'a: tensory rzędu 0, 1, 2, 3
odcinki odpowiadają indeksom tensorów

Działania na tensorach

Suma, iloczyn przez stałą

- iloczyn tensora przez stałą: każdy element jest mnożony przez stałą

$$\mathbf{B}_{i_1,..,i_n} = c\mathbf{A}_{i_1,..,i_n}$$

- suma (podobnie: różnica) tensorów o jednakowych rozmiarach: każdy element jest sumą odpowiednich elementów tensorów składowych

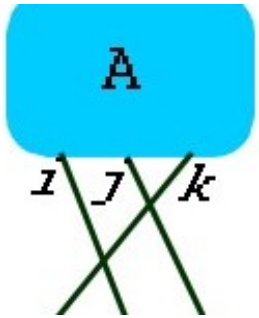
$$\mathbf{C}_{i_1,..,i_n} = \mathbf{A}_{i_1,..,i_n} + \mathbf{B}_{i_1,..,i_n}$$

- własności: działania przemienne i łączne

Działania na tensorach

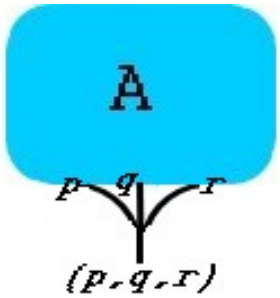
Permutacja, zmiana kształtu, wektoryzacja

- *permutacja* tensora

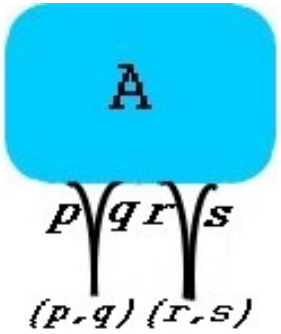


permutacja tensora
$\mathbf{A}_{ijk}(3,1,2) = \mathbf{A}_{kij}$

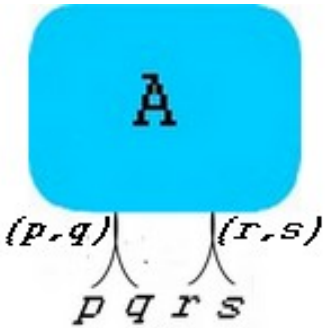
- *zmiana kształtu* (ang. *reshape*) tensora - zmiana kształtu tensora
- w szczególności *wektoryzacja* (spłaszczenie) tensora: zapis tensora w postaci wektora $\boldsymbol{a} = \text{vec}(\mathbf{A})$



zmiana kształtu: wektoryzacja



zmiana kształtu: sprowadzenie do macierzy



zmiana kształtu: redukcja macierzy

Działania na tensorach

Operacje Hadamarda

- *iloczyn Hadamarda* (*iloczyn Schura*) $\odot$ $\circ$ lub $\odot$ tensorów o identycznych rozmiarach: iloczyn element po elemencie:

$$\mathbf{P} = \mathbf{A} \,.\, \mathbf{B} \quad \text{jeśli} \quad \mathbf{P}_{i_1,..,i_n} = \mathbf{A}_{i_1,..,i_n} \mathbf{B}_{i_1,..,i_n}$$

  - przemienność, łączność, rozdzielność względem dodawania
  - $\oplus$ Matlab, Gauss: $.*$, Fortran, R, Wolfram Language (Mathematica), Python (NumPy, SymPy): $*$

- inne *operacje Hadamarda*
  - potęgowanie $\mathbf{P} = \mathbf{A}^{\cdot m}$ jeśli $\mathbf{P}_{i_1,..,i_n} = \mathbf{A}^{m}_{i_1,..,i_n}$
  - pierwiastkowanie $\mathbf{P} = \mathbf{A}^{\cdot 1/2}$ jeśli $\mathbf{P}_{i_1,..,i_n} = \mathbf{A}^{1/2}_{i_1,..,i_n}$ (o ile elementy są nieujemne)
  - dzielenie $\mathbf{P} = \mathbf{A} \,./\, \mathbf{B}$ jeśli $\mathbf{P}_{i_1,..,i_n} = \mathbf{A}_{i_1,..,i_n} / \mathbf{B}_{i_1,..,i_n}$ (o ile wszystkie elementy B są niezerowe)
- uwaga: w kontekście operacji na funkcjach przez $\circ$ oznaczamy złożenie funkcji

Działania na tensorach

Kontrakcja indeksów i ślad

- *kontrakcja* indeksów i_p, i_q, $p < q \leqslant n$ tensora $\mathbf{A}_{i_1,..,i_n}$:

$$\sum_z \mathbf{A}_{i_1,..,i_{p-1},z,i_{p+1},..,i_{q-1},z,i_{q+1},..,i_n}$$

  to zsumowanie elementów tensora dla których indeksy te są identyczne
- tensor którego indeksy i_p, i_q uległy kontrakcji nazywa się często *śladem* (ang. *trace*) oryginalnego tensora wzdłuż indeksów i_p, i_q

$$\mathrm{tr}_{i_p,i_q}\, \mathbf{A}_{i_1,..,i_n} \stackrel{\text{def}}{=} \sum_z \mathbf{A}_{i_1,..,i_{p-1},z,i_{p+1},..,i_{q-1},z,i_{q+1},..,i_n}$$

- ślad tensora rzędu n jest tensorem rzędu $n-2$
  - liczba indeksów (rząd tensora) musi być > 1
  - wymiary indeksów podlegających kontrakcji muszą być identyczne
  - z (dowolne oznaczenie) - wartości tzw. indeksu niemego (ang. *dummy*)
  - ⊛ w ogólnym przypadku jeden z wektorów powinien być kowariantny a drugi kontrawariantny

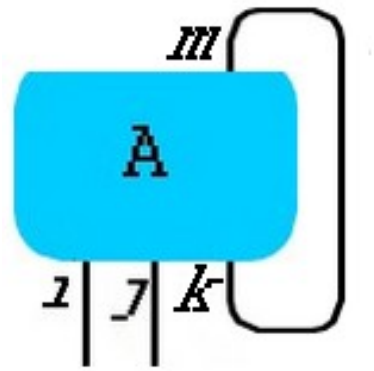
⊛ *notacja Einsteina*: sumowanie względem par identycznych indeksów z pominięciem symbolu sumy

$$\sum_z \mathbf{A}_{i_1,..,i_{p-1},z,i_{p+1},..,i_{q-1},z,i_{q+1},..,i_n} \underset{\text{notacja Einsteina}}{=} \mathbf{A}_{i_1,..,i_{p-1},z,i_{p+1},..,i_{q-1},z,i_{q+1},..,i_n}$$

Działania na tensorach

Kontrakcja indeksów: diagram Penrose'a

- (E) przykład: dla macierzy: $\mathbf{A}_{i,i} = \operatorname{tr} \boldsymbol{A}$
- (E) przykład: tensor $\mathbf{A}_{pqr}$ 4-wymiarowy; kontrakcja indeksów p i r: $\mathbf{A}_q = \mathbf{A}_{1q1} + \mathbf{A}_{2q2} + \mathbf{A}_{3q3} + \mathbf{A}_{4q4}$, $q = 1, 2, 3, 4$ ("ślad" wzdłuż $p = r$)
- (E) przykład: kontrakcja indeksów i, q tensora 3-wymiarowego $\mathbf{A}_{p,i,q,j}$
  $\mathbf{A}_{p,j} = \mathbf{A}_{p,z,z,j} = \sum_{i=q} \mathbf{A}_{p,i,q,j} = \mathbf{A}_{p,1,1,j} + \mathbf{A}_{p,2,2,j} + \mathbf{A}_{p,3,3,j}$



kontrakcja indeksów j, m
$\mathbf{B}_{ik} = \sum_{j=m} \mathbf{A}_{ijkm} = \sum_x \mathbf{A}_{ixkx}$

Działania na tensorach

Kontrakcja indeksów - przykłady

Ⓔ przykład: $\mathbf{A}_{i,j,k} = \left(\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 2 \\ 1 & 3 \\ 2 & 1 \end{bmatrix} \right)$ (rozmiar $3 \times 2 \times 2$)

$$\begin{aligned} A_{111} &= 1, A_{121} = 3 \quad & A_{112} &= -1, A_{122} = 2 \\ A_{211} &= 2, A_{221} = 4 \quad & A_{212} &= 1, A_{222} = 3 \\ A_{311} &= 1, A_{321} = 1 \quad & A_{312} &= 2, A_{322} = 1 \end{aligned}$$

kontrakcja indeksów j, k: $\boldsymbol{b}_i = \sum_x A_{i,x,x}$, gdzie

$$\begin{aligned} b_1 &= A_{111} + A_{122} = 1 + 2 = 3 \\ b_2 &= A_{211} + A_{222} = 2 + 3 = 5 \\ b_3 &= A_{311} + A_{322} = 1 + 1 = 2 \end{aligned}$$

czyli $\boldsymbol{b} = (3, 5, 2)$

Działania na tensorach

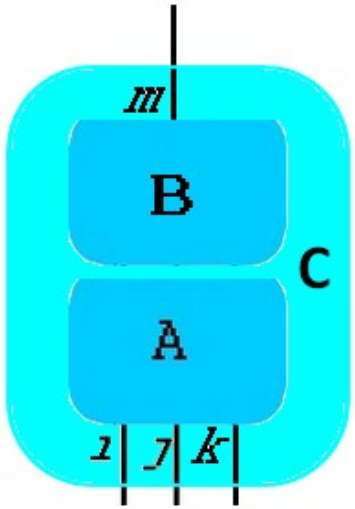
Iloczyn zewnętrzny

- *iloczyn zewnętrzny* (ang. *outer product*) tensorów $\mathbf{A}_{i_1,..,i_p}$, $\mathbf{B}_{j_1,..,j_q}$

$$(\mathbf{A} \otimes \mathbf{B})_{i_1,..,i_p,j_1,..,j_q} \stackrel{\text{def}}{=} \mathbf{A}_{i_1,..,i_p} \mathbf{B}_{j_1,,..,j_q}$$

to tensor o elementach równych iloczynom odpowiadających sobie par elementów czynników

- rząd wyniku jest sumą rzędów czynników
- indeksy elementów prosuktuq są złożeniem indeksów czynników
- produkt zewnętrzny nie jest przemienny
- produkt zewnętrzny jest rozłączny względem dodawania
- produkt zewnętrzny zapisuje się również jako $\mathbf{AB}$, ale działanie to nie jest to zgodne z mnożeniem macierzy



iloczyn zewnętrzny tensorów
$\mathbf{C}_{ijkm} = \mathbf{A}_{ijk} \otimes \mathbf{B}_m$

Działania na tensorach

Iloczyn zewnętrzny -przykłady

Ⓔ przykład

$$\mathbf{A} = \begin{bmatrix} 1 & -1 \\ 2 & 4 \end{bmatrix} \quad = (\,(1,-1),(2,4)\,)$$

$$\mathbf{B} = \begin{bmatrix} -2 & 1 \end{bmatrix} \quad = (-2,1)$$

$\mathbf{A} \otimes \mathbf{B} = (\,(\,(-2,1),(2,-1)\,),(\,(-4,2),(-8,4)\,)\,)$
$\mathbf{B} \otimes \mathbf{A} = (\,(\,(-2,2),(-4,-8)\,),(\,(1,-1),(2,4))\,)$

Ⓔ przykład: diada (ang. *dyad*): dla tensorów 1 rzędu (wektorów) $(\boldsymbol{a} \otimes \boldsymbol{b})_{ij} = \boldsymbol{a}_i \boldsymbol{b}_j^{\mathsf{T}}$

⊕ tf.tensordot(a,b,0,name=None), pytorch.tensordot(a,b,0)

Działania na tensorach

Iloczyn wewnętrzny

- *iloczyn wewnętrzny tensorów* (ang. *inner product*) $\mathbf{A}_{i_1,..,i_{k-1},i_k}$, $\mathbf{B}_{k,j_2,..,j_m}$

$$(\mathbf{A} \bullet \mathbf{B})_{i_1,..,i_{n-1}j_2,..,j_m} = \sum_k \mathbf{A}_{i_1,..,i_{k-1},k}\mathbf{B}_{k,j_2,..,j_m}$$

jest równoważny dwóm kolejnym operacjom:

  - iloczyn zewnętrzny tensorów $\mathbf{A}_{i_1,..,i_n} \otimes \mathbf{B}_{j_1,..,j_m}$
  - kontrakcji "przyległych" indeksów i_n, j_1

$$\mathbf{C}_{i_1,..,i_{n-1}j_2,..,j_m} = \sum_{i_n=j_1} A_{i_1,..,i_n} B_{j_1,..,j_m}$$

⊛ w notacji Einsteina

$$\mathbf{A} \bullet \mathbf{B} = (\mathbf{A} \otimes \mathbf{B})_{i_1,..,i_{n-1},j_2,..,j_m} = \underbrace{\mathbf{A}_{i_1,..,i_{n-1},z}\mathbf{B}_{z,j_2,..,j_m}}_{\text{notacja Einsteina}}$$

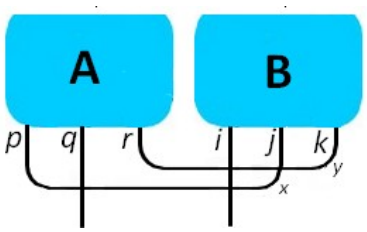
  - nie jest przemienny (z wyjątkiem iloczynu tensorów rzędu 1)
  - jest rozłączny względem dodawania

- iloczyn wewnętrzny można przeprowadzić względem indeksów innych niż przyległe

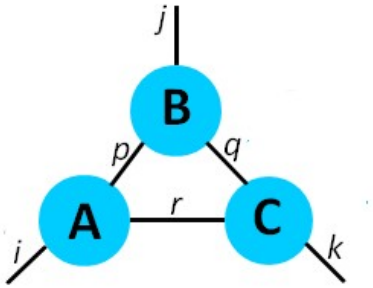
$$\mathbf{A} \underset{i_k,j_\ell}{\bullet} \mathbf{B} = (\mathbf{A} \otimes \mathbf{B})_{i_1,..,i_k z i_{k+1},..,i_n j_1,..,j_{\ell-1} z j_{\ell+1},..,j_m}$$

Działania na tensorach

Iloczyn wewnętrzny - Diagram Penrose'a



iloczyn wewnętrzny
$\mathbf{A}_{xqy} \bullet \mathbf{B}_{ixy}$



iloczyn wewnętrzny
$\mathbf{A}_{irp} \bullet \mathbf{C}_{kqr} \bullet \mathbf{B}_{jpq}$

⊕ tf.tensordot(a, b, axes, name=None), numpy.tensordot(a,b,axes), pytorch.tensordot(a,b,axes); axes =number lub axes = tensor 2× n

Działania na tensorach

Iloczyn wewnętrzny: przykłady

Ⓔ iloczyn wewnętrzny dwu tensorów 1 rzędu

$$\underbrace{\mathbf{A}_i \bullet \mathbf{B}_j}_{\text{skalar}} = \sum_{i=j} A_i B_j \quad \underbrace{= \mathbf{A}_z \mathbf{B}_z}_{\text{notacja Einsteina}}$$

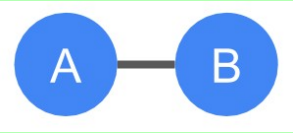


diagram Penrose'a

tzn. w notacji wektorowej jest to iloczyn skalarny wektorów $\boldsymbol{A}^\mathsf{T}\boldsymbol{B}$

Ⓔ iloczyn wewnętrzny tensora 2 rzędu i tensora 1 rzędu

$$\text{tzn.} \quad \mathbf{A}_{ij} \bullet \mathbf{B}_k = \sum_p \mathbf{A}_{ip}\mathbf{B}_p \quad \underbrace{= \mathbf{A}_{ip}\mathbf{B}_p}_{\text{notacja Einsteina}}$$

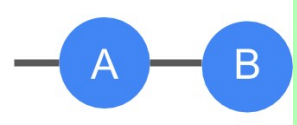


diagram Penrose'a

tzn. w notacji macierzowej działanie jest równoważne mnożeniu macierzy przez wektor: $\boldsymbol{AB}$

Ⓔ iloczyn wewnętrzny tensorów 2 rzędu

$$\mathbf{A}_{ij} \bullet \mathbf{B}_{k\ell} = \sum_{j=k} A_{ij} B_{k\ell} \quad \underbrace{= \mathbf{A}_{iz}\mathbf{B}_{z\ell}}_{\text{notacja Einsteina}}$$

diagram Penrose'a

tzn. w notacji macierzowej działanie odpowiada mnożeniu macierzy: $\boldsymbol{AB}$

Tensory

Przykłady działań

$$\mathbf{A} = \left(\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 2 \\ 1 & 3 \\ 2 & 1 \end{bmatrix} \right), \quad \mathbf{B} = \left(\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -2 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \right) \quad \mathbf{C} = \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix}$$

Ⓔ $\mathbf{A} = (\,(\,(1,3),(2,4),(1,1)\,),(\,(-1,2),(1,3),(2,1)\,)\,)$

Ⓔ $\mathbf{A}$, $\mathbf{B}$: wymiar 3, rozmiar $2 \times 3 \times 2$, $\mathbf{C}$: wymiar 2, rozmiar 2×3

,

Ⓔ $\mathrm{vec}(\mathbf{A}) = (1,3,2,4,1,1,-1,2,1,3,2,1)$

Ⓔ $\mathbf{A} \,.\, \mathbf{B} = \mathbf{B} \,.\, \mathbf{A} = \left(\begin{bmatrix} 1 & 9 \\ 4 & 16 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -4 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \right)$

,

Ⓔ

$$\mathbf{B} \bullet \mathbf{C} = \left(\begin{bmatrix} 1\cdot 1 + 3\cdot 4 & 1\cdot 3 + 3\cdot 0 & 1\cdot 2 + 3\cdot 1 \\ 2\cdot 1 + 4\cdot 4 & 2\cdot 3 + 4\cdot 0 & 2\cdot 2 + 4\cdot 1 \\ 0\cdot 1 + 1\cdot 4 & 0\cdot 3 + 1\cdot 0 & 0\cdot 2 + 1\cdot 1 \end{bmatrix}, \right.$$

$$\left. \begin{bmatrix} 0\cdot 1 + -2\cdot 4 & 0\cdot 3 - 2\cdot 0 & 0\cdot 2 - 2\cdot 1 \\ 1\cdot 1 + 0\cdot 4 & 1\cdot 3 + 0\cdot 0 & 1\cdot 2 + 0\cdot 1 \\ 1\cdot 1 + 1\cdot 4 & 1\cdot 3 + 1\cdot 0 & 1\cdot 2 + 1\cdot 1 \end{bmatrix} \right)$$

$$= \left(\begin{bmatrix} 13 & 3 & 5 \\ 18 & 6 & 8 \\ 4 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -8 & 0 & -2 \\ 1 & 3 & 2 \\ 5 & 3 & 3 \end{bmatrix} \right)$$

Tensory

Przykłady działań (2)

$$\mathbf{C} = \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} = (\,(1,3,2),(4,0,1)\,)$$

Ⓔ $$\mathbf{D} = \mathbf{C} \otimes \mathbf{C} = (\,(\,(\,(1,3,2),(4,0,1)\,),(\,(3,9,6),(12,0,3)\,),(\,(2,6,2),(8,0,2)\,)\,),$$
$$(\,(\,(4,12,8),(16,12,4)\,),(\,(0,0,0),(0,0,0)\,),(\,(1,3,2),(4,0,1)\,)\,)\,)$$

3.1 Sygnały w sztucznych sieciach neuronowych
3.2 Sztuczny neuron
3.3 Warstwy
3.4 Konstrukcja sztucznych sieci neuronowych
3.5 Uczenie sieci
3.6 Sieci statyczne a sieci dynamiczne
3.7 Tablice wielowymiarowe i tensory
3.8 ® Splot 2D

® Splot 2D

- obraz 2D w skali szarości: *tablica pikseli* $\boldsymbol{U}_{ij} = \{U_{ij},\ (i,j) \in I\}$
- *filtr* $\boldsymbol{F}_{i'j'} = \{F_{i'j'},\ (i',j') \in I'\}$: zestaw pikseli (wag) zapisanych zwykle w tablicy prostokątnej (masce filtru) I'
  - piksele maski są zwykle indeksowane względem centrum maski $(0,0)$
  - w sieciach neuronowych rozmiar maski filtru jest określony przez pole recepcyjne neuronu
  - w (głębokich) sieciach neuronowych maska ma zwykle nieparzystą liczbę pikseli, np. 3×3 lub 5×5 px
- *splot 2D* obrazu $\boldsymbol{U}$ z filtrem $\boldsymbol{F}$:

$$\boldsymbol{U}^F = \boldsymbol{U} * \boldsymbol{F} = \boldsymbol{F} * \boldsymbol{U}$$

  jest kombinacją liniową (sumą iloczynów) pikseli obrazu i odpowiednich pikseli filtru

$$U^F_{ij} = \sum_{(i'j') \in I'} U_{i-i',j-j'} F_{i',j'}$$

  przy czym obliczenia są przeprowadzane dla każdego lub dla wybranych punktów (i,j) obrazu
- bezpośrednie uogólnienia na obrazy 3D (np. obrazy kolorowe) i na problemy nD

® Splot 2D

Postać korelacyjna

- jeśli piksele maski są indeksowane przeciwnie niż piksele obrazu, tzn. stosowana jest *maska odbita* $\bar{\boldsymbol{F}}$

$$\bar{F}_{i',j'} = F_{-i',-j'}$$

to można zapisać splot w wygodniejszej *postaci korelacyjnej*

$$U^F_{i,j} = \sum_{(i',j')\in I'} U_{i+i',j+j'}\bar{F}_{i',j'} \quad \text{tzn.} \quad U^F_{\boldsymbol{i}} = \sum_{\boldsymbol{x}'\in I'} U_{\boldsymbol{x}+\boldsymbol{x}'}\bar{F}_{\boldsymbol{x}'}$$

tzn. dla punktu (i,j) obrazu, centrum maski odbitej "umieszczane jest" w tym punkcie i obliczana jest kombinacja liniowa punktów maski i pikseli obrazu

- jeśli - jak dla sieci neuronowych - współczynniki maski są wynikiem treningu, to otrzymane maski są od razu traktowane jako maski odbite
- zawsze będziemy domyślnie używać maski odbitej (bez stosowania oznaczenia $\bar{I}$)

® Splot 2D

Efekty brzegowe (1)

- *przycinanie* (ang. *cropping*): maska filtru nie wychodzi poza obraz
  - splot może być obliczony co najwyżej w $(I - I' + 1) \times (J - J' + 1)$ punktach obrazu
  - matematycznie zgodny z definicją splotu
  - w sieciach neuronowych stosowane najczęściej
- *uzupełnianie stałą* (ang. *padding*): przyjmuje się, że piksele poza brzegiem obrazu są zerowe (ang. *zero padding*) lub równe średniej jasności obrazu (ang. *mean padding*)
  - może powadzić do dodatkowe krawędzie na brzegach (efekt mniejszy dla uzupełniania średnią)
  - brak uzasadnienia matematycznego
  - czasem stosowane w sieciach neuronowych w wersji uzupełniania zerami
- *zakleszczanie* (ang. *clamping*) pikseli obrazu: brakujące wartości pikseli obrazu są zastępowane wartościami najbliższych pikseli obrazu

® Splot 2D

Efekty brzegowe (2)

- *przedłużanie okresowe, zawijanie* (ang. *wrapping*): brakujące wartości są uzupełniane wartościami z przeciwnej krawędzi obrazu
  - tworzenie struktury *podwójnego torusa*
  - otrzymywany automatycznie, jeśli splot jest dokonywany w dziedzinie Fouriera
  - efekt dodatkowych krawędzi na brzegach, jeśli jasności przeciwległych krawędzi obrazu są różne
- *odbicie obrazu* (ang. *mirroring*): lustrzane odbicie obrazu poza krawędziami
  - równoważne stosowaniu transformaty kosinusowej w miejsce transformaty Fouriera
- *obcinanie maski* (ang. *kernel cropping*): zerować współczynniki maski poza obrazem, równocześnie normując współczynniki wewnątrz obrazu

Literatura

Materiały źródłowe

Nair, V. and G. E. Hinton (2010). "Rectified linear units improve restricted Boltzmann machines". In: *Proc. 27th International Conference on Machine Learning*.

Materiały pomocnicze

Arbib, M. A., ed. (1998). *The handbook of Brain Theory and Neural Networks*. Cambridge, Massachusetts: MIT Press.

Brannon, R. M. (2004). *Functional and Structured Tensor Analysis for Engineers. A casual (intuition-based) introduction to vector and tensor analysis with reviews of popular notations used in contemporary materials modeling*. UNM BOOK DRAFT. University of New Mexico, Albuquerque.

Clarke, D. A. (2011). *A Primer on Tensor Calculus*. Tech. rep. Saint Mary's University, Halifax NS, Canada.

Dullemond, K. and K. Peeters (2010). *Introduction to Tensor Calculus*. Tech. rep. URL: maths.dur.ac.uk/users/kasper.peeters/pdf/tensor_en.pdf.

Feng, J. C. (2017). *The Poor Man's Introduction to Tensors*. Tech. rep. Universty of Texas. URL: ph.utexas.edu/~jcfeng/notes/Tensors_Poor_Man.pdf.

Hackl, K. and M. Goodarzi (2010). *A Small Compendium on Vector and Tensor Algebra and Calculus*. Institute of Mechanics, Faculty of Civil and Environmental Engineering, Ruhr-University Bochum. URL: http://web.iitd.ac.in/~pmvs/courses/mcl702/tensors.pdf.

Literatura (cd.)

Ioffe, S. and C. Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32 nd International Conference on Machine Learning*. Vol. 37. JMLR: W and CP. Lille, France.

Kolecki, J. C. (2002). *An Introduction to Tensors for Students of Physics and Engineering*. Tech. rep. NASA/TM—2002-211716. Glenn Research Center, Cleveland, Ohio. URL: gltrs.grc.nasa.gov.

Ostrowska-Maciejewska, J. (2007). "Podstawy zastosowania rachunku tensorowego. Wykład na Studiach Doktoranckich w IPPT PAN". In: *Prace Instytutu Podstawowych Problemów Techniki PAN* 1, pp. 5–185.

Porat, B. (2014). *A Gentle Introduction to Tensors*. Tech. rep. Department of Electrical Engineering, Technion – Israel Institute of Technology.

Schüz, A. (n.d.). "Neuroanatomy in Computational Perspective". In: *Arbib 1998*, pp. 622–626.

Sochi, T. (2016). *Introduction to Tensor Calculus*. Tech. rep. Department of Physics & Astronomy, University College London. URL: www.researchgate.net/publication/296707240_Introduction_to_Tensor_Calculus.

Część II: Sieci perceptronowe a klasyfikacja

Temat 4. Obrazy i klasyfikacja

Temat 5. Perceptron Rosenblatta

Temat 6. Maszyny wektorów podpierających

Temat 4: Obrazy i klasyfikacja

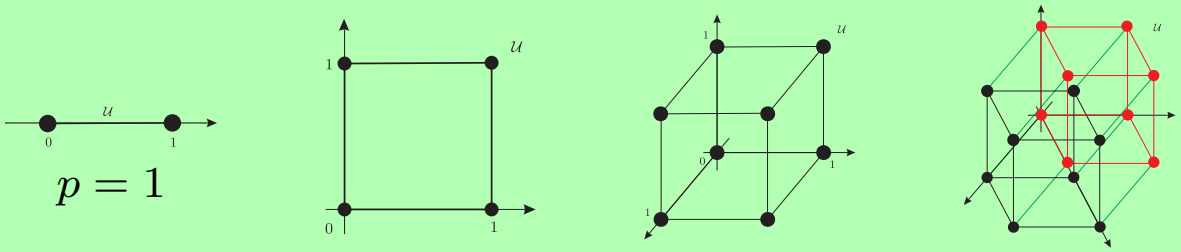
4.1 Problem klasyfikacji obrazów
4.2 Klasyfikacja na dwie klasy
4.3 Klasyfikacja liniowa
4.4 Podejście probabilistyczne

4.1 Problem klasyfikacji obrazów
4.2 Klasyfikacja na dwie klasy
4.3 Klasyfikacja liniowa
4.4 Podejście probabilistyczne

Reprezentacje obrazów

obrazy	obraz jako tablica	obraz jako wektor	oznaczenie
obraz $p_1 \times p_2$ **pikseli**	tablica $p_1 \times p_2$	wektor $p_1 p_2$-elementowy	$\boldsymbol{u} \in U \subset \mathbb{R}^{p_1 p_2}$
piksel	element tablicy	element wektora	$\boldsymbol{u} \in U$
liczba pikseli	liczba elementów tablicy		
przestrzeń obrazów	zbiór $\mathbb{R}^{p_1} \times \mathbb{R}^{p_2}$	zbiór $\mathbb{R}^{p_1 p_2}$	$U \subset \mathbb{R}^{p_1 p_2}$

Ⓔ Obrazy jako wektory



zbiór wszystkich obrazów binarnych o p pikselach

zbiór wszystkich obrazów o p pikselach

obrazy zmienne w czasie $\mathcal{T} \mapsto U$

Reprezentacje obrazów

Przykłady

obraz	tablica	wektor	U	$\lvert U \rvert$
obraz 32×32 px binarny	tablica 32×32 binarna	wierzchołek kostki 1024-wymiarowej	$\{0,1\}^{1024}$	$2^{1024} \approx G^3$
obraz 32×32 px o głębi szarości 1 bpp	tablica 32×32 o elementach $0,..,255$	punkt siatki 1024-wymiarowej o kroku $1/256$	$\{0, \frac{1}{256}, .., \frac{255}{256}\}^{1024}$	$256^{1024} \approx G^{26}$
obraz 32×32 px	tablica 32×32 o elementach w $[0,1)$	punkt kostki 1024-wymiarowej	$[0,1)^{1024}$	—
obraz binarny 1920×1200 px	tablica 1920×1200 binarna	wierzchołek kostki 2304000-wymiar.	$\{0,1\}^{2304000}$	$2^{2304000} \approx G^{6935}$
obraz 1920×1200 px o głębi szarości 1 bpp	tablica 1920×1200 o elementach $0,..,255$	punkt siatki 2304000-wymiar. o kroku $1/256$	$\{0, \frac{1}{256}, .., \frac{255}{256}\}^{p}$ $p = 2304000$	$256^{2304000} \approx G^{55486}$
obraz 1920×1200 px	tablica 1920×1200 o elementach w $[0,1)$	punkt kostki 2304000-wymiar.	$[0,1)^{2304000}$	—

$$\text{googol} = 10^{100}$$
$$\text{googolplex} = 10^{\text{googol}}$$
liczba atomów we wszechświecie: 10^{80}

Obrazy i wagi sieci jako punkty przestrzeni o bardzo dużym wymiarze

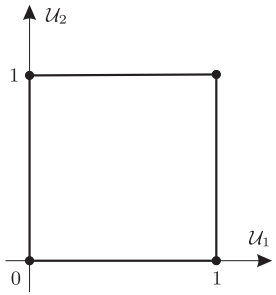
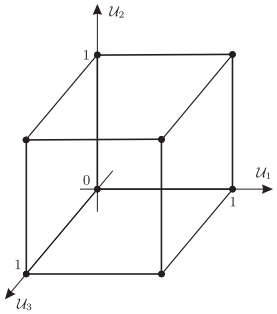
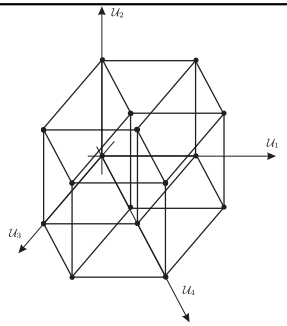
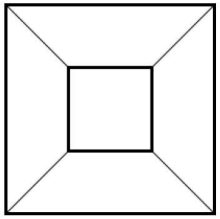
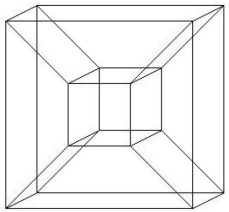
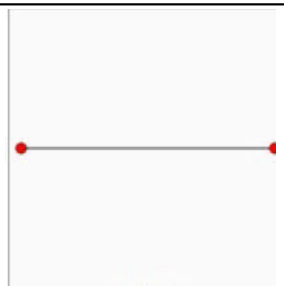
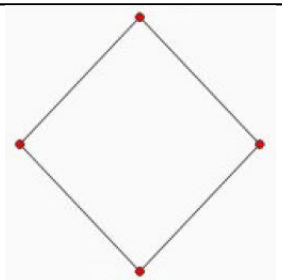
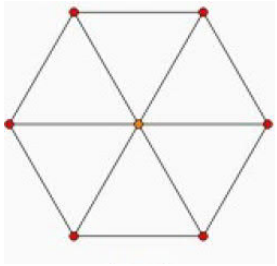
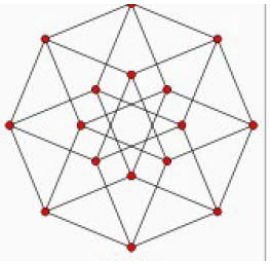
- nawet najprostsze obrazy są równoważne punktom przestrzeni o bardzo dużych wymiarach
- punkty bliskie sobie postrzegane są jednak jako te same obrazy
- wektory parametrów współczesnych sieci neuronowych są również równoważne punktom przestrzeni o bardzo dużym wymiarze
- nasze intuicje dotyczące przestrzeni wielowymiarowych bazują na geometrii obiektów 2 i 3 wymiarowych
- pokażemy, że intuicje te mogą być błędne,

Geometria przestrzeni wielowymiarowych

Kostka

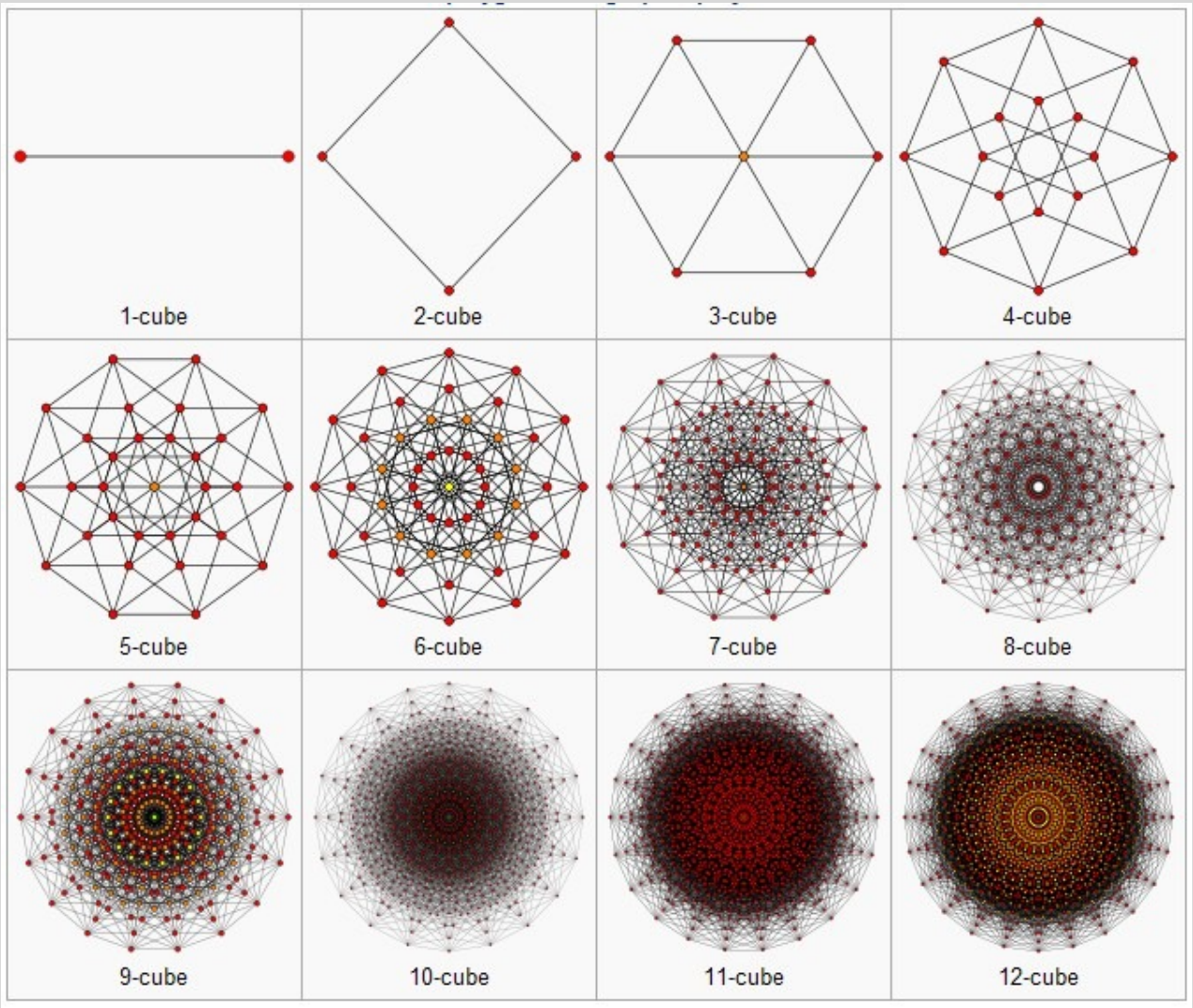
- p-kostka o boku s

$$C^p(s) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \mathbb{R}^p : x_i \in [-s/2, s/2], i = 1, \ldots, p\} = [-s/2, s/2]^p$$

$p = 1$	$p = 2$	$p = 3$	$p = 4$
			
			
			

⊕ Geometria przestrzeni wielowymiarowych

Kostka



Autodesk Reasearch https://autodeskresearch.com/files/n-cube.jpg
Pinterest https://www.pinterest.com.au/pin/774124922411178/

⊕ Geometria przestrzeni wielowymiarowych
"Objętość" kostki

- "objętość" (miara Lebesgue'a) p-kostki

$$\text{vol}(C^p(s)) = s^p \underset{p\to\infty}{\longrightarrow} \begin{cases} 0 & \text{jeśli } s < 1 \\ = 1 & \text{jeśli } s = 1 \\ \infty & \text{jeśli } s > 1 \end{cases}$$

- długość przekątnej (średnica) p-kostki

$$\text{diam}(C^p(s)) = \|(s, \ldots, s)\| = s\sqrt{p}$$

tzn. dla p-kostki jednostkowej $C^p \stackrel{\text{def}}{=} C^p(1)$

$$\text{vol}(C^p) = 1 \qquad \text{diam}(C^p) = \sqrt{p}$$

a więc **"objętość" p-kostki jednostkowej jest stała, ale jej średnica rośnie do nieskończoności z wymiarem przestrzeni** !

⊕ Geometria przestrzeni wielowymiarowych
Powłoka kostki

- ϵ-*powłoka* p-kostki:

$$S^p(s,\epsilon) = C^p(s) - C^p(s-2\epsilon)$$

  (ϵ jest grubością powłoki)
- dla dowolnego ϵ, "objętość" ϵ-powłoki p-kostki jednostkowej dąży do 1 z wymiarem przestrzeni
- tzn. ze wzrostem p **cała objętość p-kostki skupia się na jej ϵ-powłoce**
- "objętość" ϵ-powłoki p-kostki jednostkowej dla $\epsilon = c/p$

$$\text{vol}(S^p(1,c/p)) \underset{p\to\infty}{\longrightarrow} 1 - \mathrm{e}^{-2c}$$

⊕ Geometria przestrzeni wielowymiarowych

Kula

- *p-kula* $B^p(R)$: kula o promieniu R w $\mathbb{R}^p$

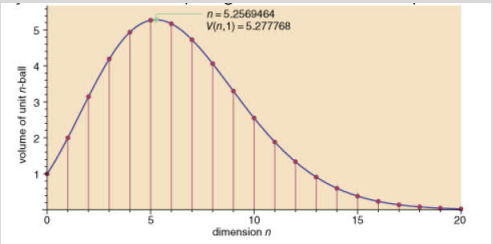
Ⓡ "objętość" p-kuli

- $\mathrm{vol}\Big(B^p(R)\Big) = R^p \mathrm{vol}\Big(B^p(1)\Big)$
- $\mathrm{vol}\Big(B^p(1)\Big) = \pi^{p/2}/\Gamma(p/2+1)$

Ⓡ $\Gamma(1/2) = \sqrt{\pi}$, $\Gamma(x+1) = x\Gamma(x)$, $\Gamma(x) \approx \sqrt{2\pi}\,\mathrm{e}^x\, x^{x-1/2}\Big(1 + 1/(12x) + \ldots\Big)$

- asymptotycznie dla dużych p, "objętość" p-kuli jednostkowej jest równa $\approx 1/\sqrt{2\pi}(2\pi\,\mathrm{e}\,/p)^{p/2}$ przy czym dla parzystych $p = 2k$

$$\mathrm{vol}\Big(B^p(1)\Big) = (\pi)^{p/2}/(p/2)!$$



zatem ze wzrostem p **"objętość" p-kuli jednostkowej początkowo rośnie a potem maleje do zera**; ekstremum dla $p = 5$

- wynik jest prawdziwy dla p-kuli o dowolnym promieniu (ale ekstremum zachodzi dla innego p).

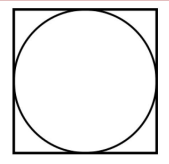
⊕ Geometria przestrzeni wielowymiarowych

Powłoka kuli

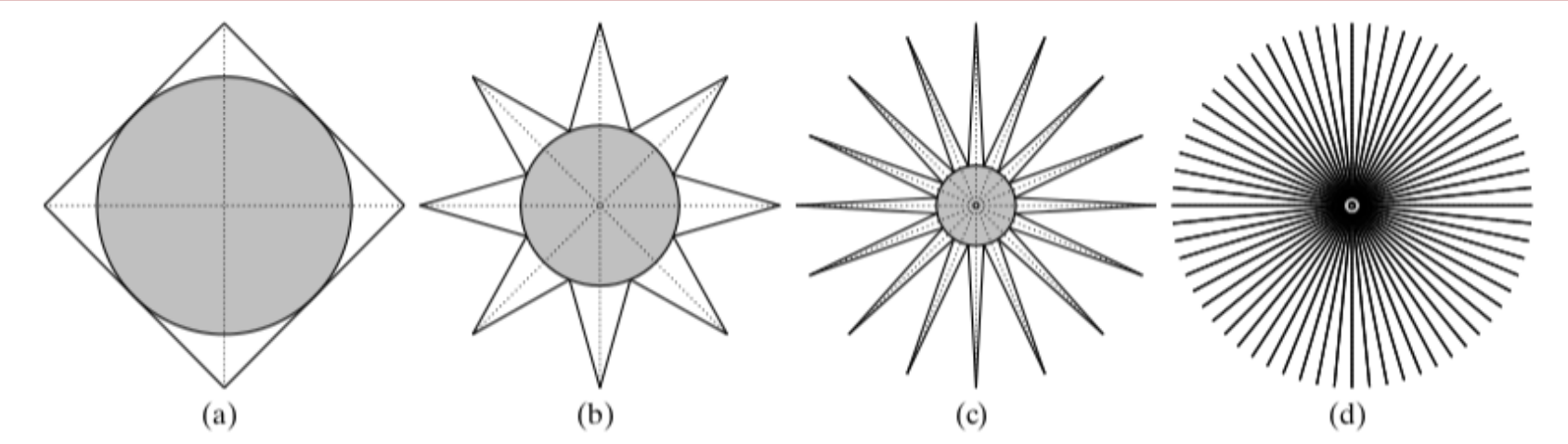
- ϵ-powłoka p-kuli
$$T^p(R,\epsilon) = B^p(R) - B^p(R-\epsilon)$$
- względna "objętość" powłoki
$$\frac{\text{vol}\, T^p(R,\epsilon)}{\text{vol}\, B^p(R)} = 1 - (1-\epsilon/R)^p \underset{\infty \to p}{\longrightarrow} 0$$
rośnie do jedności z p (ale objętość p-kuli jednostkowej zmierza do zera !)
- tzn. ze wzrostem p, **"objętość" p-kuli koncentruje się w jej ϵ-powłoce** dla dowolnego ϵ!
- ułamek objętości p-kuli pomiędzy przekrojami przez hiperpłaszczyzny $x = \sin\theta_1$ i $x = \sin\theta_2$ dla dowolnych $\theta_1 < 0 < \theta_2$, zmierza do 1 ze wzrostem p do nieskończoności, zatem ze wzrostem p, **objętość p-kuli koncentruje się wokół jej dowolnego równoleżnika**!

⊛ Geometria przestrzeni wielowymiarowych

Kula wpisana w kostkę



- **promień p-kuli wpisanej** w p-kostkę o boku s **nie zależy od p** (jest równy $s/2$)
- **"objętość" p-kuli wpisanej w p-kostkę o boku s maleje do zera** ze wzrostem p (nawet dla $s > 1$!)



Stack Exchange `https://tex.stackexchange.com/questions/207772/hypersphere-inscribed-within-hypercube`

⊛ Geometria przestrzeni wielowymiarowych

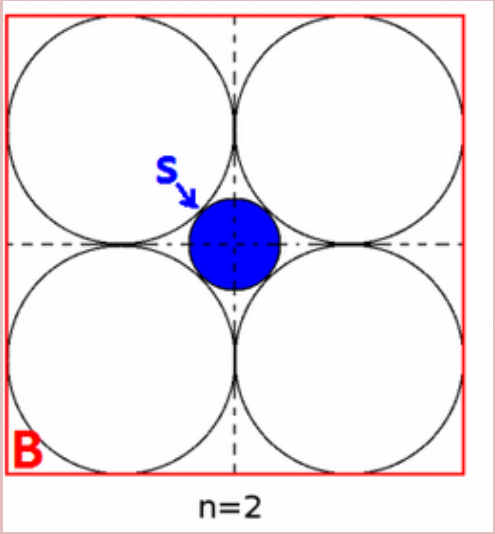
Kula styczna do kul wpisanych w kostkę

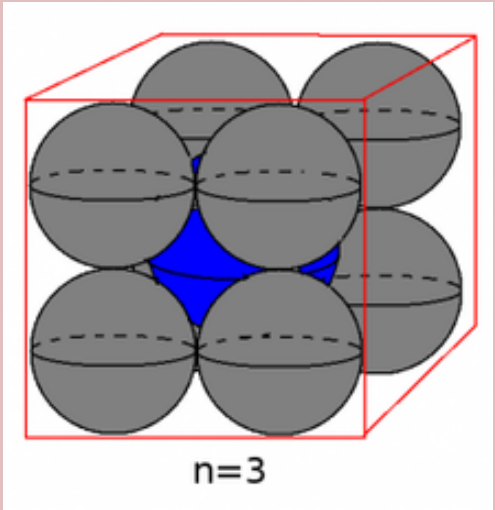
Cipra (0191) and Hayes (2011)

- 2^p p-kul jednostkowych B wpisanych w p-kostkę o boku 4 (kule o środkach w węzłach siatki o boku 2):
  - $p = 2$: 4 koła jednostkowe stycznych do boków kwadratu,
  - $p = 3$: 8 kul jednostkowych stycznych do ścian sześcianu,
  - ...
  - dowolne p: 2^p p-kul jednostkowych stycznych do ścian p-kostki o boku 4
- w środek kostki wpisana "mała" p-kula S styczna do wszystkich p-kul B

  nieintuicyjne własności (łatwe do wyprowadzenia): promień p-kuli S jest równy $\sqrt{p} - 1$

  dla $p = 4$ promień S jest jednostkowy
  dla $p = 9$ promień S jest równy 2: kula S jest **styczna do ścian w p-kostki**
  dla $p > 9$ S "mała" kula S **przecina kostkę!**

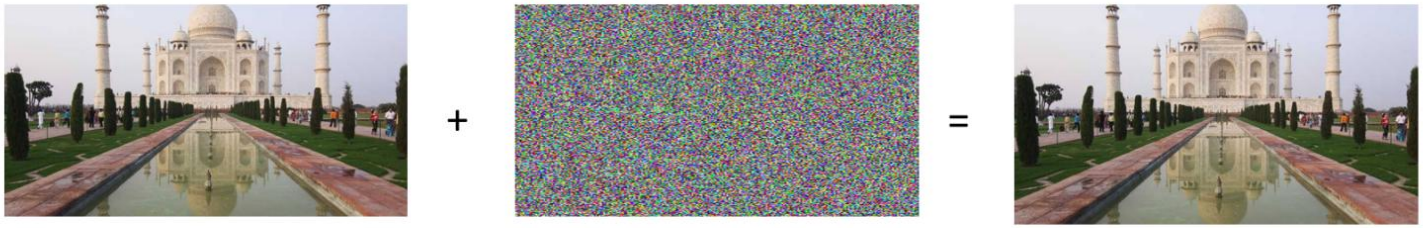




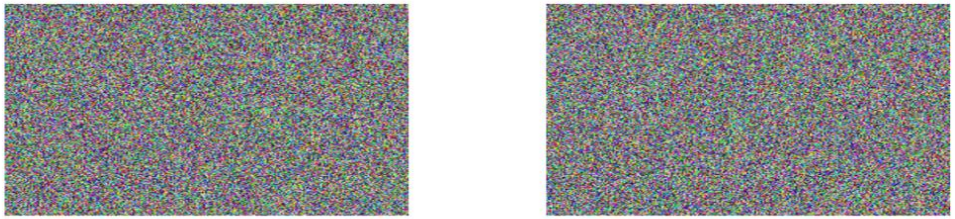
https://datascience.stackexchange.com/questions/9038/

Związek z sieciami antagonistycznymi

- własności klasyfikatorów obrazują własności przestrzeni bardzo wysoko-wymiarowych
- wzrok człowieka inaczej przetwarza obraz



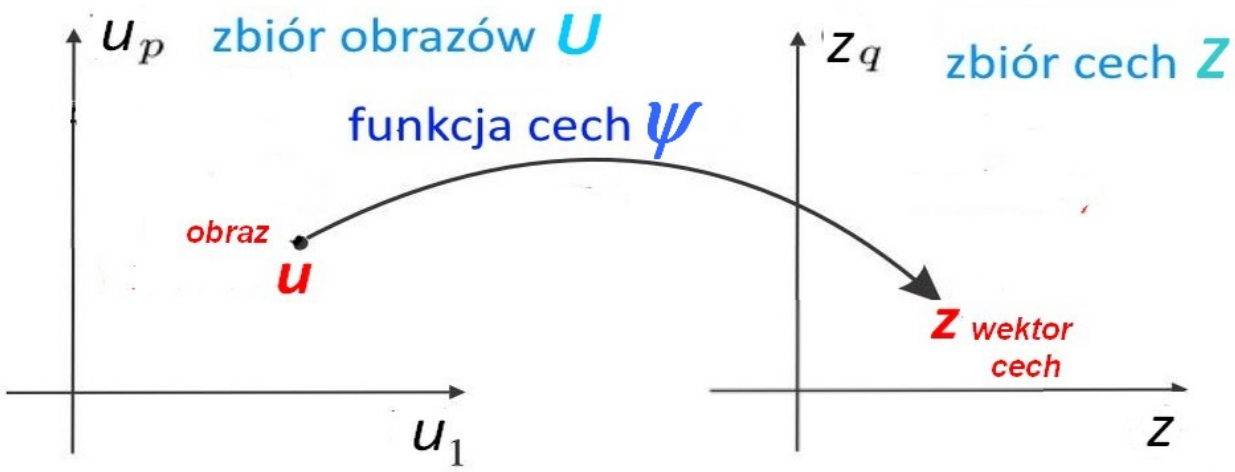
Classified as "Taj Mahal" | Small Perturbation | Classified as "Not Taj Mahal"



Fake Unrecognizable Images
Classified as "Taj Mahal"

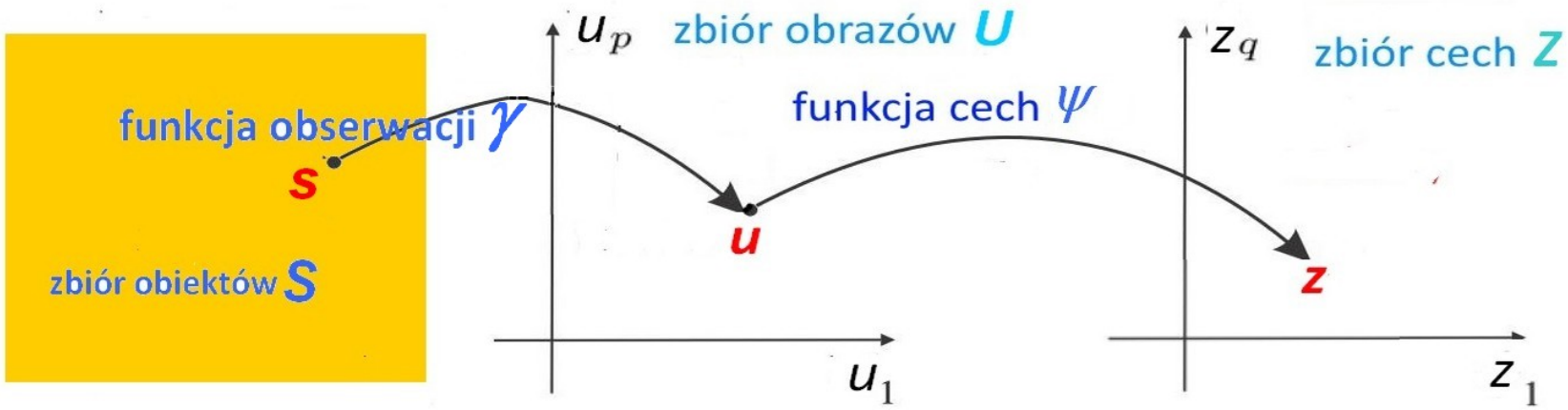
C. Szegedy et al. (2014). "Intriguing properties of neural networks". In: *Proceedings of ICLR 2014*. URL: arxiv.org/abs/1312.6199,
A. Nguyen, J. Yosinski, and J. Clune (2015). "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *Proc. of CVPR 2015*. URL: arxiv.org/pdf/1412.1897.pdf

Cechy obrazów



- *obraz* - zwykle tablica 2d lub 3d $\boldsymbol{u} \in U \subset \mathbb{R}^{p1 \times p2}$ lub wektor $\boldsymbol{u} \in U \subset \mathbb{R}^{p1p2}$
- *cechy obrazu* - zwykle wektor $\boldsymbol{z} \in \boldsymbol{Z} \subset \mathbb{R}^h$
- *funkcja cech* $\varphi : U \mapsto \boldsymbol{Z}$ wyznacza cechy danego obrazu
  - uproszczona reprezentacja obrazu
    - wystarczająco prosta ze względów obliczeniowych
    - wystarczająco złożona by wydobyć istotne własności
    - wystarczająco złożona, by stosować klasyfikacje liniowe
  - pełna reprezentacja obrazu
    - wektoryzacja obrazu

Cechy obiektów



- *obiekty* $\mathscr{S} \in \mathscr{S}$ nie są tworami matematycznymi, ale abstrakcyjnymi elementami rzeczywistości
- obrazy $\boldsymbol{u} \in U$ mogą być wynikiem obserwacji *obiektów* fizycznych (biologicznych, ekonomicznych)
- *funkcja obserwacji* $\gamma : \mathscr{S} \mapsto U$ odwzorowuje obiekty w obrazy
- funkcja $\varphi^* = \varphi\gamma$ bezpośrednio odwzorowuje obiekty w cechy
- tworzenie funkcji obserwacji: nagrania wideo, stosowanie czujników pomiarowych etc.

Klasyfikacja

Funkcja klasyfikująca i funkcje przynależności

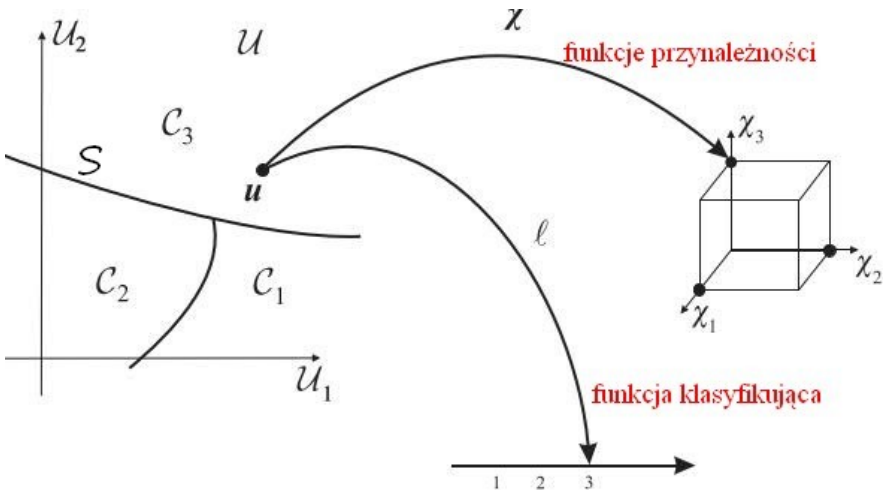
- klasa: podzbiory zbioru obrazów (zwykle rozłączne)
- *zbiór separujący* (zbiór decyzyjny): brzeg wszystkich klas

  $$\mathcal{D} = \bigcup_k \partial \mathcal{C}_k$$

  gdzie $\partial \mathcal{C}_k$: brzeg zbioru $\mathcal{C}_k$
- *funkcja klasyfikująca* ℓ

  $$\ell(\boldsymbol{u}) = k \iff \boldsymbol{u} \in \mathcal{C}_k$$
- *funkcje przynależności* χ_k

  $$\chi_k(\boldsymbol{u}) = \begin{cases} 1 \iff \boldsymbol{u} \in \mathcal{C}_k \\ 0 \iff \boldsymbol{u} \notin \mathcal{C}_k \end{cases}$$



  - suma funkcji przynależności jest stała równa 1
  - wektor funkcji przynależności $\boldsymbol{\chi}(\boldsymbol{u}) = \begin{bmatrix} \chi_1(\boldsymbol{u}) & \cdots & \chi_c(\boldsymbol{u}) \end{bmatrix}^\mathsf{T}$ zwany także funkcją *WTA* (ang. *winner takes all*) — zwycięzca bierze wszystko

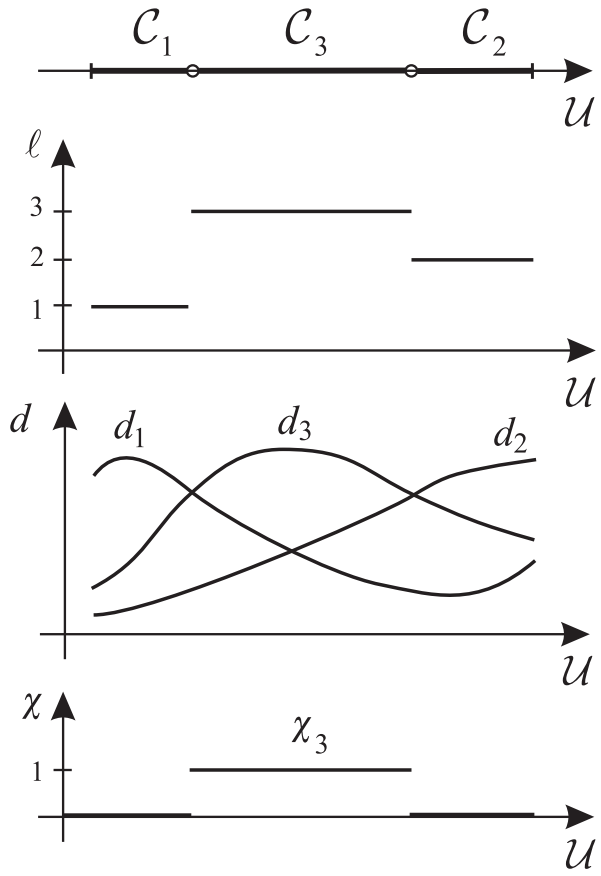
Klasyfikacja

Funkcje decyzyjne

- *funkcje decyzyjne*

$$d_k(\boldsymbol{u}) > d_j(\boldsymbol{u}) \text{ dla wszystkich } j \neq k$$
$$\iff \boldsymbol{u} \in \mathcal{C}_k$$

- funkcje decyzyjne nie są jednoznaczne i dobierane są np. z punktu widzenia wygody klasyfikacji
- przykład: funkcje przynależności χ_k



klasy,
funkcja klasyfikująca,
funkcje decyzyjne,
funkcja przynależności

Klasyfikacja

Podsumowanie pojęć

nazwa	oznaczenie	definicja
klasyfikacja	$\mathscr{C}$	$\{\mathcal{C}_1, \ldots, \mathcal{C}_c\}$
funkcja klasyfikująca	ℓ	$\ell(\boldsymbol{u}) = k \iff \boldsymbol{u} \in \mathcal{C}_k$
funkcje przynależności	χ_k	$\chi_k(\boldsymbol{u}) = \begin{cases} 1 \iff \boldsymbol{u} \in \mathcal{C}_k \\ 0 \iff \boldsymbol{u} \notin \mathcal{C}_k \end{cases}$
	$\boldsymbol{\chi}$	$\boldsymbol{\chi}(\boldsymbol{u}) = \begin{bmatrix} \chi_1(\boldsymbol{u}) \\ \vdots \\ \chi_c(\boldsymbol{u}) \end{bmatrix}$
funkcje decyzyjne	d_k	$d_k(\boldsymbol{u}) > d_j(\boldsymbol{u})$ dla wszystkich $j \neq k$ $\iff \boldsymbol{u} \in \mathcal{C}_k$

Klasyfikatory uczące się

- *klasyfikator*: algorytm lub urządzenie realizujące klasyfikację
- klasyfikatory są definiowane podobnie jak klasyfikacje (funkcja klasyfikująca, funkcja przynależności, funkcja decyzyjna)
- jeśli klasyfikator nie jest znany, należy go wybrać w procesie uczenia z zadanej rodziny klasyfikatorów
- najczęściej klasyfikator jest znany z dokładnością do skończenie wielu parametrów i wtedy jego parametry trzeba określić w procesie uczenia
- *zbiór treningowy* (*zbiór uczący*): zbiór danych (obserwacji) wykorzystywanych do doboru klasyfikatora
- dobór klasyfikatora polega zwykle na minimalizacji "kosztu" niewłaściwej klasyfikacji

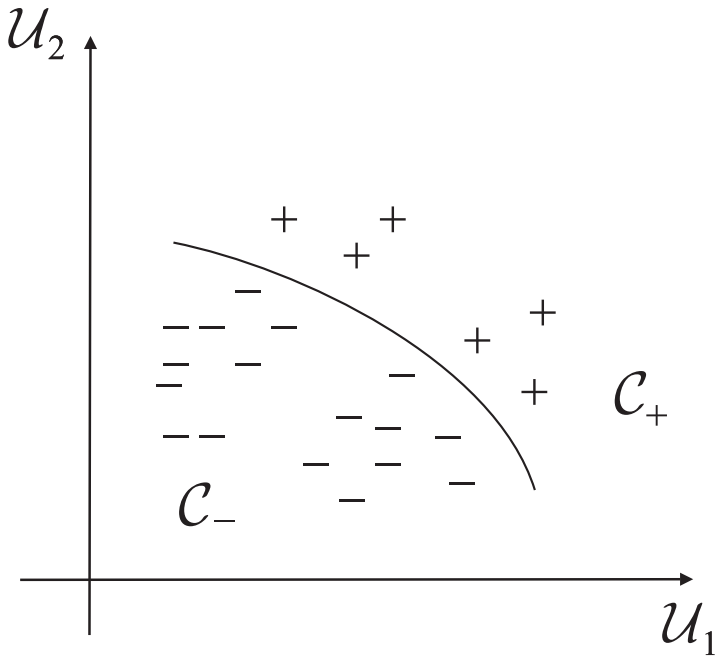
⊕ *metoda klasyfikacji*: sposób konstrukcji klasyfikatora na podstawie zbioru treningowego

Klasyfikatory: Dostępna informacja

Problemy nadzorowane

- *problemy nadzorowane*
  - liczba klas c **jest znana**
  - przykład trenujący $(\boldsymbol{u}_n, \ell(\boldsymbol{u}_n))$
  - zbiór trenujący

$$\mathcal{S} = \Big\{ \big(\boldsymbol{u}_n, \ell(\boldsymbol{u}_n)\big),\ n = 1, \ldots, N \Big\}$$

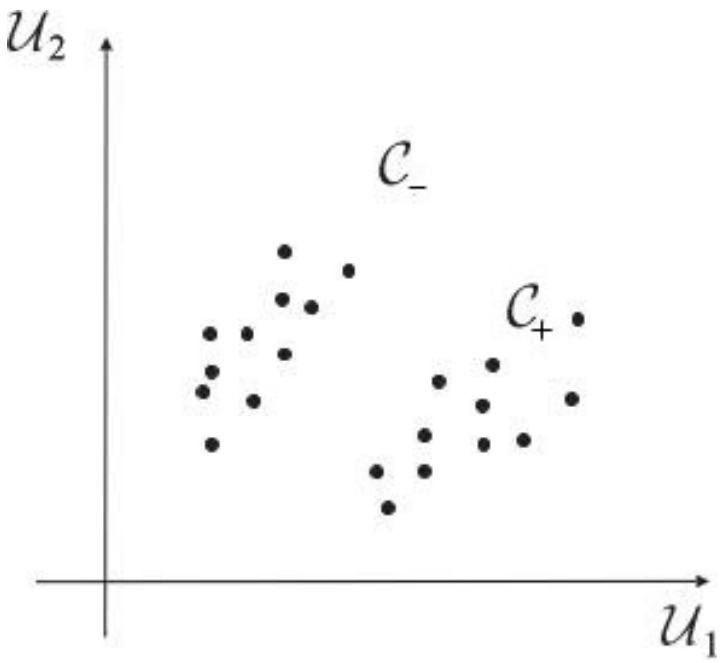


Klasyfikatory: Dostępna informacja

Problemy nienadzorowane

- problemy *nienadzorowane*, analiza skupień
  - liczba klas c **znana lub nie**
  - przykład trenujący $\boldsymbol{u}_n$ (**klasa nieznana**)
  - zbiór trenujący

$$\mathcal{S} = \{\boldsymbol{u}_n,\ n = 1, \ldots, N\}$$



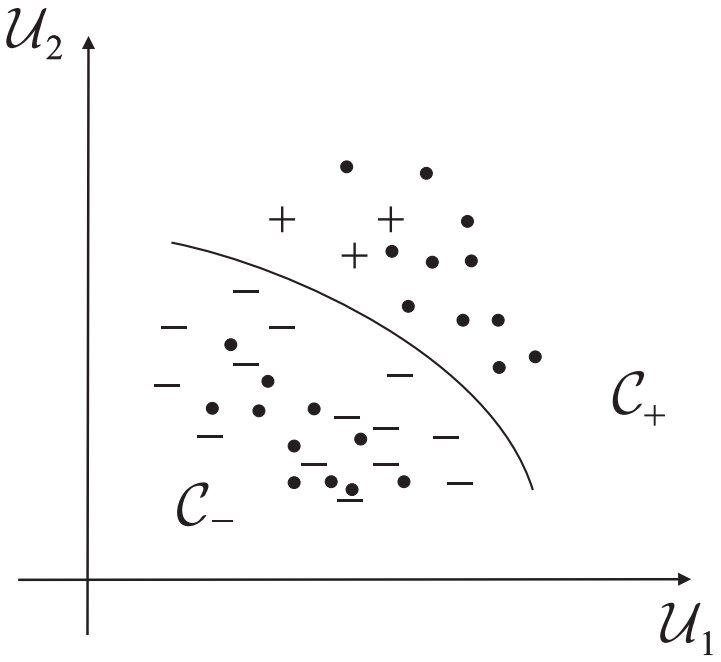
Klasyfikatory: Dostępna informacja

Problemy częściowo nadzorowane

- *problemy częściowo nadzorowane* (transdukcyjne)
  - liczba klas c **znana**
  - przykład trenujący $(\boldsymbol{u}_n, \ell(\boldsymbol{u}_n))$ lub tylko $\boldsymbol{u}_n$
  - zbiór trenujący

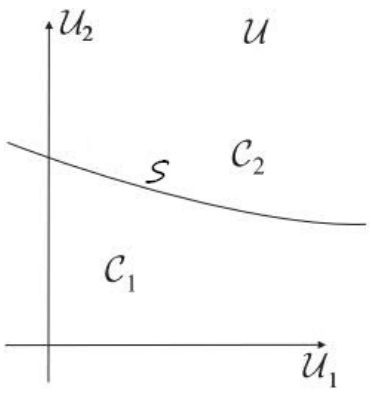
$$\mathcal{S} = \Big\{ \underbrace{\big(\boldsymbol{u}_n, \ell(\boldsymbol{u}_n)\big)}_{\text{nadzorowane}}, n = 1, \ldots, N' \Big\}$$

$$\cup \Big\{ \underbrace{\boldsymbol{u}_m}_{\text{nienadzorowane}}, m = N'+1, \ldots, N \Big\}$$



4.1 Problem klasyfikacji obrazów
4.2 Klasyfikacja na dwie klasy
4.3 Klasyfikacja liniowa
4.4 Podejście probabilistyczne

Klasyfikacja na dwie klasy



- *klasyfikacja na dwie klasy* (klasyfikacja binarna) $c = 2$

  etykiety klas $1, -1$ lub $1, 0$, lub $+, -$
- funkcja klasyfikująca: przyjmuje dwie wartości, np. -1, 1
- funkcje przynależności: wystarczy znajomość jednej z nich $\chi_-(\boldsymbol{u}) = 1 - \chi_+(\boldsymbol{u})$
- funkcje decyzyjne: wystarczy znajomość jednej funkcji $d(\boldsymbol{u}) = d_+(\boldsymbol{u}) - d_-(\boldsymbol{u})$

$$d(\boldsymbol{u}) > 0 \iff \boldsymbol{u} \in \mathcal{C}_+$$

Klasyfikacja a funkcje logiczne

- klasyfikacje binarne obrazów binarnych odpowiadają funkcjom logicznym

język klasyfikacji	język logiki
piksel binarny	zmienna logiczna
-1 (lub 0)	FALSE
1	TRUE
liczba pikseli p	liczba zmiennych logicznych
obraz	wektor zmiennych logicznych
klasy	wartości zmiennych logicznych
$\mathcal{C}_{-1}$ (lub $\mathcal{C}_0$)	FALSE
$\mathcal{C}_1$	TRUE
klasyfikacja wzorców w $\mathbb{R}^p$	funkcja logiczna o p argumentach

Klasyfikacja binarna

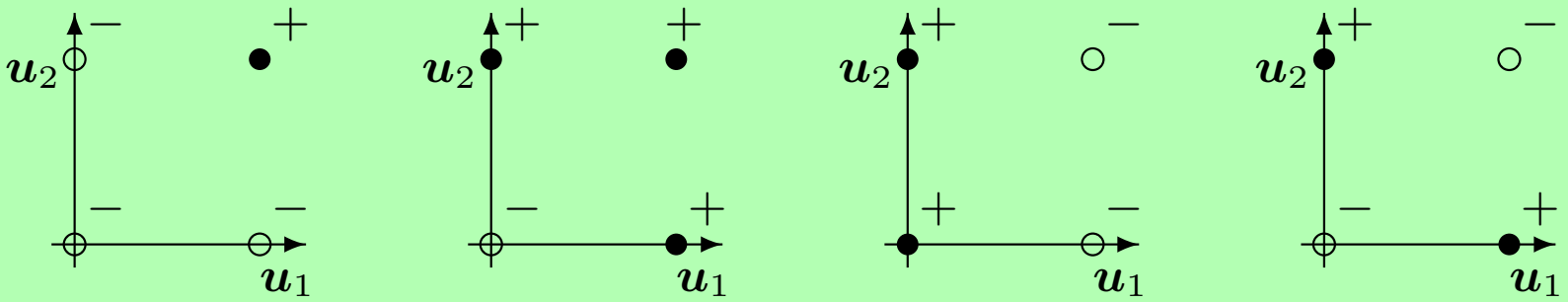
Przykład

Ⓔ Funkcje binarne 2 zmiennych binarnych

$U = \{0,1\}^2 = \big\{(0,0),(0,1),(1,0),(1,1)\big\}$

$n = 4$ obrazy, każdy o 2 pikselach binarnych

klasyfikacje odpowiadają funkcjom logicznym 2 zmiennych



u_1 AND u_2 $\quad$ u_1 OR u_2 $\quad$ NOT u_1 $\quad$ u_1 XOR u_2

4.1 Problem klasyfikacji obrazów
4.2 Klasyfikacja na dwie klasy
4.3 Klasyfikacja liniowa
4.4 Podejście probabilistyczne

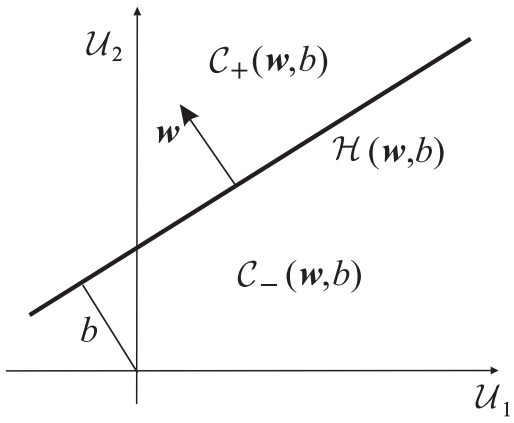
Klasyfikacja liniowa

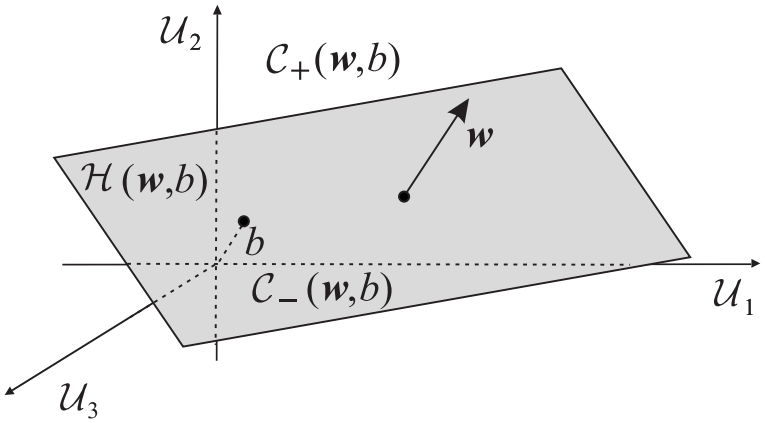
Hiperpłaszczyzna

- *hiperpłaszczyzna zorientowana*: hiperpłaszczyzna z jej wektorem normalnym

$$\mathcal{H}(\boldsymbol{v}, \beta) = \{\boldsymbol{u} \in \mathbb{R}^p : \underset{\substack{\uparrow \\ \text{wektor normalny}}}{\boldsymbol{v}}{}^{\mathsf{T}} \boldsymbol{u} + \underset{\substack{\uparrow \\ \text{przesunięcie} \\ \text{(odległość od } \mathbf{0})}}{\beta} = 0\}$$

- *półprzestrzeń dodatnia* względem hiperpłaszczyzny zorientowanej $\mathcal{H}(\boldsymbol{w}, b)$: $\mathcal{C}_+(\boldsymbol{w}, b) = \{\boldsymbol{u} \in \mathbb{R}^p : \boldsymbol{w}^{\mathsf{T}}\boldsymbol{u} + b > 0\}$; *półprzestrzeń ujemna*: $\mathcal{C}_-(\boldsymbol{w}, b)$



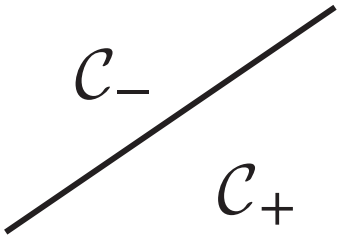


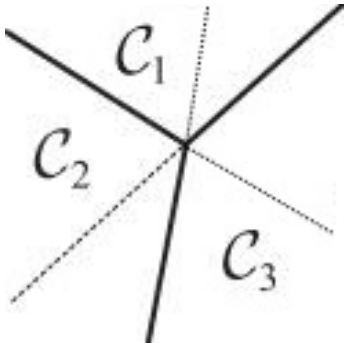
- $\mathcal{H}(\alpha\boldsymbol{w}, \alpha b) = \mathcal{H}(\boldsymbol{w}, b)$ dla dowolnego $\alpha > 0$

Klasyfikacja liniowa

Definicja ***Klasyfikacja liniowa***

Jeśli każde dwie klasy można separować hiperpłaszczyzną to *klasyfikację nazywamy liniową*





Funkcje separujące dla klasyfikacji liniowej

- do określenia klasyfikacji liniowej obrazów w $\mathbb{R}^p$ na dwie klasy wystarczy znajomość wektora $\boldsymbol{w} \in \mathbb{R}^p$ i przesunięcia $b \in \mathbb{R}$
- do określenia klasyfikacji liniowej obrazów w $\mathbb{R}^p$ na c klas w wystarczy znajomość macierzy c wektorów hiperpłaszczyzn i wektora c skalarnych przesunięć

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^\mathsf{T} \\ \vdots \\ \boldsymbol{w}_c^T \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_c \end{bmatrix}$$

- *afiniczna funkcja decyzyjna* $\quad d_k(\boldsymbol{u}) = b_k + \boldsymbol{w}_k^\mathsf{T}\boldsymbol{u}, \;\; k = 1, \ldots, c$

$$\boldsymbol{d}(\boldsymbol{u}) = \begin{bmatrix} b_1 + w_1\boldsymbol{u} \\ \vdots \\ b_c + w_c\boldsymbol{u} \end{bmatrix} = \boldsymbol{b} + \boldsymbol{W}\,\boldsymbol{u}$$

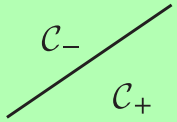
(!) Funkcje decyzyjne dla klasyfikacji liniowej

klasy są liniowo separowalne $\Leftrightarrow$ istnieją afiniczne funkcje decyzyjne

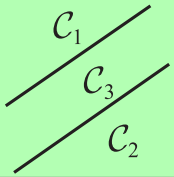
Klasyfikacja liniowa

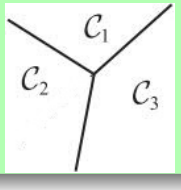
Przykłady

Ⓔ Klasyfikacja liniowa: dwie klasy w $\mathbb{R}^2$

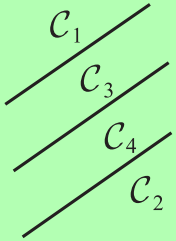


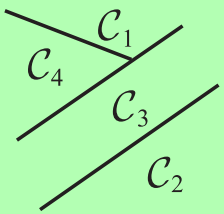
Ⓔ Klasyfikacja liniowa: trzy klasy w $\mathbb{R}^2$

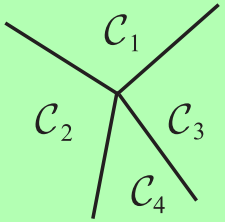


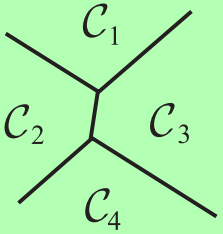


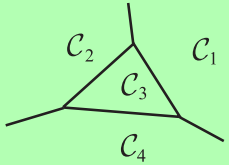
Ⓔ Klasyfikacja liniowa: cztery klasy w $\mathbb{R}^2$







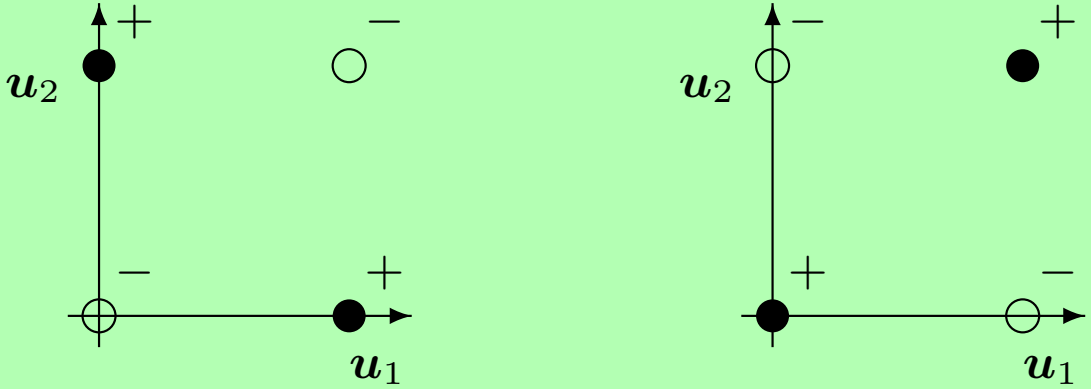




Proste przykłady klasyfikacji nieliniowych

Ⓔ Klasyfikacja nieliniowa: 3 obrazy w $\mathbb{R}$, 2 klasy

Ⓔ Klasyfikacja nieliniowa: 4 obrazy w $\mathbb{R}^2$, 2 klasy



Przykład: Klasyfikacja liniowa na dwie klasy

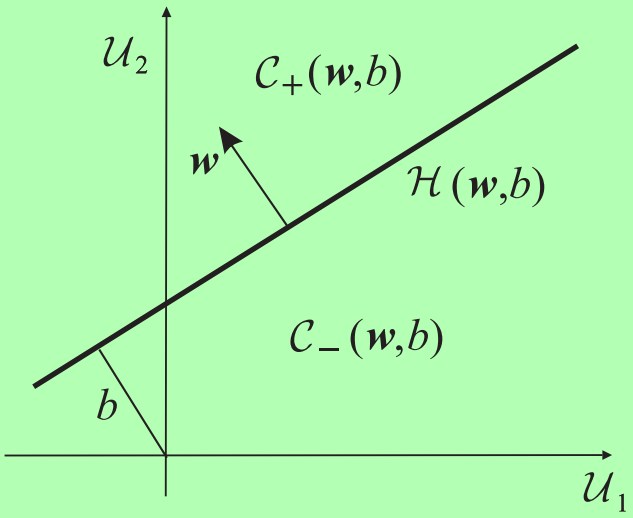
Ⓔ Przykład na płaszczyźnie

- klasyfikator liniowy

$$\boldsymbol{d}(\boldsymbol{u}) = \begin{bmatrix} b_+ + \boldsymbol{w}_+^\mathsf{T}\boldsymbol{u} \\ b_- + \boldsymbol{w}_-^\mathsf{T}\boldsymbol{u} \end{bmatrix}$$

$$\chi(\boldsymbol{u}) = \begin{cases} -1 & \text{jeżeli } \boldsymbol{w}^\mathsf{T}\boldsymbol{u} + b < 0 \\ 1 & \text{jeżeli } \boldsymbol{w}^\mathsf{T}\boldsymbol{u} + b > 0 \end{cases}$$

gdzie $\boldsymbol{w} = \boldsymbol{w}_+ - \boldsymbol{w}_-$, $b = b_+ - b_-$



- zmodyfikowana funkcja decyzyjna: porównania z zerem

$$d(\boldsymbol{u}) = b + \boldsymbol{w}^\mathsf{T}\boldsymbol{u}$$

- hiperpłaszczyzna decyzyjna $\mathcal{H}(\boldsymbol{w}, b) = \{\boldsymbol{u} : d(\boldsymbol{u}) = 0\}$
- półprzestrzeń dodatnia $\mathcal{H}_+(\boldsymbol{w}, b) = \{\boldsymbol{u} : d(\boldsymbol{u}) > 0\}$
  (podobnie półprzestrzeń ujemna)

Liczba klasyfikacji liniowych na dwie klasy

- obraz złożony z p pikseli = obraz w $\mathbb{R}^p$, zatem hiperpłaszyzna decyzyjna ma wówczas $p+1$ *wolnych parametrów*, tzn. parametrów do określenia w ramach klasyfikacji
- dla n obrazów o p pikselach (obrazów w pozycji ogólnej), liczba klasyfikacji liniowych na dwie klasy jest równa

$$L(n,p) = \begin{cases} 2^n & \text{dla } n \leqslant p+1 \\ 2\sum_{i=0}^{p}\binom{n-1}{i} & \text{dla } n > p+1 \end{cases} \quad \text{(„więcej obrazów niż wolnych parametrów")}$$

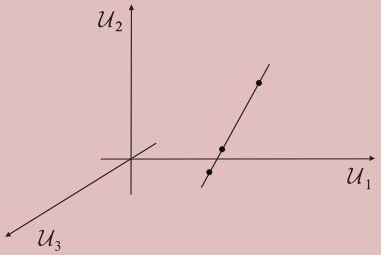
- dla obrazów w $\mathbb{R}^p$
  - dla $n \leqslant p+1$, liczba klasyfikacji liniowych **rośnie wykładniczo z liczbą obrazów** n
  - dla $n > p+1$, liczba klasyfikacji liniowych **rośnie potęgowo z liczbą obrazów** n (jak n^p) (**wolniej!**)
- "przełomowa" liczba obrazów $n_{\text{vc}} = p+1$ jest tu równa liczbie „wolnych parametrów" określających klasyfikację liniową

⊛ Liczba binarnych klasyfikacji liniowych

Położenie ogólne a położenie specjalne obrazów

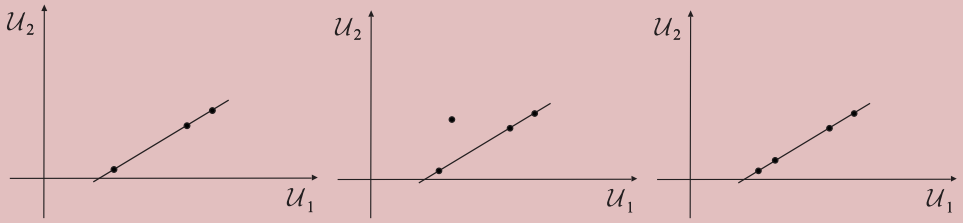
- n obrazów w $\mathbb{R}^p$ ma *położenie specjalne* (*położenie zdegenerowane*)
  - dla $3 \leqslant n \leqslant p$ (**"mało obrazów"**):
    jeżeli wszystkie obrazy należą do pewnej $(n-2)$-wymiarowej podprzestrzeni afinicznej

p=3

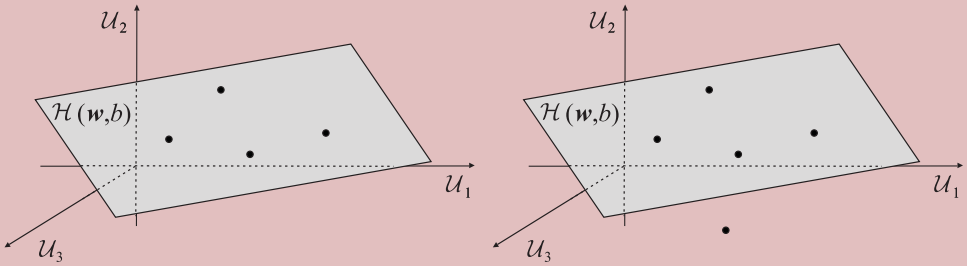


  - dla $n > p$ (**"dużo obrazów"**):
    jeżeli $p+1$ obrazów należy do pewnej hiperpłaszczyzny, tzn. $p-1$-wymiarowej podprzestrzeni afinicznej

p=2



p=3



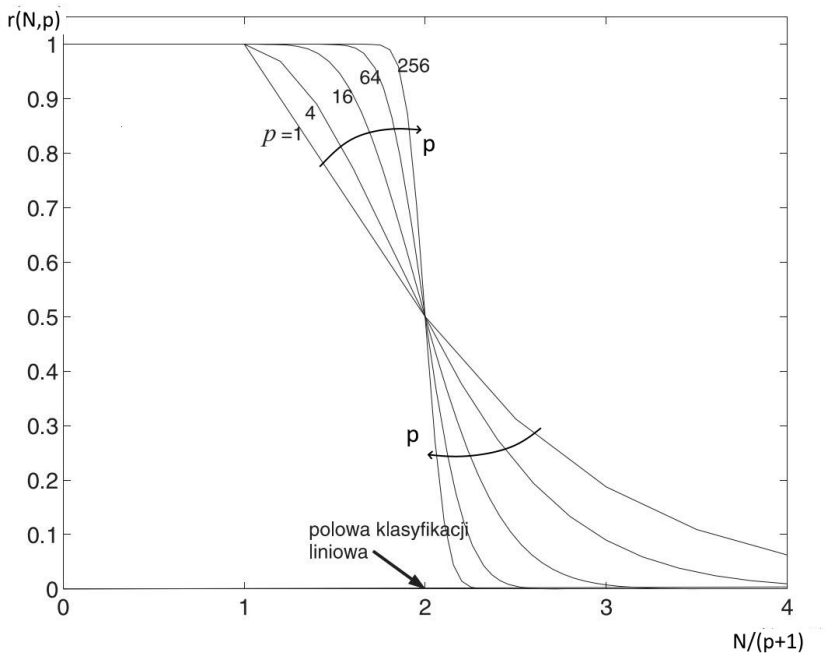
- w przeciwnym przypadku obrazy mają *położenie ogólne*

Liczba klasyfikacji liniowych na dwie klasy

Pojemność klasyfikacji liniowej

- ułamek klasyfikacji liniowych wśród wszystkich klasyfikacji na dwie klasy $r(n,p)$ (n-liczba obrazów, p- rozmiar obrazów)

$$r(n,p) = \frac{L(n,p)}{B(n)} = \begin{cases} 1 & \text{dla } \frac{n}{p+1} \leqslant 1 \\ 2^{1-n} \sum_{i=0}^{p} \binom{n-1}{i} & \text{dla } \frac{n}{p+1} \geqslant 1 \\ (0.5 & \text{dla } \frac{n}{p+1} = 2) \end{cases}$$



nieliniowa SVM

- **do $p+1$ obrazów w $\mathbb{R}^p$: wszystkie klasyfikacje są liniowe**
- dla $2(p+1)$ obrazów w $\mathbb{R}^p$: co najwyżej połowa klasyfikacji jest liniowa

Liczba klasyfikacji liniowych na dwie klasy

Ilustracja

Ⓔ Liczba klasyfikacji i liczba klasyfikacji liniowych

liczba cech lub pikseli p	liczba obrazów n						
	2	3	4	5	6	7	8
	liczba klasyfikacji $B(n) = 2^n$						
	4	8	16	32	64	128	256
	liczba klasyfikacji liniowych $L(n,p)$						
1	4	6	8	10	12	14	16
2	4	8	14	22	32	44	58
3	4	8	16	30	52	84	128
4	4	8	16	32	62	114	198
5	4	8	16	32	64	126	240
	lewy + górny lewy $L(n,p) = L(n-1,p) + L(n-1,p-1)$						

wszystkie klasyfikacje (binarne) są liniowe

połowa klasyfikacji jest liniowa

Liczba klasyfikacji liniowych

„Przekleństwo wymiarowości”

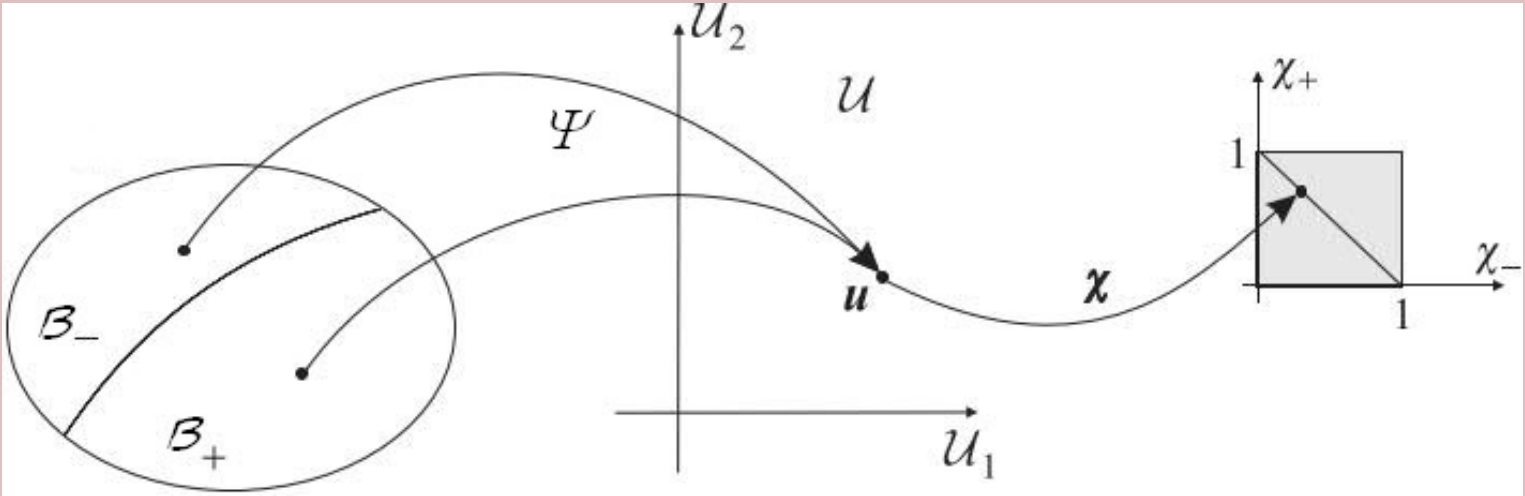
Ⓔ Liczba klasyfikacji obrazów binarnych

liczba pikseli p	maks. liczba obrazów binarnych $n_{\max} = 2^p$	liczba klasyfikacji $B(n_{\max}) = 2^p$	liczba klasyfikacji **liniowych** $L(n_{\max}, p)$	proporcja klasyfikacji liniowych $r = L/B$
2	4	16	14	0.875
3	8	256	128	0.500
4	16	65536	3882	0.059
5	32	$4.3\ 10^{9}$	412736	$9.6\ 10^{-5}$
8	256	10^{77}	10^{15}	10^{-63}
16	65536	10^{19728}	10^{64}	10^{-19664}

- prawdopodobieństwo, że dla 32 obrazów o $p = 5$ cechach binarnych, losowo wybrana klasyfikacja p obrazów jest liniowa jest znikomo mała, rzędu 0.01 %

4.1 Problem klasyfikacji obrazów
4.2 Klasyfikacja na dwie klasy
4.3 Klasyfikacja liniowa
4.4 Podejście probabilistyczne

⊛ Dziedziczenie probabilistyczne



- niejednoznaczność $\mathcal{C}_i^0 \cap \mathcal{C}_j^0 \neq \emptyset$
- rozkład obiektów $\mathscr{P}$
- probabilistyczna funkcja przynależności do klas w U: $\chi_k(\boldsymbol{u}) = \mathscr{P}\{f \in m_k\}$
- probabilistyczna funkcja przynależności $\boldsymbol{\chi}$ przyjmuje wartości z hiperpłaszczyzny $\sum_k \chi_k(\boldsymbol{u}) = 1$

⊛ Klasyfikatory probabilistyczne

Dyskretne rozkłady a posteriori

- skończony zbiór obrazów $U = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N\}$
- skończony zbiór klas $\mathcal{C}_k, k = 1, \ldots, c$
- jeśli $\boldsymbol{\gamma}(\mathscr{S}_i) \cap \boldsymbol{\gamma}(\mathscr{S}_j) \neq \emptyset$ to klasyfikacja obrazów nie jest jednoznaczna
- znany jest *rozkład klas a priori* $\pi_k \stackrel{\text{def}}{=} \mathscr{P}\{\boldsymbol{u} \in \mathcal{C}_k\},\ k = 1, \ldots, c$
- dla każdej klasy znany jest *rozkład wewnątrz-klasowy obrazów*

$$p_{m\,|\,k} \stackrel{\text{def}}{=} \mathscr{P}(\boldsymbol{u} = \boldsymbol{u}_m \,|\, \boldsymbol{u} \in \mathcal{C}_k), \quad m = 1, \ldots, N$$

- dla danego obrazu $\boldsymbol{u} = \boldsymbol{u}_m$ *można wyznaczyć rozkład klas a posteriori*

$$\pi_{k\,|\,m} \stackrel{\text{def}}{=} \mathscr{P}\{\boldsymbol{u} \in \mathcal{C}_k \,|\, \boldsymbol{u} = \boldsymbol{u}_m\} \underset{\text{reguła Bayesa}}{=} \frac{\pi_k\, p_{m\,|\,k}}{p_m} = \frac{\pi_k\, p_{m\,|\,k}}{\sum_{i=1}^{c} \pi_i\, p_{m\,|\,i}}$$

⊛ Klasyfikatory probabilistyczne

Ciągłe rozkłady obrazów

- nieskończony zbiór obrazów $U = \{\boldsymbol{u}\}$
- skończony zbiór klas $\mathcal{C}_k, k = 1, \ldots, c$
- dla każdej klasy $\mathcal{C}_k, k = 1, \ldots, c$, znany jest *rozkład wewnątrz-klasowy obrazów*

$$p_{\nu \mid k} \stackrel{\text{def}}{=} f_{\boldsymbol{u} \mid k}(\nu), \quad \nu \in U$$

- dla danego obrazu $\boldsymbol{u} = \nu$ *można wyznaczyć rozkład klas a posteriori*

$$\pi_{k \mid \nu} \stackrel{\text{def}}{=} \mathscr{P}\{\boldsymbol{u} \in \mathcal{C}_k \mid \boldsymbol{u} = \nu\} \underset{\textit{reguła Bayesa}}{=} \frac{\pi_k \, p_{\nu \mid k}}{p_\nu} = \frac{\pi_k \, p_{\nu \mid k}}{\sum_{i=1}^{c} \pi_i \, p_{\nu \mid i}}$$

⊛ Klasyfikatory probabilistyczne

Klasyfikator Bayesowski

- reguła: przyporządkować obraz do klasy obiektów o **najwyższym prawdopodobieństwie a posteriori**
- funkcja decyzyjna dla *klasyfikatora bayesowskiego*

$$\ell_k(\boldsymbol{u}_m) = \pi_{k\,|\,m} = \frac{\pi_k\, p_{m\,|\,k}}{\sum_{i=1}^{c} \pi_i\, f_{m\,|\,i}}$$

- równoważna funkcja decyzyjna: $\ell_k(\boldsymbol{u}_j) = \pi_k\, p_{j\,|\,k}$
- dla rozkładów ciągłych:

$$\ell_k(\boldsymbol{u}) = \pi_{k\,|\,\boldsymbol{u}} = \frac{\pi_k\, f_{\boldsymbol{u}\,|\,k}(\boldsymbol{u})}{\sum_{i=1}^{c} \pi_i\, f_{\boldsymbol{u}\,|\,i}(\boldsymbol{u})}$$

lub, równoważnie $\ell_k(\boldsymbol{u}_j) = \pi_k\, f_{\boldsymbol{u}\,|\,k}(\boldsymbol{u})$

Literatura

Materiały źródłowe

Duda, R. O., P. E. Hart, and D. G. Stork (2001). *Pattern Recognition, Second Edition*. New York: J. Wiley & Sons.

Fomin, W. (1976). *Matematiczeskaja teorija obuczajemych opoznajuszczich sistem*. Leningrad: Leningrad University Press.

Materiały pomocnicze

Cipra, B. (191). "Here's looking at Euclid". In: *What's Happening in the Mathematical Sciences*. Ed. by B. Cipra and D. Mackenzie. Vol. 1. Providence: American Mathematical Society.

Hayes, B. (2011). "An adventure in the nth dimension". In: *American Scientist*.

Nguyen, A., J. Yosinski, and J. Clune (2015). "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *Proc. of CVPR 2015*. URL: arxiv.org/pdf/1412.1897.pdf.

Nilsson, N. J. (1965). *Learning Machines - Foundations of Trainable Pattern Classifying Systems*. New York: McGraw-Hill.

Szegedy, C. et al. (2014). "Intriguing properties of neural networks". In: *Proceedings of ICLR 2014*. URL: arxiv.org/abs/1312.6199.

Temat 5: Perceptron Rosenblatta

5.1 Struktura perceptronu Rosenblatta
5.2 Uczenie perceptronu Rosenblatta
5.3 Budowa i uczenie maszyny liniowej

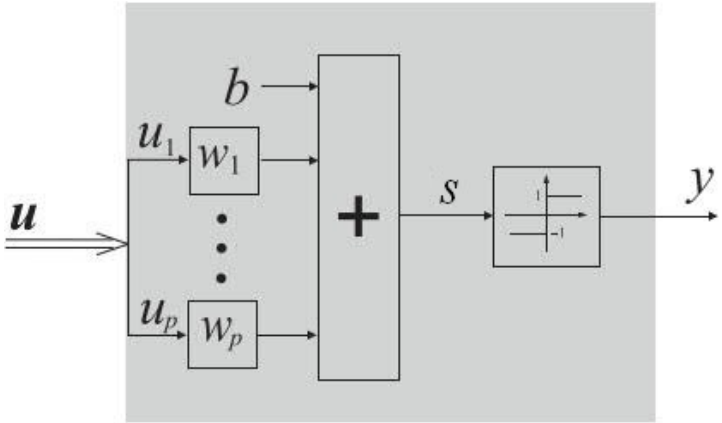
Krótka historia perceptronu Rosenblatta

- Rosenblatt (1958) (Cornell U., Ithaka, N.Y.)
  - "uniwersalna maszyna do rozpoznawania i klasyfikacji wzorców"
  - "sztuczny mózg", histeryczna reakcja mediów
- sprzęt: perceptron "Mark 1" (1960)

  matryca 20×20 fotodiod, każda połączona losowo z 40 neuronami warstwy ukrytej
  512 neuronów warstwy ukrytej
  8 neuronów wyjściowych, potencjometry wag napędzane silnikami
- Minsky and Papert (1969) (MIT, Cambridge, MA)
  - "...większość tych publikacji nie ma naukowej treści"
  - "...entuzjazm nie skażony wiedzą"
  - perceptron jednowarstwowy – tylko klasyfikacja liniowa
- pozostało do dzisiaj:
  - 'koneksjonizm'
  - architektura warstwy
  - obliczenia równoległe
  - uczenie przez adaptację wag

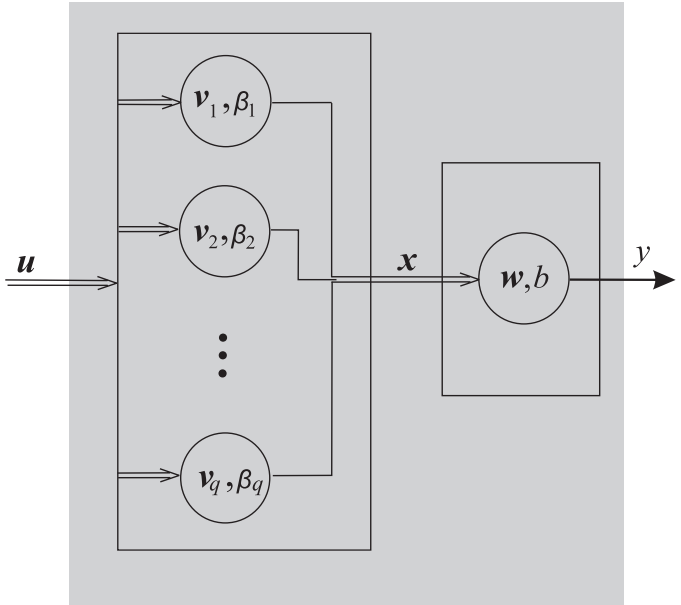
5.1 Struktura perceptronu Rosenblatta
5.2 Uczenie perceptronu Rosenblatta
5.3 Budowa i uczenie maszyny liniowej

Struktura perceptronu Rosenblatta



jednowarstwowy perceptron Rosenblatta

- **pojedynczy neuron**
- wejścia: obrazy $\boldsymbol{u} \in \mathbb{R}^p$ (*warstwa p sensorów*)
- neuron dwustanowy (*aktywny - nieaktywny*)
- typowa funkcja aktywacji: sign
- z – *liniowa aktywacja neuronu*
- **wagi adaptacyjne**



dwuwarstwowy perceptron Rosenblatta

- wejścia: obrazy $\boldsymbol{u} \in \mathbb{R}^p$ (*warstwa p sensorów*)
- warstwa ukryta (*asocjacyjna*): **stałe wagi**
- warstwa wyjściowa (*reakcji*): **wagi adaptacyjne**

Perceptron Rosenblatta jako klasyfikator

- zadana klasyfikacja na dwie klasy $\mathcal{C}_+$ i $\mathcal{C}_-$

$$y^o = \begin{cases} -1 & \text{jeżeli } \boldsymbol{u} \in \mathcal{C}_- \\ 1 & \text{jeżeli } \boldsymbol{u} \in \mathcal{C}_+ \end{cases}$$

Perceptron

$$y = \begin{cases} -1 & \text{jeżeli } \boldsymbol{w}^\mathsf{T}\boldsymbol{u} + b < 0 \\ 1 & \text{jeżeli } \boldsymbol{w}^\mathsf{T}\boldsymbol{u} + b > 0 \end{cases} = \operatorname{sign}(\boldsymbol{w}^\mathsf{T}\boldsymbol{u} + b) = d(\boldsymbol{u})$$

- wyjście perceptronu jest pewną funkcją klasyfikująca, tzn. **perceptron jest klasyfikatorem binarnym**
- **funkcja liniowa** $\boldsymbol{w}^\mathsf{T}\boldsymbol{u} + b$ jest **funkcją decyzyjną** perceptronu jako klasyfikatora
- klasyfikacja obrazu $\boldsymbol{u}$ przez perceptron jest poprawna (tzn. zgodna z pożądaną) $\iff$

$$y = y^o \quad \text{tzn.} \quad y^o\, y = 1 \quad \text{tzn.} \quad \underbrace{y^o}_{\substack{\text{klasa}\\ \text{poprawna}}} \underbrace{(\boldsymbol{w}^\mathsf{T}\boldsymbol{u} + b)}_{\substack{\text{aktywacja}\\ \text{perceptronu}}} > 0$$

Zagadnienie reprezentacji

- *zagadnienie reprezentacji*: czy rodzina funkcji reprezentowana przez perceptron Rosenblatta jest wystarczająco bogata, by dowolnie klasyfikować dowolne zbiory obrazów ?

(!) Reprezentacja funkcji dla jednowarstwowego perceptronu Rosenblatta

Dowolna klasyfikacja liniowa skończonego zbioru obrazów może być zrealizowana przez pewien **jednowarstwowy** perceptron Rosenblatta

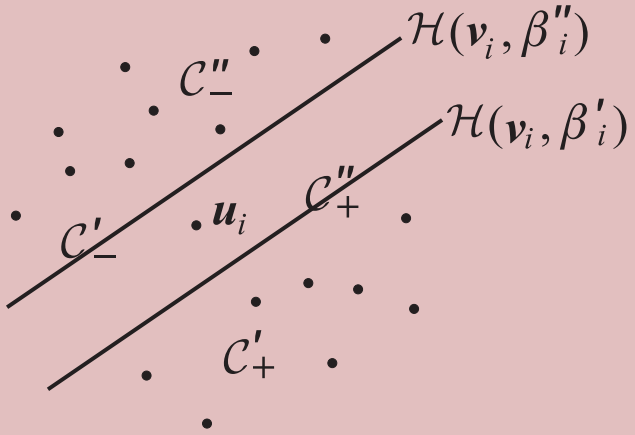
(!) Reprezentacja funkcji dla dwuwarstwowego perceptronu Rosenblatta

Dowolna klasyfikacja skończonego zbioru obrazów może być zrealizowana przez pewien **dwuwarstwowy** perceptron Rosenblatta o odpowiednio dużej warstwie ukrytej

- **problem**: w jaki sposób dobrać właściwe parametry perceptronu Rosenblatta?

⊛ Własności reprezentacji dwuwarstwowego perceptronu Rosenblatta

- *zbiór treningowy* $U = \{\boldsymbol{u}_1, \ldots \boldsymbol{u}_N\}$, $\boldsymbol{u}_i \in \mathbb{R}^p$
- struktura *warstwy ukrytej*: $2N$ neuronów
  - przypisać dwa neurony warstwy ukrytej (indeksy i' i i'') każdemu obrazowi $\boldsymbol{u}_i$:
  - określić hiperpłaszczyzny $\mathcal{H}(\boldsymbol{v}_i, \beta'_i)$, $\mathcal{H}(\boldsymbol{v}_i, \beta''_i)$ takie, że dla $j \neq i$
    $\boldsymbol{u}_i \in U_-(\boldsymbol{v}_i, \beta'_i) \cap U_+(\boldsymbol{v}_i, \beta''_i)$
    $\boldsymbol{u}_j \in U_+(\boldsymbol{v}_i, \beta'_i) \cup U_-(\boldsymbol{v}_i, \beta''_i)$
  - definicja wyjść neuronów ukrytych
    $s_{i'} = 0, \quad s_{i''} = 1$
    $s_{j'} = s_{j''}$ dla $j \neq i$
- *neuron wyjściowy*
  - $w_{i'} = 1$, $w_{i''} = -1$ jeżeli $\boldsymbol{u}_i \in U_-$
  - $w_{i'} = -1$, $w_{i''} = 1$ jeżeli $\boldsymbol{u}_i \in U_+$
    $(w_{j'} s_{j'} + w_{j''} s_{j''} = 0$ dla $j \neq i)$
  - $b = 0$



- brak algorytmów uczenia: **jak znaleźć $\boldsymbol{v}_i, \beta_{i'}, \beta_{i''}$?**
- rozwiązanie **nieefektywne**: więcej neuronów niż obrazów uczących i polega na **wyodrębnianiu obrazów** przy użyciu hiperpłaszczyzn
- **brak generalizacji** dla rzeczywistych problemów

5.1 Struktura perceptronu Rosenblatta
5.2 Uczenie perceptronu Rosenblatta
5.3 Budowa i uczenie maszyny liniowej

Uczenie jednowarstwowego perceptronu Rosenblatta

Zbiór uczący

- liniowo separowalny zbiór obrazów: $U = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N\}$
- *zbiór treningowy (uczący)*: zbiór **par** (obraz, klasa)

$$\{(\boldsymbol{u}_1, y_1^o),\ \ldots,\ (\boldsymbol{u}_N, y_N^o)\}$$

(każdy obraz pojawia się jeden raz)

- *ciąg uczący*: ciąg par $\big(\boldsymbol{u}(t), y^o(t)\big)$ losowanych ze zbioru uczącego

$$\big(\boldsymbol{u}(1), y^o(1)\big),\ \ldots,\ \big(\boldsymbol{u}(T), y^o(T))\big),$$

gdzie t jest numerem kroku uczenia; obrazy nogą się powtarzać

- *uczenie*: korekcja wag (i obciążeń) na podstawie wyjścia rzeczywistego i *wyjścia pożądanego* sieci w poszczególnych krokach uczenia

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \delta\boldsymbol{w}(t)$$

- *epoka*: prezentacja wszystkich par uczących; w procesie uczenia niezbędne jest użycie wielu epok

Uczenie jednowarstwowego perceptronu Rosenblatta

! Podstawowy algorytm uczenia, Rosenblatt (1962)

- arbitralne początkowe wagi $\boldsymbol{w}$ i obciążenie b
- prezentacja par $(\boldsymbol{u}, y)$ z **liniowo separowalnego** zbioru uczącego epoka po epoce, **z losowaniem bez zwracania** w każdej epoce
- *korekcja* wag i obciążenia jeśli klasyfikacja obrazu $\boldsymbol{u}$ jest błędna

$$\Delta \boldsymbol{w}(t) = y^o(t)\,\boldsymbol{u}(t) \qquad \Delta b(t) = y^o(t)$$

natomiast brak korekcji, jeśli klasyfikacja obrazu $\boldsymbol{u}(t)$ jest poprawna

$$\Delta \boldsymbol{w}(t) = 0 \qquad \Delta b(t) = 0$$

- postać algorytmu uczenia uwzględniająca oba przypadki (klasyfikacji poprawnej i błędnej)

$$\Delta \boldsymbol{w}(t) = \boldsymbol{u}(t)\ \mathrm{sign}\big(y^o(t) - y(t)\big)$$
$$\Delta b(t) = \mathrm{sign}\big(y^o(t) - y(t)\big)$$

SVM

Uczenie jednowarstwowego perceptronu Rosenblatta

Modyfikacje

Zmienna szybkość uczenia

Wielkość korekcji może być zmodyfikowana przez *współczynnik uczenia* γ (może być zmienny w czasie)

$$\Delta \boldsymbol{w} = \gamma\, \boldsymbol{u}\, \mathrm{sign}\big(y^o - y\big), \qquad \Delta b = \gamma\ \mathrm{sign}\big(y^o - y\big)$$

gdzie $0 < \gamma_{\min} < \gamma(t) < \gamma_{\max}$

Korekcje bezwzględne

Dla każdego obrazu ta sama korekcja wag jest powtarzana dopóki nie nastąpi poprawna klasyfikacja tego obrazu

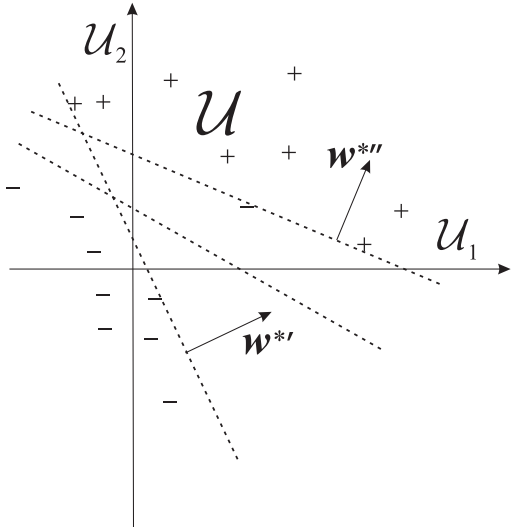
Uczenie jednowarstwowego perceptronu Rosenblatta

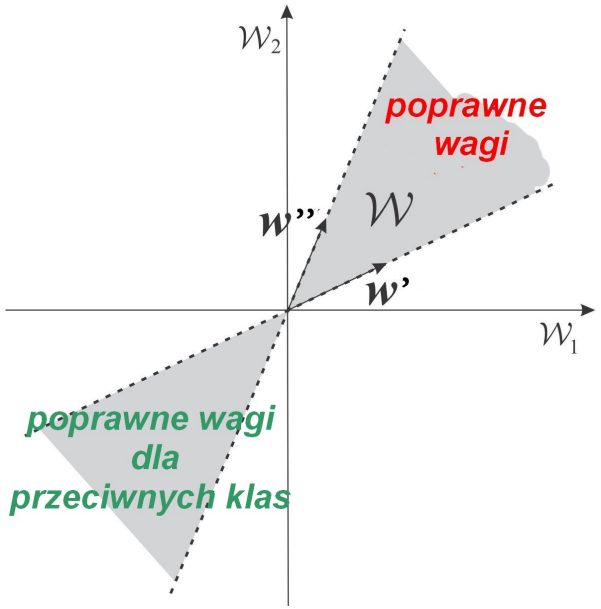
Zbieżność uczenia

Zbieżność uczenia perceptronu, Rosenblatt

Dla **liniowo separowalnego zbioru uczącego** perceptron Rosenblatta trenowany przy użyciu podstawowego algorytmu uczenia klasyfikuje poprawnie **wszystkie obrazy zbioru uczącego** po **skończenie wielu korekcjach wag**

- hiperpłaszczyzny klasyfikujące poprawnie (a zatem odpowiadające im wagi i obciążenia **nie są jednoznaczne**





⊛ Uczenie perceptronu

Liczba korekcji

⊛ Liczba korekcji (Novikoff (1962), Cristianini and Shawe-Taylor (2001), and Cristianini and Shawe-Taylor (2007))

Dla liniowo separowalnego zbioru trenującego i poprawnie klasyfikującej hiperpłaszczyzny decyzyjnej $\mathcal{H}(\boldsymbol{w}^o, b^o)$, liczba korekcji nie przekracza ρ^2/Δ_g^2 gdzie

$$\Delta_g = \inf_{\boldsymbol{u} \in \mathcal{S}} d(\boldsymbol{u}, \mathcal{H})$$

jest odległością obrazów uczących od hiperpłaszczyzny decyzyjnej (minimum indywidualnych odległości d) a ρ

$$\rho = \max_{U} \|\boldsymbol{u}\|$$

jest *średnicą zbioru* obrazów

⊛ Uczenie jednowarstwowego perceptronu Rosenblatta

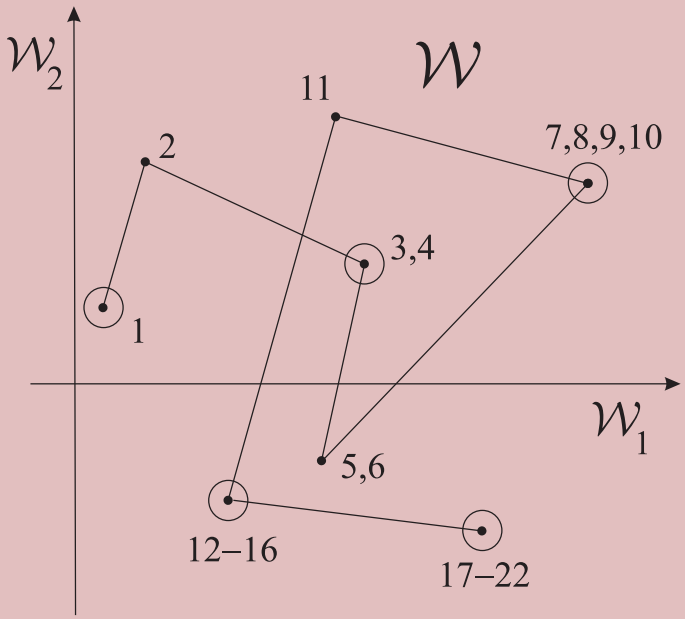
Algorytm kieszeniowy

Gallant (1990)

- *waga "w kieszeni"* $\boldsymbol{w}^{\mathsf{s}}(t)$ – waga dla najdłuższej serii do chwili t

  $\widetilde{N}(\boldsymbol{w})$ - liczba niewłaściwych klasyfikacji dla zbioru uczącego dla wag $\boldsymbol{w}$

  $\widetilde{N}$ - najmniejsza możliwa liczba niewłaściwych klasyfikacji dla zbioru uczącego



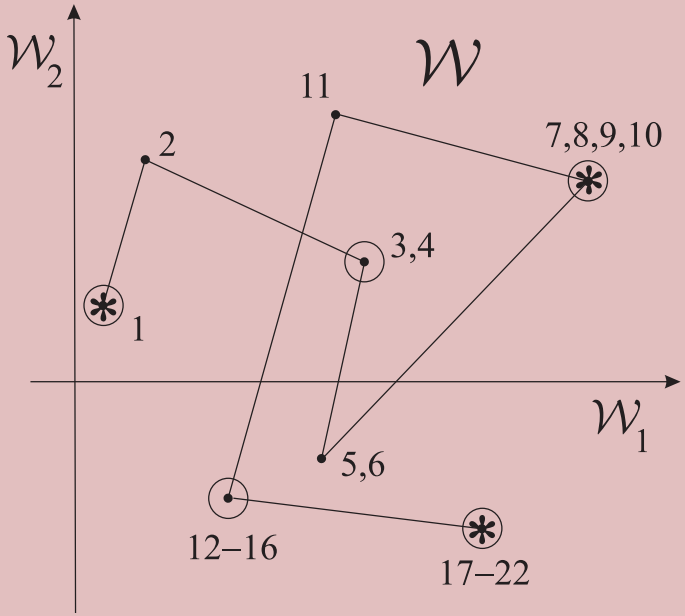
⊛ Uczenie kieszeniowe z grzechotką

Gallant (1990)

- *mechanizm grzechotki* - waga "w kieszeni" tylko jeśli dodatkowo powoduje mniej błędów na zbiorze uczącym

$$\boldsymbol{w}^{\mathsf{r}}(t) = \begin{cases} \boldsymbol{w}^{\mathsf{s}}(t) & \text{jeżeli} \quad \widetilde{N}(\boldsymbol{w}^{\mathsf{r}}(t)) < \widetilde{N}(\boldsymbol{w}^{\mathsf{s}}(t)) \\ \boldsymbol{w}^{\mathsf{r}}(t-1) & \text{w przeciwnym razie} \end{cases}$$

- metoda wymaga obliczenia liczby błędnych klasyfikacji dla $\boldsymbol{w}^{\mathsf{r}}(t) = \boldsymbol{w}^{\mathsf{s}}(t)$



⊛ Uczenie perceptronu jednowarstwowego

Zbieżność algorytmów kieszeniowych

⊛ Uczenie kieszeniowe (Muselli 1997)

Jeśli obrazy losowane są ze zwracaniem ze skończonego choć **niekoniecznie liniowo separowalnego** zbioru uczącego, to liczba błędnych klasyfikacji dla wag w kieszeni jest asymptotycznie optymalna, tzn.

$$\text{dla każdego } \eta > 0: \quad \mathscr{P}(\widetilde{N}(\boldsymbol{w}^{\mathsf{s}}(t)) - \widetilde{N} < \epsilon) \underset{t\to\infty}{\longrightarrow} 1$$

⊛ Zbieżność uczenia z grzechotką (Gallant 1990; Muselli 1997)

Jeśli obrazy losowane są ze zwracaniem ze skończonego choć **niekoniecznie liniowo separowalnego** zbioru uczącego, to liczba błędnych klasyfikacji dla wag z grzechotką jest optymalna po skończone liczbie kroków uczenia (*skończenie optymalna*), tzn. dla pewnego t^o

$$\mathscr{P}\left(\forall_{t>t^o}\ \widetilde{N}(\boldsymbol{w}^{\mathsf{r}}(t)) = \widetilde{N}\right) = 1$$

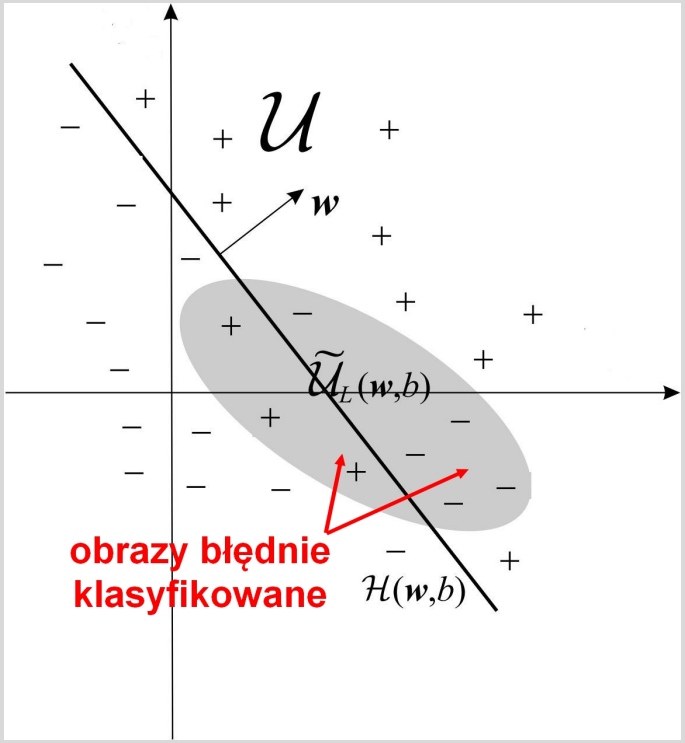
- liczba błędnych klasyfikacji dla wag z grzechotką maleje monotonicznie, ale nie jest tak dla wag w kieszeni

⊕ Interpretacja algorytmu Rosenblatta

Koszt klasyfikacji dla perceptronu Rosenblatta (1)

- zbiór tych obrazów uczących, które są **błędnie klasyfikowane** przez hiperpłaszczyznę decyzyjną $\mathcal{H}(\boldsymbol{w}, b)$

$$\widetilde{U}(\boldsymbol{w}, b) = \{\boldsymbol{u} \in \mathcal{S} : (b + \boldsymbol{w}^{\mathsf{T}}\boldsymbol{u})\, y^o < 0\}$$



⊕ Interpretacja algorytmu Rosenblatta

Koszt klasyfikacji dla perceptronu Rosenblatta (2)

- *koszt niewłaściwej klasyfikacji* obrazu $\boldsymbol{u}_n$

$$R(\boldsymbol{w}, b) = \sum_{\boldsymbol{u}_n \in \widetilde{U}(\boldsymbol{w},b)} | \underbrace{b + \boldsymbol{w}^{\mathsf{T}} \boldsymbol{u}_n}_{\substack{\text{aktywacja} \\ \text{perceptronu}}} | = - \sum_{\boldsymbol{u}_n \in U} (b + \boldsymbol{w}^{\mathsf{T}} \boldsymbol{u}_n) \operatorname{sign}(y_n^o - y_n)$$

- gradient

$$\frac{\partial R(\boldsymbol{w}, b)}{\partial \boldsymbol{w}} = - \sum_{\boldsymbol{u}_n \in U} \boldsymbol{u}_n \operatorname{sign}(y_n^o - y_n)$$

$$\frac{\partial R(\boldsymbol{w}, b)}{\partial b} = - \sum_{\boldsymbol{u}_n \in U} \operatorname{sign}(y_n^o - y_n)$$

(istnieje poza $\leqslant N$ hiperpłaszczyznami $\{(\boldsymbol{w}, b) : b + \boldsymbol{u}_n^{\mathsf{T}} \boldsymbol{w} = 0\}$ w $\mathbb{R}^{p+1}$)

- gradient jako suma po wszystkich źle klasyfikowanych obrazach zbioru uczącego dla zadanych wag - **nie jest dostępny**
- możliwości aproksymacji gradientu
  - aproksymacja gradientu przez gradient w danym kroku (gradient chwilowy)
  - aproksymacja gradientu raz na epokę (akumulacja gradientu podczas epoki bez zmiany wag)

⊕ Interpretacja algorytmu Rosenblatta

Minimalizacja kosztu

- metoda najszybszego spadku z *akumulacją gradientu podczas każdej epoki* $T_k = (t_k, \ldots, t_{k+T-1})$

$$\boldsymbol{w}(t_{k+1}) = \boldsymbol{w}(t_k) - \gamma \frac{\mathrm{d}R(\boldsymbol{w}, b)}{\mathrm{d}\boldsymbol{w}} \approx \boldsymbol{w}(t_k) + \gamma \sum_{t \in T_k} \boldsymbol{u}(t)\ \mathrm{sign}\big(y^o(t) - y(t)\big)$$

$$b^+(t_{k+1}) = b(t_k) - \gamma \frac{\mathrm{d}R(\boldsymbol{w}, b)}{\mathrm{d}b} \approx b(t_k) + \gamma \sum_{t \in T_k} \mathrm{sign}\big(y^o(t) - y(t)\big)$$

— **identyczna z uczeniem perceptronu Rosenblatta z korekcją wag raz na epokę**

- metoda najszybszego spadku z *chwilową aproksymacją gradientu*

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \gamma \frac{\mathrm{d}R(\boldsymbol{w}, b)}{\mathrm{d}\boldsymbol{w}} \approx \boldsymbol{w}(t) + \gamma\, \boldsymbol{u}(t)\ \mathrm{sign}\big(y^o(t) - y(t)\big)$$

$$b(t+1) = b(t) - \gamma \frac{\mathrm{d}R(\boldsymbol{w}, b)}{\mathrm{d}b} \approx b(t) + \gamma\, \mathrm{sign}\big(y^o(t) - y(t)\big)$$

— **identyczna z uczeniem Rosenblatta**

⊕ Warstwa ukryta dwuwarstwowego perceptronu Rosenblatta (1)

Klasyfikacja n obrazów binarnych o p pikselach (tzn. obrazy w $\mathbb{R}^p$)
Problem: Znaleźć rozmiar h i sposób określania wag warstwy ukrytej

- *zakodowanie obrazów uczących w warstwie ukrytej*
  - konstrukcja warstwy ukrytej o rozmiarze $h = 2n$ (dwa neurony na obraz trenujący), tak by dowolna klasyfikacja liniowa obrazów trenujących mogła być zrealizowana;
  - metoda **praktycznie bezużyteczna**: nie działa dla obrazów "podobnych" (brak *generalizacji*), działa tylko dla zadanego zbioru obrazów w zadanych położeniach,
  - uwzględnienie zakłóceń wymaga "ręcznego" określania obrazów podobnych i drastycznie zwiększa liczbę obrazów do zakodowania
- *pełne kodowanie* — zakodowanie wszystkich możliwych 2^p obrazów
  - warstwa ukryta o rozmiarze $h = 2^{p+1}$ (dwa neurony warstwy na obraz) pozwala na bezbłędną klasyfikację dowolnych obrazów
  - prowadziłoby to do nierealistycznie dużej warstwy ukrytej i konieczności uprzedniej znajomości klasy dowolnego obrazu - rozwiązanie niesensowne
  - ...

⊕ Warstwa ukryta dwuwarstwowego perceptronu Rosenblatta (2)

- *uczenie warstwy ukrytej* - brak algorytmu
- *kodowanie losowe* warstwy ukrytej
  - jeśli $N \leqslant p+1$, wszystkie klasyfikacje są liniowe zatem wystarczy wówczas zastosować losowe wagi warstwy ukrytej a dowolna klasyfikacja będzie liniowa
  - jeśli $N \leqslant 2(p+1)$ (*pojemność klasyfikacji liniowej*), to z prawdopodobieństwem 0.5 co najmniej połowa klasyfikacji jest liniowa zatem dla losowych wag bardziej prawdopodobna jest poprawna klasyfikacja
  - *odpowiednio duża warstwa ukryta*: aby dowolna klasyfikacja N obrazów była liniowa potrzebna jest warstwa ukryta o rozmiarze co najmniej $h = N - 1$; aby bardziej prawdopodobny losowy wybór klasyfikacji był liniowy potrzebna jest warstwa ukryta o rozmiarze co najmniej $h = N/2 - 1$ - zwiększanie liczby pikseli !
  - inspiracja maszyn wektorów podpierających SVM

5.1 Struktura perceptronu Rosenblatta
5.2 Uczenie perceptronu Rosenblatta
5.3 Budowa i uczenie maszyny liniowej

Maszyny liniowe

- klasyfikacja liniowa skończonego zbioru **liniowo separowalnego** na q klas: *maszyna liniowa* o q neuronach (Nilsson 1965):

  dla $k = 1, \ldots, q$

$$z_k = b_k + \boldsymbol{w}_k^{\mathsf{T}} \boldsymbol{u}, \quad \boldsymbol{u} \in \mathbb{R}^p \qquad \text{(funkcje decyzyjne)}$$

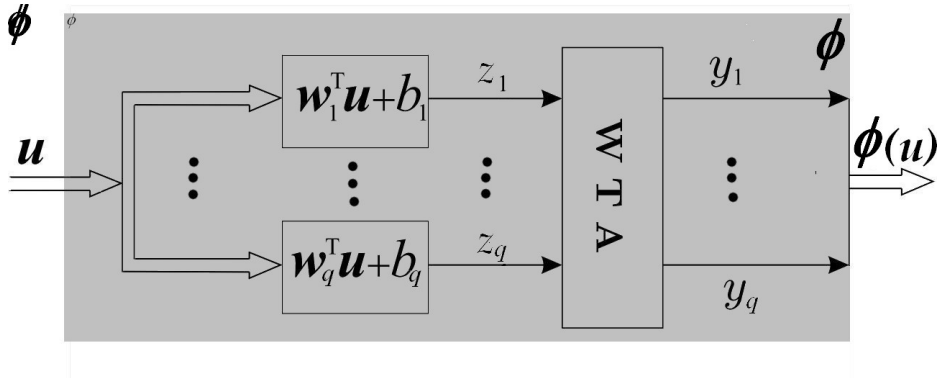
$$y_k = \begin{cases} 1 & \text{dla } k = \arg\max z_k \ (\textit{klasa wygrywająca}) \\ 0 & \text{w przeciwnym razie} \end{cases}$$

  i wówczas obraz u jest klasyfikowany do klasy wygrywającej $\mathcal{C}_{\arg\max z_k}$

- w postaci macierzowej

$$\boldsymbol{z} = \mathrm{WTA}(\boldsymbol{W}\boldsymbol{u} + \boldsymbol{b})$$

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1{}^{\mathsf{T}} \\ \vdots \\ \boldsymbol{w}_q{}^{\mathsf{T}} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_q \end{bmatrix}$$



gdzie WTA$=$

$$\begin{cases} 1 \text{ dla składowej największej} \\ 0 \text{ dla pozostałych} \end{cases}$$

(ang. *winner takes all*))

Uczenie maszyny liniowej

Algorytm korekcji wag

- arbitralna inicjacja wag, zazwyczaj $\boldsymbol{w}_k(0) = \mathbf{0}$, $b_k = 0$, $k = 1, \ldots, q$
- losowanie bez zwracania par uczących $(\boldsymbol{u}(t), \ell^o(t))$ epoka-po-epoce
  1. znaleźć klasę wygrywającą $\ell^*(t)$
  2. *korekcja* wag i obciążeń jeśli klasyfikacja obrazu $\boldsymbol{u}(t)$ jest błędna (klasa wygrywająca $\ell^*(t)$ jest inna niż właściwa $\ell^o(t)$)

$$\Delta \boldsymbol{w}_k = \begin{cases} \boldsymbol{u}_k & \\ -\boldsymbol{u}_k & \\ \mathbf{0} & \end{cases}, \quad \Delta b_k = \begin{cases} 1 & \text{dla } k = \ell^o \ \ (\textbf{klasa właściwa}) \\ -1 & \text{dla } k = \ell^* \ \ (\textbf{klasa wygrywająca}) \\ 0 & \text{dla pozostałych neuronów } k \end{cases}$$

  3. brak korekcji, jeśli klasyfikacja obrazu $\boldsymbol{u}(t)$ jest poprawna

$$\Delta \boldsymbol{w}_k = 0, \qquad \Delta b_k = 0, \quad k = 1, \ldots, q$$

- sformułowanie równoważne: korekcja macierzy wag

$$\Delta \boldsymbol{W} = \boldsymbol{u}^\mathsf{T} \operatorname{sign}(\boldsymbol{y}^o - \boldsymbol{y}), \qquad \Delta \boldsymbol{b} = \operatorname{sign}(\boldsymbol{y}^o - \boldsymbol{y})$$

- zbieżność: dla klas liniowo separowalnych poprawne wagi są osiągane po **skończenie wielu korekcjach**

Maszyna liniowa a perceptron Rosenblatta

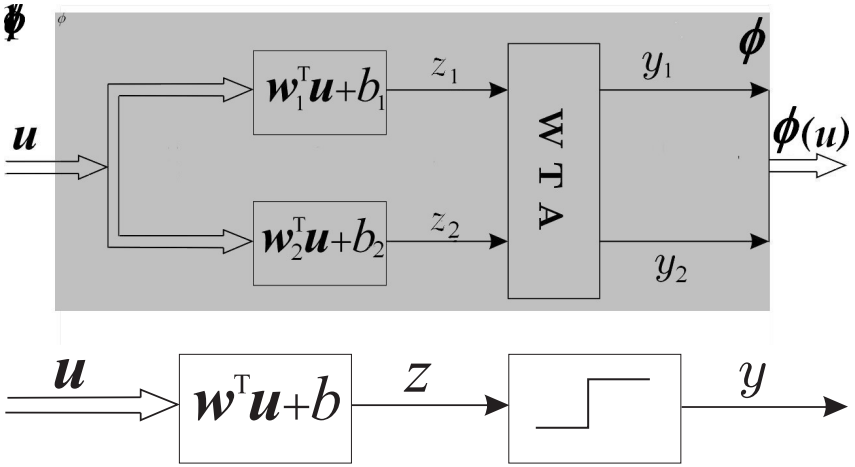
Równoważność maszyny liniowej i perceptronu Rosenblatta, dwie klasy

Maszyna liniowa z dwoma wyjściami

$$y_1 = \begin{cases} 1 & \text{jeśli } \boldsymbol{w}_1{}^\mathsf{T}\boldsymbol{u} + b_1 > \boldsymbol{w}_2{}^\mathsf{T}\boldsymbol{u} + b_2 \\ 0 & \text{w przeciwnym razie} \end{cases}$$

$$y_2 = 1 - y_1$$

jest równoważna jednowarstwowemu perceptronowi Rosenblatta o parametrach $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_2$, $b = b_1 - b_2$ i z korekcją wag identyczną z podstawowym algorytmem uczenia Rosenblatta



Literatura

Materiały źródłowe

Minsky, M. L. and S. A. Papert (1969). *Perceptrons*. Cambridge, Massachusetts: MIT Press. Also available: Expanded Edition, MIT Press, Cambridge, Ma., 1988.

Rosenblatt, F. (1958). "The perceptron: a probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6, pp. 386–408. Reprinted in **AnderRosen88**, Ch. 8, pp. 89-114, 1988.

Materiały pomocnicze

Cristianini, N. and J. Shawe-Taylor (2001). *An Introduction to Support Vector Machines*. Cambridge, UK: Cambridge University Press.

— (2007). *Introduction to Support Vector Machines*. URL: http://www.support-vector.net.

Gallant, S. I. (June 1990). "Perceptron-based Learning Algorithms". In: *IEEE Transactions on Neural Networks* 1.2, pp. 179–191.

Muselli, M. (May 1997). "On Convergence Properties of Pocket Algorithm". In: *IEEE Transactions on Neural Networks* 8.3, pp. 623–629.

Nilsson, N. J. (1965). *Learning Machines - Foundations of Trainable Pattern Classifying Systems*. New York: McGraw-Hill.

Novikoff, A. B. (1962). "On convergence proofs on perceptrons". In: *Symp. on the Mathematical Theory of Automata*. Vol. 12, pp. 615–622.

Literatura (cd.)

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Washington, D.C.: Spartan Books.

Temat 6: Maszyny wektorów podpierających

6.1 Maksymalizacja obszaru separującego
6.2 Wyznaczenie optymalnej hiperpłaszczyzny decyzyjnej
6.3 Maszyny wektorów podpierających
6.4 Problemy nieseparowalne liniowo
6.5 Metody obliczeniowe

6.1 Maksymalizacja obszaru separującego
6.2 Wyznaczenie optymalnej hiperpłaszczyzny decyzyjnej
6.3 Maszyny wektorów podpierających
6.4 Problemy nieseparowalne liniowo
6.5 Metody obliczeniowe

Reprezentacje wag perceptronu Rosenblatta

- wagi i obciążenia wytrenowanego perceptronu (niejednoznaczne !)
  - klasy liniowo separowalne, zerowe wagi początkowe, dowolny ciąg uczący, stały współczynnik uczenia η

Reprezentacja pierwotna uczenia

Sumowanie względem elementów **ciągu trenującego** $\big(\boldsymbol{u}(1), y^o(1)\big)$,.., $\big(\boldsymbol{u}(T), y^o(T)\big)$ Rosenblatt

$$\boldsymbol{w} = \sum_{t=1}^{T} \gamma\, \boldsymbol{u}(t)\, \mathrm{sign}\big(y^o(t) - y(t)\big), \qquad b = \sum_{t=1}^{T} \gamma\, \mathrm{sign}\big(y^o(t) - y(t)\big)$$

Reprezentacja dualna uczenia

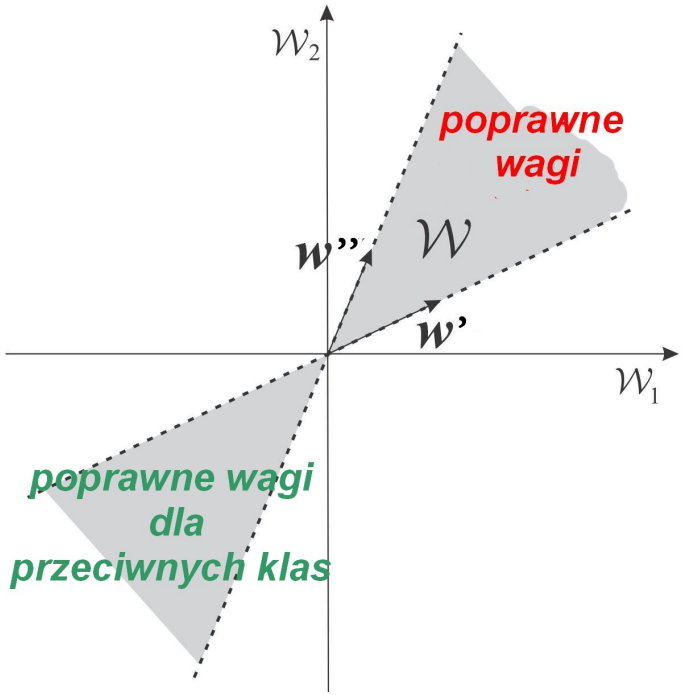
Sumowanie względem elementów **zbioru trenującego** $\big(\boldsymbol{u}_1, y_1^o\big)$,.., $\big(\boldsymbol{u}_N, y_N^o\big)$

$$\boldsymbol{w} = \sum_{n=1}^{N} \gamma\, \nu_n\, \boldsymbol{u}_n\, y_n^o, \qquad b = \sum_{n=1}^{N} \gamma\, \nu_n\, y_n^o$$

gdzie *siła obrazu* ν_n = krok uczenia · liczba błędnych klasyfikacji elementu $\boldsymbol{u}_n$ ($\nu_n \geqslant 0$)

Reprezentacja dualna a niejednoznaczność wag perceptronu Rosenblatta

- wagi wytrenowanego perceptronu (nawet dla zerowych wag początkowych) i dowolnego ciągu uczącego dla liniowo separowalnych klas **nie są jednoznaczne**
  - dla reprezentacji pierwotnej niejednoznaczność jest związana z doborem ciągu trenującego
  - dla reprezentacji dualnej niejednoznaczność jest wyrażona przez siły obrazów



Wytrenowany perceptron Rosenblatta

- wytrenowany perceptron Rosenblatta $\phi(\boldsymbol{u}) = \mathrm{sign}\big(b^o + \boldsymbol{u}^\mathsf{T}\boldsymbol{w}^o\big)$

(!) wytrenowany perceptron Rosenblatta — *postać pierwotna*

$$\phi(\boldsymbol{u}) = \mathrm{sign}\Big(\sum_{t=1}^{T}(1 + \underbrace{\boldsymbol{u}^\mathsf{T}\boldsymbol{u}(t)}_{\text{iloczyn skalarny}})\,\mathrm{sign}\big(y^o(t) - y(t)\big)\Big)$$

jest zależna od obrazów losowanych w procesie uczenia $\boldsymbol{u}(t)$, ich pożądanych klas $y^o(t)$ i wyjść perceptronu dla tych obrazów $y(t)$

(!) wytrenowany perceptron Rosenblatta — *postać dualna*

$$\phi(\boldsymbol{u}) = \mathrm{sign}\Big(\sum_{n=1}^{N}\nu_n\,(1 + \underbrace{\boldsymbol{u}^\mathsf{T}\boldsymbol{u}_n}_{\text{iloczyn skalarny}})\,y_n^o\Big)$$

jest zależna od ν_n (sił osadzenia obrazów w procesie uczenia)

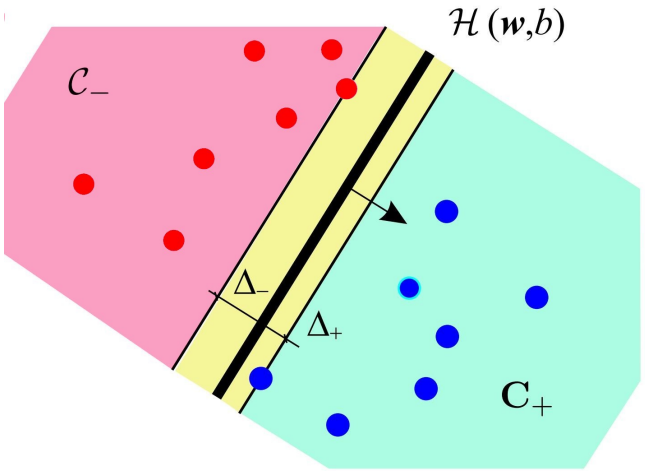
- hiperpłaszczyzna separująca i wynik klasyfikacji
  - zależy **iloczynów skalarnych obrazów** a nie samych obrazów
  - **zależy tylko od obrazów o dodatniej sile**

Obszar separujący

- istnieje nieskończenie wiele hiperpłaszczyzn separujących dwie klasy obrazów liniowo separowalnych
- dla każdej hiperpłaszczyzny separującej

$$b^o + \boldsymbol{w}^{o\top}\boldsymbol{u} = 0$$

  (tażdego kierunku $\mathbf{w}$) określimy maksymalny *obszar separujący* $\mathcal{S}^*$ obie klasy, zawarty pomiędzy dwoma hiperpłaszczyznami do niej równoległymi
- na każdym brzegu obszaru separującego leży co najmniej jeden obraz, gdyż inaczej można ten obszar zwiększyć
- wnętrze obszaru separującego nie zawiera obrazów uczących



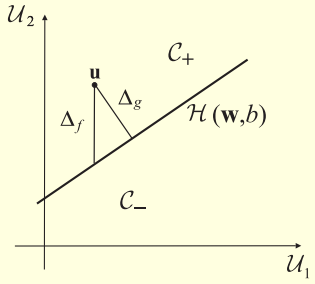
Ⓡ Odległość

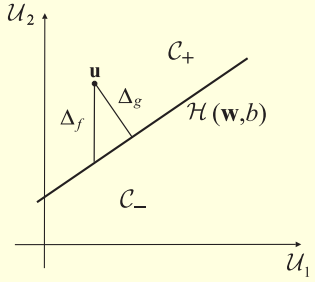
Ⓡ Odległości punkt-zbiór, zbiór-zbiór

- *odległość* punktu od zbioru $d(x, B) = \inf_{y \in B} d(x, y)$ (wynik 0 jeśli $x \in B$)
- *odległość* zbioru od zbioru dla zbiorów rozłącznych $d(UA, B) = \inf_{x \in A} d(x, B) = \inf_{x \in A, y \in B} d(x, y)$

Ⓡ Odległości punkt - płaszczyzna, płaszczyzna - płaszczyzna

- odległość punkt-płaszczyzna: $d\big(\boldsymbol{u}_0, \mathcal{H}(\boldsymbol{w}, b)\big) = \frac{|\boldsymbol{w}^{\mathsf{T}}\boldsymbol{u}_0 + b|}{\|\boldsymbol{w}\|}$,
- odległość płaszczyzn równoległych: $d\big(\mathcal{H}(\boldsymbol{w}, b_1), \mathcal{H}(\boldsymbol{w}, b_2)\big) = \frac{|b_1 - b_2|}{\|\boldsymbol{w}\|}$
- jeżeli odległość hiperpłaszczyzny i punktu wynosi Δ (*margines geometryczny*), to różnica wyrazów wolnych równań tych hiperpłaszczyzn wynosi $\Delta/|\boldsymbol{w}|$ (*margines funkcjonalny*)





Margines separujący

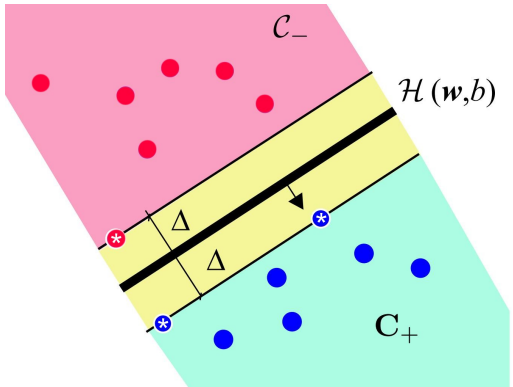
- *margines separujący* (geometryczny) Δ: odległość hiperpłaszczyzny separującej $\mathcal{H}$ od najbliższego obrazu dowolnej klasy

$$\Delta = \min_{b^o, \boldsymbol{w}^o} \Big(d(\mathcal{H}, \mathcal{C}_-),\ d(\mathcal{H}, \mathcal{C}_+) \Big) = \min_{b^o, \boldsymbol{w}^o} (\Delta^-, \Delta^+)$$

- im większy margines separujący, tym bardziej klasyfikacja jest odporna na fluktuacje danych czy obserwacje odstające (ang. *outliers*)
- *klasyfikator optymalny* klasyfikuje poprawnie i maksymalizuje margines separujący (względem b^o i $\boldsymbol{w}^o$)
  1. dla **zadanego** $\boldsymbol{w}$, margines separujący jest największy, gdy hiperpłaszczyzna separująca leży w połowie obszaru separującego
  2. największy margines separujący otrzymamy poprzez maksymalizację powyższego maksimum względem wektora $\boldsymbol{w}$

$$\Delta^* = \max_{\boldsymbol{w}} \Delta(\boldsymbol{w}), \qquad \boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} \Delta(\boldsymbol{w})$$

  3. rozwiązanie nie zależy od długości $\boldsymbol{w}$ a tylko od jego kierunku



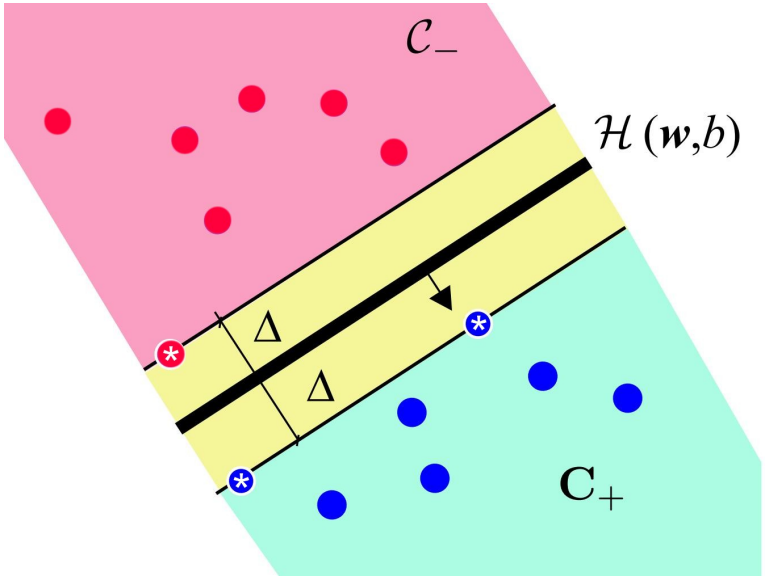
Wektory podpierające

(!) wektory $\boldsymbol{u}_n^*$ leżące **na brzegach obszaru separującego**

$$b + \boldsymbol{w}^\mathsf{T}\boldsymbol{u}_n^* = -\Delta_f$$
$$b + \boldsymbol{w}^\mathsf{T}\boldsymbol{u}_n^* = \Delta_f$$

(zawsze istnieją) zwane są *wektorami podpierającymi* (ang. *support vectors*) lub obrazami nośnymi

- na wektorach podpierających "opiera się" obszar separujący
- optymalne wartości wag i obciążeń zależą **tylko od obrazów podpierających**



Warunek poprawnej separacji

Postać kanoniczna

- margines separujący $\Delta = \frac{\Delta_f}{\|\boldsymbol{w}\|} = \frac{b+\boldsymbol{w}^\mathsf{T}\boldsymbol{u}_n^* \, y_n^o}{\|\boldsymbol{w}\|} = \frac{|b+\boldsymbol{w}^\mathsf{T}\boldsymbol{u}_n^*|}{\|\boldsymbol{w}\|}$ zależy od ilorazów parametrów $b/\|\boldsymbol{w}\|$, $\boldsymbol{w}/\|\boldsymbol{w}\|$
- *postać kanoniczna* $\Delta_f = 1$ (usuwa niejednoznaczność)

  ! separacja jest poprawna, jeśli dla każdego obrazu uczącego (tzn. $\boldsymbol{u}_n \in \mathcal{S}$)
  $$(b + \boldsymbol{w}^\mathsf{T}\boldsymbol{u}_n)y_n^o \geqslant 1$$
  - obszar separujący
  $$\{\boldsymbol{u} \in \mathbb{R}^p : \ |b + \boldsymbol{w}^\mathsf{T}\boldsymbol{u}| < 1\}$$
  - margines separujący: $1/\|\boldsymbol{w}\|$

Optymalna hiperpłaszczyzna separująca

DEFINICJA *Liniowy klasyfikator optymalny*

Klasyfikatorem liniowym optymalnym

$$\phi(\boldsymbol{u}) \stackrel{\text{def}}{=} \operatorname{sign}(\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u} + b^*)$$

nazywamy klasyfikator poprawnie klasyfikujący i maksymalnie separujący wszystkie obrazy uczące, tzn.

$$(\boldsymbol{w}^*, b^*) : \quad \begin{aligned} &\boldsymbol{w}^* = \arg\min \|\boldsymbol{w}\| && (1) \\ &(\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u}_n + b^*)\, y_n^o \geqslant 1 \text{ dla } \boldsymbol{u}_n \in \mathcal{S} && (2) \end{aligned}$$

- należy rozwiązać problem minimalizacji funkcji kwadratowej $p+1$ zmiennych $(\boldsymbol{w}, b)$ (1) z N ograniczeniami afinicznymi (2)

6.1 Maksymalizacja obszaru separującego
6.2 Wyznaczenie optymalnej hiperpłaszczyzny decyzyjnej
6.3 Maszyny wektorów podpierających
6.4 Problemy nieseparowalne liniowo
6.5 Metody obliczeniowe

Optymalna hiperpłaszczyzna separująca

Maksymalizacja marginesu - Problem pierwotny

minimalizować	$\ell(\boldsymbol{w}, b) = \frac{1}{2}\|\boldsymbol{w}\|^2$	(*wypukła na* $\mathbb{R}^{p+1}$)
przy ograniczeniach	$y_n^o \left(\boldsymbol{w}^\mathsf{T}\boldsymbol{u}_n + b\right) \geqslant 1, \quad n = 1, \ldots, N$	(*afiniczne*)

- problem ten jest zwykle sprowadzany do równoważnego *problemu dualnego*

opuścić: minimalizacja z ograniczeniami ↷ ⟲ nieseparowalne

⊛ Problem dualny

- *funkcja Lagrange'a* $L(\boldsymbol{w},\, b\,;\, \boldsymbol{\alpha}) = \frac{1}{2}\,\boldsymbol{w}^{\mathsf{T}}\boldsymbol{w} + \underbrace{\sum_{n=1}^{N} \overset{\substack{\text{mnożniki}\\ \text{Lagrange'a}\\ \geqslant 0}}{\alpha_n} \left(1 - y_n^o\left(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{u}_n + b\right)\right)}_{\substack{\text{kara za naruszenie}\\ \text{ograniczeń}}}$
- minimalizacja względem $\boldsymbol{w}$ i b

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0 \quad \Longrightarrow \quad \boldsymbol{w}^* = \sum_{i=1}^{N} \alpha_n\, \boldsymbol{u}_n\, y_n^o \qquad \text{podstawić do funkcji Lagrange'a}$$

$$\frac{\partial L}{\partial b} = 0 \quad \Longrightarrow \quad \sum_{i=1}^{N} \alpha_n\, y_n^o = 0 \qquad \text{ograniczenie w problemie dualnym}$$

- *funkcja dualna*

$$L^o(\boldsymbol{\alpha}) = -\tfrac{1}{2}\,\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{w}^* + \sum_{n=1}^{N} \alpha_n = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n\, \alpha_m y_n^o\, y_m^o\, \boldsymbol{u}_n^{\mathsf{T}}\boldsymbol{u}_m$$

- w rezultacie problem pierwotny jest równoważny problemowi **maksymalizacji** funkcji dualnej z ograniczeniami

$$\sum_{n=1}^{N} \alpha_n\, y_n^o = 0, \qquad \text{gdzie } \alpha_n \geqslant 0, \quad n = 1, \ldots, N$$

⊛ Maksymalizacja marginesu

Postać dualna

! Maksymalizacja marginesu - Problem dualny

Maksymalizować funkcję

$$L^o(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \, \alpha_m y_n^o \, y_m^o \, \boldsymbol{u}_n^\mathsf{T} \boldsymbol{u}_m$$

przy ograniczeniach $\sum_{n=1}^{N} \alpha_n \, y_n^o = 0, \quad \alpha_n \geqslant 0, \quad n = 1, \ldots, N$

- *programowanie kwadratowe*, N zmiennych $(\boldsymbol{\alpha})$, $N+1$ ograniczeń
- rozwiązanie $\boldsymbol{\alpha}^*$ **zależy od $\boldsymbol{u}_n$ tylko przez iloczyny skalarne $\boldsymbol{u}_n^\mathsf{T} \boldsymbol{u}_m$**
- w wyniku otrzymujemy *siły α_n obrazów*, a stąd wartości wag i obciążeń
- rozwiązanie $\boldsymbol{\alpha}^*$ **zależy tylko** od obrazów $\boldsymbol{u}_n$, dla których $\boldsymbol{\alpha}_n > 0$ - obrazy podpierające ($\boldsymbol{\alpha}_n$ - siły obrazów)

opuść: szczegóły ↷

⊛ Maksymalizacja marginesu

Rozwiązanie

- $\boldsymbol{\alpha}^*$ **zależy od $\boldsymbol{u}_n$ tylko przez iloczyny skalarne $\boldsymbol{u}_n^\mathsf{T}\boldsymbol{u}_j$**
- trzeci warunek Karusha-Kuhna-Tuckera: dla $n = 1, \ldots, N$

$$y_n^o \left(\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u}_n + b^*\right) \geqslant 1 \qquad \text{KKT-ne}$$

$$\alpha_n^* \geqslant 0 \qquad \text{KKT-ne'}$$

$$\alpha_n^* \left(1 - y_n^o \left(\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u}_n + b^*\right)\right) = 0 \qquad \text{KKT-ne''}$$

$\alpha_n^* = 0$	$y_n^o \left(\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u}_n + b^*\right) > 1$	$\boldsymbol{u}_n$ w obszarze dopuszczalnym	KKT-ar
$\alpha_n^* > 0$	$y_n^o \left(\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u}_n + b^*\right) = 1$	$\boldsymbol{u}_n$ – wektor podpierający	KKT-sv

- b^* może być określony na podstawie ograniczeń pierwotnych:

dla arbitralnego **wektora podpierającego** $\boldsymbol{w}_m^*$: $y_m^o \left(\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u}_m + b^*\right) = 1$ (KKT-sv) zatem

$$\begin{aligned} b^* &= y_m^o - \boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u}_m, \quad \boldsymbol{u}_m \in \mathcal{S}^* \\ &= y_m^o - \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \, y_n^o \, \boldsymbol{u}_n^\mathsf{T}\boldsymbol{u}_m \end{aligned}$$

Maszyna wektorów podpierających

Jednowarstwowa maszyna liniowa

Jednowarstwowa linowa maszyna wektorów podpierających - podsumowanie

$$AAA\alpha_n^* \, y_n^o \, \boldsymbol{u}_n^\mathsf{T} \boldsymbol{u}\Big) \qquad \text{gdzie}$$

- $\boldsymbol{\alpha}^*$ jest jednoznacznym rozwiązaniem problemu maksymalizacji funkcji

$$L^o(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \, \alpha_m y_n^o \, y_m^o \, \boldsymbol{u}_n^\mathsf{T} \boldsymbol{u}_m$$

przy ograniczeniach $\alpha_n \geqslant 0, \quad n = 1, \ldots, N, \qquad \sum_{n=1}^{N} \alpha_n \, y_n^o = 0$

- $\mathcal{S}^* = \{\boldsymbol{u}_n \in \mathcal{S} : \alpha_n^* > 0\}$ — zbiór wektorów podpierających,
- $b^* = y_m^o - \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \, y_n^o \, \boldsymbol{u}_n^\mathsf{T} \boldsymbol{u}_m$ dla dowolnego $\boldsymbol{u}_m \in \mathcal{S}^*$

- przypomnienie dowolny perceptron Rosenblatta w postaci dualnej:

$$\phi(\boldsymbol{u}) = \operatorname{sign}\Big(b + \sum_{n=1}^{N} \nu_n \, y_n^o \, \boldsymbol{u}_n^\mathsf{T} \boldsymbol{u}\Big)$$

czyli siła obrazu $\alpha_n^* = \nu_n$

Maszyna wektorów podpierających

Jednowarstwowa maszyna liniowa - własności rozwiązania

- $\boldsymbol{\alpha}^*$ **zależy od $\boldsymbol{u}_n$ tylko przez iloczyny skalarne $\boldsymbol{u}_n^\mathsf{T}\boldsymbol{u}_j$**

(!) siły α_n^* obrazów
  - równe zero wewnątrz obszaru dopuszczalnego - obrazy nie są podpierające
  - dodatnie na granicy obszaru dopuszczalnego -obrazy podpierające $\boldsymbol{u}_n \in \mathcal{S}^*$

- wagi optymalne

$$\boldsymbol{w}^* = \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \, \boldsymbol{u}_n \, y_n^o \qquad \alpha_n^* : \text{siła obrazu podpierającego } \boldsymbol{u}_n$$

- norma wag

$$\|\boldsymbol{w}^*\|^2 = \boldsymbol{w}^{*\mathsf{T}} \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* y_n^o \, \boldsymbol{u}_n \underset{\text{(KKT-ne")}}{=} \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \left(1 - y_n^o \, b\right) \underset{\text{(ogr. dualne)}}{=} \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^*$$

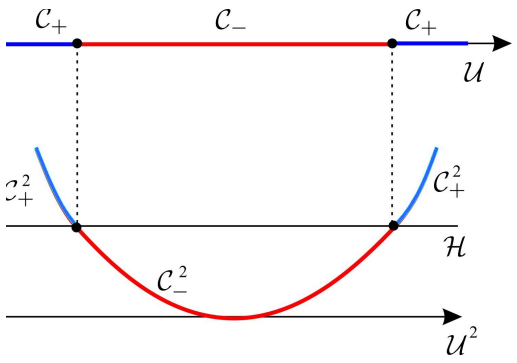
- największy margines geometryczny

$$\Delta_g^* = 1 / \sqrt{\sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^*}$$

6.1 Maksymalizacja obszaru separującego
6.2 Wyznaczenie optymalnej hiperpłaszczyzny decyzyjnej
6.3 Maszyny wektorów podpierających
6.4 Problemy nieseparowalne liniowo
6.5 Metody obliczeniowe

Idea transformacji do zbioru liniowo separowalnego

- klasyfikacja nieliniowa może stać się liniowa po przekształceniu obrazów („zanurzeniu") do odpowiedniej przestrzeni (przestrzeni cech) o **większej** liczbie wymiarów



- w odpowiednio wielowymiarowej przestrzeni wszystkie klasyfikacje są liniowe (do $p+1$ obrazów w $\mathbb{R}^p$) **liczba klasyfikacji liniowych $\leftrightarrow$**
- *przestrzeń cech*: $S = \varphi^{[h]}(U)$, przy czym $\dim(S) > \dim(U)$
- idea: *maszyna dwuwarstwowa*: pierwsza warstwa oblicza cechy a druga maksymalizuje margines w przestrzeni cech

Dwuwarstwowa maszyna maksymalizująca margines dla cech

- dwuwarstwowa maszyna maksymalizująca margines dla cech
  - pierwsza warstwa: obliczanie cech $\boldsymbol{s} = \boldsymbol{\psi}(\boldsymbol{u})$
  - druga warstwa: maksymalizacja marginesu w przestrzeni cech (poprzez znalezienie "cech podpierających" $\mathcal{S}^*$)

$$\phi(\boldsymbol{u}) = \operatorname{sign}\left(b^* + \sum_{\boldsymbol{s}_n \in \mathcal{S}^*} \alpha_n^* y_n^o \underbrace{\boldsymbol{s}_n^\mathsf{T} \boldsymbol{s}}_{\substack{\text{iloczyn skalarny} \\ \textbf{w przestrzeni cech}}}\right)$$

a stąd klasyfikator ma postać

$$\phi(\boldsymbol{u}) = \operatorname{sign}\left(b^* + \sum_{\substack{\boldsymbol{u}_n \in \psi^{-1}\mathcal{S}^* \\ \textbf{obrazy} \\ \textbf{podpierające}}} \alpha_n^* \, y_n^o \underbrace{\boldsymbol{\psi}^\mathsf{T}(\boldsymbol{u}_n)\boldsymbol{\psi}(\boldsymbol{u})}_{\substack{\text{iloczyn skalarny} \\ \textbf{w przestrzeni obrazów}}}\right)$$

$\psi^{-1}\mathcal{S}^*$: przeciwobraz zbioru "cech podpierających" tzn. zbiór obrazów $\mathcal{S}^*$ przekształconych do cech podpierających

- wada: duży nakład obliczeniowy ze względu na - zwykle - duży wymiar obrazów czyli złożoność funkcji $\boldsymbol{\psi}$

Dwuwarstwowa maszyna wektorów podpierających

"Kernel trick"

® Iloczyn skalarny

dodatnia określoność $k(\boldsymbol{u},\boldsymbol{u}) \geqslant 0, \;\; k(\boldsymbol{u},\boldsymbol{u}) = 0 \Leftrightarrow \boldsymbol{u} = 0$

biliniowość: $k(\alpha'\boldsymbol{u}' + \alpha''\boldsymbol{u}'', \boldsymbol{u}) = \alpha' k(\boldsymbol{u}', \boldsymbol{u}) + \alpha'' k(\boldsymbol{u}'', \boldsymbol{u})$

symetria: $k(\boldsymbol{u}', \boldsymbol{u}'') = k(\boldsymbol{u}'', \boldsymbol{u}')$

[*warunek Cauchy'ego Schwarza*: $k^2(\boldsymbol{u}', \boldsymbol{u}'') \leqslant k(\boldsymbol{u}', \boldsymbol{u}')\, k(\boldsymbol{u}'', \boldsymbol{u}'')$]

! Alternatywna konstrukcja maszyny dwuwarstwowej: *Kernel trick*

pominąć obliczanie cech, ale zastąpić iloczyn skalarny w przestrzeni obrazów przez funkcję obrazu, która jest iloczynem skalarnym w przestrzeni cech bez określania funkcji cech ψ

$$\phi(\boldsymbol{u}) = \operatorname{sign}\Bigg(b^* + \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \, y_n^o \, \underbrace{k(\boldsymbol{u}_n, \boldsymbol{u})}_{\substack{\textit{jądro} \\ \textit{SVM}}}\Bigg)$$

Dwuwarstwowa maszyna wektorów podpierających

Przykłady transformacji

Ⓔ Znaleźć jądro dla zadanej funkcji cech

$$\boldsymbol{u} = (u_1, u_2)$$

funkcja cech: $\boldsymbol{s} = \boldsymbol{\psi}(\boldsymbol{u}) = (u_1, u_2, u_1^2 + u_2^2)$

jądro : $k(\boldsymbol{u}', \boldsymbol{u}'') = \boldsymbol{\psi}^\mathsf{T}(\boldsymbol{u}')\boldsymbol{\psi}(\boldsymbol{u}'') = \boldsymbol{u}'^\mathsf{T}\boldsymbol{u}'' + \|\boldsymbol{u}'\|^2\|\boldsymbol{u}''\|^2$

Ⓔ Znaleźć funkcję cech dla zadanego jądra

jądro
$$\begin{aligned} k(\boldsymbol{u}', \boldsymbol{u}'') &= (1 + \boldsymbol{u}'^\mathsf{T}\boldsymbol{u}'')^2 \\ &= 1 + 2u_1'u_1'' + 2u_2'u_2'' + u_1'^2u_1''^2 + u_2'^2u_2''^2 + 2u_1'u_2'u_1''u_2'' \\ &= \boldsymbol{\psi}(\boldsymbol{u}')^\mathsf{T}\boldsymbol{\psi}(\boldsymbol{u}'') \end{aligned}$$

funkcja cech: $\boldsymbol{\psi}(\boldsymbol{u}) = (1, \sqrt{2}u_1, \sqrt{2}u_2, u_1^2, u_2^2, \sqrt{2}u_1u_2)$

Dwuwarstwowa maszyna wektorów podpierających

Warunek Mercera

- **nie każda funkcja symetryczna k może być jądrem w przestrzeni cech**, tzn. być wynikiem mapowania iloczynu skalarnego $\boldsymbol{u}^\mathsf{T}\boldsymbol{u}'$ w przestrzeni obrazów przez pewną funkcję cech $\boldsymbol{\psi}$

$$k(\boldsymbol{u}, \boldsymbol{u}') = \boldsymbol{\psi}(\boldsymbol{u})^\mathsf{T}\boldsymbol{\psi}(\boldsymbol{u}')$$

Warunek Mercera

Funkcja symetryczna $k(\boldsymbol{u}, \boldsymbol{u}')$ jest jądrem w pewnej przestrzeni cech, tzn. jest wynikiem mapowania iloczynu skalarnego w przestrzeni obrazów przez pewną funkcję cech $\boldsymbol{\psi}$

$$k(\boldsymbol{u}, \boldsymbol{u}') = \boldsymbol{\psi}(\boldsymbol{u})^\mathsf{T}\boldsymbol{\psi}(\boldsymbol{u}')$$

jeśli dla dowolnych obrazów $\in U$, macierz $\boldsymbol{K} = \{k(\boldsymbol{u}_m, \boldsymbol{u}_{m'})\}$ jest dodatnio półokreślona
tzn. dla każdego każdej macierzy $\boldsymbol{K}$ o dowolnym rozmiarze $M \times M$ i wektorów $\boldsymbol{\alpha}, \boldsymbol{\alpha}' \in \mathbb{R}^M$

$$\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{K}\,\boldsymbol{\alpha}' \geqslant 0$$

Dwuwarstwowa maszyna wektorów podpierających

Własności warunku Mercera

⊕ Własności warunku Mercera

Jeśli funkcje symetryczne k', k'' spełniają warunek Mercera, to również spełnia go

1. ich dowolna kombinacja liniowa $k(\boldsymbol{u}, \boldsymbol{u}') = a'k'(\boldsymbol{u}, \boldsymbol{u}') + a''k''(\boldsymbol{u}, \boldsymbol{u}')$
2. ich iloczyn $k(\boldsymbol{u}, \boldsymbol{u}') = k'(\boldsymbol{u}, \boldsymbol{u}')\, k''(\boldsymbol{u}, \boldsymbol{u}')$
3. złożenie k' i dowolnej funkcji f, tzn. $k(\boldsymbol{u}, \boldsymbol{u}') = k'\big(f(\boldsymbol{u}'), f(\boldsymbol{u}'')\big)$
4. wielomian W takiej funkcji o nieujemnych współczynnikach, tzn. $W\big(t'(\boldsymbol{u}', \boldsymbol{u}'')\big)$
5. funkcja wykładnicza $k(\boldsymbol{u}, \boldsymbol{u}') = \exp\big(t'(\boldsymbol{u}', \boldsymbol{u}'')\big)$

Ⓔ Przykład: Uzasadnić spełnienie warunku Mercera dla $k(\boldsymbol{u}, \boldsymbol{u}') = (1 + \boldsymbol{u}^\top \boldsymbol{u}')^2 \boldsymbol{u}^\top \boldsymbol{u}$

$k(\boldsymbol{u}, \boldsymbol{u}')$ jest wielomianem $W(x) = (1+x)^2 x$ funkcji $x = \boldsymbol{u}^\top \boldsymbol{u}$, która oczywiście spełnia warunek Mercera (dla $\psi \equiv 1$). Zgodnie z własnością 4, k spełnia warunek Mercera.

Dwuwarstwowa maszyna wektorów podpierających

Przykłady jąder dla SVM

César Souza, Kernel Functions for Machine Learning Applications, 2010
<http://crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html>

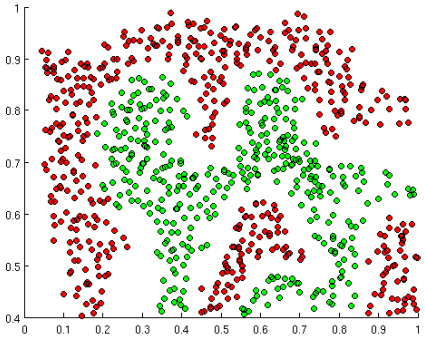
jądro	$k(\boldsymbol{u}', \boldsymbol{u}'') \qquad \boldsymbol{u}', \boldsymbol{u}'' \in \mathbb{R}^p$	uwagi
wielomianowe	$(\boldsymbol{u}'^{\mathsf{T}}\boldsymbol{u}'' + 1)^p$	MC
gaussowskie	$\exp\left(\frac{-1}{2\sigma^2}\,\lVert\boldsymbol{u}' - \boldsymbol{u}''\rVert^2\right) = \exp\left(-\gamma\,\lVert\boldsymbol{u}' - \boldsymbol{u}''\rVert^2\right)$	MC
wykładnicze	$\exp\left(\frac{-1}{\sigma}\,\lVert\boldsymbol{u}' - \boldsymbol{u}''\rVert\right)$	MC
sigmoidalne	$\mathrm{tgh}(\alpha\,\boldsymbol{u}'^{\mathsf{T}}\boldsymbol{u}'' + c)$	perceptron 2-warstwowy
uogólnione t-Studenta	$1/\left(1 + \lVert\boldsymbol{u}' - \boldsymbol{u}''\rVert^d\right)$	MC
min (*histogram*)	$\sum_{i=1}^{p} \min(u_i', u_i'')$	MC
wymierne kwadratowe	$1 - \frac{\lVert\boldsymbol{u}' - \boldsymbol{u}''\rVert^2}{\lVert\boldsymbol{u}' - \boldsymbol{u}''\rVert^2 + c}$	MC, zastępuje gauss.
odwrotne kwadratowe	$1/\sqrt{\lVert\boldsymbol{u}' - \boldsymbol{u}''\rVert^2 + c}$	MC
kwadratowe	$-\sqrt{\lVert\boldsymbol{u}' - \boldsymbol{u}''\rVert^2 + c^2}$	
chi-kwadrat	$1 - 2\sum_{i=1}^{p} \frac{\lVert u_i' - u_i''\rVert^2}{\lVert u_i' + u_i''\rVert}$	
logarytmiczne	$-\log(\lVert\boldsymbol{u}' - \boldsymbol{u}''\rVert^d + 1)$	
potęgowe	$-\lVert\boldsymbol{u}' - \boldsymbol{u}''\rVert^d$	

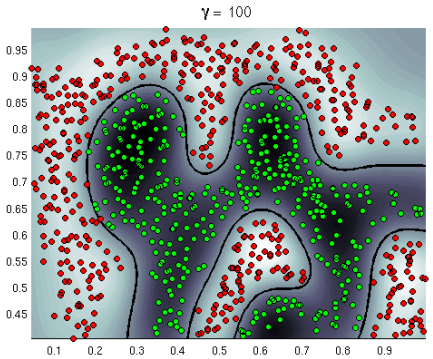
Dwuwarstwowa maszyna wektorów podpierających

Dobór jądra

Andrew Ng, Support Vector Machines Using An SVM - Machine Learning
https://www.youtube.com/watch?v=i25MEJeX0Eg
stats.stackexchange.com/questions/131138/what-makes-the-gaussian-kernel-so-magical

- liczba obrazów jest "mała" (tzn. mniejsza od rozmiaru obrazu)
  zastosować jądro liniowe (ale: niebezpieczeństwo zbytniego dopasowania)
- liczba obrazów jest większa od rozmiaru obrazu ale mniejsza niż ok. 500
  zastosować jądro gaussowskie
- liczba obrazów jest większa od 50 000
  zastosować jądro liniowe (duży czas uczenia dla jądra gaussowskiego)
- szczególnie polecane, gdy nie ma dodatkowej wiedzy na temat danych



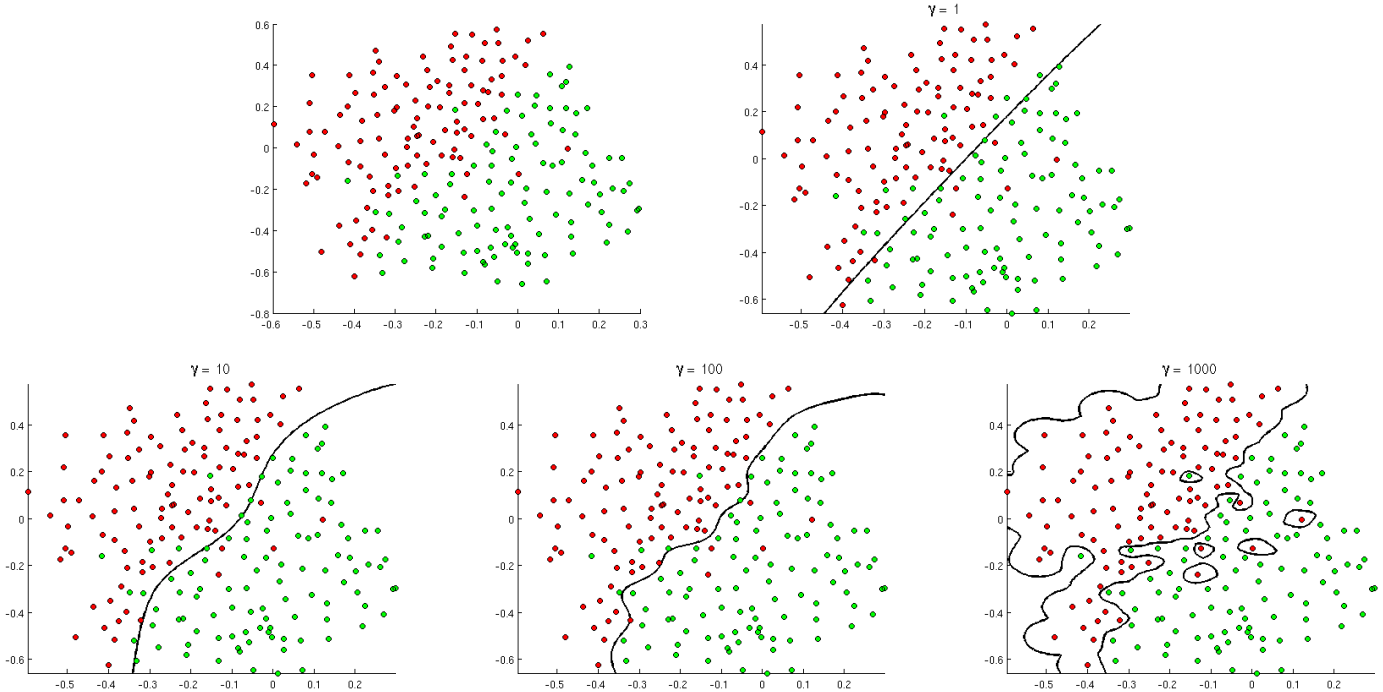


Andrew Ng "Machine Learning", Exercise 8: Non-linear SVM classification with kernels,
nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-the-linearly-separable-case-1.html
$\gamma = 100$

Dwuwarstwowa maszyna wektorów podpierających

Jądro gaussowskie - wpływ parametru

- $\gamma = 0$ - jądro sprowadza się do jądra liniowego
- $\gamma \nearrow$ - zwiększają się częstotliwości przestrzenne funkcji separującej, zwiększa się jakość klasyfikacji na zbiorze uczącym i zbiorach testujących
- $\gamma \Uparrow$ - duże częstotliwości przestrzenne, klasyfikator silnie dopasowany do danych, mała jakość klasyfikacji na zbiorach testujących (zbytnie dopasowanie, "przeuczenie")



Andrew Ng "Machine Learning", Exercise 8: Non-linear SVM classification with kernels, nlp.stanford.edu/

Dwuwarstwowa maszyna wektorów podpierających

Dwuwarstwowa maszyna wektorów podpierających - podsumowanie

$$\phi(\boldsymbol{u}) = \operatorname{sign}\Big(b^* + \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \, y_n^o \, k(\boldsymbol{u}_n, \boldsymbol{u})\Big)$$

gdzie

- $\boldsymbol{\alpha}^o$ jest jednoznacznym rozwiązaniem problemu maksymalizacji

$$L^*(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \, \alpha_m y_n^o \, y_m^o \, k(\boldsymbol{u}_n, \boldsymbol{u}_j)$$

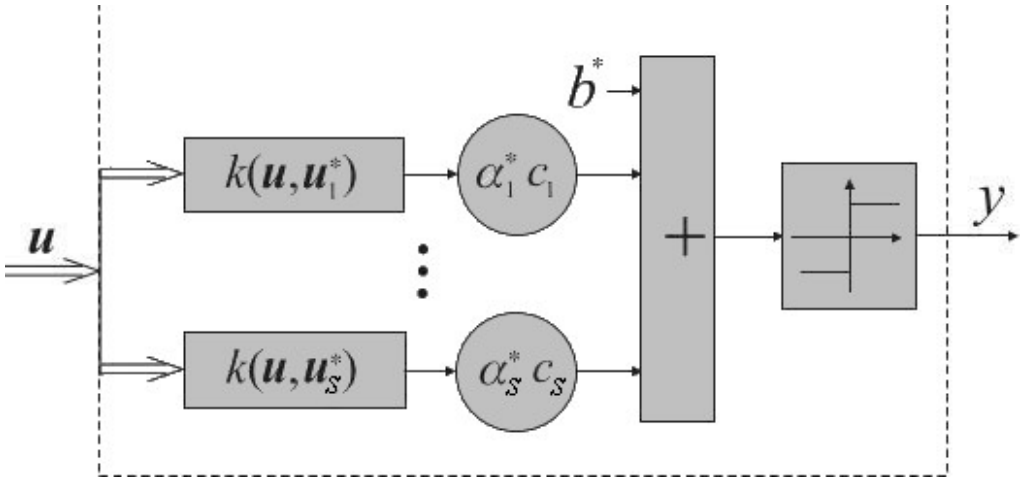
przy ograniczeniach $\alpha_n \geqslant 0, \quad n = 1, \ldots, N, \qquad \sum_{n=1}^{N} \alpha_n \, y_n^o = 0$

- $\mathcal{S}^* = \{\boldsymbol{u}_n \in \mathcal{S} : \alpha_n > 0\}$ — zbiór wektorów podpierających
- $b^* = y_m^o - \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \, y_n^o \, k(\boldsymbol{u}_n, \boldsymbol{u}_m)$ dla dowolnego $\boldsymbol{u}_m \in \mathcal{S}^*$

Dwuwarstwowa maszyna wektorów podpierających

Struktura SVM

$$\phi(\boldsymbol{u}) = \mathrm{sign}\left(b^* + \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \, y_n^o \, k(\boldsymbol{u}_n, \boldsymbol{u})\right)$$



⊕ Maszyna wielomianowa dla problemu XOR

Ilustracja: Wyprowadzenie analityczne (1)

- $\boldsymbol{u}_i$ $\begin{bmatrix}-1 & -1\end{bmatrix}^\mathsf{T}$, $\begin{bmatrix}-1 & 1\end{bmatrix}^\mathsf{T}$, $\begin{bmatrix}1 & -1\end{bmatrix}^\mathsf{T}$, $\begin{bmatrix}1 & 1\end{bmatrix}^\mathsf{T}$ $\mathcal{S}$
  y_i^o -1, 1, 1, -1
- $k(\boldsymbol{u}', \boldsymbol{u}'') = (1 + \boldsymbol{u}'^\mathsf{T}\boldsymbol{u}'')^2$
- problem dualny: maksymalizować $L^*(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\mathsf{T}\boldsymbol{i} - \frac{1}{2}\,\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{S}\,\boldsymbol{\alpha}$ przy ograniczeniach $\boldsymbol{\alpha} \geqslant \boldsymbol{0}$, $\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{y}^o \geqslant 0$

  gdzie $\boldsymbol{y}^o = \begin{bmatrix}-1\\1\\1\\-1\end{bmatrix}$, $\quad \boldsymbol{y}^o\,\boldsymbol{y}^{o\mathsf{T}} = \begin{bmatrix}1 & -1 & -1 & 1\\-1 & 1 & 1 & -1\\-1 & 1 & 1 & -1\\1 & -1 & -1 & 1\end{bmatrix}$,

  $\boldsymbol{K} = \{k(\boldsymbol{u}_i,\, \boldsymbol{u}_j)\} = \begin{bmatrix}9 & 1 & 1 & 1\\1 & 9 & 1 & 1\\1 & 1 & 9 & 1\\1 & 1 & 1 & 9\end{bmatrix}$, $\boldsymbol{S} = (\boldsymbol{y}^o\,\boldsymbol{y}^{o\mathsf{T}})\,.\,\boldsymbol{K} = \begin{bmatrix}9 & -1 & -1 & 1\\-1 & 9 & 1 & -1\\-1 & 1 & 9 & -1\\1 & -1 & -1 & 9\end{bmatrix}$

⊕ Maszyna wielomianowa dla problemu XOR

Ilustracja: Wyprowadzenie analityczne (2)

- rozwiązanie dualne bez ograniczeń: $\boldsymbol{\alpha}^o = \boldsymbol{S}^{-1}\,\boldsymbol{i} = \frac{1}{96}\begin{bmatrix} 11 & 1 & 1 & -1 \\ 1 & 11 & -1 & 1 \\ 1 & -1 & 11 & 1 \\ -1 & 1 & 1 & 11 \end{bmatrix}\boldsymbol{i} = \frac{1}{8}\,\boldsymbol{i}$
- ograniczenia dualne są spełnione, wszystkie obrazy $\boldsymbol{u}_i$ są S.V.
- $b^* = y_m^o - \sum\limits_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^*\, y_n^o\, k\Big(\boldsymbol{u}_n, \boldsymbol{u}_m\Big)$ for $\boldsymbol{u}_m \in \mathcal{S}^*$ zatem $b^* = 0$
- jądro: $k(\boldsymbol{u}, \boldsymbol{u}') = (1 + \boldsymbol{u}^\mathsf{T}\boldsymbol{u}')^2 = 1 + 2\,\boldsymbol{u}^\mathsf{T}\boldsymbol{u}' + (\boldsymbol{u}^\mathsf{T}\boldsymbol{u}')^2$
$= 1 + 2u_1u_1' + 2u_2u_2' + (u_1u_1')^2 + (u_2'u_2')^2 + 2u_1u_2u_1'u_2'$
$\underset{(\mathcal{S})}{=} 3 + 2u_1u_1' + 2u_2u_2' + 2u_1u_2u_1'u_2'$

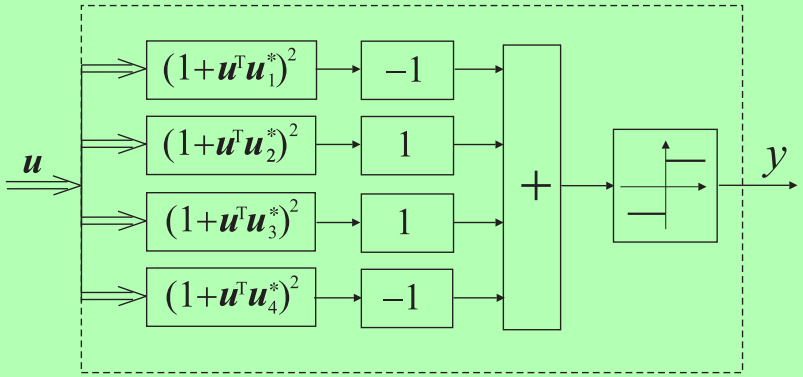
⊕ Maszyna wielomianowa dla problemu XOR

Ilustracja: Wyprowadzenie analityczne (3)

- funkcja klasyfikująca $\phi(\boldsymbol{u}) = \mathrm{sign}\Big(\sum_{\boldsymbol{u}\in\mathcal{S}} \alpha_n^* \, y_n^o \, k\big(\boldsymbol{u}, \boldsymbol{u}_n\big)\Big) = \mathrm{sign}\Big(\sum_{\boldsymbol{u}\in\mathcal{S}} y_n^o \, k\big(\boldsymbol{u}, \boldsymbol{u}_n\big)\Big)$
- równoważna transformacja obrazu

$\psi(\boldsymbol{u})$
$[1, \sqrt{2}\,u_1, \sqrt{2}\,u_2, u_1^2, u_2^2, \sqrt{2}\,u_1 u_2]^\mathsf{T}$ =

przy czym dla $\boldsymbol{u} \in \mathcal{S}$
$\psi(\boldsymbol{u}) = [\sqrt{3}, \sqrt{2}\,u_1, \sqrt{2}\,u_2, \sqrt{2}\,u_1 u_2]^\mathsf{T}$



6.1 Maksymalizacja obszaru separującego
6.2 Wyznaczenie optymalnej hiperpłaszczyzny decyzyjnej
6.3 Maszyny wektorów podpierających
6.4 Problemy nieseparowalne liniowo
6.5 Metody obliczeniowe

Problemy nieseparowalne liniowo

- przypadek liniowo separowalny: warunki poprawnej klasyfikacji

$$\left(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{u}_n + b\right) y_n^o \geqslant 1 \quad \text{for } \boldsymbol{u}_n \in \mathcal{S}$$

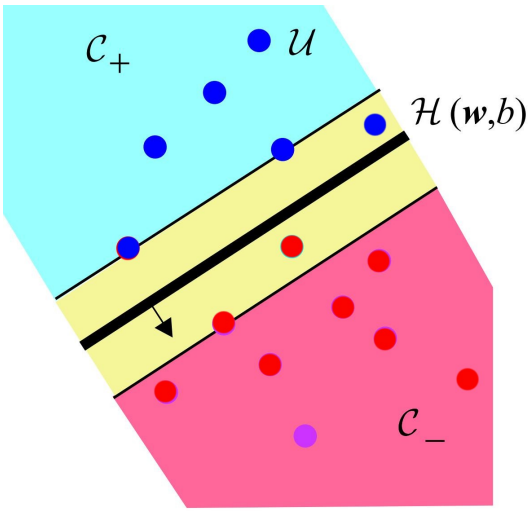
- *miękki margines*: warunki poprawnej klasyfikacji mogą być naruszone (Cortes and Vapnik 1995)

$$\left(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{u}_n + b\right) y_n^o \geqslant 1 - \xi_n, \quad \xi_n \geqslant 0 \quad \text{dla } \boldsymbol{u}_n \in \mathcal{S}$$

- ξ_n - *zmienne dopełniające*

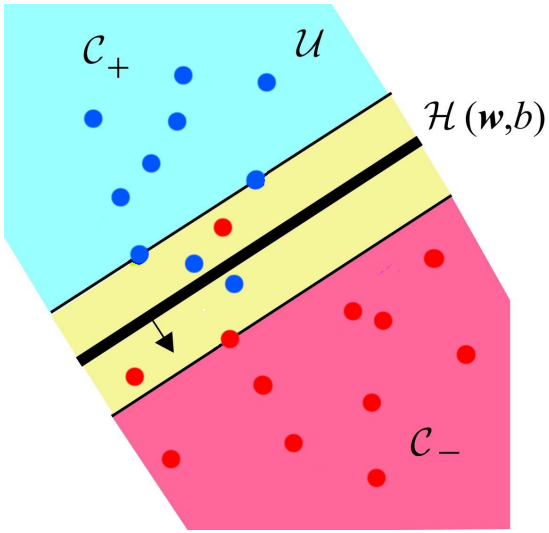
$0 < \xi_n < 1$

naruszony obszar separujący



$\xi_n > 1$

brak separacji linowej



Liniowa kara za naruszenie ograniczeń

porównanie: przypadek spełnionych ograniczeń ↔

Problem pierwotny

minimalizować funkcję $\ell(\boldsymbol{w},\, b,\, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + c\sum_{n=1}^{N}\xi_n$

przy ograniczeniach $(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{u}_n + b)\, y_n^o \geqslant 1 - \xi_n, \quad \xi_n \geqslant 0, \quad n = 1, \ldots, N$

- *współczynnik kary* c kontroluje liczbę obrazów niepoprawnie klasyfikowanych

  c większe $\Rightarrow$ większa kara $\Rightarrow$ mniej naruszonych ograniczeń
- minimalizacja funkcji kwadratowej o $p + 1 + N$ zmiennych $(\boldsymbol{w}, b, \boldsymbol{\xi})$ z $2N$ ograniczeniami afinicznymi
- obciążenie może być obliczone po minimalizacji (z problemu pierwotnego)

⊛ Liniowa kara za naruszenie ograniczeń

Funkcja Lagrange'a

- funkcja Lagrange'a $L(\boldsymbol{w},\, b,\, \boldsymbol{\xi};\, \boldsymbol{\alpha}, \boldsymbol{\lambda})$

$$= \frac{1}{2}\, \boldsymbol{w}^{\mathsf{T}} \boldsymbol{w} + c \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \left(1 - \xi_n - (\boldsymbol{w}^{\mathsf{T}} \boldsymbol{u}_n + b)\, y_n^o\right) - \sum_{n=1}^{N} \lambda_n\, \xi_n$$

- obliczenie funkcji dualnej (KKT-gr)

$$\boldsymbol{w}^* = \sum_{n1}^{N} \alpha_n\, y_n^o\, \boldsymbol{u}_n, \qquad \sum_{n=1}^{N} \alpha_n\, y_n^o = 0, \qquad \alpha_n + \lambda_n = c, \qquad n = 1, \ldots, N$$

zgodnie z KKT-ne" dodatkowe człony f.L. kasują się

⊛ Liniowa kara za naruszenie ograniczeń

Problem dualny

⊛ Kara liniowa - problem dualny

maksymalizować funkcję

$$L^*(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \, \alpha_m \, y_n^o \, y_m^o \, k(\boldsymbol{u}_n, \boldsymbol{u}_m)$$

przy ograniczeniach

$$\sum_{n=1}^{N} \alpha_n \, y_n^o = 0$$

$$0 \leqslant \alpha_n \leqslant c, \quad \text{dla} \quad n = 1, \ldots, N$$

- *programowanie kwadratowe*, N zmiennych $(\boldsymbol{\alpha})$, $2N + 1$ ograniczeń
- w wyniku otrzymujemy siły obrazów, a stąd wartości wag i obciążeń

⊛ SVM dla kary liniowej

Rozwiązanie

- warunki KKT dla ograniczeń nierównościowych: dla i=1,. . . ,N

$$\left(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{u}_n + b\right) y_n^o \geqslant 1 - \xi_n, \qquad \xi_n \geqslant 0, \qquad i = 1, \ldots, N \qquad \text{(KKT-ne)}$$

$$\alpha_n \geqslant 0, \qquad \lambda_n \geqslant 0, \qquad i = 1, \ldots, N \qquad \text{(KKT-ne')}$$

$$\alpha_n \left(1 - \xi_n - (\boldsymbol{w}^{\mathsf{T}}\boldsymbol{u}_n + b)\, y_n^o\right) = 0, \qquad \lambda_n\, \xi_n = 0, \qquad i = 1, \ldots, NtagKKT - ne''$$

$\alpha_n^* = 0$	$\left(\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u}_n + b^*\right) y_n^o > 1$	$\lambda_n^* = c$	$\xi_n^* = 0$	$\boldsymbol{u}_n$ w reg. dopuszcz.
$0 < \alpha_n^* < c$	$\left(\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u}_n + b^*\right) y_n^o = 1$	$0 < \lambda_n^* < c$	$\xi_n^* = 0$	$\boldsymbol{u}_n$ — podpierający
$\alpha_n^* = c$	$y_n^o \left(\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{u}_n + b^*\right) = 1 - \xi_n$	$\lambda_n^* = 0$	$\xi_n^* > 0$	$\boldsymbol{u}_n$ podpierający w reg

- b^* dobrane tak, by $y_n^o\, y_n = 1$ dla każdego $i : 0 < \alpha_n^* < c$

Maszyna wektorów podpierających

Maszyna dwuwarstwowa dla naruszonych ograniczeń

! SVM z możliwością naruszenia ograniczeń - podsumowanie

gdzie

$$y = \operatorname{sign}\Big(b^* + \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \, y_n^o \, k(\boldsymbol{u}, \boldsymbol{u}_n)\Big)$$

- $\boldsymbol{\alpha}^*$ jest jednoznacznym rozwiązaniem problemu maksymalizacji

$$L^*(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_n - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_n \, \alpha_j \, y_n^o \, y_j^o \, k(\boldsymbol{u}_n, \boldsymbol{u}_j)$$

przy ograniczeniach

$$0 \leqslant \alpha_n \leqslant c, \quad \text{dla} \quad i = 1, \ldots, N$$

$$\sum_{i=1}^{N} \alpha_n \, y_n^o = 0$$

- $\mathcal{S}^* = \{\boldsymbol{u}_n \in U : 0 < \alpha_n < c\}$ — zbiór wektorów podpierających,
- b^* wyznaczone z ograniczeń pierwotnych

Maszyna wektorów podpierających z możliwością naruszenia ograniczeń

Własności rozwiązania

- wagi optymalne

$$\boldsymbol{w}^* = \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \, \boldsymbol{u}_n \, y_n^o \qquad \alpha_n^* : \text{siła obrazu podpierającego } \boldsymbol{u}_n$$

(!) siły α_n^* obrazów
  - równe zero wewnątrz obszaru dopuszczalnego - obrazy nie są podpierające
  - dodatnie i mniejsze niż c: na brzegu obszaru dopuszczalnego - obrazy podpierające $\boldsymbol{u}_n \in \mathcal{S}^*$
  - równe c: poza obszarem dopuszczalnym - obrazy naruszające ograniczenia

- zmienne dopełniające $\xi_n^* = 1 - y_n^o \left(\boldsymbol{w}^{*\mathsf{T}} \boldsymbol{u}_n + b^* \right)$
- norma wag

$$\|\boldsymbol{w}^*\|^2 = \boldsymbol{w}^{*\mathsf{T}} \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* y_n^o \, \boldsymbol{u}_n \underset{\text{(KKT-ne'')}}{=} \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \left(1 - y_n^o \, b^* - \xi_n^* \right)$$

$$\underset{\text{(ogr. dualne)}}{=} \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* (1 - \xi_n^*)$$

- największy margines geometryczny $\Delta_g^* = 1 / \sqrt{\sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* (1 - \xi_n^*)}$

Maszyna wektorów podpierających

Skalowanie kary za naruszenie ograniczeń

! SVM ze skalowaniem kary za naruszenie ograniczeń - podsumowanie

gdzie
$$y = \operatorname{sign}\Big(b^* + \sum_{\boldsymbol{u}_n \in \mathcal{S}^*} \alpha_n^* \, y_n^o \, k(\boldsymbol{u}, \boldsymbol{u}_n)\Big)$$

- $\boldsymbol{\alpha}^o$ jest jednoznacznym rozwiązaniem problemu maksymalizacji
$$L^*(\boldsymbol{\alpha}) = \frac{-1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \, \alpha_m \, y_n^o \, y_m^o \, k(\boldsymbol{u}_n, \boldsymbol{u}_m) \qquad \text{(bez członu liniowego)}$$
przy ograniczeniach
$$\sum_{n=1}^{N} \alpha_n \, y_n^o = 0$$
$$0 \leqslant \alpha_n \leqslant 1/N, \qquad n = 1, \ldots, N \qquad \text{(c zastąpione przez $1/N$)}$$
$$\sum_{n=1}^{N} \alpha_n \geqslant c_0 \quad (\text{ gdzie } 0 \leqslant c_0 \leqslant 1) \qquad \text{(additional constraint)}$$
- $\mathcal{S}^* = \{\boldsymbol{u}_n \in U : 0 < \alpha_n < c\}$ — zbiór wektorów podpierających
- b^* wyznaczone z ograniczeń pierwotnych

Własności rozwiązania

- c_0 $(0 \leqslant c_0 \leqslant 1)$ - kara za naruszenie ograniczeń
- dla wektorów podpierających $0 < \alpha_n^* < \frac{1}{N} \quad \boldsymbol{u}_n \in \mathcal{S}^*$
- dla wektorów naruszających ograniczenia $\alpha_n^* = \frac{1}{N}$
- b^* wyprowadzone z ograniczeń pierwotnych dobrane tak, by dla każdego wektora podpierającego (tzn. dla $0 < \alpha_n^* < \frac{1}{N}$) spełniony był warunek

$$y_n^o \, y_n = 1$$

6.1 Maksymalizacja obszaru separującego
6.2 Wyznaczenie optymalnej hiperpłaszczyzny decyzyjnej
6.3 Maszyny wektorów podpierających
6.4 Problemy nieseparowalne liniowo
6.5 Metody obliczeniowe

⊛ Rozwiązania numeryczne

Metody największego spadku

⊛ Metoda największego spadku

$$\boldsymbol{\alpha}(t+1) = \boldsymbol{\alpha}(t) + \eta \, \frac{\mathrm{d}L^o(\boldsymbol{\alpha})}{\mathrm{d}\boldsymbol{\alpha}}$$

⊛ Sekwencyjna metoda największego spadku

powtarzać do spełnienia warunku stopu

1. wybrać m losowo
2. $\alpha_m = \alpha_m(t) + \eta_m \, \dfrac{\partial L^o(\boldsymbol{\alpha})}{\partial \alpha_m}$
3. $\alpha_m := \max(0, \alpha_m)$
4. $\alpha_m(t+1) = \min(c, \alpha_m)$ (dla problemów z karą liniową

- założyć najpierw b jeśli warunek liniowy związany z b nie może być spełniony
- modyfikacja: zastąpić ograniczenia liniowe przez $\left|\sum_{\boldsymbol{u} \in U} \alpha_m \, y_m^o\right| \leqslant z_m(t)$ gdzie $\{z_m(t)\}$ jest malejący

⊛ Rozwiązania numeryczne

Porcjowanie

⊛ Procedury dwupoziomowe

- nawet dla dużych zbiorów danych, rzędu 10^6

1. dla zadanego $\boldsymbol{\alpha}$, znaleźć wektory podpierające $U_{\boldsymbol{\alpha}} = \{\boldsymbol{u}_n \in U : \alpha_n > 0\}$
2. optymalizować względem $\boldsymbol{u} \in U_{\boldsymbol{\alpha}}$
3. iteracja dla wszystkich $\boldsymbol{u}_n \notin U_\alpha$
4. obliczyć $\boldsymbol{\alpha}$, przejść do 1

⊛ Porcjowanie

1. określić *zbiór roboczy* obrazów
2. powtarzać do spełnienia warunku stopu
   1. optymalizować względem zbioru roboczego
   2. określić nowy zbiór roboczy spomiędzy wektorów które naruszają warunki KTT

⊛ Dekompozycja

porcjowanie ze stałą wielkością zbioru roboczego

⊛ Rozwiązania numeryczne

Dekompozycja

⊛ Minimalna optymalizacja sekwencyjna

dekompozycja dla dwuelementowych zbiorów roboczych:

- ograniczenia $0 \leqslant \alpha', \alpha'' \leqslant c$ prowadzą do $U \leqslant \alpha'' \leqslant V$ gdzie

  $U = \max(0, \alpha'' - \alpha'),\ V = \min(c, c - \alpha' + \alpha'')$ dla $\boldsymbol{u}', \boldsymbol{u}''$ w różnych klasach

  $U = \max(0, \alpha' + \alpha'' - c),\ V = \min(c, \alpha' + \alpha'')$ dla $\boldsymbol{u}', \boldsymbol{u}''$ w tej samej rodzinie

- obliczyć α', α''

$$e' = \mathscr{C}(\boldsymbol{u}') - y'^o = \Big(\sum_{\boldsymbol{u}_m \in U} \alpha_m\, y_m^o\, k(\boldsymbol{u}_m, \boldsymbol{u}') + b \Big) - y'^o, \quad e'' = \mathscr{C}(\boldsymbol{u}'') - y''^o$$

$$\kappa = k(\boldsymbol{u}', \boldsymbol{u}') + k(\boldsymbol{u}'', \boldsymbol{u}'') - 2\, k(\boldsymbol{u}', \boldsymbol{u}'')$$

$$\alpha''(t) = \alpha''(t-1) + (e' - e'')\, y''^o / \kappa, \quad \alpha''(t) =: \max(U, \alpha''(t))$$

$$\alpha'(t) = \alpha'(t-1) + y'^o\, y''^o \big(\alpha''(t-1) - \alpha''(t)\big)$$

Literatura

Materiały źródłowe

Asuncion, A. and D. Newman (2016). *UCI Machine learning repository*. U. California at Irvine. URL: http://archive.ics.uci.edu/ml.

Guyon, I. (2006). *Support Vector Machines projects*. ClopiNet. URL: http://www.clopinet.com/isabelle/Projects/SVM.

Ivanciuc, O. (2005). *Support Vector Machines portal*. U. of Texas at Galvestone. URL: http://www.support-vector-machines.org.

Kernel Machines pages (2007). URL: http://www.kernel-machines.org.

Lin, C.-J. (Jan. 2013). *Support Vector Machines and Kernel Methods: Status and Challenges*. Talk at K. U. Leuven Optimization in Engineering Center. URL: www.csie.ntu.edu.tw/~cjlin/talks/kuleuven_svm.pdf.

Magdin, V. and K. Murphy (2005). *Comparison of software for SVMs*. U. of British Columbia, Vancouver, Canada. URL: http://people.cs.ubc.ca/~murphyk/Software/svm.htm.

Moore, A. (2016). *Machine Learning Tutorials*. Carnegie-Mellon University. URL: http://www.autonlab.org/tutorials.

SVM tutorials links (2004). URL: http://www.svms.org/tutorials.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag.

Literatura (cd.)

— (1998). *Statistical Learning Theory*. Wiley.

Materiały pomocnicze

Cortes, C. and V. Vapnik (1995). "Support-Vector Networks". In: *Machine Learning* 20, pp. 273–297.

Cristianini, N. and J. Shawe-Taylor (2001). *An Introduction to Support Vector Machines*. Cambridge, UK: Cambridge University Press.

— (2007). *Introduction to Support Vector Machines*. URL: http://www.support-vector.net.

Vapnik, V. and A. Chervonenkis (1981). "On the uniform convergence of relative frequencies of events to their probabilities". In: *Theory of Probability and Its Applications* 26.3, pp. 532–553.

Część III: Sieci perceptronowe a aproksymacja

Temat 7. Neurony liniowe

Temat 8. Aproksymacja funkcji

Temat 9. Błędy aproksymacji

Temat 10. Generalizacja i dobór modelu

Temat 11. Uczenie sieci neuronowych

Temat 12. Propagacja wsteczna gradientu

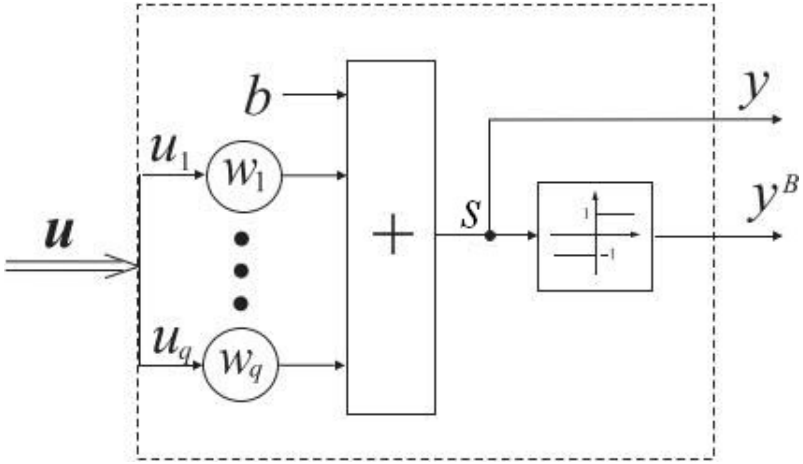
Temat 7: Neurony liniowe

7.1 Pierwszy neuron liniowy: Adaline
7.2 Uczenie sieci liniowych
7.3 Rozwiązanie analityczne

7.1 Pierwszy neuron liniowy: Adaline
7.2 Uczenie sieci liniowych
7.3 Rozwiązanie analityczne

Adaline - neuron liniowy

- **ADA**ptive **LI**near **NE**uron (Widrow and Hoff 1960) (Stanford Uni., Stanford, CA)
- **ADA**ptive **LIN**ear **E**lement



Adaline

$$x(t) = \boldsymbol{w}(t)^{\mathsf{T}}\boldsymbol{u}(t) \qquad \textit{wyjście liniowe (korekcja wag)}$$

$$y(t) = \operatorname{sign} x(t) \qquad \textit{wyjście binarne (klasyfikacja)}$$

- współczesny perceptron wielowarstwowy

  perceptron Rosenblatta $\longrightarrow$ struktura sieci
  Adaline $\longrightarrow$ algorytm uczenia

7.1 Pierwszy neuron liniowy: Adaline
7.2 Uczenie sieci liniowych
7.3 Rozwiązanie analityczne

Problem uczenia ADALINE

- obrazy uczące $\boldsymbol{u}_k$, *wyjścia pożądane* (**analogowe!**), zbiór *par uczących* $(\boldsymbol{u}_n, y_n^o), n = 1, \ldots, N$
- *ciąg uczący* $\big(\boldsymbol{u}(t), y^o(t)\big)$, $t = 1, \ldots, T$, jest wybierany zgodnie z pewnym rozkładem $\mathscr{P}_{\boldsymbol{u},y^o}$, określającym *środowisko uczenia*
  - *odchyłka wyjścia sieci*

$$R(t) = y^o(t) - y(t) = y^o(t) - \boldsymbol{w}^\mathsf{T}\boldsymbol{u}(t)$$

- *funkcja strat* tzn. "kara" za chwilową odchyłkę wyjścia sieci jest kwadratowa

$$\ell(t; \boldsymbol{w}) = \frac{1}{2}\left|y^o(t) - \boldsymbol{w}^\mathsf{T}\boldsymbol{u}(t)\right|^2 = \frac{1}{2}|R(t)|^2$$

- koszt estymacji (błąd estymacji, *ryzyko*) tzn. suma chwilowych strat jest kwadratową funkcją odchyłek

$$R_{\mathsf{kw}}(\boldsymbol{w}) = \sum_{t=1}^{T} \ell(t; \boldsymbol{w}) = \frac{1}{2}\sum_{t=1}^{T}\left|y^o(t) - \boldsymbol{w}^\mathsf{T}\boldsymbol{u}(t)\right|^2 = \frac{1}{2}\sum_{t=1}^{T}|R(t)|^2$$

  - *średniokwadratowy* błąd estymacji $R_{\mathsf{srkw}}(\boldsymbol{w}) = R_{\mathsf{kw}}/T$
  - zadanie: dobrać $\boldsymbol{w}$ minimalizujące kwadratowy/średniokwadratowy błąd estymacji

Najmniejsze średnie kwadraty μ-LMS

Algorytm największego spadku dla błędu średniokwadratowego

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \mu \frac{\mathrm{d}R_{\mathsf{srkw}}(\boldsymbol{w})}{\mathrm{d}\boldsymbol{w}} = \boldsymbol{w}(t) + \frac{\mu}{T} \sum_{t=1}^{T} \boldsymbol{u}(t)\, R(t)$$

- najprostsze oszacowanie - przez błąd chwilowy $R_{\mathsf{srkw}}(\boldsymbol{w}) = |R(t)|^2$

Algorytm najmniejszych średnich kwadratów μ-LMS

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \mu\, \boldsymbol{u}(t)\, R(t)$$

Zbieżność algorytmu μ-LMS

Jeżeli $\boldsymbol{u}_i$ są losowane niezależnie z rozkładu o macierzy kowariancji $\boldsymbol{C}_{\boldsymbol{u}}$, to

- wartość oczekiwana wag $\mathscr{E}\boldsymbol{w}(t)$ są zbieżna jeżeli $0 < \mu \leqslant \frac{2}{\max \mathsf{eig}(\boldsymbol{C}_{\boldsymbol{u}})}$
- błąd średniokwadratowy $\mathscr{E}|R(t)|^2$ jest zbieżny, jeżeli $0 < \mu \leqslant \frac{2}{\operatorname{tr} \boldsymbol{C}_{\boldsymbol{u}}}$ $(\operatorname{tr} \boldsymbol{C}_{\boldsymbol{u}} = \sum \mathsf{eig}(\boldsymbol{C}_{\boldsymbol{u}}))$

Algorytm Kaczmarza

- podejście alternatywne
  - zamiast minimalizacji błędu modelu aż do uzyskania zerowego przyrostu wag dla bieżących obserwacji
  - minimalizacja przyrostu wag przy założeniu zerowego błędu modelu dla bieżących obserwacji - *reguła najmniejszych zmian*
- minimalizować $R(\boldsymbol{w}^+) = \frac{1}{2}\,\|\boldsymbol{w}^+ - \boldsymbol{w}\|^2$ przy ograniczeniach $x^o = \boldsymbol{w}^{+\mathsf{T}}\boldsymbol{u}$

Algorytm Kaczmarza

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \frac{1}{\|\boldsymbol{u}(t)\|^2}\,\boldsymbol{u}(t)\,R(t) \qquad (\text{for } \|\boldsymbol{u}(t)\| > 0)$$

⊛ Algorytm Kaczmarza

Szczegóły

- minimalizować $R(\boldsymbol{w}^+) = \frac{1}{2}\,\|\boldsymbol{w}^+ - \boldsymbol{w}\|^2$ przy ograniczeniach $y^o = \boldsymbol{w}^{+\mathsf{T}}\boldsymbol{u}$
- funkcja Lagrange'a

$$L(\boldsymbol{w}^+;\beta) = R(\boldsymbol{w}^+) + \beta\,(x^o - \boldsymbol{w}^{+\mathsf{T}}\boldsymbol{u})$$

- *warunki KKT* (R wypukły i ciągły wraz z pochodnymi na $\mathbb{R}^\nu$, ograniczenia afiniczne)

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{w}^+} &= \boldsymbol{w}^+ - \boldsymbol{w} - \beta\,\boldsymbol{u} = 0 \\ \frac{\partial L}{\partial \beta} &= x^o - \boldsymbol{w}^{+\mathsf{T}}\boldsymbol{u} = 0 \end{aligned} \qquad \Longrightarrow \qquad \beta\,\|\boldsymbol{u}\|^2 = x^o - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{u} = R$$

- $\boldsymbol{w}^+$ jest rzutem $\boldsymbol{w}$ na hiperpłaszczyznę $x^o = \boldsymbol{w}^{+\mathsf{T}}\boldsymbol{u}$
- $\boldsymbol{w}^+ = \boldsymbol{w} + \frac{1}{\|\boldsymbol{u}\|^2}\boldsymbol{u}\,R$

Normalizowane najmniejsze średnie kwadraty α-LMS

Normalizowane najmniejsze średnie kwadraty (α-LMS)

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \frac{\alpha}{\|\boldsymbol{u}(t)\|^2}\,\boldsymbol{u}(t)\,R(t) \qquad (\text{dla } \|\boldsymbol{u}(t)\| > 0)$$

- skorygowane wyjście (dla zmodyfikowanej wagi) $y^+(t) = \boldsymbol{w}(t+1)^\mathsf{T}\boldsymbol{u}(t)$
- korekcja wyjścia i korekcja odchyłki są proporcjonalne do odchyłki

$$x^+(t) - x(t) = \Delta\boldsymbol{w}(t)^\mathsf{T}\boldsymbol{u}(t) = \alpha\,R(t)$$
$$R^+(t) - R(t) = -\alpha\,R(t)$$

- skalowanie $\boldsymbol{u}' = \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|}$, $x' = \frac{x}{\|\boldsymbol{u}\|}$, $x^{o\prime} = \frac{x^o}{\|\boldsymbol{u}\|}$ prowadzi do μ-LMS

Zmodyfikowane α-LMS

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \frac{\alpha}{c + \|\boldsymbol{u}(t)\|^2}\,\boldsymbol{u}(t)\,R(t)$$

$0 < \alpha < 2$, *człon regularyzujący* $c > 0$

⊛ Normalizowane LMS

Zbieżność

⊛ Zbieżność modyfikowanych LMS α-LMS

- norma przyrostów wag jest zbieżna monotonicznie do zera

$$\|\boldsymbol{w}(t+1) - \boldsymbol{w}(t)\| \downarrow 0$$

- ważone reszty są zbieżne do zera: $\frac{R(t)}{\sqrt{c+\|\boldsymbol{u}(t)\|^2}} \longrightarrow 0$
- zbieżność wag $\boldsymbol{w}(t) \longrightarrow \boldsymbol{w}^o$ zachodzi dla odpowiednio małego c,

(jeśli $\displaystyle\sum_{\tau=t}^{t+k} \frac{\boldsymbol{u}(\tau)\boldsymbol{u}(\tau)^\mathsf{T}}{\|\boldsymbol{u}(\tau)\|^2} > cI$ dla każdego t i pewnego k)

7.1 Pierwszy neuron liniowy: Adaline
7.2 Uczenie sieci liniowych
7.3 Rozwiązanie analityczne

Minimalizacja błędu kwadratowego

Rozwiązanie analityczne

- błąd kwadratowy wyjścia sieci

$$R_{\text{srkw}}(\boldsymbol{w}) = \frac{1}{2}\sum_{t=1}^{T} |y^o(t) - \boldsymbol{w}^\mathsf{T}\boldsymbol{u}(t)|^2 = \frac{1}{2}\sum_{t=1}^{T} |R(t)|^2$$

- gradient i hessian

$$\frac{\mathrm{d}R_{\text{srkw}}(\boldsymbol{w})}{\mathrm{d}\boldsymbol{w}} = -\sum_{t=1}^{T} \boldsymbol{u}(t)\, R(t) = -\sum_{t=1}^{T} \boldsymbol{u}(t)\, y^o(t) + \sum_{t=1}^{T} \boldsymbol{u}(t)\, \boldsymbol{u}(t)^\mathsf{T}\, \boldsymbol{w} = \boldsymbol{0}$$

$$\frac{\mathrm{d}^2 R_{\text{srkw}}(\boldsymbol{w})}{\mathrm{d}\boldsymbol{w}^2} = \sum_{t=1}^{T} \boldsymbol{u}(t)\, \boldsymbol{u}(t)^\mathsf{T} \succ \boldsymbol{O} \quad \text{(dodatnio-określona)} \quad \textit{warunek jednoznaczności}$$

(!) błąd kwadratowy jest minimalizowany przez *wagi optymalne*

$$\boldsymbol{w}^o = \left(\sum_{t=1}^{T} \boldsymbol{u}(t)\, \boldsymbol{u}(t)^\mathsf{T}\right)^{-1} \sum_{t=1}^{T} \boldsymbol{u}(t)\, y^o(t)$$

$$= \widehat{R}_{\boldsymbol{u}}^{-1}\, \widehat{R}_{\boldsymbol{u},y^o} \quad \text{empiryczne macierze korelacji}$$

Minimalizacja błędu kwadratowego

Dodatkowa informacja o wagach

- oszacowanie wag a priori $\boldsymbol{w}_0$, jakość oszacowania $\boldsymbol{P}_0^{-1}$ ($\boldsymbol{P}_0 = \boldsymbol{P}_0^{\mathsf{T}}$, $\boldsymbol{P}_0 \succ \boldsymbol{O}$)
- wskaźnik błędu uwzględniający oszacowaniem a'priori

$$R_{\text{a priori}}(\boldsymbol{w}) = R_{\text{srkw}}(\boldsymbol{w}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_0)^{\mathsf{T}} \boldsymbol{P}_0^{-1} (\boldsymbol{w} - \boldsymbol{w}_0)$$

- wagi optymalne z uwzględnieniem informacji a priori

$$\boldsymbol{w}^o = \left(\boldsymbol{P}_0^{-1} + \underbrace{\sum_{t=1}^{T} \boldsymbol{u}(t)\, \boldsymbol{u}(t)^{\mathsf{T}}}_{T\, \widehat{R}_{\boldsymbol{u}}}\right)^{-1} \left(\boldsymbol{P}_0^{-1} \boldsymbol{w}_0 + \underbrace{\sum_{t=1}^{T} \boldsymbol{u}(t)\, x^o(T)}_{T\, \widehat{R}_{\boldsymbol{u}, x^o}}\right)$$

$$= \left(I + \boldsymbol{P}_0 \sum_{t=1}^{T} \boldsymbol{u}(t)\, \boldsymbol{u}(t)^{\mathsf{T}}\right)^{-1} \left(\boldsymbol{w}_0 + \boldsymbol{P}_0 \sum_{t=1}^{T} \boldsymbol{u}(t)\, x^o(T)\right)$$

Minimalizacja błędu kwadratowego

Rekurencyjne najmniejsze kwadraty

- implementacja rekurencyjna z zastosowaniem informacji a priori: przypadek szczególny *filtru Kalmana*

Rekurencyjne najmniejsze kwadraty

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \boldsymbol{P}(t)\,\boldsymbol{u}(t)\,R(t) \qquad \textit{korekcja wag}$$

$$\boldsymbol{P}(t+1) = \boldsymbol{P}(t) - \frac{\boldsymbol{P}(t)\,\boldsymbol{u}(t)\,\boldsymbol{u}^{\mathsf{T}}(t)\,\boldsymbol{P}(t)}{1+\boldsymbol{u}^{\mathsf{T}}(t)\,\boldsymbol{P}(t)\,\boldsymbol{u}(t)} \qquad \textit{korekcja wzmocnień}$$

$$\boldsymbol{w}(0) = \boldsymbol{w}_0, \quad \boldsymbol{P}(0) = \boldsymbol{P}_0 \succ 0 \qquad \textit{warunki początkowe}$$

- równoważna postać korekcji wzmocnień dla $\boldsymbol{P}(t) \succ 0$

$$\boldsymbol{P}(t)^{-1} = \boldsymbol{u}(t)\,\boldsymbol{u}(t)^{\mathsf{T}} + \boldsymbol{P}(t-1)^{-1} = \boldsymbol{P}_0^{-1} + \sum_{s=1}^{t} \boldsymbol{u}(s)\,\boldsymbol{u}(s)^{\mathsf{T}}$$

korekcja wag wymaga odwracania macierzy

⊛ Rekurencyjne najmniejsze kwadraty

Zbieżność

oznaczmy $\kappa(t) = \max \text{eig}(\boldsymbol{P}(t))$, $\kappa_0 = \max \text{eig}(\boldsymbol{P}_0)$

⊛ Zbieżność rekurencyjnych najmniejszych kwadratów

- norma korekcji wag jest monotonocznie zbieżna do zera

$$\|\boldsymbol{w}(t) - \boldsymbol{w}(t-1)\| \downarrow 0$$

nie implikuje zbieżności wag!

- zbieżność wag:

$$\text{jeżeli } \kappa(t) \longrightarrow 0 \quad \text{wówczas} \quad \boldsymbol{w}(t) \longrightarrow \boldsymbol{w}^o$$

- ważone błędy resztowe dążą do zera

$$\frac{R(t)}{\sqrt{1 + \kappa_0 \|\boldsymbol{u}(t)\|^2}} \longrightarrow 0$$

Trening ADALINE: podsumowanie

algorytm	korekcja wag	zbieżność
μ-LMS	$\Delta \boldsymbol{w}(t) = \mu\, \boldsymbol{u}(t)\, R(t)$	$0 < \mu \leqslant \frac{2}{\mathrm{tr}(\boldsymbol{R}_{\boldsymbol{u}})}$
Kaczmarza	$\Delta \boldsymbol{w}(t) = \frac{1}{\lVert \boldsymbol{u}(t) \rVert^2}\, \boldsymbol{u}(t)\, R(t)$ dla $\lVert \boldsymbol{u}(t) \rVert > 0$	
α-LMS	$\Delta \boldsymbol{w}(t) = \frac{\alpha}{\lVert \boldsymbol{u}(t) \rVert^2}\, \boldsymbol{u}(t)\, R(t)$ dla $\lVert \boldsymbol{u}(t) \rVert > 0$	$0 < \alpha < 2$
α-LMS, modyf.	$\Delta \boldsymbol{w}(t) = \frac{\alpha}{c + \lVert \boldsymbol{u}(t) \rVert^2}\, \boldsymbol{u}(t)\, R(t)$	$c > 0,\ \ 0 < \alpha < 2$
rekurencyjne LS	$\Delta \boldsymbol{w}(t) = \boldsymbol{P}(t)\, \boldsymbol{u}(t)\, R(t)$ $\Delta \boldsymbol{P}(t) = -\frac{\boldsymbol{P}(t)\, \boldsymbol{u}(t)\, \boldsymbol{u}^{\mathsf{T}}(t)\, \boldsymbol{P}(t)}{1 + \boldsymbol{u}^{\mathsf{T}}(t)\, \boldsymbol{P}(t)\, \boldsymbol{u}(t)}\ \ \boldsymbol{P}(0) = \boldsymbol{P}_0 \succ 0$ lub $\Delta(\boldsymbol{P}(t)^{-1}) = \boldsymbol{u}(t)\, \boldsymbol{u}(t)^{\mathsf{T}}$, dla $\boldsymbol{P}(t) \succ 0$	

Literatura

Materiały źródłowe

Widrow, B. and M. E. Hoff (1960). "Adaptive switching circuits". In: *1960 IRE WESCON Convention Record*. Vol. 4. New York, pp. 96–104.

Materiały pomocnicze

Widrow, B. and M. Lehr (Sept. 1990). "30 years of adaptive neural networks: perceptron, madaline, and backpropagation". In: *Proceedings of IEEE* 78.9.

Temat 8: Aproksymacja funkcji

8.1 Problem aproksymacji
8.2 Struktura aproksymatorów
8.3 Możliwości aproksymacyjne sieci neuronowych

8.1 Problem aproksymacji
8.2 Struktura aproksymatorów
8.3 Możliwości aproksymacyjne sieci neuronowych

Zadanie aproksymacji

- *rodzina funkcji aproksymowanych* (rodzina obiektów, rodzina funkcji docelowych) $\mathscr{F}$
  - funkcja aproksymowana (obiekt): funkcja $f \in \mathscr{F}$; w szczególności, funkcja losowa
- *rodzina aproksymatorów* (rodzina hipotez, rodzina modeli, rodzina sieci neuronowych):
  - aproksymator (model obiektu): funkcja ϕ z założonej klasy funkcji Φ
- *koszt aproksymacji* $R(\phi) \geqslant 0$ określa karę za niepodobieństwo aproksymatora ϕ do funkcji aproksymowanej f koszt aproksymacji $\leftrightarrow$

Aproksymacja funkcji

Dla dowolnej funkcji z *rodziny funkcji aproksymowanych* dobrać funkcję z *rodziny aproksymatorów* tak, aby minimalizować *koszt aproksymacji* tych funkcji

- jeśli wymagamy, by niepodobieństwo było zerowe, mówimy o *reprezentacji funkcji*

Aproksymacja

Przykłady

Ⓔ Prognozowanie

- u interpretuje się jako dane dostępne w chwili prognozowania
- funkcja $\phi \in \Phi$ przyporządkowuje tym danym żądaną prognozę

Ⓔ Podejmowanie decyzji / sterowanie

- u interpretuje się jako dane dostępne w chwili analizy systemu
- funkcja $\phi \in \Phi$ przyporządkowuje tym danym decyzję w danej chwili

Klasyfikacja jako zadanie aproksymacji

- wejście $\boldsymbol{u}$ aproksymatora można interpretować jako obraz należący do jednej z rozłącznych klas $\mathcal{C}_1, \ldots, \mathcal{C}_c$
- konstrukcja aproksymatora dla zagadnień klasyfikacji
  - aproksymacja **funkcji klasyfikującej**

$$\phi(\boldsymbol{u}) = k \quad \Longleftrightarrow \quad \boldsymbol{u} \in \mathcal{C}_k$$

  - aproksymacja **funkcji decyzyjnej**

$$\boldsymbol{u} \in \mathcal{C}_k \text{ jeśli } \phi_k(\boldsymbol{u}) = \max_j(\phi_j(\boldsymbol{u}))$$

  - aproksymacja **prawdopodobieństwa przynależności** do klasy

$$\mathscr{P}(\boldsymbol{u} \in \mathcal{C}_k) = \mathrm{e}^{\phi_k(\boldsymbol{u})} \Big/ \sum_j \mathrm{e}^{\phi_j(\boldsymbol{u})}$$

miękkie maksimum (ang. *soft max*)

Klasyfikacja jako zadanie aproksymacji

Przykłady

Ⓔ Przykład: Detekcja (twarzy, sylwetki, przedmiotu, etc.) na obrazie

- $\boldsymbol{u}$ interpretuje się jako obrazy w zbiorze obrazów U
- $\phi(\boldsymbol{u})$ - liczba twarzy na obrazie albo współrzędne ramek zawierających twarze

Ⓔ Przykład: Badanie, czy na dwu obrazach znajduje się element tej samej klasy (np. ta sama osoba)

- para obrazów $\boldsymbol{u}^{(1)}$ i $\boldsymbol{u}^{(2)}$
- $\phi(\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)})$ - prawdopodobieństwo, że obrazy dotyczą tej samej osoby

8.1 Problem aproksymacji
8.2 Struktura aproksymatorów
8.3 Możliwości aproksymacyjne sieci neuronowych

Aproksymatory

Aproksymatory liniowe

- *aproksymator liniowy* funkcji skalarnej wektora $\boldsymbol{u} \in \mathbb{R}^p$

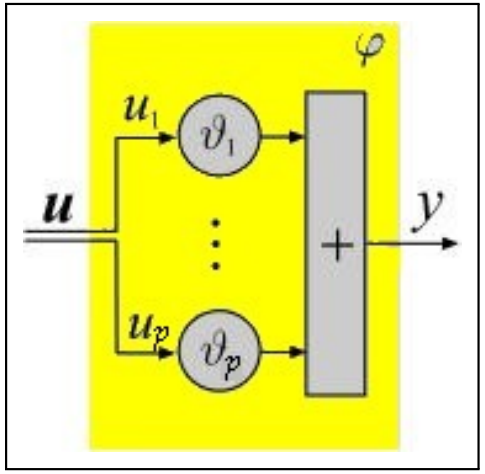
$$\phi(\boldsymbol{u}; \boldsymbol{\vartheta}) = \boldsymbol{\vartheta}^\mathsf{T} \boldsymbol{u} = \sum_{i=1}^{p} \vartheta_i u_i$$

gdzie $\boldsymbol{\vartheta} \in \mathbb{R}^p$ – wektor parametrów

- przykład: Adaline
- *aproksymator liniowy* funkcji wektorowej ma postać *stosu* aproksymatorów liniowych funkcji skalarnych

$$\boldsymbol{\varphi}(\boldsymbol{u}; \boldsymbol{\Theta}) = \begin{bmatrix} \boldsymbol{\vartheta}_1^\mathsf{T} \\ \vdots \\ \boldsymbol{\vartheta}_q^\mathsf{T} \end{bmatrix} \boldsymbol{u} = \boldsymbol{\Theta} \boldsymbol{u}$$

gdzie $\boldsymbol{\Theta} \in \mathbb{R}^p \times \mathbb{R}^h$ - macierz parametrów



⊛ Aproksymatory liniowe

Aproksymatory liniowe wielowymiarowe

- jeśli analizowane sygnały są wielowymiarowe i przedstawione w postaci tensorów (np. dla analizy obrazów RGB) to

$$\phi(\mathbf{U};\boldsymbol{\Theta}) = \boldsymbol{y}_{i_1,..,i_m} = \sum_{j_1,..,j_n} \vartheta_{i_1,..,i_m,j_1,..,j_n} \boldsymbol{U}_{j_1,..,j_n}$$

jest iloczynem wewnętrznym tensora parametrów $\boldsymbol{\Theta}_{i_1,..,i_m,j_1,..,j_n}$ i tensora wejść $\mathbf{U}_{j_1,..,j_n}$, tzn. w notacji Einsteina (i z użyciem multiindeksów)

$$\boldsymbol{y}_{i_1,..,i_m} = \boldsymbol{\Theta}_{\boldsymbol{i},\boldsymbol{j}} \bullet_{\boldsymbol{j}} \mathbf{U}_{\boldsymbol{j}}$$

- tensor parametrów $\boldsymbol{\Theta}_{\boldsymbol{i},\boldsymbol{j}}$ można interpretować jako parametry dla elementu wyjściowego o indeksie $\boldsymbol{i}$
- jeśli tensor parametrów $\boldsymbol{\Theta}_{\boldsymbol{i},\boldsymbol{j}}$ nie zależy od $\boldsymbol{i}$, to parametry modelu są identyczne dla wszystkich indeksów $\boldsymbol{i}$, co odpowiada współdzieleniu parametrów dla sieci głębokich

Aproksymatory

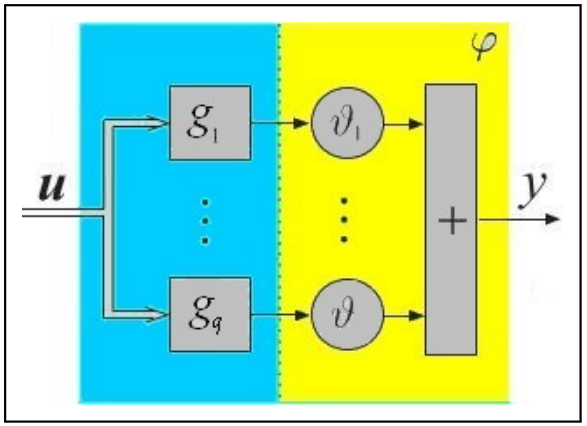
Aproksymatory liniowe względem parametrów

- *aproksymator liniowy względem parametrów funkcji skalarnej* (*struktura Hammersteina*)

$$\phi(\boldsymbol{u};\boldsymbol{\vartheta}) = \boldsymbol{\vartheta}^{\mathsf{T}}\boldsymbol{g}(\boldsymbol{u}) = \sum_{i=1}^{h} \vartheta_i g_i(\boldsymbol{u})$$

gdzie

  - *rozmiar* h i funkcje $g_1, \ldots, g_h$ **są dane**
  - $\boldsymbol{\vartheta} \in \mathbb{R}^h$ jest wektorem parametrów



Ⓔ Przykłady: Modele liniowe względem parametrów

- *model wielomianowy* $\phi(u) = \vartheta_0 + \vartheta_1 u + \ldots + \vartheta_h u^h = \boldsymbol{\vartheta}^{\mathsf{T}}\boldsymbol{\varphi}(u)$ gdzie $\boldsymbol{\varphi}(u) = [1, u, \ldots, u^h]^{\mathsf{T}}$, $\boldsymbol{\vartheta} = [\vartheta_0, \ldots, \vartheta_h]^{\mathsf{T}}$
- *model kwadratowy* $\mathbb{R}^2 \mapsto \mathbb{R}$:
  $\phi(\boldsymbol{u}) = \vartheta_0 + \vartheta_1 u_1 + \vartheta_2 u_2 + \vartheta_{11} u_1^2 + \vartheta_{12} u_1 u_2 + \vartheta_{22} u_2^2 = \boldsymbol{\vartheta}^{\mathsf{T}}\boldsymbol{\varphi}(\boldsymbol{u})$ gdzie
  $\boldsymbol{\varphi}(\boldsymbol{u}) = [1, u_1, u_2, u_1^2, u_1 u_2, u_2^2]^{\mathsf{T}}$, $\boldsymbol{\vartheta} = [\vartheta_0, \vartheta_1, \vartheta_2, \vartheta_{11}, \vartheta_{12}, \vartheta_{22}]^{\mathsf{T}}$

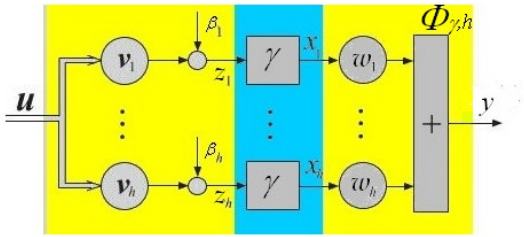
Aproksymatory

Typowa warstwa neuronowa

- *warstwa neuronowa* $\mathbb{R}^p \mapsto \mathbb{R}^q$

$$\varphi(\boldsymbol{u}; \boldsymbol{b}, \boldsymbol{W}) = \begin{bmatrix} g_1(b_1 + \boldsymbol{w}_1^\mathsf{T}\boldsymbol{u}) \\ \vdots \\ g_q(b_q + \boldsymbol{w}_q^\mathsf{T}\boldsymbol{u}) \end{bmatrix} = \boldsymbol{g}(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{u})$$

gdzie



  - wektor funkcji aktywacji $\boldsymbol{g}(\boldsymbol{z}) \stackrel{\text{def}}{=} \begin{bmatrix} g_1(\boldsymbol{z}_1) \\ \vdots \\ g_q(\boldsymbol{z}_q) \end{bmatrix}$ **jest dany**
  - parametry: wektor $\boldsymbol{b} = \begin{bmatrix} b_1^\mathsf{T} \\ \vdots \\ b_q^\mathsf{T} \end{bmatrix} \in \mathbb{R}^q$, macierz $\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^\mathsf{T} \\ \vdots \\ \boldsymbol{w}_q^\mathsf{T} \end{bmatrix} \in \mathbb{R}^q \times \mathbb{R}^p$
- warstwa ma *strukturę Wienera* $\varphi \in \text{NONLIN} \circ \text{LIN}$

Aproksymatory

Podstawowa sieć dwuwarstwowa

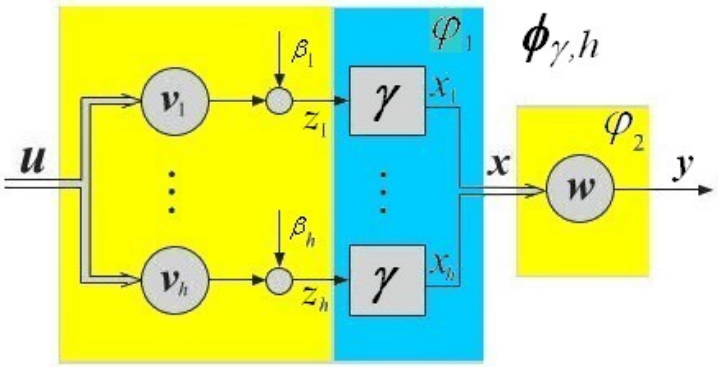
- *podstawowa sieć dwuwarstwowa* $\mathbb{R}^p \mapsto \mathbb{R}^q$

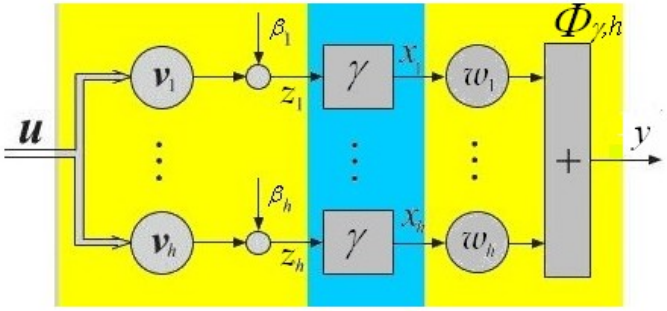
$$\boldsymbol{\varphi}^1(\boldsymbol{u}) = \boldsymbol{g}^1(\boldsymbol{b}^1 + \boldsymbol{W}^1\boldsymbol{u}) \quad \in \mathbb{R}^h \qquad \boldsymbol{\varphi}^1 \in \text{NONLIN} \circ \text{LIN}$$

$$\boldsymbol{\varphi}^2(\boldsymbol{s}) = \boldsymbol{W}^2\boldsymbol{s} \quad \in \mathbb{R}^q \qquad \boldsymbol{\varphi}^2 \in \text{LIN}$$

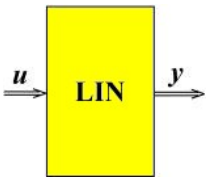
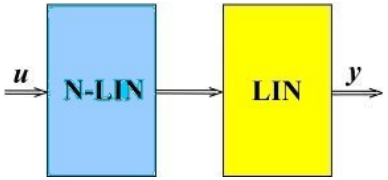
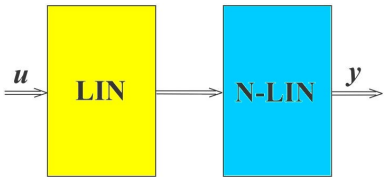
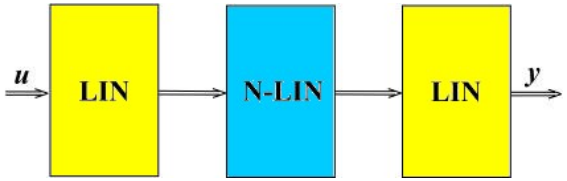
$$\boldsymbol{\varphi}(\boldsymbol{u}) = \boldsymbol{\varphi}^2 \circ \boldsymbol{\varphi}^1(\boldsymbol{u}) = \boldsymbol{W}_2\,\boldsymbol{g}^1(\boldsymbol{b}^1 + \boldsymbol{W}^1\boldsymbol{u}) \qquad \boldsymbol{\varphi} \in \text{LIN} \circ \text{NONLIN} \circ \text{LIN}$$

- złożenie struktury Hammersteina i struktury Wienera
- szczególny przypadek *perceptronu wielowarstwowego* MLP (ang. *multiple layer perceptron*)





Porównanie struktur aproksymatorów

liniowy	*liniowy względem parametrów*	*warstwa*	*podstawowa sieć dwuwarstwowa*
$\widehat{\boldsymbol{y}} = \boldsymbol{\Theta}\boldsymbol{u}$	$\widehat{\boldsymbol{y}} = \boldsymbol{\Theta}\,\boldsymbol{g}(\boldsymbol{u})$	$\widehat{\boldsymbol{y}} = \boldsymbol{g}(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{u})$	$\widehat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{\gamma}(\boldsymbol{\beta} + \boldsymbol{V}\boldsymbol{u})$
		hiperparametry	
—	h i funkcja $\boldsymbol{g} : \mathbb{R}^p \mapsto \mathbb{R}^h$	funkcja aktywacji $\boldsymbol{g} : \mathbb{R}^{p+1} \mapsto \mathbb{R}^q$	h i funkcja aktywacji warstwy ukrytej $\boldsymbol{\gamma} : \mathbb{R}^{h+1} \mapsto \mathbb{R}^q$
		parametry	
$\boldsymbol{\Theta} \in \mathbb{R}^{pq}$	$\boldsymbol{\Theta} \in \mathbb{R}^{hq}$	$(\boldsymbol{b}, \boldsymbol{W}) \in \mathbb{R}^{(p+1)q}$	$(\boldsymbol{\beta}, \boldsymbol{V}, \boldsymbol{W}) \in \mathbb{R}^{(p+1)h+(h+1)q}$
			

Aproksymatory

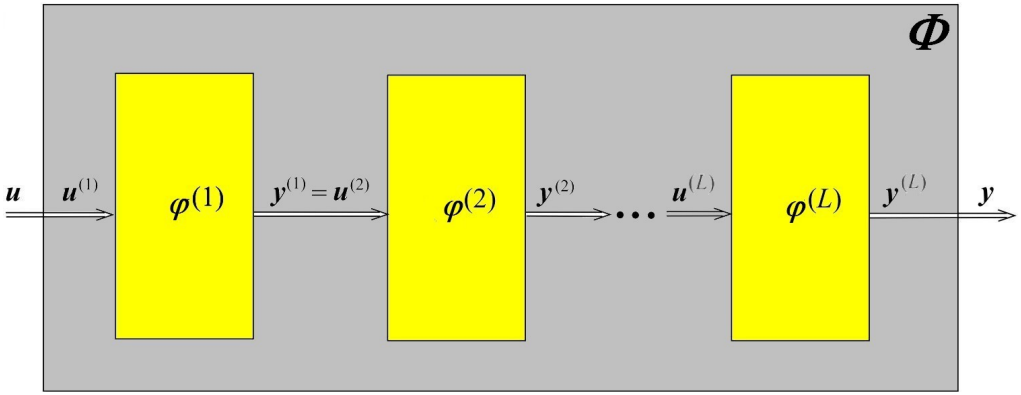
Wielowarstwowy aproksymator neuronowy

- *perceptron wielowarstwowy* jest złożeniem wielu (L) standardowych warstw neuronowych $\boldsymbol{\varphi}(\boldsymbol{x}) = \boldsymbol{g}^j(\boldsymbol{b}^j + \boldsymbol{W}^j\boldsymbol{x})$ dla $j = 1, \ldots, L$

$$\boldsymbol{\varphi}(\boldsymbol{u}; \boldsymbol{\vartheta}) = \boldsymbol{\phi}_{\boldsymbol{\vartheta}}(\boldsymbol{u}) = \boldsymbol{\varphi}_{\boldsymbol{b}^L, \boldsymbol{W}^L} \circ \ldots \circ \boldsymbol{\varphi}_{\boldsymbol{b}^1, \boldsymbol{W}^1}(\boldsymbol{u})$$

gdzie

- $\boldsymbol{\vartheta} = (\boldsymbol{b}^1, \boldsymbol{W}^1, \ldots, \boldsymbol{b}^L, \boldsymbol{W}^L)$ oznacza listę parametrów
- liczba warstw L, rozmiary warstw $q^1, \ldots, q^L$ i funkcje aktywacji warstw $\boldsymbol{g}^1, \ldots, \boldsymbol{g}^L$ są dane



Aproksymatory uniwersalne

Aproksymatory uniwersalne

Φ nazywamy rodziną *uniwersalnych aproksymatorów* dla pewnej klasy funkcji $\mathscr{F}$, jeżeli dla każdej funkcji $f \in \mathscr{F}$ istnieje aproksymator $\phi \in \Phi$ dowolnie jej bliski

tzn. dla każdego $f \in \mathscr{F}$ i dowolnie małego $\epsilon > 0$ istnieje $\phi \in \Phi$ dla którego

$$R(\phi) < \epsilon$$

- czy dla zadanej rodziny $\mathscr{F}$ i ryzyka R, określona rodzina aproksymatorów Φ jest wystarczająco „bogata", żeby być rodziną uniwersalnych aproksymatorów?
- **czy sieci neuronowe są uniwersalnymi aproksymatorami ?**

8.1 Problem aproksymacji
8.2 Struktura aproksymatorów
8.3 Możliwości aproksymacyjne sieci neuronowych

13. Hipoteza Hilberta

- 23 problemy Hilberta, International Congress of Mathematicians, Paryż 1900

13. Hipoteza Hilberta (1900)

Hipoteza: Rozwiązanie równania $x^7 + ax^4 + bx^3 + cx + 1 = 0$ **nie może być** przedstawione w postaci złożenia skończenie wielu funkcji dwóch zmiennych

Uogólnienie 13. Hipotezy Hilberta

Hipoteza: istnieją funkcje ciągłe trzech zmiennych, których **nie można** przedstawić w postaci złożenia skończenie wielu ciągłych funkcji dwu zmiennych

- Kolmogorov (1956): **13. Hipoteza Hilberta jest fałszywa**

Twierdzenie Kołmogorowa-Arnolda o reprezentacji

(Kolmogorov 1956; Kolmogorov 1957; Arnold 1957)
Każdą funkcję ciągłą n zmiennych określoną na kostce n-wymiarowej można przedstawić w postaci złożenia ciągłych funkcji jednej zmiennej i dodawań

approx-nn-back

Reprezentacja Kołmogorowa

! Reprezentacja Kołmogorowa (Kolmogorov 1957; Lorentz 1962)

Każda ciągła funkcja $p > 2$ zmiennych $f : I^p \mapsto \mathbb{R}$, może być przedstawiona jako suma funkcji zmiennych rzeczywistych, z których każda jest sumą „funkcji wagowych" składowych wejściowych

„warstwa wyjściowa": 1 „neuron", $2p+1$ „funkcji wagowych"

$$f(\boldsymbol{u}) = \sum_{i=0}^{2p} g_i(\boldsymbol{x}_i)$$

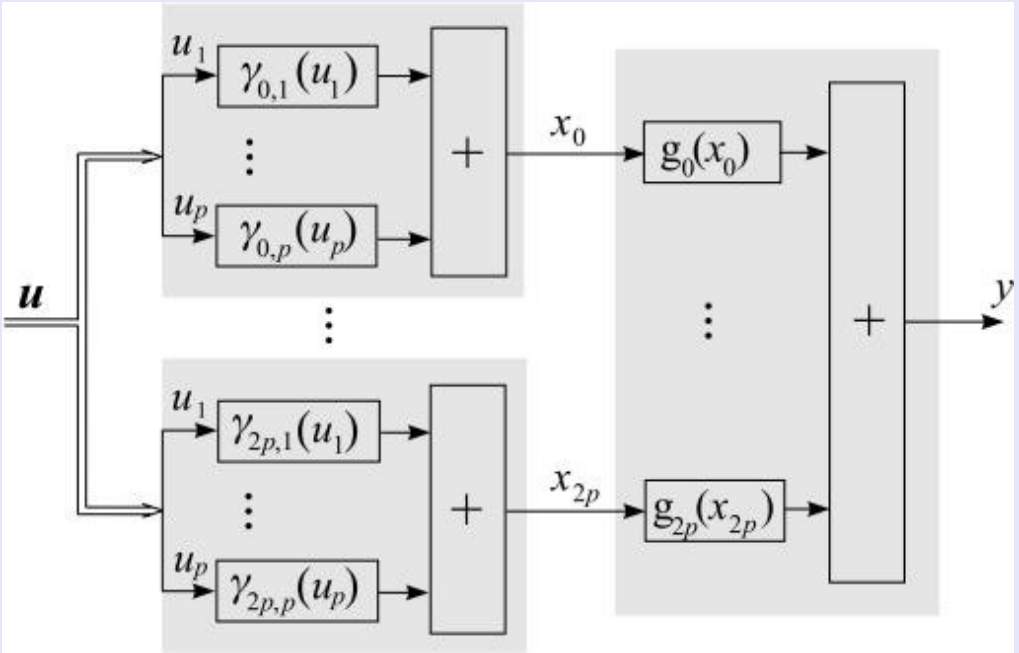
funkcje wagowe ciągłe zależne od f

⇐**Arnold**: funkcje g_i mogą być identyczne

„warstwa ukryta": $2p+1$ „neuronów", $2(p+1)p$ „funkcji wagowych"

$$\boldsymbol{x}_i = \sum_{j=1}^{p} \gamma_{i,j}(u_j), \quad i = 0, \ldots, 2p$$

uniwersalne funkcje monotoniczne ciągłe



warstwa uniwersalna warstwa zależna od f

- reprezentacja Kołmogorowa ma postać zbliżoną do sieci o dwu warstwach ukrytych (Hecht-Nielsen 1987; Hecht-Nielsen 1990)

⊛ Reprezentacja Kołmogorowa

Złożoność reprezentacji (1)

- *funkcja Lipszyca* L_α: istnieje takie $b > 0$, że dla każdego $\boldsymbol{u}$, $\boldsymbol{u}'$

$$|f(\boldsymbol{u}) - f(\boldsymbol{u}')| \leqslant b \, \|\boldsymbol{u} - \boldsymbol{u}'\|^\alpha$$

- jeśli r-te pochodne f są Lipschitza L_α wówczas $\chi = \frac{p}{r+\alpha}$ może być użyte jako miara złożoności funkcji p zmiennych

⊛ Twierdzenie (Vitushkin 1954)

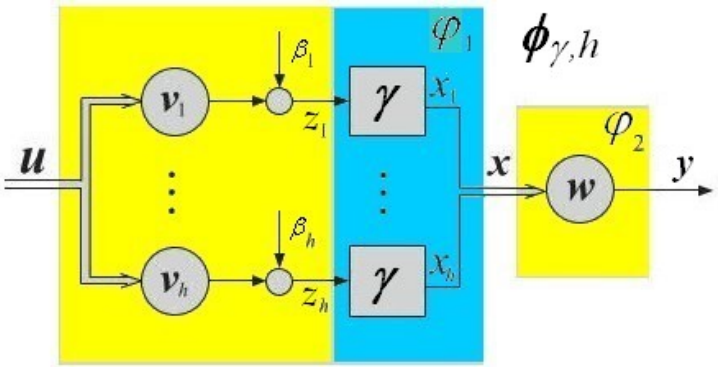
Jeśli $r + \alpha \geqslant 1$ to istnieją funkcje o złożoności χ_0 których **nie można** reprezentować przez superpozycję funkcji o złożoności niższej niż χ_0

- (Girosi and Poggio 1989): Reprezentacja Kołmogorowa nie jest prawdziwa gdy wymaga się, by funkcje uniwersalne $\gamma_{i,j}$ były gładkie
- efektywna parametryzacja funkcji g_i "warstwy wyjściowej" (wagi sieci) nie jest możliwa, bo funkcje g_i nie są gładkie (mają złożoność taką jak f)

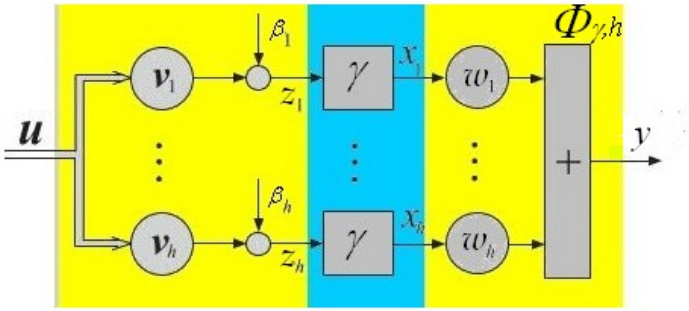
⊛ Twierdzenie (Vitushkin 1954)

Dla $p > 2$ istnieją $r-$krotnie różniczkowalne funkcje p zmiennych, których **nie można** przedstawić jako superpozycję r-krotnie różniczkowalnych w sposób ciągły funkcji mniej niż p zmiennych

Podstawowa sieć dwuwarstwowa



podstawowa sieć dwuwarstwowa



podstawowa sieć dwuwarstwowa jako suma funkcji 1D typu LNL

- *podstawowa sieć dwuwarstwowa* $\phi_{\gamma,h}$: dwuwarstwowy perceptron

$$\phi_{\gamma,h}(\boldsymbol{u}) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\gamma}(\beta + \boldsymbol{V}\boldsymbol{u})$$
$$= \sum_{i=1}^{h} w_i \gamma_i(\beta + \boldsymbol{V}\boldsymbol{u})$$

  którego
  - warstwa ukryta o h neuronach ma identyczne funkcje aktywacji γ
  - warstwa wyjściowa jest liniowa o zerowym obciążeniu
- czy rodzina *podstawowych sieci dwuwarstwowych* $\boldsymbol{\Phi}_{\gamma,h}$ jest rodziną uniwersalnych aproksymatorów?

Aproksymacja funkcji ciągłych

Twierdzenie (Cybenko 1989): Aproksymacja funkcji ciągłych

Jeżeli funkcja aktywacji g jest

ciągła, **sigmoidalna**

wówczas rodzina $\boldsymbol{\Phi}_{\gamma,h}$ sieci o wystarczająco dużej warstwie ukrytej może dowolnie dobrze (w sensie supremum) aproksymować dowolną funkcję ciągłą na zbiorze domkniętym ograniczonym

tzn. dla dowolnej funkcji f ciągłej na kostce p-wymiarowej U i dowolnego $\epsilon > 0$ istnieje sieć ϕ w $\boldsymbol{\Phi}_{\gamma,h}$ taka, że

$$R(\phi) = \sup_{\boldsymbol{u} \in U} |\phi(\boldsymbol{u}) - f(\boldsymbol{u})| < \epsilon$$

⊛ dowód korzysta z tw. Banacha-Hahna

Aproksymacja funkcji ciągłych

Twierdzenie (Hornik 1991) - Aproksymacja funkcji ciągłych

jeżeli funkcja aktywacji g jest

ciągła, **ograniczona** i **różna od stałej**

wówczas rodzina $\boldsymbol{\Phi}_{\gamma,h}$ sieci o wystarczająco dużej warstwie ukrytej może dowolnie dobrze (w sensie supremum) aproksymować dowolną funkcję ciągłą na kostce p-wymiarowej

tzn. dla dowolnej funkcji f ciągłej na kostce p-wymiarowej U i dowolnego $\epsilon > 0$ istnieje sieć ϕ w $\boldsymbol{\Phi}_{\gamma,h}$ taka, że

$$R(\phi) = \sup_{\boldsymbol{u} \in U} |\phi(\boldsymbol{u}) - f(\boldsymbol{u})| < \epsilon$$

⊛ Aproksymacja funkcji ciągłych

Warunek konieczny i wystarczający

- funkcję $g : \mathbb{R} \mapsto \mathbb{R}$ nazywamy *lokalnie istotnie ograniczoną*, jeżeli jest *ograniczona prawie wszędzie* (tzn. za wyjątkiem zbioru μ_{L} o zerowej mierze Lebesgue'a) na każdym podzbiorze domkniętym i ograniczonym
- *istotne supremum* funkcji

$$\operatorname*{ess\,sup}_{U} |g(u)| = \inf\Big\{\delta : \mu_{\mathrm{L}}\{s : |g(s)| > \delta\} = 0\Big\}$$

⊛ Twierdzenie (Leshno et al. 1993)

Jeżeli

- funkcja aktywacji g jest lokalnie istotnie ograniczona
- domknięcie zbioru punktów nieciągłości funkcji aktywacji g ma miarę zero

to sieć typu $\boldsymbol{\Phi}_{\gamma,h}$ o dostatecznie dużej warstwie ukrytej może dowolnie dokładnie (w sensie istotnego supremum) aproksymować dowolną funkcję ciągłą na zbiorze domkniętym i ograniczonym **wtedy i tylko wtedy gdy g nie jest wielomianem prawie wszędzie**

⊛ Aproksymacja funkcji wraz z pochodnymi

- jeżeli ϕ aproksymuje f to jej pochodna $\dot{\phi}$ może nie aproksymować $\dot{f}$
- przykład: aproksymacja za pomocą funkcji skokowej
- $\mathscr{C}_\ell(U)$ – rodzina funkcji ℓ-krotnie różniczkowalnych w sposób ciągły na U

⊛ Twierdzenie (Hornik, Stinchcombe, and White 1990)

Jeżeli funkcja aktywacji g jest

ℓ-krotnie różniczkowalna w sposób ciągły, **ograniczona** i **różna od stałej**

to sieć $\boldsymbol{\Phi}_{\gamma,h}$ o wystarczająco dużej warstwie ukrytej może dowolnie dokładnie (w sensie supremum) aproksymować dowolną funkcję i jej pochodne do rzędu ℓ na zbiorze domkniętym ograniczonym U

tzn. dla każdego zbioru domkniętego ograniczonego $U \in \mathbb{R}^p$, każdej funkcji $\phi \in \mathscr{C}_\ell(U)$, i każdego $\epsilon > 0$ istnieje sieć $\boldsymbol{\Phi}_{\gamma,h}$ taka, że

$$R(\phi) = \sup_{\boldsymbol{u} \in U} |D^{\boldsymbol{\ell}}\phi(\boldsymbol{u}) - D^{\boldsymbol{\ell}} f(\boldsymbol{u})| \leqslant \epsilon$$

dla wszystkich pochodnych $D^{\boldsymbol{\ell}} = \frac{\partial^{\ell_1+\cdots+\ell_p}}{\partial u_1^{\ell_1} \ldots \partial u_p^{\ell_p}}$, $0 \leqslant \ell_1 + \cdots + \ell_p \leqslant \ell$

⊛ Aproksymacja funkcji nieciągłych

- μ_{L}: miara Lebesgue'a

⊛ Twierdzenie (Hornik 1991)

Jeżeli funkcja aktywacji g jest

ciągła, **ograniczona** i **różna od stałej**

wówczas sieć $\boldsymbol{\Phi}_{\gamma,h}$ o dostatecznie dużej warstwie ukrytej może dowolnie dokładnie (w sensie supremum) aproksymować dowolną funkcję (mierzalną) w jej dziedzinie z wyjątkiem podzbioru o arbitralnie małej mierze Lebesgue'a

tzn. dla każdego ograniczonego domkniętego zbioru U, każdej (mierzalnej) funkcji określonej na U i dowolnej dokładności $\epsilon > 0$ istnieje

- sieć $\boldsymbol{\Phi}_{\gamma,h}$
- i domknięty ograniczony podzbiór $U_\epsilon \subset U$ dowolnie bliski zbiorowi U ($\mu_{\mathrm{L}}(U - U_\epsilon) \leqslant \epsilon$),

taki, że

$$R(\phi) = \sup_{\boldsymbol{u} \in U_\epsilon} |f(\boldsymbol{u}) - \phi(\boldsymbol{u})| \leqslant \epsilon$$

Aproksymacja funkcji całkowalnych

- aproksymacja funkcji całkowalnych z p-tą potęgą na zbiorze ograniczonym $U \in \mathbb{R}^p$
- p-norma: $\|f\|_p = \left(\int_U |f(\boldsymbol{u})|^p \, d\boldsymbol{u}\right)^{1/p}$

Twierdzenie (Hornik 1991) - Aproksymacja funkcji całkowalnych

Jeśli funkcja aktywacji g jest

ograniczona i **różna od stałej**

wówczas sieć $\boldsymbol{\Phi}_{\gamma,h}$ o dostatecznie dużej warstwie ukrytej może dowolnie dokładnie (w sensie p-normy) aproksymować dowolną funkcję całkowalną z p-tą potęgą na zbiorze ograniczonym

tzn. dla dowolnego ograniczonego zbioru $U \in \mathbb{R}^p$, dowolnej funkcji w $\mathscr{L}_p(U)$ i dowolnego $\epsilon > 0$ istnieje sieć $\boldsymbol{\Phi}_{\gamma,h}$ dla której

$$R(\phi) = \|\phi - f\|_p < \epsilon$$

- jeżeli U jest skończona (lub przeliczalna) twierdzenie pozostaje prawdziwe, przy czym całkę należy rozumieć jako sumowanie

Aproksymacja funkcji losowej

- $\boldsymbol{u}$: zmienna losowa określona na $U \subset \mathbb{R}^p$
- rozkład $\boldsymbol{u}$ określa wagę wejść
- funkcja f zmiennej losowej $\boldsymbol{u}$ jest *rzędu p* jeśli ma skończone momenty rzędu p tzn. $\mathcal{E}|f(\boldsymbol{u})|^p < \infty$ $(1 \leqslant p < \infty)$

Twierdzenie (Hornik 1991) - Aproksymacja funkcji losowych

Jeżeli funkcja aktywacji g jest

ograniczona i **różna od stałej**

wówczas sieć $\boldsymbol{\Phi}_{\gamma,h}$ o dostatecznie dużej warstwie ukrytej może dowolnie dokładnie (w sensie p-tych momentów) aproksymować dowolną funkcję p-tego rzędu zmiennej losowej $\boldsymbol{u}$

tzn. dla dowolnej funkcji rzędu p (tzn. dla której $\mathcal{E}|\phi^o(\boldsymbol{u})|^p < \infty$ i każdego $\epsilon > 0$ istnieje sieć $\boldsymbol{\Phi}_{\gamma,h}$ taka. że

$$R(\phi) = \mathcal{E}|f(\boldsymbol{u}) - \phi(\boldsymbol{u})|^p < \epsilon$$

⊛ Aproksymacja funkcji zmiennej losowej

Warunek konieczny i wystarczający

- $\boldsymbol{u}$ jest ciągłą zmienną losową określoną na zbiorze $U \subset \mathbb{R}^p$
- funkcje aproksymowane: funkcje p-tego rzędu zmiennej losowej $\boldsymbol{u}$, $1 \leqslant p < \infty$

⊛ Leshno et al. (1993)

jeżeli

- funkcja aktywacji g jest lokalnie istotnie ograniczona
- domknięcie zbioru punktów nieciągłości funkcji g ma miarę zero

to sieć $\boldsymbol{\Phi}_{\gamma,h}$ o dostatecznie dużej liczbie neuronów ukrytych może aproksymować dowolnie dokładnie dowolną funkcję p-tego rzędu ciągłej zmiennej losowej $\boldsymbol{u}$ **wtedy i tylko wtedy gdy g nie jest wielomianem prawie wszędzie**

Twierdzenia aproksymacyjne - Podsumowanie

rodzina funkcji aproksymowanych	funkcja aproksymowana		koszt	twierdzenie	autor
	dziedzina	rodzina funkcji			
ciągłe	ograniczona domknięta	$\mathscr{C}$	$\sup_{\boldsymbol{u}} \lvert f(\boldsymbol{u}) - \phi(\boldsymbol{u}) \rvert$	g ciągła, ograniczona na $\mathscr{C}$, nie stała $\Rightarrow \mathcal{A}$	Hornik (1991), Leshno et al. (1993)
różniczkowalna ℓ-krotnie w sposób ciągły	ograniczona domknięta	$\mathscr{C}_\ell$	$\max_{k \leqslant \ell} \sup_{\boldsymbol{u}} \lvert D^k f(\boldsymbol{u}) - D^k \phi(\boldsymbol{u}) \rvert$	g ℓ-krotnie różniczkowalna w sposób ciągły, ograniczona, nie stała $\Rightarrow \mathcal{A}$	HSW 1990
całkowalna	ograniczona	$\mathscr{L}_p$	$\lVert f - \phi \rVert_p$	g ograniczona nie stała $\Rightarrow \mathcal{A}$	Hornik (1991)
dowolna (mierzalna)	ograniczona domknięta	dowolna (mierzalna)	$\sup_{\boldsymbol{u} \in U_\epsilon} \lvert f(\boldsymbol{u}) - \phi(\boldsymbol{u}) \rvert$, gdzie $\mu(U - U_\epsilon) < \epsilon$	g ciągła, ograniczona, nie stała $\Rightarrow \mathcal{A}$	Hornik (1991)
losowe	ograniczone	$\mathscr{E} \lvert f(\boldsymbol{u}) \rvert^p < \infty$	$\mathscr{E} \lvert f - \phi \rvert^p$	g ograniczona, nie stała $\Rightarrow \mathcal{A}$	Hornik (1991), Leshno et al. (1993)

$\mathcal{A}$: aproksymacja; μ: miara Lebesque'a; HSW: Hornik, Stinchcombe, and White (1990)

Podsumowanie

- **sieci neuronowe są uniwersalnymi aproksymatorami i** w szczególności **uniwersalnymi klasyfikatorami**
- mogą one przy spełnieniu słabych dodatkowych warunków dowolnie dokładnie - w określonym sensie - aproksymować funkcje ciągłe, ciągłe wraz z pochodnymi, całkowalne, losowe, mierzalne
- własności aproksymacyjne sformułowane są jednak w *sposób egzystencjalny*, tzn. **wiadomo, że istnieją** sieci aproksymujące z odpowiednio małym kosztem, ale **nie wiadomo jak je konstruować**, a w szczególności jaki powinien być rozmiar warstwy ukrytej
- obliczenie kosztu aproksymacji wymaga znajomości funkcji aproksymowanej **dla wszystkich argumentów** - ale funkcja ta jest znana tylko dla skończenie wielu "przykładów" argumentów i wartości funkcji
- zgodnie z twierdzeniami aproksymacyjnymi, rozmiar warstwy ukrytej powinien być "odpowiednio duży"
  - co się stanie, jeśli jest on "za mały"
  - czy to źle, jeśli jest "za duży"
  - jak rozmiar ten wpływa na jakość aproksymacji
- jeśli nawet znany jest niezbędny rozmiar warstwy ukrytej, jak określić strukturę sieci i jak wyznaczyć parametry sieci ?

zagadnienia te omawiamy w dalszej części wykładu

Literatura

Materiały źródłowe

Arnold, V. I. (1957). "On the functions of three variables". In: *Doklady Akademii Nauk SSSR* 114.4, pp. 679–681. English transl., Amer. Math. Soc. Transl. (2) 28 (1963), 51-54.

Barron, A. (1993). "Universal approximation bounds for superposition of a sigmoidal function". In: *IEEE Transactions on Information Theory* 3, pp. 930–945.

Chen, T. and H. Chen (1993). "Approximations of Continuous Functionals by Neural Networks with Applications to Dynamic Systems". In: *IEEE Transactions on Neural Networks* 4.6, pp. 910–918.

Chen, T., H. Chen, and R. Liu (1995). "Approximation capability in $C(\bar{\mathbb{R}}^n)$ by multilayer feedforward networks and related problems". In: *IEEE Transactions on Neural Networks* 6.1, pp. 25–30.

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2, pp. 303–314.

Funahashi, K. (1989). "On the approximate realizations of continuous mappings by neural networks". In: *Neural Networks* 2.3, pp. 183–192.

Girosi, F. and T. Poggio (1989). "Representation properties of network: Kolmogorov's theorem is irrelevant". In: *Neural Computation* 1, pp. 465–469.

Hornik, K. (1991). "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4, pp. 251–257.

Literatura (cd.)

Hornik, K., M. B. Stinchcombe, and H. White (1989). "Multilayer feedforward networks are univeral approximators". In: *Neural Networks* 2.5, pp. 359–366.

— (1990). "Univeral approximation of an unknown mapping and its derivatives using multilayer neural networks". In: *Neural Networks* 3.5, pp. 551–560.

Kolmogorov, A. N. (1956). "On the representation of continuous functions of several variables by superposition of continuous functions of a smaller number of variables". In: *Doklady Akademii Nauk USSR* 108.2, pp. 179–182.

— (1957). "On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition". In: *Doklady Akademii Nauk USSR* 114.5, pp. 953–956.

Leshno, M. et al. (1993). "Multilayer feedforward networks with a nonpolynomial activation funtion can approximate any function". In: *Neural Networks* 6, pp. 861–867.

Sprecher, D. A. (Mar. 1965). "On the structure of continuous functions of several variables". In: *Trans. Amer. Math. Society* 115, pp. 340–355.

Vitushkin, A. G. (1954). "On Hilbert's thirteenth problem". In: *Doklady Acad. Nauk SSSR* 95, pp. 701–704.

Materiały pomocnicze

Fine, T. L. (1999). *Feedforward Neural Network Methodology*. New York: Springer Verlag.

Literatura (cd.)

Haykin, S. (1999). *Neural Networks. A Comprehensive Foundation*. Second Edition. Upper Saddle River, New Jersey, USA: Prentice Hall, Inc.

Hecht-Nielsen, R. (1987). "Kolmogorov's mapping neural network existence theorem". In: *Proceedings of the IEEE International Conference on Neural Networks*. Vol. I, pp. 11–13.

— (1990). *Neurocomputing*. Reading, Mass.: Addison-Wesley.

Lorentz, G. (1962 1962). "Metric entropy, widths, and superpositions of functions". In: *American Mathematical Monthly* 69.6, pp. 469–485. URL: www.tandfonline.com/doi/abs/%2010.1080/00029890.1962.11989915.

Sandberg, I. W. (May 1991). "Approximation theorems form discrete-time systems". In: *IEEE Trans. Circuits and Systems* 38.5, pp. 564–566.

Widrow, B. and M. Lehr (Sept. 1990). "30 years of adaptive neural networks: perceptron, madaline, and backpropagation". In: *Proceedings of IEEE* 78.9.

Temat 9: Błędy aproksymacji

9.1 Teoretyczny koszt aproksymacji
9.2 Aproksymator bayesowski
9.3 Aproksymator optymalny w zadanej rodzinie
9.4 Aproksymator empiryczny i koszt empiryczny

9.1 Teoretyczny koszt aproksymacji
9.2 Aproksymator bayesowski
9.3 Aproksymator optymalny w zadanej rodzinie
9.4 Aproksymator empiryczny i koszt empiryczny

Niepodobieństwo

Funkcja strat - problem aproksymacji

- *funkcja strat* (ang. *loss function*) ℓ: funkcja określająca niepodobieństwo **dwu skalarów lub wektorów**
- definicja ℓ wykorzystuje zwykle metryki oparte o p-normy $\|\boldsymbol{z}\|_p = \left(\sum_i |z_i|^p\right)^{1/p}$
  - $p = 1$: $\|\boldsymbol{z}\|_1 = \sum_i |z_i|$
  - $p = 2$: $\|\boldsymbol{z}\|_2 = \sqrt{\sum_i |z_i|^2}$
  - $p = \infty$: $\|\boldsymbol{z}\|_\infty = \max_i |z_i|$

(E) przykłady funkcji strat

- *strata bezwzględna* $\ell(\boldsymbol{z}, \boldsymbol{y}) = \sum_{i=1}^n |z_i - y_i|$ (ang. *sum of absolute differences*) SAD
- *strata kwadratowa* $\ell(\boldsymbol{z}, \boldsymbol{y}) = \sum_{i=1}^n |z_i - y_i|^2$ (ang. *square error*)
- *strata średniokwadratowa* $\ell(\boldsymbol{z}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^n |z_i - y_i|^2$ (ang. *mean square error*) MSE
- *strata maksymalna* $\ell(\boldsymbol{z}, \boldsymbol{y}) = \max_i |z_i - y_i|$ (strata L_∞)

(E) przykład: funkcja strat dla wektorów binarnych

$$\ell(\boldsymbol{z}, \boldsymbol{y}) = \begin{cases} 0 & \boldsymbol{z} = \boldsymbol{y} \\ 1 & \text{w przeciwnym przypadku} \end{cases}$$

Niepodobieństwo

Funkcje strat - klasyfikacja

- funkcja aproksymowana może być wektorową funkcją wskaźnikową klas, tzn. $f_i(\boldsymbol{u}) = p_i = \begin{cases} 1 & \text{jeśli } \boldsymbol{u} \in \mathcal{C}_i \\ 0 & \text{dla pozostałych } i \end{cases}$
- jeśli wyjście y klasyfikatora (sieci neuronowej) jest liczbą rzeczywistą, to często stosuje się przekształcenie *softmax* $s : \mathbb{R} \mapsto [0, 1]$

$$\pi_i = s(y_i) = \frac{\mathrm{e}^{y_i}}{\sum_j \mathrm{e}^{y_j}}$$

  sprowadzające wszystkie wyjścia y_i klasyfikatora do empirycznych "prawdopodobieństw" $\pi_i(u)$ klas odpowiadających tym wyjściom
- najczęściej stosowaną funkcją strat (asymetryczną!) w zadaniach klasyfikacji jest *entropia skrośna* (ang. *cross entropy*), będąca wskaźnikiem bliskości rozkładów $\{p_i, i = 1,.., c\}$, $\{\pi_i, i = 1,.., c\}$

$$\ell(\boldsymbol{p}, \boldsymbol{\pi}) = -\sum_{i=1}^{c} p_i \ln \pi_i, \quad p_i \in [0, 1],\ \pi_i \in (0, 1)$$

Modele aproksymacji

Źródła losowości

- rozwiązanie zadań aproksymacji powinna uwzględniać niepewność pochodzącą z kilku źródeł
  - sygnał $\boldsymbol{u}$ (np. obraz) analizowany przez aproksymator nie jest z góry znany
  - wartość funkcji aproksymowanej y odpowiadająca sygnałowi $\boldsymbol{u}$ może być obciążona błędem ("omylny nauczyciel")
  - algorytm uczenia $\mathscr{L}$ uczenia aproksymatora może mieć różną efektywność (np. szybkość uczenia)

(!) zakładamy, że istnieje *mechanizm niepewności* pozwalający na stosowanie otrzymanych wyników w innych zagadnieniach generowanych przez ten sam mechanizm

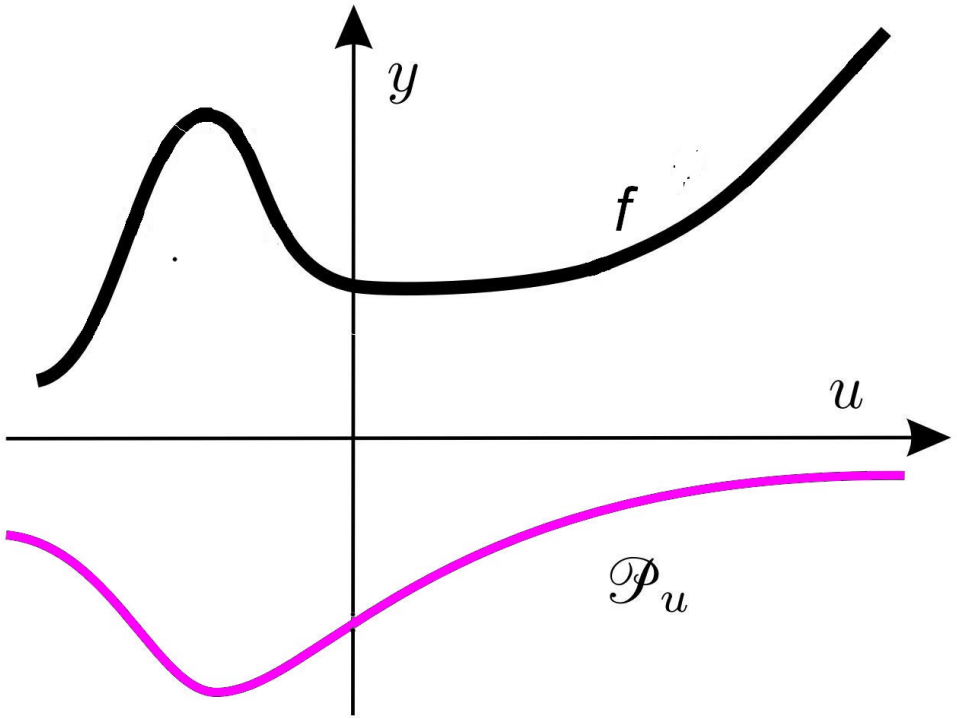
- zakłada się że mechanizm ten jest rządzony przez reguły probabilistyczne, tzn. $(\boldsymbol{u}, y)$ jest zmienną losową

Modele losowości (I)

Bezbłędne obserwacje

- **bezbłędne obserwacje, argumenty losowe**:
  wartości funkcji są losowane ze znanego rozkładu, dla określonej wartości wejścia funkcja aproksymowana jest znana bezbłędnie

$$\boldsymbol{y} \stackrel{\text{def}}{=} f(\boldsymbol{u}), \quad \boldsymbol{u} \sim \mathscr{P}_{\boldsymbol{u}}$$



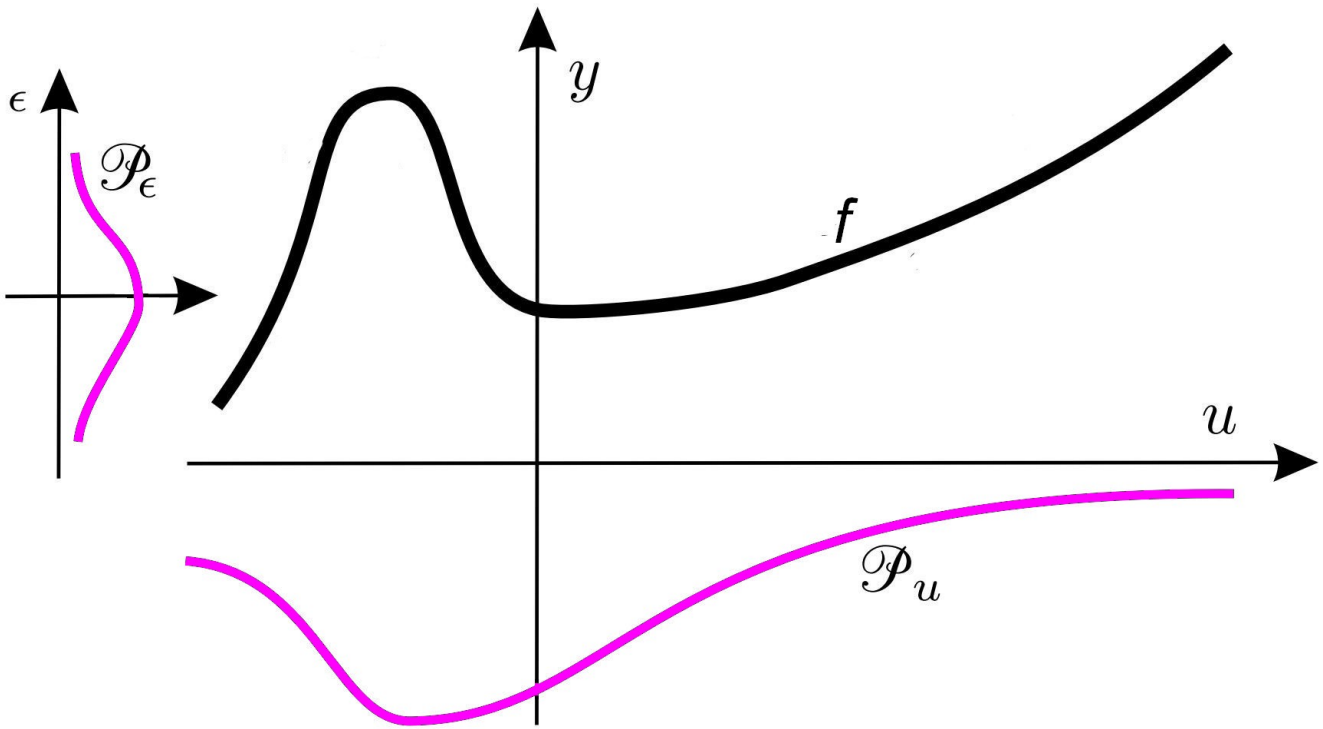
Modele losowości (II)

Niezależne błędy obserwacji

- **niezależne addytywne błędy obserwacji** funkcji f

$$\boldsymbol{y} \stackrel{\text{def}}{=} f(\boldsymbol{u}) + \epsilon, \quad \boldsymbol{u} \sim \mathscr{P}_{\boldsymbol{u}}, \quad \epsilon \sim \mathscr{P}_{\epsilon}, \quad \boldsymbol{u}, \epsilon, \text{niezależne}$$

- $(\boldsymbol{u}, \boldsymbol{y}) \sim \mathscr{P}_{\boldsymbol{u}, f(\boldsymbol{u})+\epsilon}$

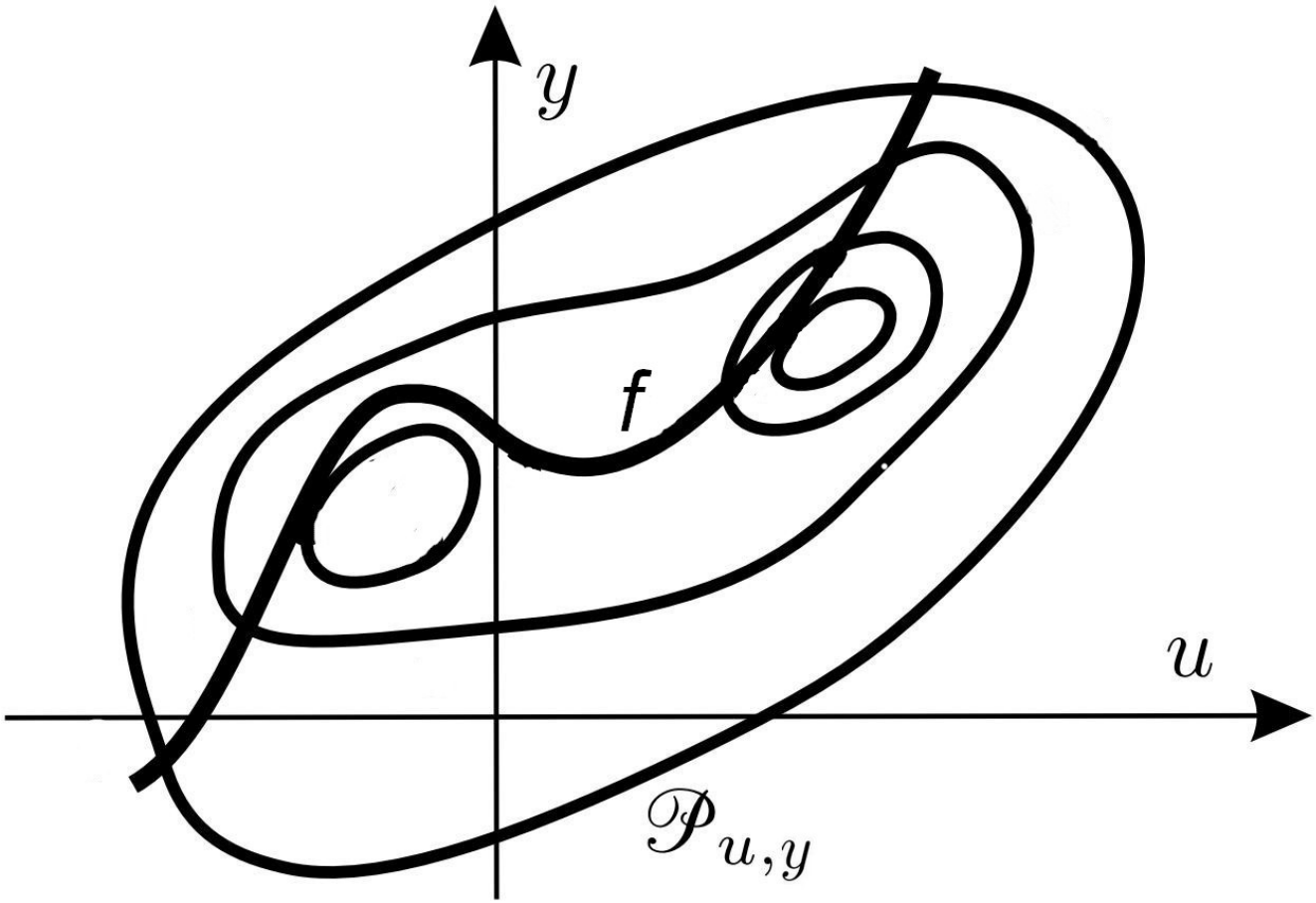


Modele losowości (III)

Zależne błędy obserwacji

- **addytywne błędy obserwacji** funkcji f zależne od argumentu funkcji

$$\boldsymbol{y} \stackrel{\text{def}}{=} f(\boldsymbol{u}) + \epsilon, \quad (\boldsymbol{u}, \epsilon) \sim \mathscr{P}_{\boldsymbol{u},\epsilon}$$

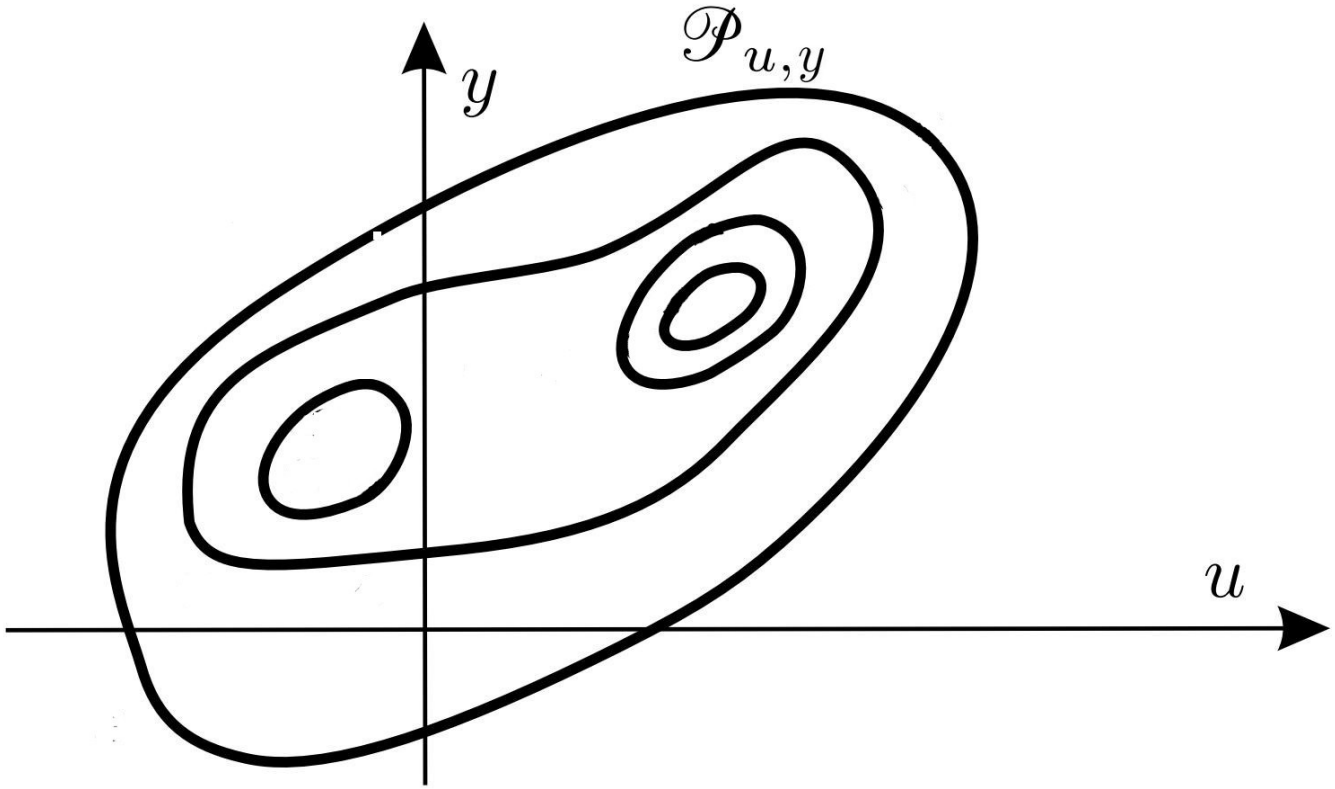


Modele losowości (IV)

Ogólna zależność losowa

- **obserwacje zależności losowej**

$$(\boldsymbol{u}, \boldsymbol{y}) \sim \mathscr{P}_{\boldsymbol{u},\boldsymbol{y}}$$



Koszt aproksymacji

- koszt aproksymacji jest średnią (lub sumą) strat, a więc wpływ losowości jest redukowany

(!) *koszt* (ryzyko) aproksymacji przez aproksymator (np. sieć neuronową) ϕ definiujemy jako **wartość oczekiwaną funkcji strat**

$$R(\phi) \stackrel{\text{def}}{=} \underset{\substack{\uparrow \\ \text{uśrednianie kosztu} \\ \text{(względem } \boldsymbol{u} \text{ i } \boldsymbol{y})}}{\mathscr{E}} \ell\Big(\phi(\boldsymbol{u}), \boldsymbol{y}\Big)$$

$$= \underset{\substack{\uparrow \\ \text{uśrednianie} \\ \text{względem } \boldsymbol{u}}}{\mathscr{E}} \underset{\substack{\uparrow \\ \text{uśrednianie kosztu} \\ \text{dla ustalonego } \boldsymbol{u}}}{\mathscr{E}} \Big(\ell\Big(\phi(\boldsymbol{u}), \boldsymbol{y}\Big) \,|\, \boldsymbol{u}\Big)$$

przypomnienie

$$\mathscr{E}\eta = \mathscr{E}(\mathscr{E}\eta \,|\, \xi)$$

- dalsze rozważania będą dotyczyć głównie kosztów kwadratowych

$$R(\phi) = \mathscr{E}\,\|\phi(\boldsymbol{u}) - \boldsymbol{y}\|^2 = \mathscr{E}\,\mathscr{E}\Big(\|\phi(\boldsymbol{u}) - \boldsymbol{y}\|^2 \,|\, \boldsymbol{u}\Big)$$

tzn. dla dyskretnego zbioru argumentów (obrazów, cech)

$$R(\phi) = \sum p_k \mathscr{E}\Big(\|\phi(\boldsymbol{u}_k) - \boldsymbol{y}_k\|^2 \,|\, \boldsymbol{u}_k\Big)$$

gdzie $p_k = \mathscr{P}(\boldsymbol{u} = \boldsymbol{u}_k) = \mathscr{P}(\boldsymbol{u}_k^k)$

® Koszt aproksymacji

Szczegóły obliczeniowe

⊛ w ogólnym przypadku $R(\phi) = \mathcal{E}\ell\Big(\phi(\boldsymbol{u}), \boldsymbol{y}\Big) = \int \ell\Big(\phi(\boldsymbol{u}), \boldsymbol{y}\Big)\,\mathrm{d}\mathcal{P}_{\boldsymbol{u},\boldsymbol{y}} = \mathcal{E}\mathcal{E}\Big(\ell\Big(\phi(\boldsymbol{u}), \boldsymbol{y}\Big) \mid \boldsymbol{u}\Big)$

$$= \begin{cases} \int f_{\boldsymbol{u}}(\boldsymbol{u}) \underbrace{\int f_{\boldsymbol{y}\mid\boldsymbol{u}}(\boldsymbol{y}\mid\boldsymbol{u})\,\ell\Big(\phi(\boldsymbol{u}), \boldsymbol{y}\Big)\,\mathrm{d}\boldsymbol{y}}_{\mathcal{E}\left(\ell\left(\phi(\boldsymbol{u}),\boldsymbol{y}\right)\mid\boldsymbol{u}\right)}\,\mathrm{d}\boldsymbol{u} & \text{dla ciągłych } \mathcal{P}_{\boldsymbol{u},\boldsymbol{y}} \\ \sum_k p_k \underbrace{\int f_{\boldsymbol{y}\mid\boldsymbol{u}_k}(\boldsymbol{y}\mid\boldsymbol{u}_k)\,\ell\Big(\phi(\boldsymbol{u}_k), \boldsymbol{y}\Big)\,\mathrm{d}\boldsymbol{y}}_{\mathcal{E}\left(\ell\left(\phi(\boldsymbol{u}_k),\boldsymbol{y}\right)\mid\boldsymbol{u}_k\right)} & \text{dla dyskretno/ciągłych } \mathcal{P}_{\boldsymbol{u},\boldsymbol{y}} \\ \sum_k p_k \underbrace{\sum_j p_{j\mid k}\,\ell\Big(\phi(\boldsymbol{u}_k), \boldsymbol{y}_j\Big)}_{\mathcal{E}\left(\ell\left(\phi(\boldsymbol{u}_k),\boldsymbol{y}_j\right)\mid\boldsymbol{u}_k\right)} & \text{dla dyskretnych } \mathcal{P}_{\boldsymbol{u},\boldsymbol{y}} \end{cases}$$

- jeśli w szczególności aproksymowana funkcja f jest znana a nieznany (losowy) jest jej argument $\boldsymbol{u} \sim \mathcal{P}_{\boldsymbol{u}}$ funkcji, to

$$R(\phi) = \mathcal{E}\ell\Big(\phi(\boldsymbol{u}), \boldsymbol{y}\Big) = \int \ell\Big(\phi(\boldsymbol{u}), f(\boldsymbol{u})\Big)\,\mathrm{d}\mathcal{P}_{\boldsymbol{u}}$$

$$= \begin{cases} \int \phi(\boldsymbol{u})\,\ell\Big(\phi(\boldsymbol{u}), f(\boldsymbol{u})\Big)\,\mathrm{d}\boldsymbol{u} & \text{dla ciągłych rozkładów } \mathcal{P}_{\boldsymbol{u}} \\ \sum_k p_k\,\ell\Big(\phi(\boldsymbol{u}_k), f(\boldsymbol{u}_k)\Big) & \text{dla dyskretnych rozkładów } \mathcal{P}_{\boldsymbol{u}} \end{cases}$$

Koszt aproksymacji

Koszt kwadratowy

- dalsze rozważania będą dotyczyć głównie kosztów kwadratowych

$$R(\phi) = \mathscr{E}\,\|\phi(\boldsymbol{u}) - \boldsymbol{y}\|^2 = \mathscr{E}\,\mathscr{E}\left(\|\phi(\boldsymbol{u}) - \boldsymbol{y}\|^2 \mid \boldsymbol{u}\right)$$

tzn. dla dyskretnego zbioru argumentów (obrazów, cech)

$$R(\phi) = \sum_k p_k \mathscr{E}\left(\|\phi(\boldsymbol{u}_k) - \boldsymbol{y}_k\|^2 \mid \boldsymbol{u}_k\right)$$

gdzie $p_k = \mathscr{P}(\boldsymbol{u} = \boldsymbol{u}_k)$; skrótowo zapisujemy warunek $\mid (\boldsymbol{u} = \boldsymbol{u}_k)$ w postaci $\mid \boldsymbol{u}_k$

Koszt kwadratowy aproksymatora

Przykłady

Ⓔ jeśli funkcja aproksymowana f jest znana (model niepewności I) to

$$R_A(\phi) = \sum_k p_k \left\|\phi(\boldsymbol{u}_k) - f(\boldsymbol{u}_k)\right\|^2$$

Ⓔ koszt kwadratowy aproksymatora ϕ dla funkcji aproksymowanej f obarczonej addytywnym niezależnym błędem o zerowej wartości oczekiwanej (model II)

$$\begin{aligned} R(\phi) &= \sum_k p_k \, \mathcal{E}\left(\left\|\phi(\boldsymbol{u}_k) - f(\boldsymbol{u}_k) - \epsilon\right\|^2 \,|\, \boldsymbol{u}_k\right) \\ &= \sum_k p_k \left(\left\|\phi(\boldsymbol{u}_k) - f(\boldsymbol{u}_k)\right\|^2 + \mathcal{E}\,\epsilon^2\right) = R_A(\phi) + \mathcal{V}\epsilon \end{aligned}$$

Ⓔ koszt kwadratowy aproksymatora ϕ dla funkcji aproksymowanej f obarczonej addytywnym błędem (model III) o zerowej warunkowej wartości oczekiwanej $\mathcal{E}\,\epsilon \,|\, \boldsymbol{u}$

$$\begin{aligned} R(\phi) &= \sum_k p_k \, \mathcal{E}\left(\left\|\phi(\boldsymbol{u}_k) - f(\boldsymbol{u}_k) - \epsilon\right\|^2 \,|\, \boldsymbol{u}_k\right) \\ &= \sum_k p_k \left(\left\|\phi(\boldsymbol{u}_k) - f(\boldsymbol{u}_k)\right\|^2 + \mathcal{E}\left(\epsilon^2 \,|\, \boldsymbol{u}_k\right)\right) = R_A(\phi) + \mathcal{V}\epsilon \end{aligned}$$

9.1 Teoretyczny koszt aproksymacji
9.2 Aproksymator bayesowski
9.3 Aproksymator optymalny w zadanej rodzinie
9.4 Aproksymator empiryczny i koszt empiryczny

Aproksymator globalnie optymalny

Aproksymator globalnie optymalny

Aproksymator **globalnie optymalny** funkcji losowej, tzn. optymalny w rodzinie **wszystkich** aproksymatorów, zwany jest *aproksymatorem bayesowskim* ϕ^B

Koszt bayesowski

Koszt bayesowski R^B, tzn. koszt aproksymatora bayesowskiego jest **najniższym możliwym kosztem aproksymacji** zadanej funkcji losowej, zatem koszt (teoretyczny) dowolnego aproksymatora ϕ jest nie mniejszy niż koszt bayesowski

$$0 \leqslant R^B = R(\phi^B) \leqslant R(\phi)$$

Aproksymator bayesowski i koszt bayesowski

! Aproksymator globalnie optymalny

Aproksymator bayesowski funkcji losowej jest równy **warunkowej wartości oczekiwanej aproksymowanej funkcji losowej pod warunkiem jej argumentu**

$$\phi^B(\boldsymbol{u}) = \arg\inf_{\phi} \mathcal{E}\left(\|\phi(\boldsymbol{u}) - \boldsymbol{y}\|^2 \,|\, \boldsymbol{u}\right) = \mathcal{E}\left(\boldsymbol{y} \,|\, \boldsymbol{u}\right)$$

a jego koszt teoretyczny tzn. koszt bayesowski, jest równy **wariancji warunkowej aproksymowanej funkcji pod warunkiem jej argumentu**

$$R^B = R(\phi^B) = \mathcal{E}\mathcal{V}\left(\boldsymbol{y} \,|\, \boldsymbol{u}\right)$$

- aproksymator bayesowski może nie istnieć, albo może nie być jednoznaczny

- uzasadnienie rozwiązania: ozn. $\mathcal{E}_{\boldsymbol{u}}\xi = \mathcal{E}(\xi \,|\, \boldsymbol{u})$ mamy $\inf_\phi R(\phi) = \mathcal{E}\inf_\phi \mathcal{E}_{\boldsymbol{u}}\|\phi(\boldsymbol{u}) - \boldsymbol{y}\|^2$ (bo wyniki dla różnych $\boldsymbol{u}$ nie są powiązane), czyli $\mathcal{E}_{\boldsymbol{u}}\|\phi(\boldsymbol{u}) - \boldsymbol{y}\|^2 = \mathcal{E}_{\boldsymbol{u}}\|\phi(\boldsymbol{u}) - \mathcal{E}_{\boldsymbol{u}}\boldsymbol{y} + \mathcal{E}_{\boldsymbol{u}}\boldsymbol{y} - \boldsymbol{y}\|^2$
$$= \underbrace{\mathcal{E}_{\boldsymbol{u}}\|\phi(\boldsymbol{u}) - \mathcal{E}_{\boldsymbol{u}}\boldsymbol{y}\|^2}_{\substack{\text{minimum dla}\\ \phi(\boldsymbol{u})=\mathcal{E}_{\boldsymbol{u}}\boldsymbol{y}}} + 2\underbrace{\left(\phi(\boldsymbol{u}) - \mathcal{E}_{\boldsymbol{u}}\boldsymbol{y}\right)^{\mathsf{T}}}_{\substack{\text{wyłączone przed } \mathcal{E}_{\boldsymbol{u}}\\ \text{jako funkcja } \boldsymbol{u}}}\underbrace{\mathcal{E}_{\boldsymbol{u}}\left(\mathcal{E}_{\boldsymbol{u}}\boldsymbol{y} - \boldsymbol{y}\right)}_{\substack{=\mathcal{E}_{\boldsymbol{u}}\boldsymbol{y} - \mathcal{E}_{\boldsymbol{u}}\boldsymbol{y}\\ =0}} + \underbrace{\mathcal{E}_{\boldsymbol{u}}\|\boldsymbol{y} - \mathcal{E}_{\boldsymbol{u}}\boldsymbol{y}\|^2}_{\substack{\mathcal{V}_{\boldsymbol{u}}\boldsymbol{y}\\ \text{nie zależy od } \phi}}$$

Aproksymator bayesowski i koszt bayesowski

Aproksymacja funkcji nielosowej

Aproksymacja bayesowska w pełni znanej funkcji nielosowej

Jeśli aproksymowana jest funkcja nielosowa (model niepewności I), tzn. $\boldsymbol{y}$ i $\boldsymbol{u}$ są **funkcyjnie zależne**

$$\boldsymbol{y} = f(\boldsymbol{u})$$

to aproksymator najlepszy globalnie jest identyczny z aproksymowaną funkcją

$$\phi^B(\boldsymbol{u}) = f(\boldsymbol{u})$$

a **koszt bayesowski jest zerowy**

$$R^B = 0$$

Aproksymator bayesowski i koszt bayesowski

Obserwacje obarczone błędem

- w ogólnym przypadku koszt bayesowski jest niezerowy

(E) jeśli obserwacje obciążone są błędem ϵ niezależnym od $\boldsymbol{u}$ (model II), to

$$\phi^B(\boldsymbol{u}) = f(\boldsymbol{u}) + \mathscr{E}\epsilon$$
$$R^B = \mathscr{V}\epsilon$$

(E) jeśli obserwacje obciążone są błędem ϵ zależnym od $\boldsymbol{u}$ (model III), to

$$\phi^B(\boldsymbol{u}) = f(\boldsymbol{u}) + \mathscr{E}\epsilon \,|\, \boldsymbol{u}$$
$$R^B = \mathscr{V}\epsilon \,|\, \boldsymbol{u}$$

(E) klasyfikacja: jeśli klasy są obserwowane bezbłędnie z prawdopodobieństwem p, tzn.

$$\boldsymbol{y} = \begin{cases} f(\boldsymbol{u}) & \text{z prawd. } p \\ 1 - f(\boldsymbol{u}) & \text{z prawd. } 1-p \end{cases}$$

wówczas

$$\mathscr{E}\boldsymbol{y} \,|\, \boldsymbol{u} = f(\boldsymbol{u})p + (1 - f(\boldsymbol{u}))(1-p)$$
$$\mathscr{E}\boldsymbol{y}^2 \,|\, \boldsymbol{u} = f^2(\boldsymbol{u})p + (1 - f(\boldsymbol{u}))^2(1-p)$$

a stąd

$$\phi^B(\boldsymbol{u}) = (2p-1)f(\boldsymbol{u}) + (1-p)$$
$$R^B = p(1-p)$$

9.1 Teoretyczny koszt aproksymacji
9.2 Aproksymator bayesowski
9.3 Aproksymator optymalny w zadanej rodzinie
9.4 Aproksymator empiryczny i koszt empiryczny

Aproksymator optymalny w zadanej rodzinie

- *minimalny koszt teoretyczny* R^* (krótko: koszt minimalny) w rodzinie modeli Φ

$$R^* = \inf_{\phi \in \Phi} R(\phi)$$

- *aproksymator optymalny* w rodzinie Φ

$$\phi^* = \arg\inf_{\phi \in \Phi} R(\phi)$$

- aproksymator optymalny w rodzinie Φ na ogół nie jest jednoznaczny

⊛ własności aproksymatora i kosztu empirycznego są trudne do wyprowadzeń, gdyż - inaczej niż dla aproksymatora bayesowskiego - funkcje ϕ są powiązane ze względu na przynależność do tej samej rodziny Φ

- z definicji minimalny koszt (teoretyczny) spełnia relację

$$0 \leqslant R^B \leqslant R^* \leqslant R(\phi)$$

gdzie ϕ jest dowolnym aproksymatorem z rodziny Φ

Od aproksymatora do aproksymatora empirycznego

- do określenia aproksymatora optymalnego w zadanej rodzinie niezbędna jest znajomość kosztu, który zależy od rozkładu $\mathscr{P}_{u,y}$ (środowiska aproksymacji)
- rozkład $\mathscr{P}_{u,y}$ i wielkości od niego zależne (np. koszt teoretyczny aproksymacji, najlepszy estymator w zadanej rodzinie) **nie są znane** ale mogą być szacowane przez *wartości empiryczne* obliczone na podstawie *obserwacji (próby)*

9.1 Teoretyczny koszt aproksymacji
9.2 Aproksymator bayesowski
9.3 Aproksymator optymalny w zadanej rodzinie
9.4 Aproksymator empiryczny i koszt empiryczny

® Wartości empiryczne

- przez *populację* (populację generalną) rozumiemy zbiór wszystkich **możliwych** obserwacji tej zmiennej losowej
- *próba* $\mathcal{S}$ (zbiór obserwacji) to reprezentatywny podzbiór populacji, tzn. taki, który ma własności populacji
- próba jest *losowa prosta*, tzn. i.i.d (ang. *independent identical distributions*) jeśli każdy element populacji generalnej ma jednakową i niezależną od innych elementów możliwość znalezienia się w próbie, określonej w wyniku losowania wartości tej zmiennej losowej
- dla nieskończonych populacji: własność powyższa ma być prawdziwa dla każdego skończonego podzbioru populacji generalnej
- średnia empiryczna $\mathscr{A}\boldsymbol{\xi}$ zmiennej losowej (funkcji zmiennej losowej) definiowana jest jako jej średnia wartość dla próby $\mathcal{S} = \{\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_N\}$

$$\mathscr{A}\boldsymbol{\xi} = 1/N \sum_{n=1}^{N} \boldsymbol{\xi}_n, \qquad \mathscr{A} f(\boldsymbol{\xi}) = 1/N \sum_{n=1}^{N} f(\boldsymbol{\xi}_n)$$

- wartości empiryczne stosowane są do szacowania wartości oczekiwanych

$$\begin{aligned} \mathscr{A}\boldsymbol{\xi} \quad &\Rightarrow \quad \mathscr{E}\boldsymbol{\xi} \\ \mathscr{A} f(\boldsymbol{\xi}) \quad &\Rightarrow \quad \mathscr{E} f(\boldsymbol{\xi}) \end{aligned}$$

® Wartości empiryczne

- *prawdopodobieństwo empiryczne*: prawdopodobieństwo dowolnego zdarzenia Z jest wartością oczekiwaną funkcji wskaźnikowej $\chi(Z)$ tego zdarzenia

$$\mathscr{P}(Z) = \mathscr{P}(\chi(Z) = 1) = \mathscr{E}(\chi(Z))$$

więc wartością empiryczna prawdopodobieństwa zdarzenia jest częstość empiryczna tego zdarzenia

$$\mathscr{P}Z \quad \Rightarrow \quad \mathscr{A}\chi(Z) = 1/N \sum_{n=1}^{N} \chi(Z_n) = |Z_n|/N$$

gdzie $|A|$ to liczba elementów zbioru A

Potrzeba eksperymentu

- rozkład $\mathscr{P}_{\boldsymbol{u},\boldsymbol{y}}$ i wielkości z nim związane **nie są znane** ale mogą być szacowane przez *wartości empiryczne* obliczone na podstawie próby
- dla sieci neuronowych próba ma zwykle postać *zbioru uczącego*

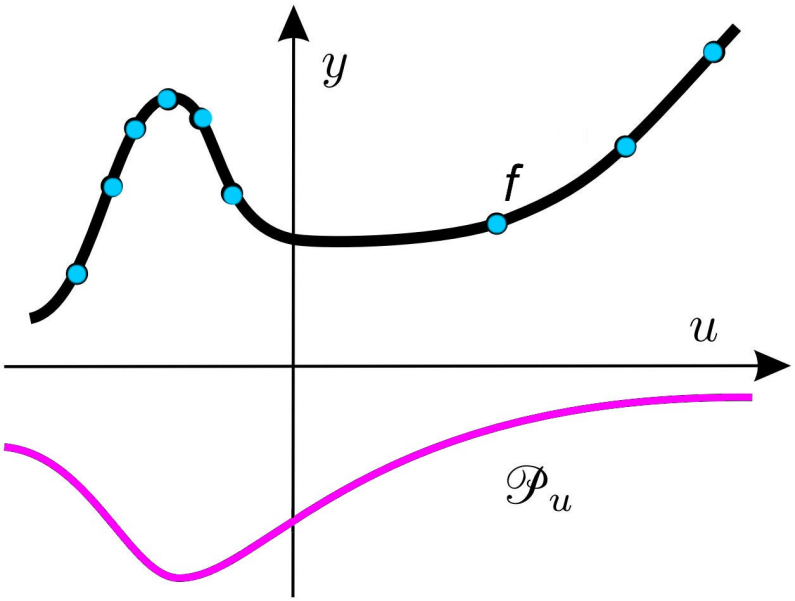
$$\mathcal{S} = \{\big(\boldsymbol{u}_1, \boldsymbol{y}_1\big)\big), \ldots, \big(\boldsymbol{u}_N, \boldsymbol{y}_N\big)\}$$

  tzn. zbioru par (obserwacji wejścia i wyjścia funkcji aproksymowanej) wylosowanych niezależnie z rozkładu środowiska
- dla zbioru uczącego $\mathcal{S}$ mogą być określane m.in.
  - **empiryczna** częstość występowania punktów $\boldsymbol{u}$
  - **empiryczna** częstość pojawiania się par $(\boldsymbol{u}, \boldsymbol{y})$
  - **empiryczne** prawdopodobieństwa warunkowe wyjścia modelu $\boldsymbol{y}$ dla każdego wylosowanego punktu $\boldsymbol{u}$
  - **empiryczne** wersje aproksymatorów i ich empiryczne koszty

Modele obserwacji (I)

Bezbłędne obserwacje

- **bezbłędne obserwacje**: $\boldsymbol{y}_i \stackrel{\text{def}}{=} f(\boldsymbol{u}_n), \quad \underbrace{\boldsymbol{u}_n \sim \mathscr{P}_{\boldsymbol{u}}}_{\substack{\text{dobór } \boldsymbol{u}_n \\ \text{stacjonarny}}}, \quad \boldsymbol{u}_1, \ldots, \boldsymbol{u}_N$ niezależne
  - wartości funkcji f znane tylko dla pewnej liczby (N) losowo wybranych argumentów $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N$
  - reguły doboru argumentów sa określone i niezmienne
  - dobór każdego z argumentów nie wpływa na dobór pozostałych



Modele obserwacji (II)

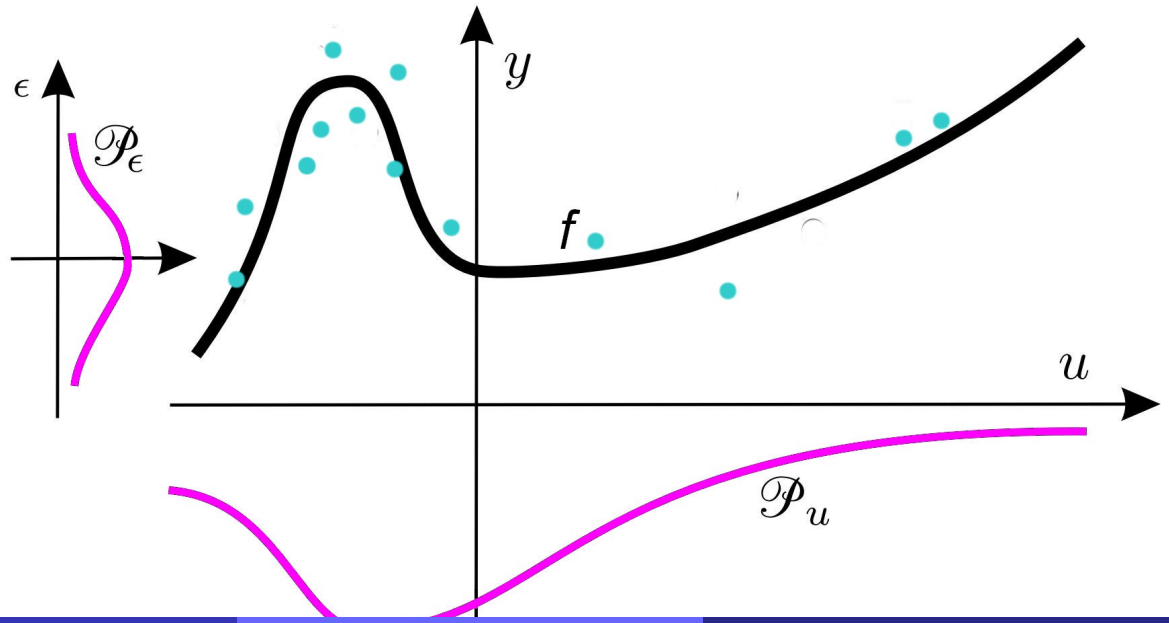
Niezależne błędy obserwacji

- **niezależne addytywne błędy obserwacji** funkcji f

$$\boldsymbol{y}_n \stackrel{\text{def}}{=} f(\boldsymbol{u}_n) + \epsilon_n, \quad \underbrace{\boldsymbol{u}_n \sim \mathscr{P}_{\boldsymbol{u}}}_{\substack{\text{dobór } \boldsymbol{u}_n \\ \text{stacjonarny}}}, \quad \underbrace{\epsilon_n \sim \mathscr{P}_{\epsilon}}_{\substack{\text{mechanizm} \\ \text{błędów} \\ \text{stacjonarny}}},$$

$\boldsymbol{u}_1, \epsilon_1, \ldots, \boldsymbol{u}_N, \epsilon_N$ niezależne

- $\left(\boldsymbol{u}_n, \boldsymbol{y}_n\right) \sim \mathscr{P}_{\boldsymbol{u}, f(\boldsymbol{u})+\epsilon}$



Modele obserwacji (III)

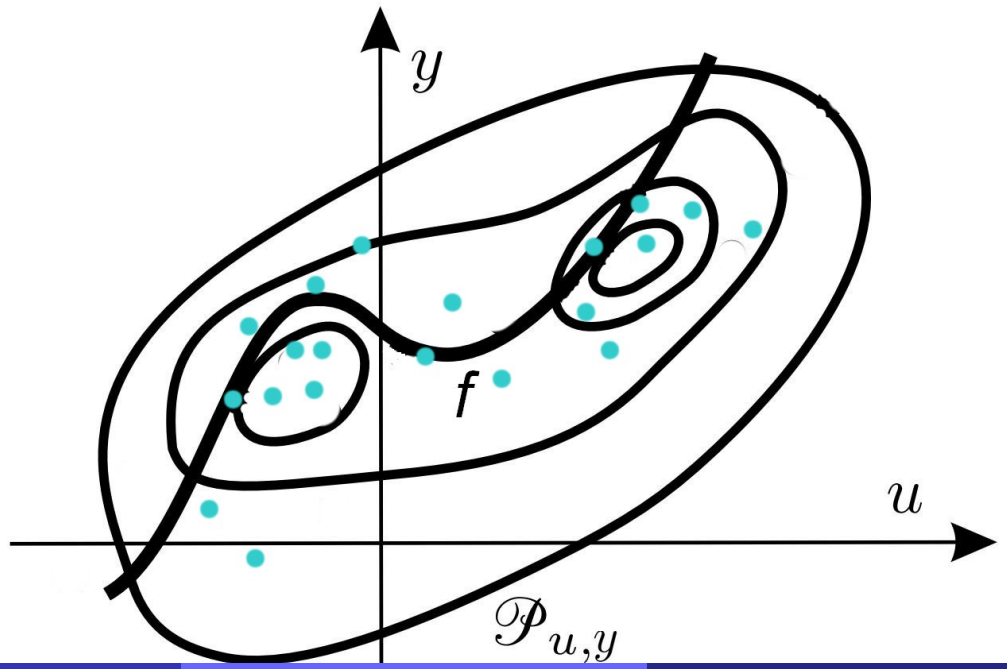
Zależne błędy obserwacji

- **addytywne błędy obserwacji** funkcji f zależne od argumentu funkcji

$$\boldsymbol{y}_n \stackrel{\text{def}}{=} f\big(\boldsymbol{u}_n + \epsilon_n\big), \qquad \underbrace{\big(\boldsymbol{u}_n, \epsilon_n\big) \sim \mathscr{P}_{\boldsymbol{u},\epsilon}}_{\substack{\text{dobór } \boldsymbol{u}_n, \epsilon_n \\ \text{stacjonarny}}}$$

$$(\boldsymbol{u}_1, \epsilon_1), \ldots, (\boldsymbol{u}_N, \epsilon_N), \ldots \quad \text{niezależne}$$

uwaga: czy to samo co w modelu II ?

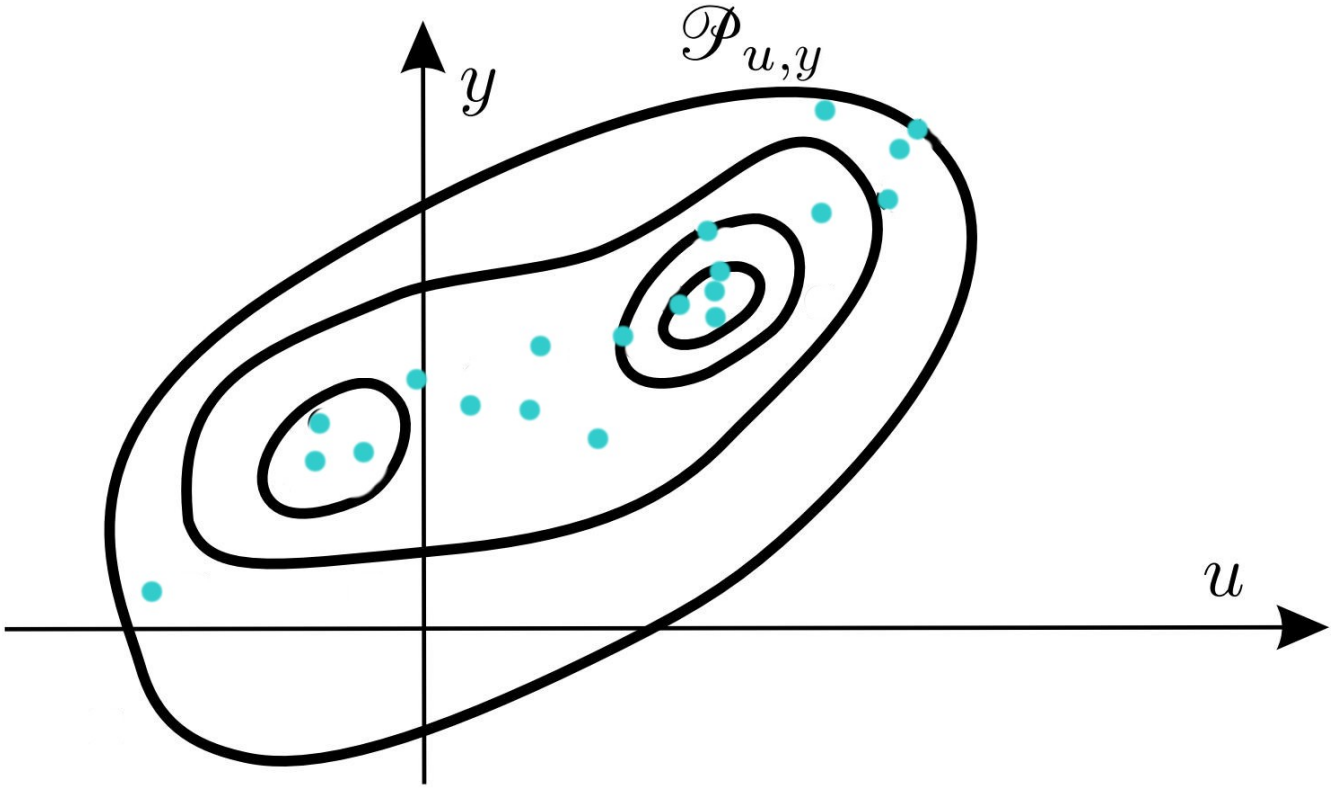


Modele obserwacji (IV)

Ogólna zależność losowa

- **obserwacje zależności losowej**

$$\left(\boldsymbol{u}_n, \boldsymbol{y}_n\right) \sim \mathscr{P}_{\boldsymbol{u},\boldsymbol{y}}, \quad \left(\boldsymbol{u}_1, \boldsymbol{y}_1\right), \ldots, \left(\boldsymbol{u}_N, \boldsymbol{y}_N\right), \text{ niezależne}$$



Koszt empiryczny

- *koszt empiryczny* (*ryzyko empiryczne*) jest odpowiednikiem kosztu teoretycznego, obliczonym na podstawie zbioru uczącego

$$\underbrace{R_{\mathcal{S}}(\phi)}_{\substack{\text{losowy}\\ \text{(jako funkcja}\\ \text{próby } \mathcal{S})}} = \underset{\substack{\uparrow\\ \text{średnia}\\ \text{dla próby } \mathcal{S}}}{\mathscr{A}}\ell(\phi(\boldsymbol{u}), \boldsymbol{y}) = \frac{1}{N}\sum_{n=1}^{N}\ell\big(\phi(\boldsymbol{u}_n, \boldsymbol{y}_n\big)$$

! Koszt teoretyczny a koszt empiryczny

koszt teoretyczny $R(\phi)$	koszt empiryczny $R_{\mathcal{S}}(\phi)$
źródła losowości: u, y	źródła losowości: $u, y, \mathcal{S}$
deterministyczny	**losowy** (losowe dane)
wartości niedostępne	**realizacje dostępne**
rzeczywisty koszt	estymuje koszt
zależny od rozkładu danych	zależny od danych

Koszt empiryczny

Przykłady

Ⓔ Koszt empiryczny -typowe przypadki

- dla aproksymacji funkcji

$$R(\phi) = \mathscr{E}\,\|\phi(\boldsymbol{u}) - \boldsymbol{y}\|^2 \quad \rightarrow \quad R_{\mathcal{S}}(\phi) = \mathscr{A}\|\phi(\boldsymbol{u}) - \boldsymbol{y}\|^2 = \frac{1}{N}\sum_{n=1}^{N}\|\phi_n - \boldsymbol{y}_n\|^2$$

- w szczególności, dla klasyfikacji obrazów na dwie klasy:

$$R(\phi) = \underbrace{\mathscr{P}\Big(\phi(\boldsymbol{u}) \neq y\Big)}_{\substack{\text{prawdopodobieństwo}\\ \text{błędnej klasyfikacji}\\ 1-p^*}} = \mathscr{E}\Big[\phi(\boldsymbol{u}_n) \neq y_n\Big]$$

$$R_{\mathcal{S}}(\phi) = \underbrace{\mathscr{A}\Big[\phi(\boldsymbol{u}_n) \neq y_n\Big]}_{\substack{\text{częstość}\\ \text{błędnej klasyfikacji}\\ 1-\pi^*}} = \frac{1}{N}\sum_{n=1}^{N}\Big(\phi(\boldsymbol{u}_n) - y_n\Big)^2$$

gdzie dla dowolnego zdania Z oznaczono: $[Z] = \begin{cases} 1 & \text{jeśli } Z \text{ jest prawdziwe} \\ 0 & \text{jeśli } Z \text{ jest fałszywe} \end{cases}$

Aproksymator empiryczny

- *minimalizacja kosztu (ryzyka) empirycznego* ERM (ang. *empirical risk minimization*)

$$\underset{\substack{\uparrow \\ \text{zmienna} \\ \text{losowa}}}{R^*_{\mathcal{S}}} \stackrel{\text{def}}{=} \min_{\phi \in \Phi} \underbrace{R_{\mathcal{S}}(\phi)}_{\text{losowe}}$$

  jest popularną metodą aproksymacji

- *optymalny aproksymator empiryczny* minimalizuje koszt empiryczny w klasie aproksymatorów Φ

$$\underset{\substack{\uparrow \\ \text{zmienna} \\ \text{losowa}}}{\phi^*_{\mathcal{S}}} \stackrel{\text{def}}{=} \arg\min_{\phi \in \Phi} \underbrace{R_{\mathcal{S}}(\phi)}_{\text{losowe}}$$

- optymalny aproksymator empiryczny na ogół nie jest jednoznaczny
- minimalny koszt empiryczny i optymalny aproksymator empiryczny zależą od
  - zbioru uczącego $\mathcal{S}$
  - rodziny Φ aproksymatorów

Koszt a koszt empiryczny

- posługiwanie się kosztem empirycznym w zastępstwie kosztu (teoretycznego) jest uzasadnione, o ile są dostatecznie bliskie sobie

$$|R(\phi) - R_{\mathcal{S}}(\phi)| < \epsilon$$

tzn.

$$\overbrace{R_{\mathcal{S}}(\phi) - \epsilon \leqslant R(\phi)}^{\text{ograniczenie dolne kosztu teoretycznego}} \leqslant R_{\mathcal{S}}(\phi) + \epsilon$$

ograniczenie górne kosztu teoretycznego

- wystarczy, by koszt empiryczny nie był "zbyt optymistyczny", tzn. by ograniczyć oszacowanie górne kosztu teoretycznego

$$R(\phi) - R_{\mathcal{S}}(\phi) \leqslant \epsilon$$

- jeśli powyższy warunek jest spełniony z dostatecznie dużym prawdopodobieństwem (*ufnością*) $1 - \delta$ dla **wszystkich** klasyfikatorów zadanej rodziny

$$\underbrace{\sup_{\phi \in \Phi} \Big(R(\phi) - R_{\mathcal{S}}(\phi) \Big)}_{\textit{luka generalizacji}} \leqslant \epsilon$$

to również koszt optymalnego klasyfikatora empirycznego jest ograniczony

Luka generalizacji

(!) oszacowania luki generalizacji $\sup_{\phi \in \Phi} \Big(R(\phi) - R_{\mathcal{S}}(\phi) \Big)$ mają zwykle jedną z równoważnych postaci:
Dla wszystkich klasyfikatorów danej klasy,

- dowolnego rozmiaru próby $N \geqslant N^o$ i dowolnego poziomu ufności $1 - \delta^o$, luka generalizacji nie przekracza pewnego $\epsilon^o(N^o, \delta^o)$
- i dowolnego poziomu ufności $1 - \delta^o$, luka generalizacji ϵ jest dowolnie mała ($\epsilon < \epsilon^o$), o ile próba jest odpowiednio duża $N > N^o(\delta^o, \epsilon^o)$
- i dowolnego rozmiaru próby N^o, luka generalizacji jest dowolnie mała ($\epsilon < \epsilon^o$) z prawdopodobieństwem co najmniej $1 - \delta^o(N^o, \epsilon^o)$

- **uwaga:** zwykle zakładamy, że próba jest i.i.d., ale losowe modyfikacje obrazów w celu zwiększenia rozmiaru próby powodują, że **próba nie jest i.i.d.**

Oszacowanie luki generalizacji

Jeden klasyfikator

Ⓔ przykład: klasyfikator neuronowy o zadanymi parametrami

Ⓘ dla dowolnych: klasyfikatora, próby i.i.d., poziomu ufności $1-\delta$ i rozmiaru próby N

$$R(\phi) < R_{\mathcal{S}}(\phi) + \sqrt{\ln(1/\delta)\,/(2N)}$$

⊛ powyższa własność może być traktowana jako postać prawa wielkich liczb dla klasyfikatora: dla zadanego klasyfikatora koszt empiryczny jest zbieżny wg. prawdopodobieństwa z rozmiarem próby do kosztu teoretycznego $R_{\mathcal{S}}(\phi) \xrightarrow[N\to\infty]{\mathscr{P}} R(\phi)$, tzn. koszt empiryczny jest *estymatorem zgodnym* kosztu zadanego klasyfikatora

⊛ rezultat wynika z tw. Hoeffdinga Hoeffding ↔ dotyczącego różnicy między średnią a wartością oczekiwaną dla ciągów binarnych i.i.d.; warunek wielomianowej zależności funkcji μ od $1/\epsilon, 1/\delta$ jest spełniony gdyż $\frac{\ln(1/\delta)}{2\epsilon^2} \leqslant \frac{1}{2}\frac{1}{\delta}(\frac{1}{\epsilon})^2$ jest wielomianem zmiennych $1/\epsilon$, $1/\delta$

- powyższe oszacowania są konserwatywne
- z własności dla jednego klasyfikatora **nie wynika**, że jest spełniona równocześnie dla dowolnej klasy klasyfikatorów

Oszacowanie luki generalizacji

Skończone rodziny klasyfikatorów

Ⓔ przykład skończonej rodziny klasyfikatorów: sieci neuronowe o ν parametrach binarnych (rodzina 2^ν klasyfikatorów)

ⓘ Luka generalizacji skończonej rodziny klasyfikatorów

Luka generalizacji m-elementowej (skończonej!) rodziny klasyfikatorów **jest dowolnie mała** z dowolnie dużym prawdopodobieństwem, o ile rozmiar N próby i.i.d. jest dostatecznie duży, tzn. ↺ szacowanie luki

$N > \frac{\ln(m/\delta)}{2\epsilon^2}$

⊛ alternatywnie: $\delta < m \exp(-2\epsilon^2 N)$

$$R(\phi) \leqslant R_{\mathcal{S}}(\phi) + \sqrt{\ln(m/\delta)/(2N)}$$

⊛ uzasadnienie: oznaczmy $B_i = \{R(\phi_i^*) - R_{\mathcal{S}}(\phi_i^*) > \epsilon\}$; z poprzedniej własności wynika, że $\mathscr{P}\left(B_1 \cup \ldots \cup B_m\right) \leqslant \sum_{i=1}^n \mathscr{P} B_i = m\,\delta$ czyli z prawdopodobieństwem $1 - m\delta$ zachodzi $\mathscr{P}\left(\forall i : R(\phi^*) - R_{\mathcal{S}}(\phi_i^*)\right) \leqslant \epsilon$

- własności
  - zwiększenie rozmiaru m rodziny klasyfikatorów (wzrost *złożoności* rodziny klasyfikatorów) zwiększa lukę dualności
  - zwiększanie liczby pomiarów przy stałej złożoności m zmniejsza lukę ϵ

⊛ Oszacowanie luki generalizacji

Skończone rodziny klasyfikatorów - Koszt teoretyczny klasyfikatora bezbłędnego empirycznie

⊛ Koszt przy zerowym koszcie empirycznym

Dla m-elementowej (skończonej!) rodziny klasyfikatorów i N-elementowej próby i.i.d., istnienie klasyfikatora ϕ **bezbłędnego empirycznie** o dodatnim koszcie teoretycznym jest dowolnie mało prawdopodobne, a dokładniej

- dla N-elementowej próby i.i.d., klasyfikator (Zhang 2011a)

$$R_{\mathcal{S}}(\phi) = 0 \wedge R(\phi) > \epsilon$$

istnieje w zbiorze m z prawdopodobieństwem co najwyżej $\delta = m \exp(-\epsilon N)$

⊛ uzasadnienie (Scott 2014, Topic 4):

$$\mathscr{P}\left(\exists_{\phi \in m} : R_{\mathcal{S}}(\phi) = 0 \wedge R(\phi) > \epsilon\right) \leqslant m\, \mathscr{P}\{R_{\mathcal{S}}(\phi) = 0 \ \wedge \ R(\phi) > \epsilon\}$$
$$\leqslant \ m\,(1-\epsilon)^N \ \leqslant \ m \exp(-\epsilon N)$$

Oszacowanie luki generalizacji

Problem nieskończonych zbiorów klasyfikatorów

- rozpatrywane dotychczas własności dotyczyły skończonych rodzin klasyfikatorów, tzn. rodzin o skończonej *złożoności* m - **dla nieskończonej rodziny klasyfikatorów wyniki te są bezużyteczne**
- problem: **rodziny sieci neuronowych są (na ogół) nieskończone !!**
- dla nieskończonej rodziny klasyfikatorów złożoność rodziny może być charakteryzowana przez *wymiar Vapnika-Chervonenkisa* d_{VC} (Vapnik and Chervonenkis 1981) **(omówiony dalej)** $\leftrightarrow$
- dla skończonych rodzin $d_{\mathrm{VC}} \leqslant \log_2 m$
- wymiar d_{VC} umożliwia uogólnienie własności aproksymatorów do przypadku nieskończonych rodzin

Oszacowanie luki generalizacji

Wykorzystanie wymiaru VC dla rodzin nieskończonych

(!) Luka generalizacji nieskończonej rodziny klasyfikatorów

Dla nieskończonej rodziny klasyfikatorów **o skończonym wymiarze d_{VC}**, luka generalizacji jest dowolnie mała z dowolnie zadanym prawdopodobieństwem $1 - \delta > 0$, o ile tylko rozmiar próby i.i.d. jest odpowiednio duży, tzn.

dla nieskończonej rodziny klasyfikatorów ϕ o skończonym wymiarze d_{VC} i próby i.i.d. o rozmiarze N, z zadanym prawdopodobieństwem $1 - \delta$

$$R(\phi) \leqslant R_{\mathcal{S}}(\phi) + \sqrt{\frac{1}{N}\left(d_{\mathsf{VC}} \ln \frac{2N\,\mathrm{e}}{d_{\mathsf{VC}}} + \ln \frac{4}{\delta}\right)}$$

- ⊛ dla założonego marginesu (klasyfikatory SVM) można zastąpić d_{VC} przez $d^{\gamma}_{\mathsf{VC}} \leqslant \min(4R^2/\gamma^2, p)$ (p - rozmiar obrazu)
- oszacowania są konserwatywne, tzn. górna granica błędu jest niższa

Klasyfikacja PAC

(!) Uczenie PAC (Valiant 1984)

Metoda klasyfikacji jest *prawdopodobnie w przybliżeniu poprawna* (ang. *probably approximately correct*) PAC dla rodziny klasyfikatorów Φ, jeśli **różnica kosztu i kosztu empirycznego jest dowolnie mała** ($\leqslant \epsilon$) dla **dowolnego klasyfikatora** $\phi \in \Phi$ i **każdej próby niezależnej** zgodnej z **dowolnym rozkładem** $\mathscr{P}$, tzn.

$$\sup_{\phi \in \Phi} R(\phi) - R_{\mathcal{S}}(\phi) \leqslant \epsilon$$

z **dowolnie dużym prawdopodobieństwem** $1 - \delta$ o ile tylko **rozmiar próby jest odpowiednio duży** $N \geqslant \nu(1/\epsilon, 1/\delta)$ gdzie ν jest pewnym wielomianem.

- powyższy warunek "bliskości" kosztu empirycznego i teoretycznego jest bardzo silny, gdyż
  - musi być spełniony jest równocześnie (supremum w warunku) dla całej klasy aproksymatorów
  - musi być spełniony jest dla próby o dowolnym rozkładzie $\mathscr{P}$ ((ang. *distribution free*))
  - wielkość próby zależy tylko wielomianowo od luki ϵ i prawdopodobieństwa δ

(!) Metoda ERM jest PAC dla skończonych rodzin klasyfikatorów i dla rodzin nieskończonych o skończonym wymiarze d_{VC}

Literatura

Materiały źródłowe

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley.

Materiały pomocnicze

Chun, S. (2014). *ML Study PAC Learning*. URL: https://www.slideshare.net/sanghyukchun/%20pac-learning-42139787.

Guestrin, C. (2007). *VC Dimension*. Machine Learning – 10701/15781.

Moore, A. W. (2001). *VC-dimension for characterizing classifiers*. URL: https://www.cmi.ac.in/~madhavan/courses/datamining13-aug/literature/vc-dimension-tutorial.pdf.

Samir, M. (2016). *Machine Learning Theory - Part 2: Generalization Bounds*. URL: https://mostafa-samir.github.io/ml-theory-pt2/.

Scott, C. (2014). *Statistical Learning Theory*. Class material EECS598. University of Michigan. URL: http://web.eecs.umich.edu/~cscott.

Valiant, L. (1984). "A theory of the learnable". In: *Communications of the ACM* 27.11, pp. 1134–1142. DOI: 10.1145/1968.1972.

Vapnik, V. and A. Chervonenkis (1981). "On the uniform convergence of relative frequencies of events to their probabilities". In: *Theory of Probability and Its Applications* 26.3, pp. 532–553.

Literatura (cd.)

Zhang, Y. (2011a). *Overfitting, PAC Learning, VC Dimension, VC Bounds, Mistake Bounds, Semi-Supervised Learning*. lecture notes Machine Learning 10-701.

— (2011b). *Overfitting, PAC Learning, VC Dimension, VC Bounds, Mistake Bounds, Semi-Supervised Learning*. 10.701 Machine Learning slides. Carneggie Mellon University. URL: https://www.cs.cmu.edu/~tom/10701_sp11/recitations/Recitation_9.pdf.

Temat 10: Generalizacja i dobór modelu

10.1 Koszt generalizacji
10.2 Redukcja nadmiernego dopasowania
10.3 Ryzyko strukturalne
10.4 Wymiar Vapnika-Chervonenkisa

10.1 Koszt generalizacji
10.2 Redukcja nadmiernego dopasowania
10.3 Ryzyko strukturalne
10.4 Wymiar Vapnika-Chervonenkisa

Własności kosztu dla innych prób

- **czy na podstawie próby można wnioskować o własnościach spoza próby** - *problem generalizacji*
  - czy estymator ERM $\phi^*_{\mathcal{S}}$ (sieć neuronowa) otrzymany dla pewnej próby $\mathcal{S}$ minimalizuje również błąd empiryczny $R_{\mathcal{V}}(\phi)$ dla dowolnej innej próby $\mathcal{V}$
  - inaczej: czy aproksymator ERM dla danej próby może być z powodzeniem zastosowany dla innej próby?
  - jak oszacować różnicę między kosztami empirycznymi tych estymatorów
- przez koszt generalizacji rozumiemy **koszt teoretyczny aproksymatora empirycznego**
  - jak koszt generalizacji zależy od liczności rodziny aproksymatorów i od rozmiaru próby?
  - oszacować koszt $R_{\mathcal{V}}(\phi^*_{\mathcal{S}})$ estymatora ERM dla próby walidacyjnej $\mathcal{V}$ (bez testowania na tej próbie)

Koszt teoretyczny estymatora empirycznego

- przypomnienie: koszt teoretyczny ustalonego estymatora jest liczbą nieujemną (nielosową)

$$R(\phi) = \overset{\substack{\text{wartość oczekiwana}\\ \text{względem } \boldsymbol{u},\boldsymbol{y}}}{\mathscr{E}}\Big(\ell(\phi(\boldsymbol{u}), \boldsymbol{y})\Big)$$

(!) Koszt (teoretyczny) optymalnego aproksymatora empirycznego otrzymanego dla znanej próby

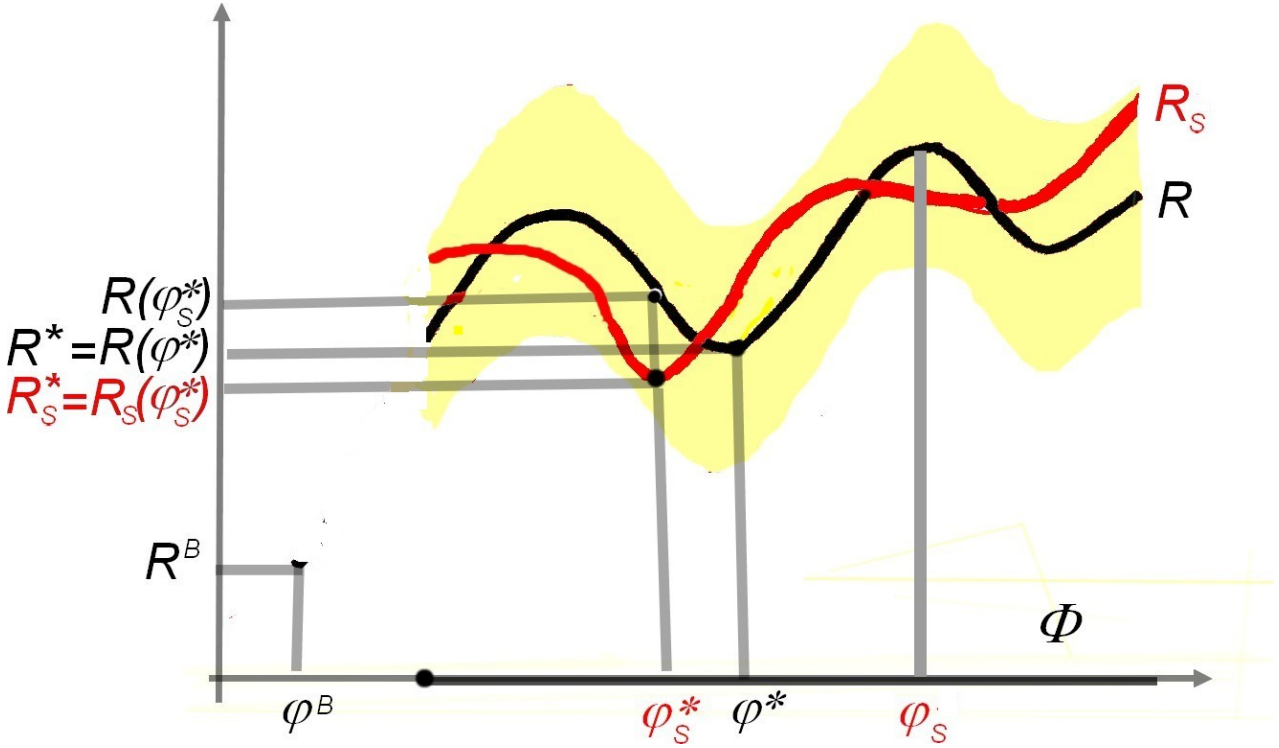
Koszt (teoretyczny) aproksymatora empirycznego $R(\phi_{\mathcal{S}})$ otrzymanego dla określonej próby $\mathcal{S}$

$$R(\phi_{\mathcal{S}}) = R(\phi_{\mathcal{S}} \,|\, \mathcal{S}) = \overset{\substack{\text{wartość oczekiwana}\\ \text{względem } \boldsymbol{u},\boldsymbol{y}\\ \text{dla ustalonego } \mathcal{S}}}{\mathscr{E}}\Big(\ell(\phi_{\mathcal{S}}(\boldsymbol{u}), \boldsymbol{y}) \,|\, \mathcal{S}\Big)$$

jest **liczbą nieujemną** (nielosową).

Struktura kosztu

Interpretacja graficzna



f	(losowa) f. aproksymowana		
ϕ	aproksymator	koszt teoretyczny	$R(\phi)$
ϕ^B	aproks. optymalny globalnie	koszt bayesowski	R^B
ϕ^*	aproks. optymalny w klasie Φ	min. koszt teoretyczny	$R^* = R(\phi^*) = \inf_{\phi\in\Phi} R(\phi)$
$\phi_{\mathcal{S}}$	aproks. empiryczny, próba $\mathcal{S}$	koszt empiryczny	$R_{\mathcal{S}}(\phi_{\mathcal{S}})$
$\phi_{\mathcal{S}}^*$	aproks. optymalny empirycznie	min. koszt empiryczny	$R_{\mathcal{S}}^* = R_{\mathcal{S}}(\phi_{\mathcal{S}}^*) = \inf_{\phi\in\Phi} R_{\mathcal{S}}(\phi)$
-"-	koszt teoretyczny najlepszego estymatora empirycznego		$R(\phi_{\mathcal{S}}^*)$

Struktura kosztu

- Struktura błędu teoretycznego dla optymalnego aproksymatora empirycznego $\phi^*_{\mathcal{S}}$ z zadanej rodziny Φ

$$\underbrace{\overbrace{R(\phi^*_{\mathcal{S}})}^{\text{koszt najlepszego aproksymatora empirycznego}}}_{\text{błąd generalizacji}} = \underbrace{\overbrace{R(\phi^B)}^{\text{koszt globalnie najlepszego aproksymatora}}}_{\substack{\text{nieredukowalny koszt bayesowski} \\ \text{zerowy dla obserwacji bezbłędnych}}} + \underbrace{\overbrace{R(\phi^*)}^{\text{koszt najlepszego aproksymatora w } \Phi} - \overbrace{R(\phi^B)}^{\text{koszt globalnie najlepszego aproksymatora}}}_{\substack{\text{błąd aproksymacji } \Delta_{\text{appr}} \\ \text{„mały” dla „dużej” złożoności klasy } \Phi \\ \text{nie zależy od liczności } N \text{ próby}}} + \underbrace{\overbrace{R(\phi^*_{\mathcal{S}})}^{\text{koszt najlepszego aproksymatora empirycznego}} - \overbrace{R(\phi^*)}^{\text{koszt najlepszego aproksymatora w } \Phi}}_{\substack{\text{błąd estymacji } \Delta_{\text{est}} \\ \text{„duży” dla „dużej” złożoności klasy } \Phi \\ \text{maleje do 0 z licznością } N \text{ próby}}}$$

- omówimy kolejno
  - *błąd aproksymacji*

$$\Delta_{\text{appr}} = R(\phi^*) - R(\phi^B)$$

  - *błąd estymacji*

$$\Delta_{\text{est}} = R(\phi^*_{\mathcal{S}}) - R(\phi^*)$$

  - *błąd (koszt) generalizacji*

$$\Delta_{\text{gen}} = R(\phi^B) + \Delta_{\text{appr}} + \Delta_{\text{est}} = R(\phi^*_{\mathcal{S}})$$

Błąd aproksymacji

- *błąd aproksymacji*: dodatkowy koszt wynikający z zastąpienia aproksymatora najlepszego globalnie przez aproksymator najlepszy w danej rodzinie (T8.3) ↺ approx-nn

  tzn. różnica kosztów (teoretycznych)
  - aproksymatora ϕ^* najlepszego w zadanej rodzinie m
  - i aproksymatora ϕ^B najlepszego globalnie

$$\Delta_{\text{appr}} = R(\phi^*) - R(\phi^B)$$

- w szczególności, dla klasyfikacji binarnej

$$\Delta_{\text{appr}} = \mathscr{P}\left(\phi^*(\boldsymbol{u}) \neq y\right) - \mathscr{P}\left(\phi^B(\boldsymbol{u}) \neq y\right) = p^* - p^B$$

- błąd aproksymacji jest nielosowy i
  - **maleje ze wzrostem złożoności** rodziny estymatorów (d_{VC} lub m)
  - **nie zależy od rozmiaru próby** N

Błąd aproksymacji

Przykład: Problem wielkości rodziny aproksymatorów

Ⓔ Zbyt duża rodzina aproksymatorów uniemożliwia generalizację:
Rozpatrzmy zbiór obrazów o p pikselach i zbiór uczący $\mathcal{S} = \{(\boldsymbol{u}_i, \boldsymbol{y}_i), i = 1, \ldots, N\}$. Łatwo określić klasyfikator w klasie wszystkich możliwych klasyfikatorów, dla którego błąd empiryczny jest zerowy dla dowolnie dużej liczby pomiarów:

$$\phi^B(\boldsymbol{u}) = \begin{cases} \boldsymbol{y}_i & \text{dla } \boldsymbol{u} = \boldsymbol{u}_i,\ i = 1, \ldots, N \\ 0 & \text{dla pozostałych } \boldsymbol{u} \end{cases}$$

Klasyfikator ten jest optymalny dla danych uczących, ale dla innych danych koszt empiryczny może być dowolnie duży, co oznacza brak możliwości generalizacji

- dla rodziny aproksymatorów o skończonym wymiarze VC, i każdego $\epsilon > 0$, istnieje taki aproksymator o odpowiednio dużej złożoności, dla którego błąd wnoszony do błędu generalizacji $R(\phi^*) - R(\phi^B)$ nie przekracza dowolnej wartości $\epsilon > 0$ (uniwersalne aproksymatory)
- z punktu widzenia błędu aproksymacji należy stosować jak najbardziej złożony model (sieć neuronową o dużej liczbie parametrów) **ALE z punktu widzenia generalizacji jest to zła polityka prowadząca do bezużytecznych rozwiązań**

Błąd estymacji

Vapnik and Chervonenkis (1981), Vapnik (1998), Zhang (2011), Moore (2001), Guestrin (2007), Samir (2016), and Chun (2014)

- *błąd estymacji*: różnica **kosztów** (teoretycznych):
  - **kosztu** optymalnego aproksymatora empirycznego w rodzinie m
  - i **kosztu** optymalnego aproksymatora teoretycznego w rodzinie m

  tzn. dodatkowy koszt wynikający z zastąpienia optymalnego aproksymatora teoretycznego w rodzinie m przez optymalny aproksymator empiryczny w tej rodzinie

$$\begin{aligned} R_{\text{est}} &= R(\phi^*_{\mathcal{S}}) - R(\phi^*) \\ &= \mathscr{E}\left\|\phi^*_{\mathcal{S}}(\boldsymbol{u}) - y\right\|^2 - \mathscr{E}\left\|\phi^*(\boldsymbol{u}) - y\right\|^2 \end{aligned}$$

- w szczególności, dla klasyfikacji

$$R_{\text{est}} = \mathscr{P}\left(\phi^*_{\mathcal{S}}(\boldsymbol{u}) \neq y\right) - \mathscr{P}\left(\phi^*(\boldsymbol{u}) \neq y\right) = p^* - \widehat{p}^*$$

- dla ustalonej rodziny klasyfikatorów o skończonym wymiarze VC, koszt ϵ wnoszony przez **estymację** do błędu generalizacji jest dowolnie mały z zadanym prawdopodobieństwem $1 - \delta < 1$, o ile tylko rozmiar próby jest odpowiednio duży (N^o jak dla błędu estymacji)

Błąd estymacji

Własności

- błąd estymacji jest losowy ze względu na losowość doboru elementów próby i
  - **rośnie ze wzrostem złożoności** (d_{VC} lub m) rodziny estymatorów
  - **maleje ze wzrostem rozmiaru próby**
- wzrost złożoności rodziny klasyfikatorów powoduje **zwiększenie** rozmiaru próby gwarantującej odchyłkę ϵ, lub **zwiększenie** odchyłki ϵ dla zadanej wielkości próby

⊛ ponieważ dla dowolnej próby N z zadanym prawdopodobieństwem nie mniejszym niż $1-\delta$ dla dowolnego ϵ zachodzą równocześnie zdarzenia

$$R(\phi^*_{\mathcal{S}}) - R_{\mathcal{S}}(\phi^*_{\mathcal{S}}) \leqslant \epsilon$$

$$R(\phi^*) - R_{\mathcal{S}}(\phi^*) \leqslant \epsilon$$

to również (dodając stronami) zachodzi

$$R(\phi^*_{\mathcal{S}}) \underbrace{-R_{\mathcal{S}}(\phi^*_{\mathcal{S}}) + R_{\mathcal{S}}(\phi^*)}_{\geqslant 0} - R(\phi^*) \leqslant 2\epsilon$$

i w rezultacie otrzymujemy $\mathcal{P}\{R(\phi^*_{\mathcal{S}}) - R(\phi^*) \leqslant 2\epsilon\} > 1-\delta$ zatem - zarówno dla skończonych jak i nieskończonych rodzin klasyfikatorów - wystarczy we wcześniejszych wzorach na lukę generalizacji zastąpić ϵ przez 2ϵ

Błąd generalizacji

- *błąd (koszt) generalizacji* definiujemy jako **koszt teoretyczny aproksymatora empirycznego**

$$\Delta_{\text{gen}} = R(\phi^*_{\mathcal{S}}) = R(\phi_{\mathcal{S}} \,|\, \mathcal{S}) = \mathscr{E}\left(\ell(\phi_{\mathcal{S}}(\boldsymbol{u}), \boldsymbol{y}) \,|\, \mathcal{S}\right)$$

- błąd generalizacji jest sumą błędu bayesowskiego, błędu aproksymacji i błędu estymacji

$$\Delta_{\text{gen}} = R(\phi^B) + \Delta_{\text{appr}} + \Delta_{\text{est}}$$

(!) przy zwiększaniu złożoności rodziny klasyfikatorów, odchyłka wnoszona do błędu generalizacji przez błąd **estymacji** rośnie, a odchyłka wnoszona przez błąd **aproksymacji** maleje

- ta ostatnia własność jest punktem wyjściowym metody estymacji poprzez minimalizację *ryzyka strukturalnego*

Szacowanie błędu generalizacji

Próby walidacyjne

- oszacowanie błędu generalizacji z zastosowaniem wielokrotnych niezależnych *prób walidujących*
  - założymy, że oprócz próby uczącej $\mathcal{S}$ dostępne są niezależne *próby walidujące* $\mathcal{V}_k$, $k = 1, \ldots, n$, losowane z z tego samego rozkładu
  - wykorzystujemy próbę uczącą $\mathcal{S}$ do utworzenia estymatora ERM $\phi_{\mathcal{S}}$
  - stosujemy estymator $\phi_{\mathcal{S}}$ dla każdej próby walidującej $\mathcal{V}_k$ i określamy empiryczny rozkład kosztów empirycznych dla prób walidujących, co pozwala na oszacowanie kosztu teoretycznego i prawdopodobieństwa odchyłek od tego kosztu
  - problem: na ogół brak dostatecznie dużego zbioru walidującego
- założymy, że oprócz próby uczącej $\mathcal{S}$ dostępna jest inna (jedna) niezależna *próba walidująca* $\mathcal{V}$ losowana z tego samego rozkładu przykładów
  - wykorzystujemy próbę uczącą $\mathcal{S}$ do utworzenia estymatora ERM $\phi^*_{\mathcal{S}}$
  - dla próby walidującej $\mathcal{V}$ o liczności $N_{\mathcal{V}}$ obliczamy koszt empiryczny $R_{\mathcal{V}}(\phi^*_{\mathcal{S}})$ dla tej próby
  - z własności luki generalizacji luka-fin wynika, że dla próby $\mathcal{V}$ i skończonego wymiaru DC rodziny klasyfikatorów, $R(\phi^*_{\mathcal{S}})$ i $R_{\mathcal{V}}(\phi^*_{\mathcal{S}})$ różnią się dowolnie mało dla dowolnego poziomu ufności

Szacowanie błędu generalizacji

Repróbkowanie

- modyfikacja technik walidacyjnych - *walidacja skrośna* (ang. *cross-validation*): wielokrotne tworzenie prób walidujących poprzez utworzenie wielokrotnych zestawów zbiór uczący - zbiór walidujący: techniki k-składania (ang. *k-folding*)
  - *k-składanie* (ang. *k-folding*), np. 10-krotna walidacja skrośna dla małych prób
  - techniki (ang. *jackknifing*); metoda *"wydziel jeden"* (ang. *leave-one-out*)
- techniki *bootstrap* losowania próby ze zwracaniem
  - niskie obciążenie i wariancja estymatorów kosztu

Szacowanie błędu generalizacji

Wykorzystanie próby uczącej

- stosuje się również szereg innych metod, np. kryterium informacyjne Akaikego AIC i jego modyfikacje (np. metoda Watanabe-Akaike, metoda Kashyapa), metody bayesowskie (np. bayesowskie kryterium informacyjne BIC), metody największej wiarygodności, metoda Fishera etc.

10.1 Koszt generalizacji
10.2 Redukcja nadmiernego dopasowania
10.3 Ryzyko strukturalne
10.4 Wymiar Vapnika-Chervonenkisa

Zmniejszanie błędu generalizacji

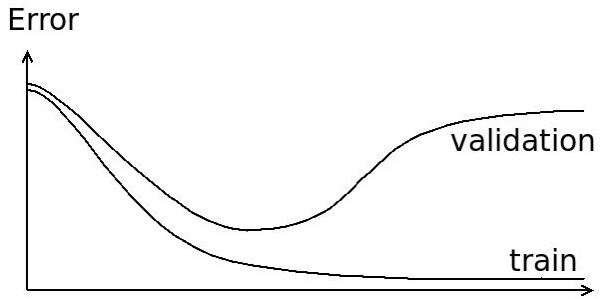
Redukcja nadmiernego dopasowania

- *wczesne zatrzymanie* uczenia
- *regularyzacja* - karanie za zbytni wzrost wag, w tym regularyzacja L1 i L2
- *transformacje danych*
- *lokalne pola receptorów*
- regularyzacja stochastyczna: *odrzucanie wag* (ang. *dropout*)
- aproksymacja: możliwa dla małych sieci, wprowadzanie szumu - uśrednianie modeli o różnej architekturze lub użycie różnych danych uczących
- *współdzielenie wag*; sieci splotowe
- *trening wstępny* (nienadzorowany) w celu doboru rejonu aproksymacji

Zmniejszanie błędu generalizacji

Wczesne zatrzymanie uczenia

- wydzielić zbiory danych: *treningowy*, *walidacyjny* i *testujący* np. w proporcji 60%, 20%, 20%
  1. założyć hiperparametry: strukturę sieci, funkcję kosztu, funkcje aktywacji, parametry minimalizacji, wagi początkowe, etc., oraz algorytm minimalizacji i jego parametry
  2. trenować sieć na zbiorze **treningowym** i obliczać (np. co epokę) błąd empiryczny **walidacji**, zatrzymując uczenie gdy spełniony jest warunek stopu, np. postaci
     - błąd walidacji przestaje maleć
     - błąd walidacji wzrasta w ciągu założonej liczby kroków/mini-porcji/epok
     - błąd uczenia wzrasta ponad założony prog w stosunku do jego wartości minimalnej
  3. jeśli zachodzi potrzeba, ponowić 1. przy innych hiperparametrach
  4. dla wybranych hiperparametrów i czasu minimalizacji wyznaczonym podczas treningu na zbiorze walidacyjnym, trenować sieć na zbiorze **testującym** przez liczbę kroków wyznaczoną podczas **walidacji**
- błąd generalizacji szacować na podstawie błędu **testowania**



Zmniejszanie błędu generalizacji

Regularyzacja

- *regularyzacja*, osłabianie wag (ang. *weights decay*)
- złożoność modelu regulowana przez *człon regularyzacyjny*

$$\widehat{R}^{+}(\phi_{\boldsymbol{w}}) = \widehat{R}(\phi_{\boldsymbol{w}}) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

gdzie $\|\boldsymbol{w}\| = \begin{cases} \sqrt{\boldsymbol{w}^{\mathsf{T}}\boldsymbol{w}} & \text{regularyzacja } L2 \text{ Tichonowa} \\ \sum_i |w_i| & \text{regularyzacja } L1 \\ \max_i |w_i| & \text{regularyzacja } L\infty \end{cases}$

⊛ przy założeniu rozkładu normalnego a'priori wag, regularyzacja L2 odpowiada minimalizacji funkcji wiarygodności: jeśli $\phi_{\boldsymbol{w}} = \mathcal{N}_{m_0,\lambda^{-1}\boldsymbol{I}}$, to $\ln f(\boldsymbol{w}) = \frac{\beta}{2}\sum_{i=1}^{N}\left(\boldsymbol{y}_i - \boldsymbol{w}^{\mathsf{T}}\phi(x_N)\right)^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \mathrm{const}$ jest funkcją wiarygodności

- wersje: regularyzacja odrębna dla wag i obciążeń, ograniczanie zakresu wag, karanie ograniczające aktywacje

Zmniejszanie błędu generalizacji

Niezmiennicze transformacje wejść

- *niezmiennicze transformacje* wejść: dopuszczalne modyfikacje, które nie zmieniają pożądanego wyniku aproksymacji (klasyfikacji), np. "niewielkie" zmiany położenia, wielkości obrazu, lokalnej jasności i in.
- dopuszczalne transformacje zwiększają liczbę danych, a więc polepszają generalizację
- wbudowanie dopuszczalnych transformacji w strukturę sieci: *lokalne pola receptorów*, *propagacja styczna*
- wbudowanie niezmienniczości w sieci: *współdzielenie wag*: sieci splotowe!

⊕ Zmniejszanie błędu generalizacji

Powiększenie zbioru danych

- sposoby powiększenia zbioru danych (ang. *image augmentation*)
  - translacje: losowy wybór 224×224 z 256×256 (2048 modyfikacji)
  - odbicia lustrzane w pionie (2 modyfikacje)
  - zmiany intensywności osi głównych kolorów:
    - $\boldsymbol{c}_j$ - wektor intensywności RGB piksela j obrazu $\boldsymbol{u}$, $j = 1, \ldots J$
    - średnia intensywność RGB: $\overline{\boldsymbol{c}} = 1/J \sum_j \boldsymbol{c}_j$, macierz kowariacji RGB $\boldsymbol{C} = 1/J \sum_j (\boldsymbol{c}_j - \overline{\boldsymbol{c}})(\boldsymbol{c}_j - \overline{\boldsymbol{c}})^\mathsf{T}$
    - PCA: wartości własne $\lambda_1 \leqslant \lambda_2 \leqslant \lambda_3$, macierz wektorów własnych $\boldsymbol{P} = [\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3]$
    - kolory we współrzędnych głównych $\boldsymbol{c}'$ vs. we współrzędnych RGB $\boldsymbol{c}$
      $\boldsymbol{c}' = \boldsymbol{P}^{-1}(\boldsymbol{c} - \overline{\boldsymbol{c}})$
      $\boldsymbol{c} = \overline{\boldsymbol{c}} + \boldsymbol{P}\boldsymbol{c}'$
    - zmodyfikowane intensywności RGB (modyfikacje wartości własnych)

$$\boldsymbol{c}_{\text{mod}\,j} = \overline{\boldsymbol{c}} + \boldsymbol{P}(\boldsymbol{P}^{-1}(\boldsymbol{c} - \overline{\boldsymbol{c}}) + \Delta) = \boldsymbol{c} + \sum_i \lambda_i \alpha_i \boldsymbol{p}_i$$

      gdzie $\alpha_i \sim \mathcal{N}_{0,0.1}$
    - dla identycznych α_i mamy po prostu $\boldsymbol{c}_{\text{mod}} = (1 + \alpha)\boldsymbol{c}$

⊛ Zmniejszanie błędu generalizacji

Propagacja styczna (1)

P. Y. Simard et al. (1998). "Transformation Invariance in Pattern Recognition - Tangent Distance and Tangent Propagation". In: *Neural Networks: Tricks on the Trade*. Vol. 1524. Lectures Notes in Computer Science. Springer-Verlag, pp. 239–274

- rozbudowa zbiorów danych zastąpiona przez wbudowany mechanizm utożsamiania danych poprzez modyfikacje funkcji kosztu
- konstrukcja kosztu: funkcja, dla której odległość między prototypem rodziny a innymi obrazami danej rodziny nie zależy od "nieistotnych" transformacji obrazu (niezmienniczy względem "nieistotnych" transformacji)
  - określamy różniczkowalną transformację obrazów $s(\boldsymbol{u}, \alpha)$ gdzie α jest parametrem, przy czym $s(\boldsymbol{u}, 0) = \boldsymbol{u}$
    - przykładowo, s może być funkcją rozmycia: $s(\boldsymbol{u}, \alpha) = \boldsymbol{u} * \phi_{0,\,\sigma^2}$
  - przekształcenie obrazu w rodzinę obrazów: dla zadanego obrazu $\boldsymbol{u}$, określamy krzywą w przestrzeni U sparametryzowaną przez α

$$\mathcal{S}_{\boldsymbol{u}} = \{\boldsymbol{u}' = s(\boldsymbol{u}, \alpha), \alpha \in \mathbb{R}\} \approx s(\boldsymbol{u}, 0) + \alpha \underbrace{\left.\frac{\partial s(\boldsymbol{u}, \alpha)}{\partial \alpha}\right|_{\alpha=0}}_{T(\boldsymbol{u})} = \boldsymbol{u} + \alpha T(\boldsymbol{u})$$

⊛ Zmniejszanie błędu generalizacji

Propagacja styczna (2)

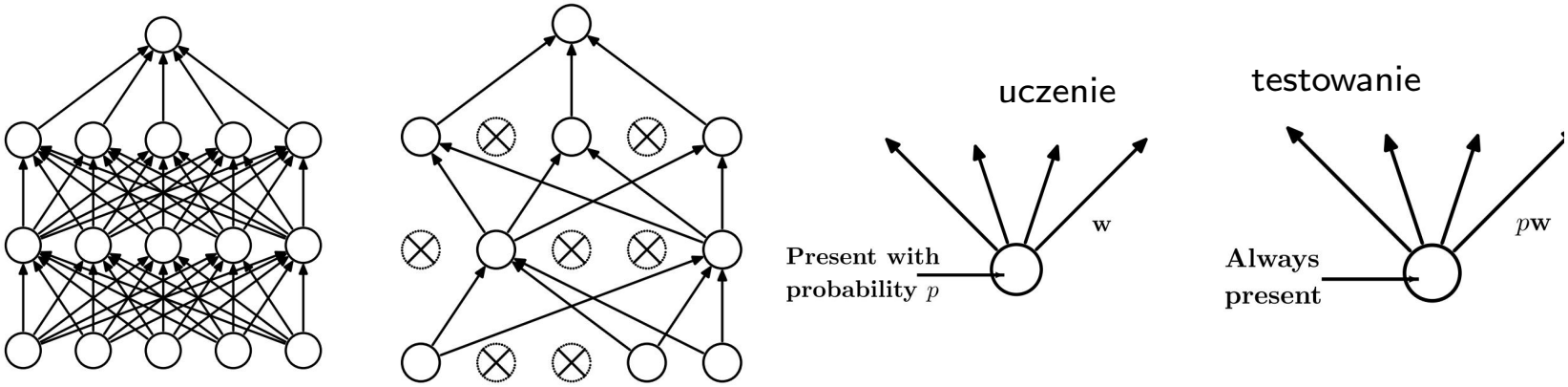
- proces uczenia: *propagacja styczna* (ang. *tangent propagation*)
  - minimalizacja odległości od rodziny obrazów (od krzywej, a nie od pojedynczego obrazu): obliczyć jej płaszczyznę styczną i minimalizować odległość między płaszczyznami
  - wprowadzić człon regularyzacyjny który karze za zmiany wyjścia przy transformacjach wejścia, które przekształcają obrazy w obrazy równoważne pod względem rozpoznawanych treści

Zmniejszanie błędu generalizacji

Odrzucanie wag (dropout)

N. Srivastava et al. (June 2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15, pp. 1929–1958

- podczas treningu losowo i **niezależnie** usuwać neurony ukryte wraz z ich połączeniami
  - prawdopodobieństwo odrzucania: $p \approx 0.2$ dla warstw bliższych wejściu do $p \approx 0.5$ dla bliższych wyjściu
  - w rezultacie otrzymujemy rozkład Bernoulliego aktywnych neuronów
  - uśrednianie wyników chwilowych sieci podczas uczenia
- podczas testowania nie odrzucać wag i stosować pełną sieć z proporcjonalnie zmniejszonymi wagami $p_i w_i$



Srivastava et al. (2014)

Zmniejszanie błędu generalizacji

Dropout - interpretacje

- w jednym kroku uczenia otrzymuje się jedną z 2^{n_ϕ} możliwych sieci z zachowaniem liczby wag, gdzie n_ϕ jest liczbą neuronów ukrytych - **próbkowanie z wykładniczej liczby sieci**
- odrzucanie wag zapobiega nadmiernej wzajemnej adaptacji neuronów podczas uczenia - każdy neuron ukryty musi działać z losowo wybranymi neuronami

⊛ związek z reprodukcją genów

- reprodukcja nieseksualna: przekazywanie zmutowanych kopii genów rodzica; optymalizuje indywidualne przystosowanie poprzez przekazywanie "dobrych" genów.
- reprodukcja seksualna: mutacja genów rodziców: może obniżać indywidualne przystosowanie ale jest ono lepsze przy konieczności działania z innym zbiorem genów (dropout)

- dropout jako *regularyzacja stochastyczna*: losowość działania neuronów ukrytych
- dropout jako minimalizacja wartości oczekiwanej funkcji strat

Nienadzorowany pre-trening

- trening wstępny
  - wprowadzenie właściwych warunków początkowych
  - użycie danych nieetykietowanych
- *autoenkodery*
- *ograniczone maszyny Boltzmanna*
- *głębokie maszyny Bolzmanna*

10.1 Koszt generalizacji
10.2 Redukcja nadmiernego dopasowania
10.3 Ryzyko strukturalne
10.4 Wymiar Vapnika-Chervonenkisa

Koszt a koszt empiryczny

Nadmierne dopasowanie

Definicja ***Nadmierne dopasowanie***

Aproksymator ϕ jest *nadmiernie dopasowany* do danych (ang. *overfits the data*), jeśli istnieje inny aproksymator ϕ' który

- jest empirycznie gorszy, tzn. $R_{\mathcal{S}}(\phi') > R_{\mathcal{S}}(\phi)$
- ale teoretycznie lepszy, tzn. $R(\phi') < R(\phi)$

- przy *parametrze ufności* δ

złożoność rodziny d_{VC}	liczność próby N	parametr ufności $1-\delta$	błąd teoretyczny R	odchyłka ϵ
—	$\nearrow$	—	—	$\searrow$
$\nearrow$	—	—	$\searrow$	$\nearrow$
—	—	$\nearrow$	—	$\nearrow$

- widać m.in., że **zwiększanie złożoności rodziny klasyfikatorów**, przy stałej liczności próby i parametrze ufności, może prowadzić do **nadmiernego dopasowania**

Minimalizacja ryzyka strukturalnego

Oszacowanie błędu generalizacji

- wzrost złożoności modelu zmniejsza ryzyko empiryczne ale zwiększa błąd generalizacji
- *ryzyko strukturalne*: ryzyko empiryczne $R_{\mathcal{S}}(\phi)$ zwiększone o *człon regularyzujący* $G(\Phi)$ reprezentujący koszt złożoności modelu

$$R_{\mathcal{S}}(\phi) + \lambda G(\Phi)$$

- koszt złożoności modelu można ocenić poprzez lukę generalizacji tzn. maksymalną - z prawdopodobieństwem $1-\delta$ - odchyłkę kosztu i kosztu empirycznego estymatora ϕ (dla liczności próby $N \geqslant d_{\mathsf{VC}}$)

$$\epsilon = R(\phi) - R_{\mathcal{S}}(\phi) = \sqrt{\frac{1}{N}\left(d_{\mathsf{VC}}\ \ln\frac{2\,\mathsf{e}\,N}{d_{\mathsf{VC}}} + \ln\frac{4}{\delta}\right)}$$

- jeśli dopuszczamy k błędów klasyfikacji, to $R_{\mathcal{S}}(\phi) = \frac{k}{N}$ i ryzyko strukturalne ma postać

$$\underbrace{\frac{k}{N}}_{\substack{\text{maks. koszt}\\ \text{empiryczny}\\ R_{\mathcal{S}}(\phi)}} + \lambda \underbrace{\sqrt{\frac{1}{N}\left(d_{\mathsf{VC}}\ \ln\frac{2\,\mathsf{e}\,N}{d_{\mathsf{VC}}} + \ln\frac{4}{\delta}\right)}}_{\substack{\text{maks. różnica kosztu}\\ \text{i kosztu empirycznego } R(\phi)-R_{\mathcal{S}}(\phi)}}$$

Minimalizacja ryzyka strukturalnego

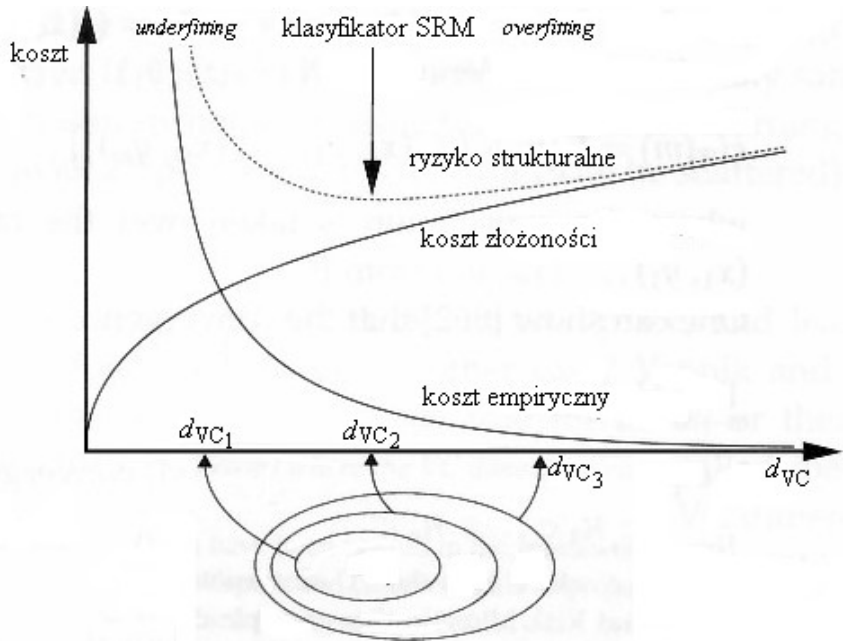
- ciąg rodzin klasyfikatorów $\Phi^1 \subset \ldots \subset \Phi^m \subset \ldots$
  - malejący koszt empiryczny (malejąca liczba błędów)

$$k^1 \geqslant \ldots \geqslant k^m \geqslant$$

  - rosnąca złożoność estymatorów

$${d_{\mathrm{VC}}}^1 \leqslant \ldots \leqslant {d_{\mathrm{VC}}}^m \leqslant \ldots$$

- *minimalizacja ryzyka strukturalnego*: wybrać d_{VC} dla którego ryzyko strukturalne jest minimalne tzn. zmiana kosztu empirycznego równoważy zmianę kosztu złożoności



Sewell 2006

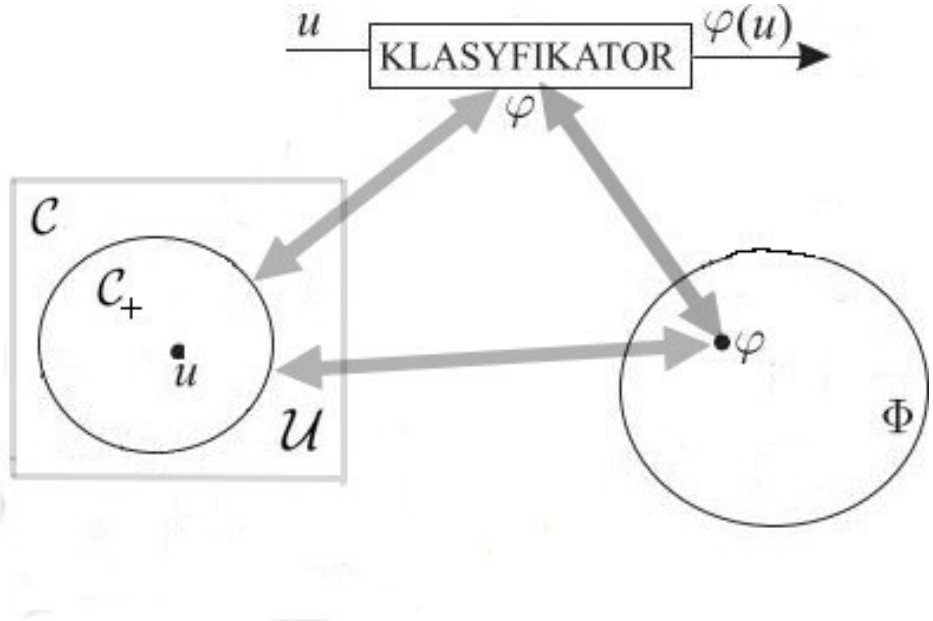
$$\frac{k}{N} + \lambda\sqrt{\frac{1}{N}\left(d_{\mathrm{VC}} \ln \frac{2eN}{d_{\mathrm{VC}}} + \ln \frac{4}{\delta}\right)} \approx c\sqrt{d_{\mathrm{VC}}/N}$$

10.1 Koszt generalizacji
10.2 Redukcja nadmiernego dopasowania
10.3 Ryzyko strukturalne
10.4 Wymiar Vapnika-Chervonenkisa

Klasyfikatory binarne a zbiory

Klasyfikacja binarna w języku zbiorów i funkcji

- dowolny klasyfikator na dwie klasy $+, -$ zadanego zbioru obrazów U jest równoważny
  - **zbiorowi** $\mathcal{C}_+ \subset U$ obrazów klasyfikowanych do klasy $+$
  - **funkcji** φ klasyfikującej obrazy, tzn. określającej indeks klasy dla każdego obrazu $\boldsymbol{u} \in U$



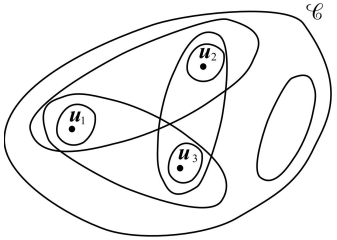
(z lewej:) każdemu klasyfikatorowi binarnemu odpowiada zbiór $\mathcal{C}+$ obrazów (punktów) w U należących do klasy $+$
(z prawej:) każdemu klasyfikatorowi binarnemu odpowiada punkt φ w zbiorze klasyfikatorów Φ;

Rozbicie przez rodzinę zbiorów

- w języku klasyfikacji : rodzina klasyfikatorów może dokonać dowolnej klasyfikacji binarnej zbioru skończonego (obrazów)
- w języku teorii zbiorów : rodzina zbiorów $\mathscr{C}$ *rozbija* (ang. *shatter*) zbiór punktów (obrazów) U jeśli może zrealizować dowolną klasyfikację tego zbioru, tzn. **każdy jego podzbiór skończony** U' może być *wyodrębniony* (odseparowany od pozostałych podzbiorów) przez pewien zbiór $\mathcal{C} \in \mathscr{C}$

$$U' = U \cap \mathcal{C}$$

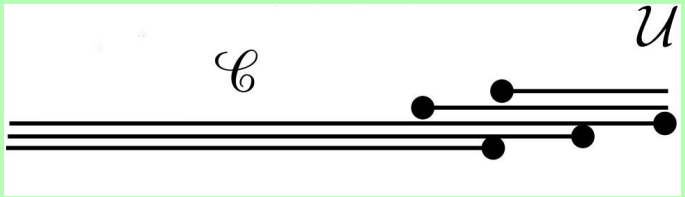
inaczej: dla każdego **skończonego** podzbioru $U' \subset U$ istnieje taki zbiór $\mathcal{C} \in \mathscr{C}$, do którego należą wyłącznie wszystkie punkty ze zbioru U'



- w języku funkcji klasyfikujących : dla skończonego zbioru (obrazów) U i dowolnej klasyfikacji $(\mathcal{C}+, \mathcal{C}-)$ punktów tego zbioru, w rodzinie funkcji klasyfikujących φ istnieje funkcja poprawnie klasyfikująca

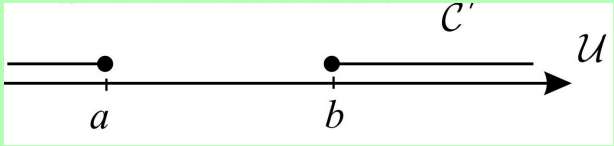
$$\varphi(\boldsymbol{u}_i) = y^i \quad \text{dla} \quad \boldsymbol{u}_i \subset U$$

Przykład: rozbicie przez półproste w $\mathbb{R}$



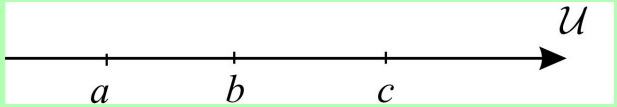
$\mathscr{C}$ rodzina wszystkich półprostych domkniętych $\mathbb{R}$

$\mathscr{C}$ rozbija dowolny zbiór dwuelementowy



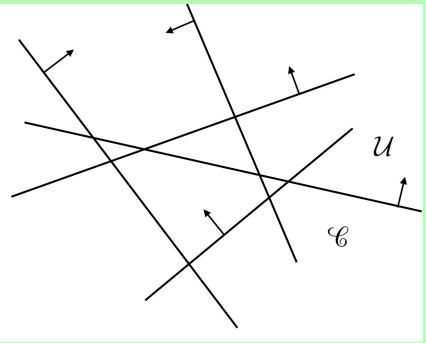
dla $a < b$: $\{a\} = (-\infty, a]$, $\{b\} = [b, \infty)$

$\mathscr{C}$ nie rozbija żadnego zbioru trójelementowego



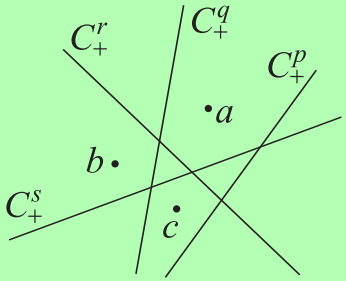
dla $a < b < c$ punkt $\{b\}$ nie może być wyodrębniony przez $\mathscr{C}$

Przykład: rozbicie przez półpłaszczyzny w $\mathbb{R}^2$



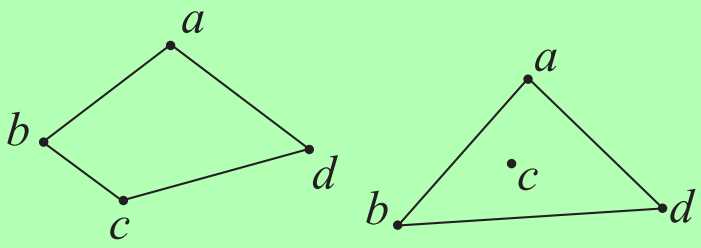
$\mathscr{C}$ = rodzina półpłaszczyzn w $\mathbb{R}^2$

$\mathscr{C}$ rozbija dowolny zbiór 3-elementowy (z wyjątkiem trzech punktów na prostej)



np. $\{a\} = \mathcal{C} \cap \mathcal{C}_+^r$, $\{a, c\} = \mathscr{C} \cap \mathcal{C}_+^q$

$\mathscr{C}$ nie rozbija żadnego zbioru 4-elementowego

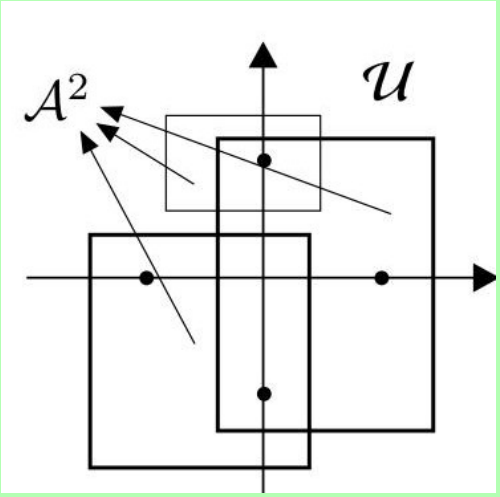


gdyż, w zależności od dwu możliwych położeń punktu c, podzbiór $\{c, a\}$ lub podzbiór $\{c\}$ nie mogą być wyodrębnione przez $\mathscr{C}$

Przykład: rozbicie przez prostokąty kolinearne w $\mathbb{R}^2$

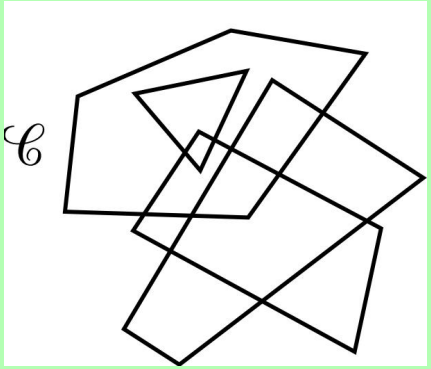
$\mathscr{C}$ = rodzina prostokątów o bokach równoległych do osi w $\mathbb{R}^2$

- $\mathscr{C}$ rozbija np. zbiór 4-elementowy $\{(1,0),(-1,0),(0,1),(0,-1)\}$



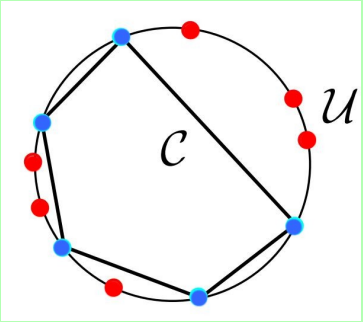
- $\mathscr{C}$ nie rozbija żadnego zbioru 5-elementowego
- w dotychczasowych przykładach liczność maksymalnego rozbijanego zbioru była równa liczbie „wolnych" parametrów rodziny $\mathscr{C}$

⊛ Przykład: rozbicie przez zbiory wypukłe w $\mathbb{R}^2$



$\mathscr{C}$ = rodzina zbiorów wypukłych w $\mathbb{R}^2$ (Sontag 1998)

- $\mathscr{C}$ rozbija dowolny zbiór punktów na okręgu



(wystarczy utworzyć powłokę wypukłą dowolnego podzbioru)

- liczba wolnych parametrów $\mathscr{C}$ jest nieograniczona; rozbijana rodzina zbiorów ma nieskończoną liczność

⊛ Przykład: rozbicie nieskończonego zbioru przez rodzinę o jednym parametrze

Ⓔ Vapnik (1995)

- funkcja klasyfikująca: $\ell_w(u) = \mathbf{1}\big(\sin(w\,u)\big)$ odpowiada zbiorom $\mathcal{C}_w = \{u : \sin(w\,u) > 0\}$
- rodzina tych zbiorów (**jeden parametr!**) rozbija zbiór c-elementowy $U = \{2^{-1}, \ldots, 2^{-c}\}$ dla dowolnego c
- dla dowolnej konfiguracji $y_1, \ldots, y_p \in \{0, 1\}$ wystarczy przyjąć $y_0 = 1$ i

$$w = \pi \sum_{i=0}^{c} y_i\, 2^i$$

- wówczas dla dowolnej konfiguracji $(y_1, \ldots, y_p)$, $y_j \in \{0, 1\}$, przynależności do klas i dowolnego elementu 2^{-j} zbioru U mamy

$$\begin{aligned}
\ell(2^{-j}) &= \mathbf{1}\Big(\sin(w\,2^{-j})\Big) = \mathbf{1}\Big(\sin(\sum_{i=0}^{c} \pi y_i\, 2^{i-j})\Big) \\
&= \mathbf{1}\sin\Big(\underbrace{\pi \sum_{i=0}^{j-1} y_i\, 2^{i-j}}_{\in(0,\pi)} + \underbrace{\pi y_j}_{0 \text{ lub } \pi} + \underbrace{\pi \sum_{i=j+1}^{c} y_i 2^{i-j}}_{\text{wielokrotność } 2\pi}\Big) \\
&= \mathbf{1}\Big(\sin(y_j \pi)\Big) = y_j \qquad \text{warunek spełniony}
\end{aligned}$$

Wymiar Vapnika-Chervonenkisa

- ⟳ infinite family *wymiar Vapnika-Chervonenkisa* rodziny $\mathscr{C}$

$$d_{\mathrm{VC}}(\mathscr{C}) = \text{największa liczność zbioru rozbijanego przez rodzinę } \mathscr{C}$$

- w języku klasyfikacji: $d_{\mathrm{VC}}(\mathscr{C})$ to **największa liczność zbioru obrazów, które być dowolnie klasyfikowane przez $\mathscr{C}$**
- jeśli wymiar VC rodziny $\mathscr{C}$ równy jest d_{VC} to
  - istnieje d_{VC}-elementowy podzbiór rozbijany przez $\mathcal{C}$
    - sieć binarna: **istnieje** d_{VC}-elementowy zbiór obrazów **poprawnie klasyfikowanych** przez sieć **dla dowolnej klasyfikacji**
  - może istnieć d_{VC}-elementowy podzbiór nie rozbijany przez $\mathcal{C}$ (o elementach w "pozycji szczególnej")
    - sieć binarna: **może istnieć** taki d_{VC}-elementowy zbiór obrazów, dla których sieć **nie potrafi dokonać dowolnej klasyfikacji**
  - nie istnieje $d_{\mathrm{VC}} + 1$-elementowy podzbiór rozbijany przez $\mathcal{C}$
    - sieć binarna: **nie istnieje** $d_{\mathrm{VC}} + 1$-elementowy zbiór obrazów, dla których sieć **może dokonać dowolnej klasyfikacji**

(E) dla rodziny $\mathscr{C}$ półprostych w $\mathbb{R}$: $d_{\mathrm{VC}}(\mathscr{C}) = p + 1$

(E) dla rodziny $\mathscr{C}$ hiperpłaszczyzn zorientowanych w $\mathbb{R}^p$, $d_{\mathrm{VC}}(\mathscr{C}) = p + 1$

- łatwo pokazać, że dla skończonych rodzin klasyfikatorów $m \leqslant \log_2 d_{\mathrm{VC}}$

⊛ VC dla sieci a VC dla zbiorów

Zbiory	Sieci wielowarstwowe	Perceptron Rosenblatta
$U \subset \mathbb{R}^p, \quad B = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_c\}$		
hipoteza $\mathcal{C}_{\boldsymbol{\vartheta}} \subset U$	sieć $\phi(\boldsymbol{u}) = \boldsymbol{w}^\mathsf{T}\gamma(\boldsymbol{\beta} + \boldsymbol{V}\boldsymbol{u})$ z rodziny $\boldsymbol{\Phi}[\gamma, h]$ o zadanych $\boldsymbol{\vartheta} = (\boldsymbol{\beta}, \boldsymbol{V}, \boldsymbol{w})$	1-warstwowy perceptron Rosenblatta o zadanych parametrach $\boldsymbol{\vartheta}$
rodzina hipotez $\mathcal{C} = \{\mathcal{C}_{\boldsymbol{\vartheta}} : \boldsymbol{\vartheta} \in \Theta\}$	rodzina sieci $\boldsymbol{\Phi}[\gamma, h]$ o dowolnych $\boldsymbol{\vartheta}$	rodzina $\mathcal{N}_R$ 1-warstwowych perceptronów Rosenblatta o dowolnych wagach
$\mathcal{B}$ jest rozbijany przez $\mathcal{C}$	$\mathcal{N}[\gamma, h]$ może dowolnie klasyfikować obrazy $\mathcal{B}$	$\mathcal{N}_R$ może dowolnie klasyfikować liniowo obrazy $\mathcal{B}$
$\nu_{\mathcal{C}}(c)$ funkcja wzrostu	maksymalna liczba różnych klasyfikacji c obrazów za pomocą rodziny sieci $\mathcal{N}[\gamma, h]$	maksymalna liczba klasyfikacji liniowych c obrazów $\nu_R(c) = \begin{cases} 2^c & \text{dla } c \leqslant p+1 \\ 2\sum_{i=0}^{p} \binom{c-1}{i} & \text{dla } c \geqslant p+1 \end{cases}$
$d_{\mathsf{VC}}(\mathcal{C})$	$d_{\mathsf{VC}}(\mathcal{N}[\gamma, h])$: największa liczba obrazów dowolnie klasyfikowanych przez $\mathcal{N}[\gamma, h]$	największa liczba obrazów dowolnie klasyfikowanych liniowo przez $\mathcal{N}_R$ $(= p + 1)$

⊛ Wymiar VC dla przykładowych klasyfikatorów

typ sieci	asympt. granica dolna	asympt. granica górna	literatura
g odcinkami wielomianowa d - max. stopień wielomianu	 $\Omega(L\nu \log_2(\nu/L))$	$O(\nu^2)$ $O(L^2\nu + L\nu \log_2 \nu)$ $O(\nu n)$	GJ95 BMM98 BHLM17
g odcinkami liniowa p - max. liczba przedziałów	 $\Omega(L\nu)$ $\Omega(L\nu \log_2(\nu/L))$	$O(\overline{\nu}\nu \log_2(pn))$ $O(L\nu \log_2 \nu)$	BHLM17 BMM98 BHLM17, HL16, BMM98
sieci ReLU	$\Omega(\nu \log_2 \nu)$ $\Omega(L\nu)$ $\Omega(L\nu \log_2(\nu/L))$	 $O(L\nu \log_2 \nu)$	M94 BMM98 BHLM17
g odcinkami stała	$\Theta(\nu \log_2 n)$	$O(\nu \log_2(pn))$	BHLM17,M94,BH89,C BHLM17
$\mathrm{sign}(\sin(w\,u)$ najbliższy sąsiad	∞ ∞		

g - funkcja aktywacji
n - liczba neuronów
ν - łączna liczba parametrów (wag, obciążeń)
L - liczba warstw
$\overline{\nu} = 1/\nu \sum_{i=1}^{L} \nu_i$, ν_i = liczba parametrów w warstwach 1 do i
$O(h(n))$: funkcja co najwyżej rzędu $h(n)$; asymptotyczna granica górna; $f(n) \leqslant c\,h(n)$;
$\Omega(h(n))$: funkcja f jest co najmniej rzędu $h(n)$; asymptotyczna granica dolna; $f(n) \geqslant c\,h(n)$
$\Theta(h(n))$: granica asymptotyczna ($O(h(n))$ i $\Omega(h(n))$)

BHLM17: P. L. Bartlett et al. (2017); **BMM98**: P. Bartlett, Maiorov, and Meir (1998); **BH98**: Baum and Haussler (1998); **GJ95**: Goldberg and Jerrum (1995); **M94**: Maass (1994); **C68**: Cover (1968)

Literatura

Materiały źródłowe

Barron, A. (1993). "Universal approximation bounds for superposition of a sigmoidal function". In: *IEEE Transactions on Information Theory* 3, pp. 930–945.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag.

— (1998). *Statistical Learning Theory*. Wiley.

Materiały pomocnicze

Bartlett, P., V. Maiorov, and R. Meir (Nov. 1998). "Almost linear VC-dimension bounds for piecewise polynomial networks". In: *Neural Computation* 10.8, pp. 2159–2173.

Bartlett, P. L. et al. (2017). "Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks". In: *Proceedings of the Conference on Learning Theory (COLT) 2016, ACM Conference on Data Science*. URL: //proceedings.mlr.press/v65/harvey17a.html,%20//ikdd.acm.org/Site/CoDS2016/keynotes.html.

Baum, E. B. and D. Haussler (1998). "What size net gives valid generalization?" In: *Neural Computation* 1.1, pp. 151–160.

Chun, S. (2014). *ML Study PAC Learning*. URL: https://www.slideshare.net/sanghyukchun/%20pac-learning-42139787.

Literatura (cd.)

Cover, T. M. (1968). *Capacity problems for linear machines. In L. Kanal, editor, Pattern Recognition*. Thompson Book Co.

Goldberg, P. W. and M. R. Jerrum (1995). "Bounding the Vapnik-Chervonenkis dimension of concept classes parameterized by real numbers". In: *Machine Learning* 18.2, pp. 131–148.

Guestrin, C. (2007). *VC Dimension*. Machine Learning – 10701/15781.

Maass, W. (Sept. 1994). "Neural nets with superlinear VC-dimension". In: *Neural Computation* 6.5, pp. 877–884.

Moore, A. W. (2001). *VC-dimension for characterizing classifiers*. URL: https://www.cmi.ac.in/~madhavan/courses/datamining13-aug/literature/vc-dimension-tutorial.pdf.

Rigollet, P. (2015). *Mathematics of Macine Learning*. OpenCourseWare 18.657. MIT. URL: ocw.mit.edu.

Samir, M. (2016). *Machine Learning Theory - Part 2: Generalization Bounds*. URL: https://mostafa-samir.github.io/ml-theory-pt2/.

Simard, P. Y. et al. (1998). "Transformation Invariance in Pattern Recognition - Tangent Distance and Tangent Propagation". In: *Neural Networks: Tricks on the Trade*. Vol. 1524. Lectures Notes in Computer Science. Springer-Verlag, pp. 239–274.

Sontag, E. (1998). "VC dimension of neural networks". In: *Neural Networks and Machine Learning*. Ed. by C. Bishop. Berlin: Springer, pp. 69–95.

Literatura (cd.)

Srivastava, N. et al. (June 2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15, pp. 1929–1958.

Vapnik, V. and A. Chervonenkis (1981). "On the uniform convergence of relative frequencies of events to their probabilities". In: *Theory of Probability and Its Applications* 26.3, pp. 532–553.

Zhang, Y. (2011). *Overfitting, PAC Learning, VC Dimension, VC Bounds, Mistake Bounds, Semi-Supervised Learning*. 10.701 Machine Learning slides. Carneggie Mellon University. URL: https://www.cs.cmu.edu/~tom/10701_sp11/recitations/Recitation_9.pdf.

Temat 11: Uczenie sieci neuronowych

11.1 Trenowanie sieci neuronowych
11.2 Metody pierwszego rzędu
11.3 Metody drugiego rzędu
11.4 Specyficzne techniki neuronowe

11.1 Trenowanie sieci neuronowych
11.2 Metody pierwszego rzędu
11.3 Metody drugiego rzędu
11.4 Specyficzne techniki neuronowe

Empiryczna minimalizacja ryzyka (ERM)

Zadanie uczenia sieci neuronowej

- metoda ERM uczenia sieci neuronowej ϕ: dobór parametrów (wag i obciążeń), minimalizujących ryzyko empiryczne

$$R_{\mathcal{S}}(\phi) = \frac{1}{N} \sum_{i=1}^{N} \ell\big(\phi(\boldsymbol{u}_i), \boldsymbol{y}_i\big)$$

- dobór wag odbywa się metodą rekurencyjną przy użyciu metod gradientowych
- na ogół obliczenie gradientu ryzyka $R_{\mathcal{S}}$ zastępuje się estymacją gradientu dla *porcji danych* (ang. *mini-batches*) o rozmiarze T, wybranych losowo w T kolejnych krokach począwszy of chwili $t_j = (j-1)T$

$$R_{\mathcal{S}\,j}(\phi) = \sum_{t=t_j+1}^{t_j+T} \ell\big(\phi(\boldsymbol{u}(t), \boldsymbol{y}(t)\big), \quad j = 1, 2, \ldots$$

przy czym T jest rzędu kilkudziesięciu, zwykle znacznie mniejsze niż długość epoki (rozmiar próby) N.

⊛ Aproksymacja stochastyczna

Podstawowy algorytm Robbinsa-Monroe

- problem: znaleźć zero $\boldsymbol{\vartheta}^*$ funkcji $Q(\boldsymbol{\vartheta})$, której obserwacje $y(\boldsymbol{\vartheta})$ są zakłócone, ale $\mathcal{E}\, y(\boldsymbol{\vartheta}) = Q(\boldsymbol{\vartheta})$

⊛ Algorytm Robbinsa-Monroe (Robbins and Monro 1951; Blum 1954)

Ciąg parametrów

$$\boldsymbol{\vartheta}_{n+1} = \boldsymbol{\vartheta}_n - \gamma_n\, y(\boldsymbol{\vartheta}_n), \quad \gamma_n > 0$$

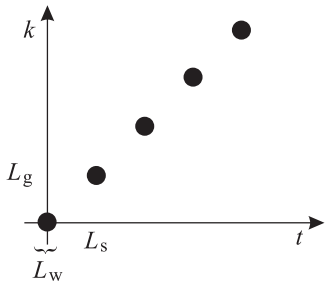
zbiega do $\boldsymbol{\vartheta}^*$ z prawdopodobieństwem 1 o ile

- funkcja $y(\boldsymbol{\vartheta})$ jest jednostajnie ograniczona
- funkcja $Q(\boldsymbol{\vartheta})$ jest niemalejąca,
- pochodna $Q'(\boldsymbol{\vartheta}^*)$ istnieje i jest dodatnia
- ciąg kroków $\{\gamma_n\}$ spełnia warunki

$$\sum_{n=0}^{\infty} \gamma_n = \infty, \qquad \sum_{n=0}^{\infty} \gamma_n^2 < \infty$$

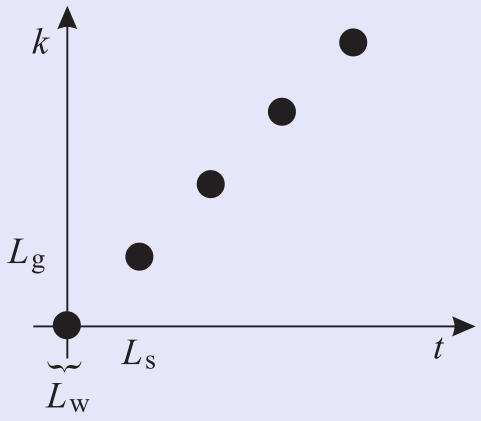
Mody minimalizacji (1)

- ogólnie: (L_w, L_g, L_s)
  - L_w - liczba kroków (długość okna) używana do obliczenia gradientu
  - L_g -liczba kroków w kierunku gradientu
  - L_s - przesunięcie okna



Moda natychmiastowa, *stochastic gradient descent* SGD $(1, 1, 1)$

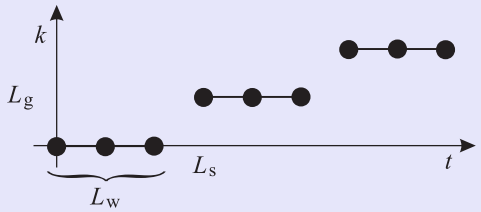
- estymatory gradientu oparte na pojedynczych próbkach! $\Rightarrow$ oszczędność pamięci, duża wariancja $\Rightarrow$ unikanie minimów lokalnych (+), problematyczne określanie minimum (-), więcej kroków treningu (-)
- koszt empiryczny = koszt dla pojedynczej pary uczącej
- jedna korekcja wag po każdej prezentacji pary uczącej



Mody minimalizacji (2)

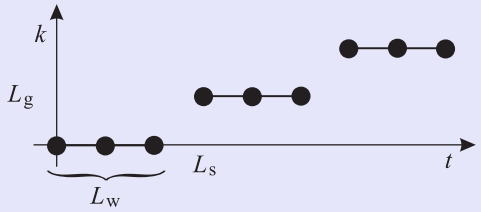
! Porcje (ang. *batches*) $(|S|, 1, |S|)$

- koszt empiryczny = koszt skumulowany uśredniony w *porcji* o $L_w > 1$ krokach
- stosowane zwykle dla "płytkich" sieci perceptronowych i skończonego zbioru uczącego $\mathcal{S}$; zwykle =epoka, $L_w = |\mathcal{S}|$
- lepsza od SGD estymacja gradientu $\Rightarrow$, szybsza zbieżność, oszczędność pamięci



! Mini-porcje (ang. *mini-batches* $(L_w, 1, L_w)$)

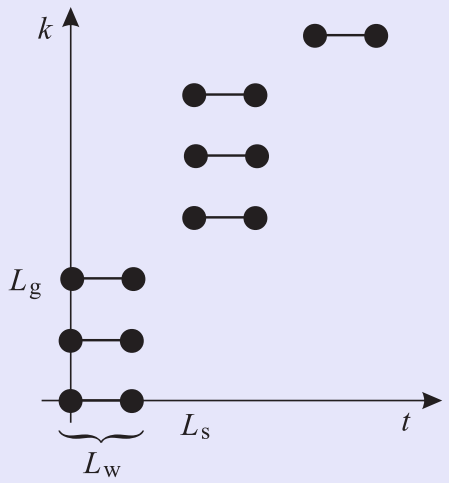
- koszt empiryczny = koszt skumulowany uśredniony w *porcji* o $L_w > 1$ krokach
- dla sieci głębokich zwykle zbiór kilkudziesięciu obrazów (32?) zależnie od GPU
- lepsza od wsadowej estymacja gradientu $\Rightarrow$, szybsza zbieżność, oszczędność pamięci



Mody minimalizacji

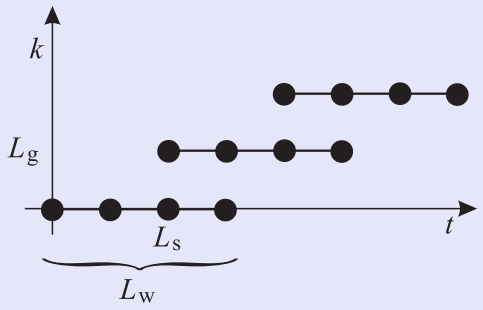
! Wielokrotne użycie gradientu + mini-porcje (L_w, L_g, L_w)

- koszt empiryczny = koszt skumulowany w mini-porcji o L_w krokach
- $L_g > 1$ kroków minimalizacji przy użyciu tego samego gradientu, np. do osiągnięcia minimum w kierunku gradientu



! Okno przesuwne (L_w, L_g, L_s)

- koszt empiryczny = koszt skumulowany w *mini-porcji* o długości L_w
- usunięcie L_s najstarszych i prezentacja L_s nowych par uczących, $L_s < L_w$
- opcjonalnie: $L_g \geqslant 1$ kroków minimalizacji przy użyciu tego samego gradientu



Problem minimalizacji kosztu empirycznego

- problem: minimalizacja funkcji
$$\widehat{R}(\boldsymbol{\vartheta}) = R_{\mathcal{S}}(\phi_{\boldsymbol{\vartheta}})$$
przy czym funkcja jest gładka tzn. ciągłe są
$$\widehat{R}'(\boldsymbol{\vartheta}) = \frac{\mathrm{d}\widehat{R}(\boldsymbol{\vartheta})}{\mathrm{d}\boldsymbol{\vartheta}}\text{: gradient kosztu empirycznego;} \quad \boldsymbol{g}(t) = \widehat{R}'(\boldsymbol{\vartheta}(t))$$
$$\widehat{R}''(\boldsymbol{\vartheta}) = \frac{\partial^2 \widehat{R}(\boldsymbol{\vartheta})}{\partial\boldsymbol{\vartheta}\,\partial\boldsymbol{\vartheta}^{\mathsf{T}}}\text{: hessian kosztu empirycznego;} \quad \boldsymbol{H}(t) = \widehat{R}''(\boldsymbol{\vartheta}(t))$$
- warunek wystarczający minimum
$$\widehat{R}'(\boldsymbol{\vartheta}) = \mathbf{0}$$
$$\widehat{R}''(\boldsymbol{\vartheta}) > \boldsymbol{O}$$

Idea rekurencyjnej minimalizacji gradientowej

Ogólna postać algorytmu

$$\boldsymbol{\vartheta}(t+1) = \boldsymbol{\vartheta}(t) + \gamma\, \boldsymbol{r}(t)$$

gdzie

- t: indeks kroku minimalizacji
- $\boldsymbol{\vartheta}(t)$: wektor parametrów sieci (wagi, obciążenia) w kroku t
- $\boldsymbol{r}(t)$: kierunek zmian wektora parametrów (zależny od gradientu i hessianu kosztu empirycznego)
- γ: szybkość uczenia (ang. *learning rate*)

Kroki uczenia

Schemat korekcji parametrów

sieć $\phi_{\boldsymbol{\vartheta}}$ o parametrach (wagi, obciążenia) $\boldsymbol{\vartheta}$
inicjalizacja: $j = 0$

1. gromadzenie danych dla mini-porcji (okna o długości L_w)
   a. inicjalizacja $i = j$; $R = 0$; $\boldsymbol{\vartheta} = \boldsymbol{\vartheta}^o$
   b. wylosować L_w par uczących $\Big(\boldsymbol{u}(t), \boldsymbol{y}(t)\Big)$
2. obliczenie kosztów dla mini-porcji i aktualnych parametrów
   a. dla każdej pary obliczyć wyjście sieci $\phi_{\boldsymbol{\vartheta}}(\boldsymbol{u}(t))$ i koszt chwilowy $R(\boldsymbol{\vartheta}) = R(\phi_{\boldsymbol{\vartheta}}(\boldsymbol{u}(t)))$
   b. kumulacja kosztu chwilowego $R(\boldsymbol{\vartheta}) \mathrel{+}= R(t)$
3. obliczenie gradientu kosztu skumulowanego względem parametrów $\boldsymbol{g}(t)$ z zastosowaniem algorytmu propagacji wstecznej
4. korekcja parametrów $\boldsymbol{\vartheta} \mathrel{+}= \eta\, \boldsymbol{r}(\boldsymbol{g}(t))$
5. przesunąć okno o L_s danych ($j \mathrel{+}= L_s$), powrót do (1)

11.1 Trenowanie sieci neuronowych
11.2 Metody pierwszego rzędu
11.3 Metody drugiego rzędu
11.4 Specyficzne techniki neuronowe

Metoda największego spadku

- klasyczny wzór Taylora 1 rzędu dla $\widehat{R}$ **jako funkcji szybkości uczenia** γ

$$\widehat{R}(\boldsymbol{\vartheta} + \gamma\, \boldsymbol{r}) = \widehat{R}(\boldsymbol{\vartheta}) + \gamma\, \boldsymbol{r}^{\mathsf{T}} \widehat{R}'(\boldsymbol{\vartheta}) + o(\gamma)$$

- jeśli przyjmiemy $\boldsymbol{r} = -\widehat{R}'(\boldsymbol{\vartheta})$ to dla wystarczająco małych γ mamy

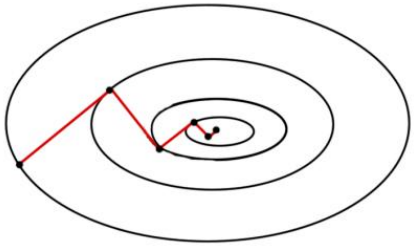
$$\widehat{R}(\boldsymbol{\vartheta} + \gamma\, \boldsymbol{r}) = \widehat{R}(\boldsymbol{\vartheta}) - \gamma \left\| \widehat{R}'(\boldsymbol{\vartheta}) \right\|^2 \leqslant \widehat{R}(\boldsymbol{\vartheta})$$

Algorytm korekcji parametrów - metoda największego spadku

Dla dostatecznie małej szybkości uczenia γ, algorytm

$$\boldsymbol{\vartheta}(t+1) = \boldsymbol{\vartheta}(t) - \gamma\, \boldsymbol{g}(t)$$

redukuje błąd empiryczny $\widehat{R}$



R. H. Refsnæs (2009). "A brief introduction to the conjugate gradient method". URL: folk.idi.ntnu.no/elster/tdt24/tdt24-f09/cg.pdf

⊛ Gradient sprzężony

- *kierunki sprzężone* $\boldsymbol{r}$, $\boldsymbol{s}$: $\boldsymbol{r}^\mathsf{T} \widehat{R}'' \boldsymbol{s} = 0$
- *gradient sprzężony* (ang. *conjugate gradient*) idea:
  $\boldsymbol{r}(t)^\mathsf{T} \boldsymbol{H}(t)\, \boldsymbol{r}(\tau) = 0$ dla $\tau = 1, \ldots, t-1$

Algorytm gradientu sprzężonego

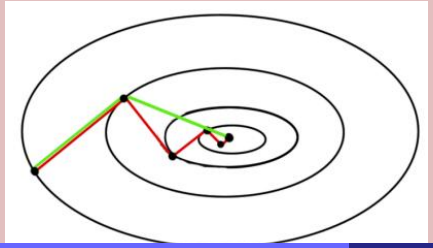
$$\boldsymbol{\vartheta}(t+1) = \boldsymbol{\vartheta}(t) + \gamma\, \boldsymbol{r}(t)$$

$$\boldsymbol{r}(t) = -\boldsymbol{g}(t) + \beta(t-1)\, \boldsymbol{r}(t-1), \qquad \boldsymbol{r}(0) = -\boldsymbol{g}(0)$$

$$\beta(t) = \frac{\|\boldsymbol{g}(t+1)\|^2}{\|\boldsymbol{g}(t)\|^2} \quad \text{Fletcher-Reeves}$$

$$\beta(t) = \frac{(\boldsymbol{g}(t+1) - \boldsymbol{g}(t))^\mathsf{T}\, \boldsymbol{g}(t+1)}{\|\boldsymbol{g}(t)\|^2} \quad \text{Polak-Ribière}$$

- dla funkcji kwadratowej w $\mathbb{R}^\nu$: F-R, P-R są identyczne, zbieżność po ν krokach
- sugeruje się restart po każdych ν krokach



11.1 Trenowanie sieci neuronowych
11.2 Metody pierwszego rzędu
11.3 Metody drugiego rzędu
11.4 Specyficzne techniki neuronowe

Metoda Newtona

- wzór Newtona 2. rzędu

$$\widehat{R}(\boldsymbol{\vartheta} + \gamma\,\boldsymbol{r}) = \widehat{R}(\boldsymbol{\vartheta}) + \gamma\,\boldsymbol{r}^{\mathsf{T}}\widehat{R}'(\boldsymbol{\vartheta}) + \frac{1}{2}\,\gamma^2\,\boldsymbol{r}^{\mathsf{T}}\widehat{R}''(\boldsymbol{\vartheta})\,\boldsymbol{r} + \mathsf{o}(\gamma^2)$$

- gradient $\widehat{R}$ w funkcji $\boldsymbol{r}$

$$\gamma\,\widehat{R}'(\boldsymbol{\vartheta} + \gamma\,\boldsymbol{r}) = \gamma\,\widehat{R}'(\boldsymbol{\vartheta}) + \gamma^2\,\widehat{R}''(\boldsymbol{\vartheta})\,\boldsymbol{r} + \mathsf{o}(\gamma^2)$$

- kierunek poprawy

$$\boldsymbol{r} = -\widehat{R}''(\boldsymbol{\vartheta})^{-1}\widehat{R}'(\boldsymbol{\vartheta})$$

Algorytm Newtona

$$\boldsymbol{\vartheta}(t+1) = \boldsymbol{\vartheta}(t) - \gamma\,\boldsymbol{H}^{-1}(t)\,\boldsymbol{g}(t)$$

- przy minimalizacji kierunkowej, zbieżność w ν krokach dla funkcji kwadratowej
- **problemy**
  - obliczenie hessianu i jego odwracanie
  - powolny "rozbieg", możliwa rozbieżność z dala od ekstremum

⊛ Metody zmiennej metryki (quasi newtonowskie)

- aproksymacja hessianu $\widehat{\boldsymbol{H}}''$ aby spełniony był warunek

$$\widehat{\boldsymbol{H}}''(\boldsymbol{\vartheta})(\boldsymbol{\vartheta}-\boldsymbol{\vartheta}^*) = \ell'(\boldsymbol{\vartheta}) - \ell'(\boldsymbol{\vartheta}^*)$$

oznaczymy $\boldsymbol{P}(t) = \widehat{\boldsymbol{H}}^{-1}(t), \quad \Delta\boldsymbol{x}(t) = \boldsymbol{x}(t+1) - \boldsymbol{x}(t)$

Algorytm zmiennej metryki

$\boldsymbol{\vartheta}(0) = \boldsymbol{\vartheta}^0, \quad \boldsymbol{P}(0) = \mathbf{1}, \quad \boldsymbol{P}(1)$ – zastosować metodę największego spadku

$\boldsymbol{\vartheta}(t+1) = \boldsymbol{\vartheta}(t) - \gamma\,\boldsymbol{P}(t)\,\boldsymbol{g}(t)$

$\boldsymbol{P}(t+1) = \boldsymbol{P}(t) + \Delta\boldsymbol{P}(t)$

wersja Davidona-Fletchera-Powella (DFP)

$$\Delta\boldsymbol{P}(t) = \frac{\Delta\boldsymbol{\vartheta}(t)\,\Delta\boldsymbol{\vartheta}(t)^T}{\Delta\boldsymbol{\vartheta}(t)^T\,\Delta\boldsymbol{g}(t)} - \frac{\boldsymbol{P}(t)\,\Delta\boldsymbol{g}(t)\,\Delta\boldsymbol{g}(t)^T\,\boldsymbol{P}(t)}{\Delta\boldsymbol{g}(t)^T\,\boldsymbol{P}(t)\,\Delta\boldsymbol{g}(t)}$$

wersja Broydena-Fletchera-Goldfarba-Shanno (BFGS)

$$\Delta\boldsymbol{P}(t) = \Delta\boldsymbol{P}(t)_{\mathsf{DFP}} + \Delta\boldsymbol{g}(t)^{\mathsf{T}}\boldsymbol{P}(t)\Delta\boldsymbol{g}(t)\,\boldsymbol{z}\,\boldsymbol{z}^{\mathsf{T}}$$

gdzie $\boldsymbol{z} = \dfrac{\Delta\boldsymbol{\vartheta}(t)}{\Delta\boldsymbol{\vartheta}(t)^{\mathsf{T}}\Delta\boldsymbol{g}(t)} - \dfrac{\boldsymbol{P}(t)\,\Delta\boldsymbol{g}(t)}{\Delta\boldsymbol{g}(t)^{\mathsf{T}}\boldsymbol{P}(t)\,\Delta\boldsymbol{g}(t)}$

Metoda Levenberga-Marquardta

- połączenie metod największego spadku i Newtona

$$\boldsymbol{r}(\boldsymbol{g}) = -\gamma(\boldsymbol{H} + \lambda\,\boldsymbol{I})^{-1}\boldsymbol{g}$$

- *parametr Marquardta* λ jest zmniejszany do zera podczas minimalizacji
  - () λ "duże" (daleko od ekstremum) – najszybszy spadek
  - () λ "małe" (w pobliżu ekstremum) – metoda Newtona
- dla funkcji kwadratowych

$$\widehat{R}(\boldsymbol{\vartheta}) = \frac{1}{2}\sum_{i=1}^{q}(y_i(\boldsymbol{\vartheta}) - y_i^o)^2 = \frac{1}{2}\sum_{i=1}^{q} R_i^2(\boldsymbol{\vartheta})$$

$$\widehat{R}'(\boldsymbol{\vartheta}) = \sum_{i=1}^{q} y_i'(\boldsymbol{\vartheta})\, R_i(\boldsymbol{\vartheta})$$

$$\widehat{R}''(\boldsymbol{\vartheta}) = \sum_{i=1}^{q} y_i''(\boldsymbol{\vartheta})\, \underbrace{R_i(\boldsymbol{\vartheta})}_{\substack{\approx 0 \text{ w pobliżu} \\ \text{ekstremum}}} + y_i'(\boldsymbol{\vartheta})\, y_i'^{\mathsf{T}}(\boldsymbol{\vartheta})$$

Algorytm Levenberga - Marquardta

$$\boldsymbol{\vartheta}(t+1) = \boldsymbol{\vartheta}(t) - \gamma\left(\sum_{i=1}^{q} y_i'(\boldsymbol{\vartheta})\, y_i'^{\mathsf{T}}(\boldsymbol{\vartheta}) + \lambda\,\boldsymbol{I}\right)^{-1}\sum_{i=1}^{q} y_i'(\boldsymbol{\vartheta})\, R_i(\boldsymbol{\vartheta})$$

11.1 Trenowanie sieci neuronowych
11.2 Metody pierwszego rzędu
11.3 Metody drugiego rzędu
11.4 Specyficzne techniki neuronowe

Specyficzne techniki neuronowe

Wygładzanie wykładnicze wag

- *współczynnik inercji* $\alpha, \quad 0 < \alpha < 1$

Wygładzanie wykładnicze wag

$$\Delta\boldsymbol{\vartheta}(t+1) = -\gamma\,\boldsymbol{g}(t) - \underbrace{\alpha\,\Delta\boldsymbol{\vartheta}(t)}_{\text{człon inercyjny}}, \quad 0 < \alpha < 1$$

- człon inercyjny **kumuluje efekty gradientów** $\boldsymbol{g}(t)$ w kierunku spadku i **redukuje efekty fluktuacji gradientu**
- zmiana wag następuje w kierunku $-$*gradientu skumulowanego ważonego* $\boldsymbol{G}(t)$ zamiast kierunku $-$ gradientu)

$$\boldsymbol{\vartheta}(t+1) = \boldsymbol{\vartheta}(t) - \gamma\,\underbrace{(\boldsymbol{g}(t) + \alpha\boldsymbol{g}(t-1) + \alpha^2\boldsymbol{g}(t-2) + \ldots)}_{\text{gradient skumulowany ważony}} = -\gamma\,\boldsymbol{G}(t)$$

- człon inercyjny jest równy gradientowi skumulowanemu ważonemu

$$\boldsymbol{\vartheta}(t+1) = \boldsymbol{\vartheta}(t) - \gamma(\boldsymbol{g}(t) + \underbrace{\alpha\,\boldsymbol{G}(t-1)}_{\substack{\text{„poprawka”: „poprzedni”} \\ \text{gradient skumulowany}}})$$

Specyficzne techniki neuronowe

Wygładzanie wykładnicze Nesterova

Y. Nesterov (1983). "A method for unconstrained convex minimization problem with the rate of convergence o(1/k2)". In: *Doklady AN SSSR (Soviet. Math. Docl.)* 269, pp. 543–547

Wygładzanie wykładnicze Nesterova (Nesterov 1983)

$$\widetilde{\boldsymbol{\vartheta}}(t) = \boldsymbol{\vartheta}(t) + \widetilde{\gamma}\Delta\boldsymbol{\vartheta}(t-1) \quad \text{punkt obliczenia gradientu}$$
$$\Delta\boldsymbol{\vartheta}(t) = -\alpha\,\widetilde{\boldsymbol{g}}(t) + \alpha\Delta\boldsymbol{\vartheta}(t-1)$$

gdzie gradient $\widetilde{\boldsymbol{g}}(t)$ jest obliczony w punkcie $\widetilde{\boldsymbol{\vartheta}}(t)$

- zbieżny dla minimalizacji funkcji wypukłych, wykorzystane do uczenia sieci (Sutskever 2012),
- problem: konieczność obliczania gradientu w punktach pośrednich

Specyficzne techniki neuronowe

Uproszczone wygładzanie wykładnicze Nesterova

Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu (2012). *Advances in optimizing recurrent networks*. Tech. rep. U. Montreal. URL: https://arxiv.org/pdf/1212.0901v2.pdf

Uproszczone wygładzanie wykładnicze Nesterova (Bengio e.a. 2012)

$$\boldsymbol{s}(t+1) = -\gamma\,\widetilde{\boldsymbol{g}}(t) + \alpha\,\boldsymbol{s}(t)$$
$$\widetilde{\boldsymbol{\vartheta}}(t+1) = \widetilde{\boldsymbol{\vartheta}}(t) - \alpha\,\boldsymbol{s}(t) + (1+\alpha)\,\gamma\,\boldsymbol{s}(t+1)$$

includegraphics

- algorytm uproszczony wykorzystuje tylko punkty pośrednie $\widetilde{\boldsymbol{\vartheta}}$
- algorytm jest zbieżny do rozwiązania Nesterova

Współczynnik kroku uczenia

- stały współczynnik kroku: $0.05 \div 0.25$ (Freeman);
  $\gamma \leqslant 1/$liczba wejść neuronu; γ mniejsze dla neuronów z wieloma wejściami; γ większe w warstwie wyjściowej (zwykle tam małe gradienty)
- redukcja wykładnicza (ang. *exponential decay*) $\gamma = \gamma_0 \exp(-kt)$
- redukcja $1/t$: $\gamma = \gamma_0/(1 + kt)$
- redukcja co kilka epok (np. 0.5 co 5 epok, 0.1 co 20 epok)

Zróżnicowane prędkości uczenia wag

Reguła delta-delta

- pochodna kosztu względem współczynnika uczenia γ_i

$$\vartheta_i(t) = \vartheta_i(t-1) - \gamma_i\, g_i(t-1)$$

$$\frac{\mathrm{d}\widehat{R}}{\mathrm{d}\gamma_i} = \frac{\mathrm{d}\widehat{R}}{\mathrm{d}\vartheta_i(t)} \frac{\mathrm{d}\vartheta_i(t)}{\mathrm{d}\gamma_i} = -g_i(t)\, g_i(t-1)$$

Algorytm δ-δ

$$\vartheta_i(t+1) = \vartheta_i(t) - \gamma_i(t)\, g_i(t)$$
$$\gamma_i(t+1) = \gamma_i(t) + \gamma\, g_i(t)\, g_i(t-1)$$

- wrażliwość na dobór γ
- szybki wzrost γ_i jeśli kolejne pochodne $g_i(t)$ są tego samego znaku

Zróżnicowane prędkości uczenia wag

Reguła delta-bar-delta

- gradient zastąpiony przez gradient wygładzany wykładniczo

Algorytm δ-$\bar{\delta}$ (delta-bar-delta) (R. A. Jacobs 1988)

$$\begin{aligned} \vartheta_i(t+1) &= \vartheta_i(t) - \gamma_i(t)\, g_i(t) \\ \gamma_i(t+1) &= \gamma_i(t) + \gamma\, g_i(t)\, \overline{g}_i(t-1) \\ \overline{g}_i(t) &= \xi\, \overline{g}_i(t-1) + (1-\xi)\, g_i(t), \quad 0 < \xi < 1 \end{aligned} \qquad (*)$$

- dodatkowe zabezpieczenia: γ rośnie liniowo jeśli znak gradientu nie zmienia się, w przeciwnym razie maleje wykładniczo

Zmodyfikowany algorytm δ-$\bar{\delta}$

$$\gamma_i(t+1) = \gamma_i(t) + \begin{cases} \kappa & \text{gdy } g_i(t)\, \overline{g}_i(t-1) > 0 \\ -\beta\gamma_i(t-1) & \text{gdy } g_i(t)\, \overline{g}_i(t-1) < 0 \end{cases} \qquad (**)$$

$0 < \kappa < 0.05, \quad 0.1 < \beta < 0.3$

Zróżnicowane prędkości uczenia wag

Adagrad

J. Duchi, E. Hazan, and Y. Singer (July 2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *JMLR* 12, pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.htm A. Karpathy (2016). *Convolutional Neural Networks for Visual Recognition*. Tech. rep. 2016 class CS231n. URL: http://cs231n.github.io/

Adagrad: Algorytm gradientu adaptacyjnego

$$S_i^2(t) = S_i^2(t-1) + g_i^2(t)$$

$$\Delta\vartheta_i(t) = -g_i(t)\underbrace{\frac{\gamma}{\sqrt{S_i^2(t)} + \epsilon}}_{\text{czynnik redukujący}}$$

- $S_i^2(t)$ jest skumulowanym kwadratem i-tej współrzędnej gradientu do chwili t
- sugerowane $\epsilon \sim 10^{-4} - 10^{-8}$
- krok jest redukowany dla wag o „dużych" fluktuacjach składowych gradientu
- **wada**: zbyt szybkie tłumienie kroku

Zróżnicowane prędkości uczenia wag

RMSprop

Hinton A. Karpathy (2016). *Convolutional Neural Networks for Visual Recognition*. Tech. rep. 2016 class CS231n. URL: http://cs231n.github.io/ S. Ruder (2017). *An overview of gradient descent optimization algorithms*. Tech. rep. arxiv.org/abs/1609.04747. Cornell University

Propagacja RMS (ang. *Root Mean Square Propagation*) (Hinton 2014)

$$\overline{g_i^2}(t) = \mu\, \overline{g_i^2}(t-1) + (1-\mu)\, g_i^2(t)$$

$$\vartheta_i(t) = \vartheta_i(t) - \gamma \underbrace{\frac{1}{\sqrt{\overline{g_i^2}(t) + \epsilon}}}_{\text{czynnik tłumiący}} g_i(t)$$

- $\overline{g_i^2}(t)$: wygładzone wykładniczo kwadraty współrzędnych gradientu $g_i^2(t)$
- *współczynnik tłumienia* (ang. *decay rate*) μ zwykle równy 0.9, 0.99, 0.999
- sugerowane $\epsilon = 10^{-8}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$

Zróżnicowane prędkości uczenia wag

Adam

D. Kingma (2015). "Adam: A Method for Stochastic Optimization". In: *ICLR*, poster

Adaptacyjna Estymacja Momentu Adam (Kingma 2015)

$$
\begin{aligned}
g(t) &= \operatorname{grad} f(\theta(t-1)) \\
\overline{g}_i(t) &= \beta_1\, \overline{g}_i(t-1) + (1-\beta_1)\, g_i(t) \\
\overline{g^2}_i(t) &= \beta_2\, \overline{g^2}_i(t-1) + (1-\beta_2)\, g_i^2(t) \\
\widehat{g}_i(t) &= \overline{g}_i(t)/(1-\beta_1^t) \\
\widehat{g}_i^2(t) &= \overline{g^2}_i(t)/(1-\beta_2^t) \\
\Delta\vartheta_i(t) &= \frac{-\gamma \widehat{g}_i(t)}{\sqrt{\widehat{g}_i^2(t)} + \epsilon}
\end{aligned}
$$

- $\widehat{g}_i(t)$ jest wygładzoną wykładniczo współrzędną gradientu
- $\widehat{g}_i^2(t)$ jest wygładzonym wykładniczo kwadratem współrzędnej gradientu
- $\gamma = 0.001$, $\mu_1 = 0.9$, $\mu_2 = 0.999$, $\epsilon \sim 10^{-4} - 10^{-8}$
- algorytm odpowiada algorytmowi RMSprop z gradientem wygładzanym wykładniczo

Inicjalizacja wag (1)

- wagi początkowe "za duże" - przedwczesne nasycenie, za "małe": zbyt małe gradienty, "śmierć" neuronów ReLU (Lee, Oh, and Kim 1991)
- jakie powinny być wartości oczekiwane i wariancje losowo inicjalizowanych wag aby wariancje wyjść poszczególnych warstw były identyczne, dla założonej liczby wejść p i wyjść q warstwy
- **uwaga** przez liczbę wejść/warstwy neuronu rozumie się liczbę neuronów w polu recepcyjnym (we wszystkich mapach aktywacji w sieciach 3D)
- popularne podejście: $\mathscr{E}w(0) = 0$, $\mathscr{V}w(0) = \frac{1}{3p}$, np.
  - $b(0) = 0$, $w(0) \sim \mathscr{U}_{(-a,a)}$, gdzie $a = \frac{1}{\sqrt{p}}$
  - $b(0) = 0$, $w(0) \sim \frac{1}{\sqrt{3p}}\mathscr{N}_{0,1}$,
- *metoda Xaviera* (Glorot and Bengio 2010): $\mathscr{V}w(0) = \frac{2}{p+q}$, np.
  - $b(0) = 0$, $w(0) \sim \mathscr{U}_{(-a,a)}$ gdzie $a = \sqrt{\frac{6}{p+q}}$
  - $b(0) = 0$, $w(0) \sim \frac{2}{p+q}\mathscr{N}_{0,1}$
- *metoda He* dla aktywacji ReLU (He et al. 2015): $b(0) = 0$, $w(0) \sim \sqrt{\frac{2}{p}}\mathscr{N}_{0,1}$

Inicjalizacja wag (2)

- *metoda Kumara* (Kumar 2017): $w(0) \sim \sigma\mathcal{N}_{0,1}$ gdzie
  - dla aktywacji ReLU $\sigma = \sqrt{\frac{2}{p}}$
  - dla różniczkowalnej funkcji aktywacji $\sigma = \frac{1}{\sqrt{p\, g'(0)^2\, (1+g^2(0))}}$
    - np. $1/\sqrt{p}$ dla $g(z) = \mathrm{tgh}(z)$
    - $3.6/\sqrt{p}$ dla sigmoidu $g(z) = 1/(1+\exp(z))$
- *randomizowana inicjalizacja asymetryczna* (Lu et al. 2019) dla funkcji aktywacji typu ReLU:
  - wybrać rozkład $\mathscr{P}$ na przedziale nieujemnym, (np. $\mathscr{B}_{2,\,1}$) i obliczyć

$$\sigma^2 = \sqrt{2}\Big(-\frac{m_1}{\sqrt{\pi}} + \sqrt{\frac{m_1^2}{\pi} + 1 - m_2}\Big)$$

    gdzie m_1, m_2 są pierwszymi dwoma momentami $\mathscr{P}$
  - $b(0) = 0$, $w(0)$: pierwsza warstwa zgodnie z He, kolejne warstwy
    - dla wybranej losowo wagi $w(0) \sim \mathscr{P}$
    - dla pozostałych wag $w(0) \sim \sigma\mathcal{N}_{0,1}$

Literatura

Materiały źródłowe

Chen, T. and H. Chen (1993). "Approximations of Continuous Functionals by Neural Networks with Applications to Dynamic Systems". In: *IEEE Transactions on Neural Networks* 4.6, pp. 910–918.

Chen, T., H. Chen, and R. Liu (1995). "Approximation capability in $C(\bar{\mathbb{R}}^n)$ by multilayer feedforward networks and related problems". In: *IEEE Transactions on Neural Networks* 6.1, pp. 25–30.

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2, pp. 303–314.

Freeman, J. A. and D. M. Skapura (1991). *Neural networks : algorithms, applications, and programming techniques*. Reading, Massachusetts, USA: Addison-Wesley.

Funahashi, K. (1989). "On the approximate realizations of continuous mappings by neural networks". In: *Neural Networks* 2.3, pp. 183–192.

Hornik, K., M. B. Stinchcombe, and H. White (1989). "Multilayer feedforward networks are univeral approximators". In: *Neural Networks* 2.5, pp. 359–366.

Jacobs, D., ed. (1977). *The State of the Art in Numerical Analysis*. London: Academic Press.

Press, W. et al. (1986, 1992). *Numerical Recipies. The Art of Scientific Computing*. Cambridge: Cambridge University Press.

Literatura (cd.)

Sprecher, D. A. (Mar. 1965). "On the structure of continuous functions of several variables". In: *Trans. Amer. Math. Society* 115, pp. 340–355.

Werbos, P. J. (1994). *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. New York: Wiley.

Materiały pomocnicze

Bengio, Y., N. Boulanger-Lewandowski, and R. Pascanu (2012). *Advances in optimizing recurrent networks*. Tech. rep. U. Montreal. URL: https://arxiv.org/pdf/1212.0901v2.pdf.

Blum, J. R. (June 1954). "Approximation Methods which Converge with Probability one". In: *The Annals of Mathematical Statistics* 25.2, pp. 382–386.

Duchi, J., E. Hazan, and Y. Singer (July 2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *JMLR* 12, pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.htm.

Fine, T. L. (1999). *Feedforward Neural Network Methodology*. New York: Springer Verlag.

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proc. of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 9. Sardinia, Italy.

Haykin, S. (1999). *Neural Networks. A Comprehensive Foundation*. Second Edition. Upper Saddle River, New Jersey, USA: Prentice Hall, Inc.

Literatura (cd.)

He, K. et al. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. Tech. rep. Microsoft Research. URL: //arxiv.org/pdf/1502.01852.pdf.

Hecht-Nielsen, R. (1987). "Kolmogorov's mapping neural network existence theorem". In: *Proceedings of the IEEE International Conference on Neural Networks*. Vol. I, pp. 11–13.

— (1990). *Neurocomputing*. Reading, Mass.: Addison-Wesley.

Jacobs, R. A. (1988). "Increased rates of convergence through learning rate adaptation". In: *Neural Networks* 1.4, pp. 295–307.

Karpathy, A. (2016). *Convolutional Neural Networks for Visual Recognition*. Tech. rep. 2016 class CS231n. URL: http://cs231n.github.io/.

Kingma, D. (2015). "Adam: A Method for Stochastic Optimization". In: *ICLR*, poster.

Kumar, S. K. (2017). *On weight initialization in deep neural networks*. Tech. rep. arXiv:1704.08863v2. URL: siddharthkumar@upwork.com,%20May%204,%202017.

Lee, Y., S. Oh, and M. Kim (1991). "The effect of initial weights on premature saturation in back-propagation learning". In: *Proc. IEEE Third Int. Joint Conf. on Neural Networks*. Vol. 1, pp. 765–770.

Lu, L. et al. (2019). *Dying ReLU and Initialization: Theory and Numerical Examples*. Tech. rep. arXiv:1903.06733.

Literatura (cd.)

Nesterov, Y. (1983). "A method for unconstrained convex minimization problem with the rate of convergence o(1/k2)". In: *Doklady AN SSSR (Soviet. Math. Docl.)* 269, pp. 543–547.

Refsnæs, R. H. (2009). "A brief introduction to the conjugate gradient method". URL: folk.idi.ntnu.no/elster/tdt24/tdt24-f09/cg.pdf.

Robbins, H. and S. Monro (1951). "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3.

Ruder, S. (2017). *An overview of gradient descent optimization algorithms*. Tech. rep. arxiv.org/abs/1609.04747. Cornell University.

Sandberg, I. W. (May 1991). "Approximation theorems form discrete-time systems". In: *IEEE Trans. Circuits and Systems* 38.5, pp. 564–566.

Sutskever, I. (2012). "Training Recurrent Neural Networks". PhD thesis. CS Dept., U. Toronto.

Widrow, B. and M. Lehr (Sept. 1990). "30 years of adaptive neural networks: perceptron, madaline, and backpropagation". In: *Proceedings of IEEE* 78.9.

Temat 12: Propagacja wsteczna gradientu

12.1 Łańcuch funkcji
12.2 Funkcje na grafie
12.3 Zastosowanie w sieciach neuronowych

12.1 Łańcuch funkcji
12.2 Funkcje na grafie
12.3 Zastosowanie w sieciach neuronowych

Pochodna funkcji złożonej

Łańcuch funkcji jednej zmiennej

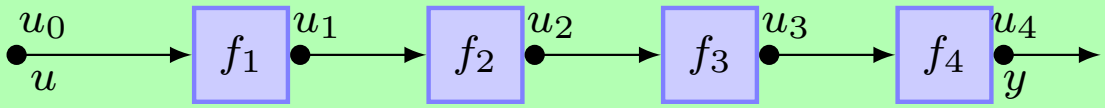
Ⓔ Przykład: Łańcuch funkcji jednej zmiennej

$$\begin{aligned} u_4 &= f_4(u_3) \\ u_3 &= f_3(u_2) \\ u_2 &= f_2(u_1) \\ u_1 &= f_1(u_0) \end{aligned}$$

$$u_4 = f_4\Big(f_3\Big(f_2(f_1(u_0))\Big)\Big)$$

$$f = f_4 \circ f_3 \circ f_2 \circ f_1$$

$$D^i_j = D^{u_i}_{u_j} \stackrel{\text{def}}{=} \frac{\mathrm{d}u_i}{\mathrm{d}u_j},\ i \geqslant j$$



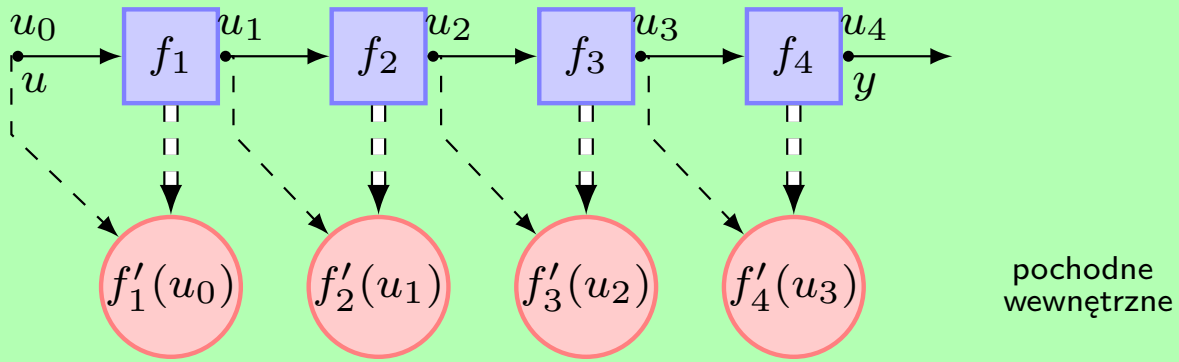
- wygodnie utożsamiać nazwę funkcji z jej wartością, np. przez dziecko u_i rozumiemy dziecko węzła f_i
- funkcja złożona reprezentowana jest przez ścieżkę grafu
- problem: znaleźć pochodną funkcji złożonej $\dfrac{\mathrm{d}y}{\mathrm{d}u}$

Pochodna funkcji złożonej

Łańcuch funkcji jednej zmiennej - pochodne wewnętrzne

Ⓔ Przykład: Pochodne wewnętrzne

$$u_4 = f_4\Big(f_3\Big(f_2(f_1(u_0))\Big)\Big) \qquad f = f_4 \circ f_3 \circ f_2 \circ f_1$$



- $\dfrac{\mathrm{d}y}{\mathrm{d}u}$: iloczyn pochodnych wewnętrznych **w dowolnej kolejności**

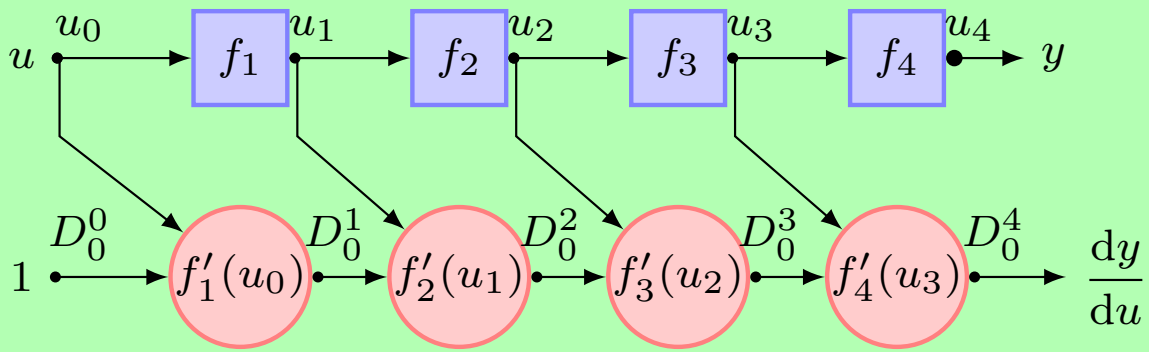
$$\Longrightarrow \qquad\qquad\qquad\qquad \Longleftarrow$$

$$\begin{aligned} D_0^1 &= f_1'(u_0) & D_3^4 &= f_4'(u_3) \\ D_0^2 &= D_0^1 f_2'(u_1) & D_2^4 &= f_3'(u_2) D_3^4 \\ D_0^3 &= D_0^2 f_3'(u_2) & D_1^4 &= f_2'(u_1) D_2^4 \\ D_0^4 &= D_0^3 f_4'(u_3) & D_0^4 &= f_1'(u_0) D_1^4 \end{aligned}$$

Pochodna funkcji złożonej

Łańcuch funkcji jednej zmiennej - propagacja prosta

Ⓔ Przykład: Łańcuch funkcji jednej zmiennej - propagacja prosta

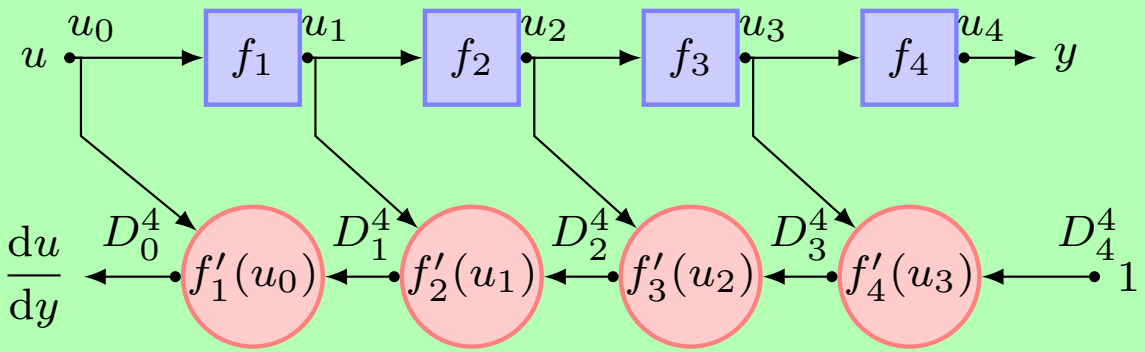


- propagacja prosta: mnożenie od wejścia
- wyniki pośrednie: pochodne kolejnych zmiennych względem wejścia - nieużyteczne dla sieci neuronowych

Pochodna funkcji złożonej

Łańcuch funkcji jednej zmiennej - propagacja wsteczna

Ⓔ Przykład: Łańcuch funkcji jednej zmiennej - propagacja wsteczna



- propagacja wsteczna: mnożenie pochodnych wewnętrznych "od wyjścia do wejścia"
- wyniki pośrednie: pochodne wyjścia względem zmiennych pośrednich

  $\Rightarrow$ wpływ wariacji zmiennych pośrednich na wariacje wyjścia

Propagacja gradientu dla łańcucha funkcji

Porównanie FP i BP

- łańcuch funkcji skalarnych jednej zmiennej f_k, $k = 1, \ldots, \ell$
- obliczyć pochodną $f'(u)$ funkcji złożonej $f = f_\ell \circ f_{\ell-1} \cdots \circ f_1$ dla danego u

Algorytm propagacji prostej (FP) dla łańcucha funkcji

- inicjalizacja: $u_0 = u$, $D^0 = 1$
- kroki "do przodu": obliczanie wartości funkcji i pochodnych: dla $k = 1, \ldots, \ell$

$$D^k = D^{k-1} \frac{\mathrm{d} f_k(u_{k-1})}{\mathrm{d} u_{k-1}}$$
$$u_k = f_k(u_{k-1})$$

- $\frac{\mathrm{d} f}{\mathrm{d} u}(u) = D^\ell$

Algorytm propagacji wstecznej (BP) dla łańcucha funkcji

- inicjalizacja: $u_0 = u$, $D_0 = 1$
- kroki "do przodu" - obliczenie wartości funkcji: dla $k = 1, \ldots, \ell$

$$u_k = f_k(u_{k-1})$$

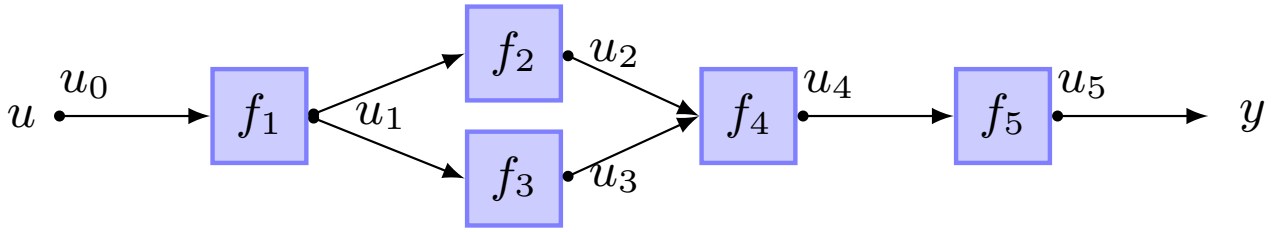
- kroki wstecz - obliczenie wartości pochodnych dla $j = \ell - 1, \ldots, 0$

$$D_j = D_{j+1} \frac{\mathrm{d} f_{j+1}(u_j)}{\mathrm{d} u_j},$$

- $\frac{\mathrm{d} f}{\mathrm{d} u}(u) = D_0$

12.1 Łańcuch funkcji
12.2 Funkcje na grafie
12.3 Zastosowanie w sieciach neuronowych

Obliczenia na grafie



- funkcja złożona reprezentowana przez *graf obliczeniowy*: *graf skierowany bez pętli* (ang. *directed acyclic graph*) DAG
  - węzły grafu: funkcje
  - krawędzie przychodzące: argumenty funkcji, krawędź wychodząca: wartość funkcji
- *kolejność topologiczna* węzłów: jeśli $A \prec B$ tzn. węzeł A jest przodkiem węzła B, to $\#A < \#B$; **uwaga**: nie jest odwrotnie, tzn. dla numeracji topologicznej węzłów jeśli zachodzi $\#A < \#B$, to niekoniecznie A jest przodkiem B
- *propagacja prosta* obliczeń na węzłach sieci : obliczanie w porządku topologicznym
- *propagacja wsteczna* obliczeń: obliczanie w kolejności przeciwnej niż porządek topologiczny

Obliczenia pochodnych funkcji złożonej

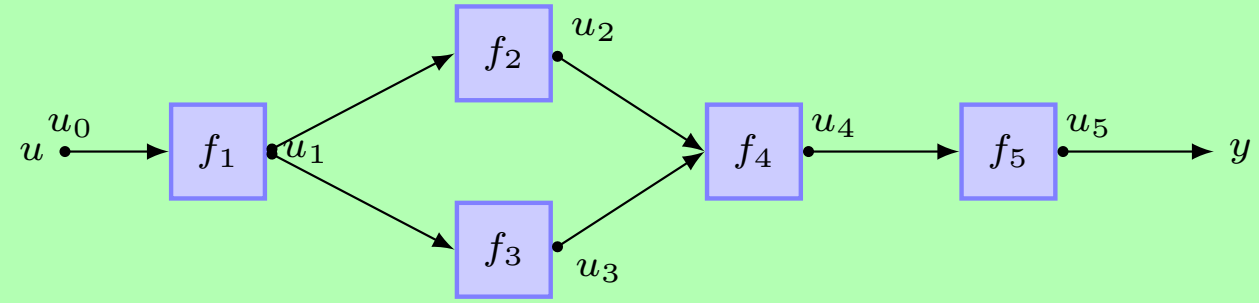
- pochodne dowolnej zmiennej y (krawędzi wychodzącej z pewnego węzła f_j) względem innej zmiennej u (krawędzi dochodzącej do pewnego węzła f_i, $i \leqslant j$) jest sumą iloczynów pochodnych wewnętrznych dla wszystkich ścieżek od f_i do f_j
- *pochodne wewnętrzne*: pochodne cząstkowa wyjścia węzła względem wejścia węzła
- iloczyn pochodnych wewnętrznych w ramach ścieżki w dowolnej kolejności

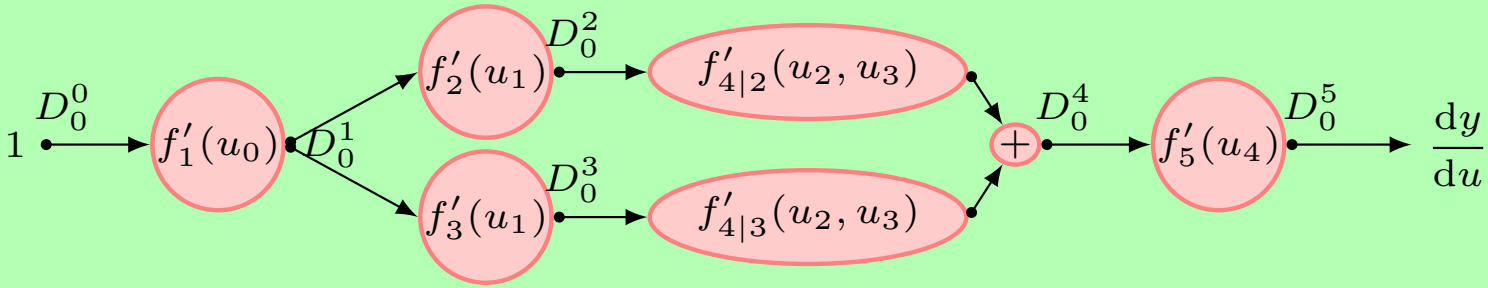
Pochodna funkcji złożonej

Idea propagacji prostej dla funkcji wielu zmiennych

Ⓔ Przykład: Propagacja prosta dla funkcji wielu zmiennych

$$
\begin{aligned}
u_1 &= f_1(u_0)\\
u_2 &= f_2(u_1)\\
u_3 &= f_3(u_1)\\
u_4 &= f_4(u_2, u_3)\\
u_5 &= f_5(u_4)
\end{aligned}
$$

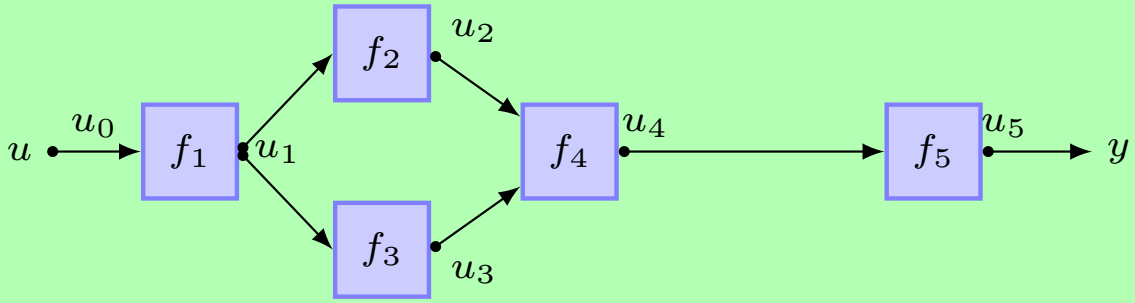




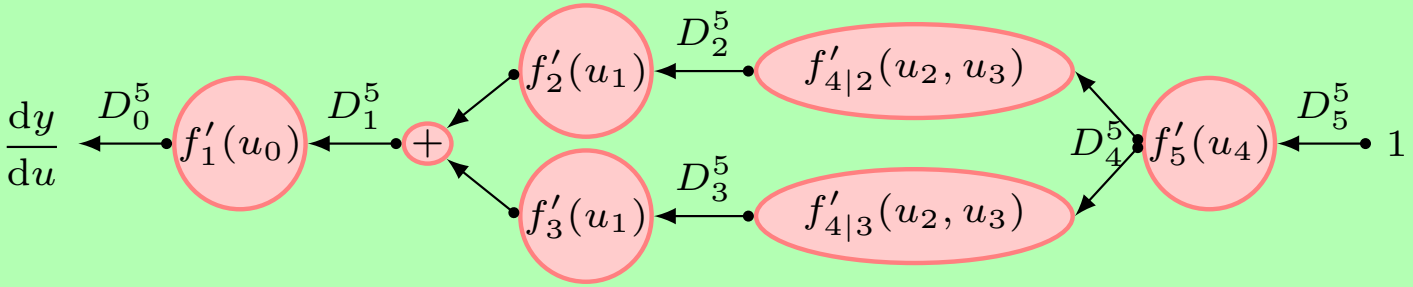
Pochodna funkcji złożonej

Idea propagacji wstecznej dla funkcji wielu zmiennych

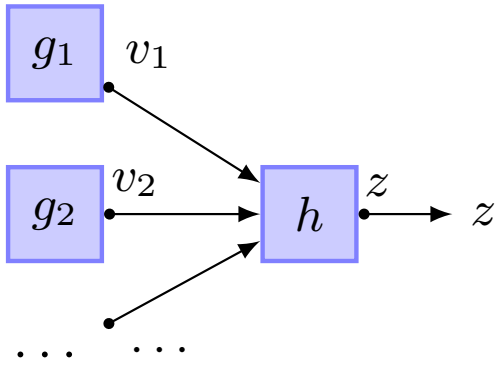
Ⓔ Przykład: Propagacja wsteczna dla funkcji wielu zmiennych



$$
\begin{aligned}
u_1 &= f_1(u_0)\\
u_2 &= f_2(u_1)\\
u_3 &= f_3(u_1)\\
u_4 &= f_4(u_2, u_3)\\
u_5 &= f_5(u_4)
\end{aligned}
$$



Propagacja prosta (FP) gradientu



Propagacja prosta (FP)

- obliczanie wartości funkcji wewnętrznych i ich pochodnych oraz wartości pochodnych D^z dla każdego węzła z w kolejności zgodnej z porządkiem topologicznym

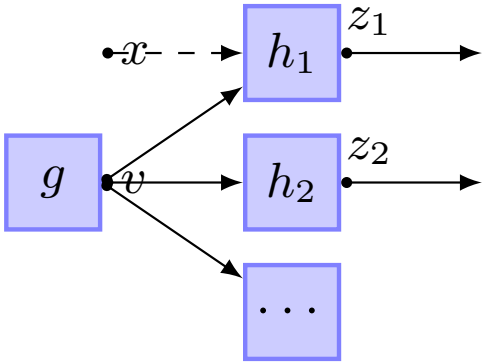
$$z = h(\boldsymbol{v})$$

$$D^z = \sum_{v_i \in \mathcal{P}h} D^{v_i} \frac{\partial h}{\partial v_i} \qquad \text{(reguła FP)}$$

gdzie

- $\mathcal{P}h$ oznacza zbiór rodziców węzła h, tzn. węzłem docelowym (ang. *target*) sygnału v jest h
- $D^z = \dfrac{\partial z}{\partial u}$ gdzie u jest argumentem funkcji złożonej

Propagacja wsteczna (BP) gradientu



Propagacja wsteczna (BP)

- ścieżka wprzód (ang. *forward pass*): obliczanie wartości funkcji wewnętrznych dla każdego węzła w kolejności zgodnej z porządkiem topologicznym
- ścieżka wstecz (ang. *backward pass*): obliczanie wartości pochodnych wewnętrznych i pochodnych D_v dla każdego węzła w kolejności przeciwnej do porządku topologicznego

$$D_v = \sum_{h_i \in \mathcal{C}v} D_{z_i} \frac{\partial h_i}{\partial v} \qquad \text{(reguła BP)}$$

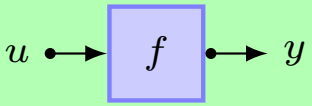
gdzie
  - $\mathcal{C}v$ oznacza dzieci węzła o wyjściu v
  - $D_v = \dfrac{\partial y}{\partial v}$ gdzie y jest wyjściem funkcji złożonej (np. wartością kosztu lub wyjściem warstwy czy sieci)

Propagacja wsteczna (BP) gradientu

Przykładowe węzły obliczeniowe

Ⓔ Funkcja $\mathbb{R} \mapsto \mathbb{R}$

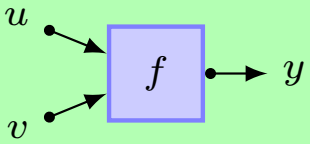
$$y = f(u)$$



$$D_u = D_y\, f'$$

Ⓔ Funkcja $\mathbb{R}^2 \mapsto \mathbb{R}$

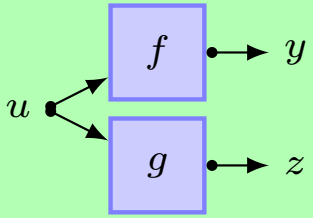
$$y = f(u, v)$$



$$D_u = D_y\, f'_u(u, v)$$
$$D_v = D_y\, f'_v(u, v)$$

Ⓔ Funkcja $\mathbb{R} \mapsto \mathbb{R}^2$

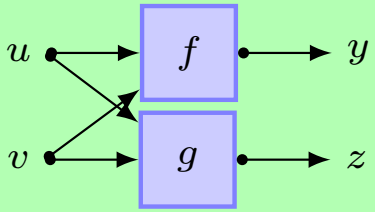
$$y = f(u)$$
$$z = g(u)$$



$$D_u = D_y\, f'(u) + D_z\, g'(u)$$

Ⓔ Funkcja $\mathbb{R}^2 \mapsto \mathbb{R}^2$

$$y = f(u, v)$$
$$z = g(u, v)$$

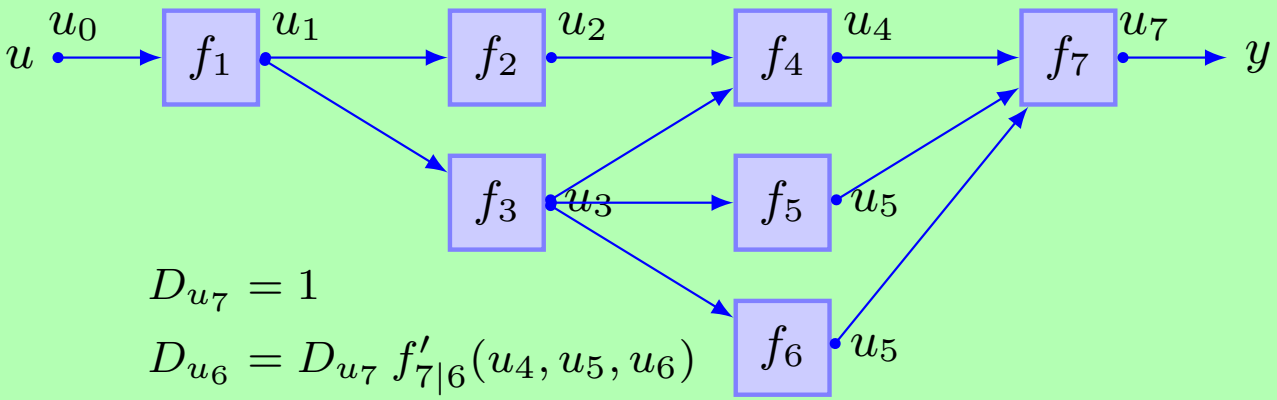


$$D_u = D_y\, f'_u(u, v) + D_z\, g'_u(u, v)$$
$$D_v = D_y\, f'_v(u, v) + D_z\, g'_v(u, v)$$

Przykład

Obliczanie pochodnych przy użyciu BP

Zapisać algorytm obliczenia $\frac{\mathrm{d}f}{\mathrm{d}u}$ przy użyciu BP dla funkcji $y = f(u)$ (rys.)



$$D_{u_7} = 1$$

$$D_{u_6} = D_{u_7}\, f'_{7|6}(u_4, u_5, u_6)$$

$$D_{u_5} = D_{u_7}\, f'_{7|5}(u_4, u_5, u_6)$$

$$D_{u_4} = D_{u_7}\, f'_{7|4}(u_4, u_5, u_6)$$

$$D_{u_3} = D_{u_4}\, f'_{4|3}(u_2, u_3) + D_{u_5}\, f'_{5|3}(u_3) + D_{u_6}\, f'_{6|3}(u_3)$$

$$D_{u_2} = D_{u_3}\, f'_{4|2}(u_2, u_3)$$

$$D_{u_1} = D_{u_2}\, f'_{2|1}(u_1) + D_{u_3}\, f'_{3|1}(u_1)$$

$$\frac{\mathrm{d}f}{\mathrm{d}u} = D_{u_0} = D_{u_1}\, f'_{1|0}(u_1)$$

Przykład

BP vs. FP

Ⓔ Przykład: Wyniki algorytmów FP i BP są identyczne

Porównać wyniki obliczenia pochodnych $\frac{\mathrm{d}x_3}{\mathrm{d}x_0}$ przy użyciu algorytmów FP i BP dla systemu:

$$x_1 = f_1(x_0)$$
$$x_2 = f_2(x_0, x_1)$$
$$x_3 = f_3(x_0, x_1, x_2)$$

propagacja prosta gradientu

$$D^{x_1} = f'_{1|0}$$
$$D^{x_2} = f'_{2|0} + f'_{2|1}\, D^{x_1}$$
$$D^{x_3} = f'_{3|0} + f'_{3|1}\, D^{x_1} + f'_{3|2}\, D^{x_2}$$

propagacja wsteczna gradientu

$$D_{x_2} = f'_{3|2}$$
$$D_{x_1} = f'_{3|1} + D_{x_2}\, f'_{2|1}$$
$$D_{x_0} = f'_{3|0} + D_{x_2}\, f'_{2|0} + D_{x_1}\, f'_{1|0}$$

Oczywiście wyniki działania obu algorytmów są identyczne

$$\frac{\mathrm{d}x_3}{\mathrm{d}x_0} = f'_{3|0} + f'_{3|1}\, f'_{1|0} + f'_{3|2}\, f'_{2|0} + f'_{3|2}\, f'_{2|1}\, f'_{1|0}$$

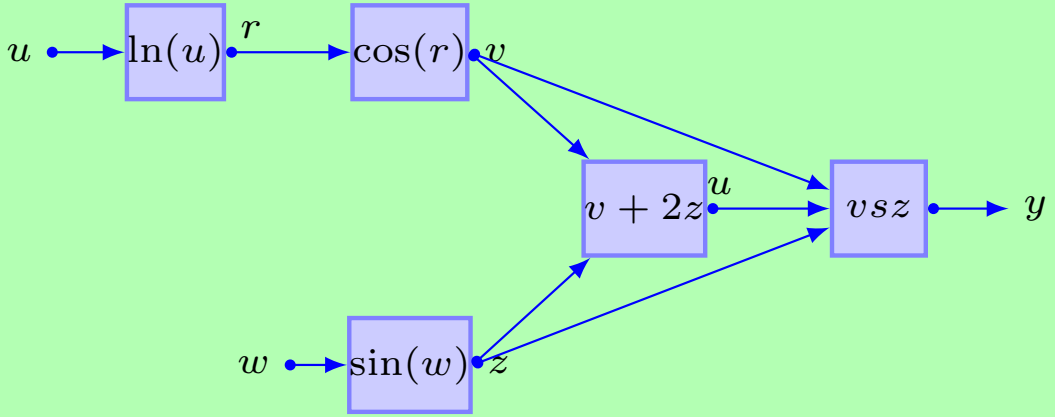
Przykład

Obliczenie pochodnych przy użyciu BP (2)

Ⓔ Przykład

Zapisać algorytm obliczenia pochodnych $\frac{\partial y}{\partial u}$, $\frac{\partial y}{\partial w}$ przy użyciu BP dla systemu

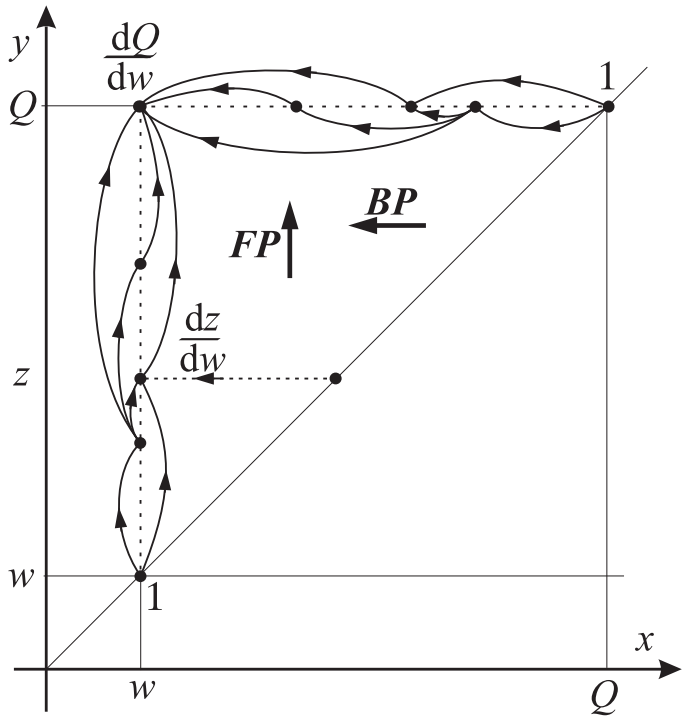
$$
\begin{aligned}
r &= \ln(u) \\
v &= \cos r \\
z &= \sin w \\
s &= v + 2z \\
y &= v\, s\, z
\end{aligned}
$$



$$
\begin{aligned}
D_y &= 1 \\
D_s &= v\, z D_y \\
D_v &= D_s + s\, z D_y \\
D_z &= 2 D_s + s\, v D_y \\
D_r &= -\sin(r) D_v \\
\frac{\partial y}{\partial u} &= D_u = \frac{1}{u} D_r \\
\frac{\partial y}{\partial w} &= D_w = \cos(w) D_z
\end{aligned}
$$

Mapa pochodnych sieci

Porównanie FP i BP



wierzchołki grafu na osiach, uporządkowane topologicznie

- propagacja prosta gradientu (FP): pochodne zmiennych pośrednich *względem zmiennych niezależnych*
- propagacja wsteczna gradientu (BP): pochodne *zmiennych zależnych* względem zmiennych pośrednich
- w sieciach neuronowych: potrzebne pochodne kosztu względem wag i obciążeń

Postać macierzowa

- macierze pochodnych dla dowolnej sieci

$$\boldsymbol{F}' = \{\frac{\partial f_i}{\partial u_j}\}_{\substack{i=0,\ldots,n \\ j=0,\ldots,n}} = \begin{bmatrix} 0 & 0 & \ldots & 0 \\ * & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \ldots & 0 \end{bmatrix}, \boldsymbol{D} = \{D^i_j\}_{\substack{j=0,\ldots,n \\ i=0,\ldots,n}} = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ * & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \ldots & 1 \end{bmatrix}$$

- w szczególności, dla łańcucha i dla sieci warstwowej (macierz blokowa)

$$\boldsymbol{F}' = \begin{bmatrix} 0 & & & \\ \frac{\mathrm{d}f_2}{\mathrm{d}u_1} & 0 & & \\ & \ddots & \ddots & \\ & & \frac{\mathrm{d}f_n}{\mathrm{d}u_{n-1}} & 0 \end{bmatrix} \quad \boldsymbol{F}' = \begin{bmatrix} \boldsymbol{O} & & & \\ \frac{\mathrm{d}\boldsymbol{f}_2}{\mathrm{d}\boldsymbol{u}_1} & \boldsymbol{O} & & \\ & \ddots & \ddots & \\ & & \frac{\mathrm{d}\boldsymbol{f}_n}{\mathrm{d}\boldsymbol{u}_{n-1}} & \boldsymbol{O} \end{bmatrix}$$

- postaci macierzowe

propagacja prosta gradientów $\boldsymbol{F}'\,\boldsymbol{D} = \boldsymbol{D} - \boldsymbol{I}$

propagacja wsteczna gradientów $\boldsymbol{D}\,\boldsymbol{F}' = \boldsymbol{D} - \boldsymbol{I}$

12.1 Łańcuch funkcji
12.2 Funkcje na grafie
12.3 Zastosowanie w sieciach neuronowych

Propagacja wsteczna kosztu dla perceptronu dwuwarstwowego

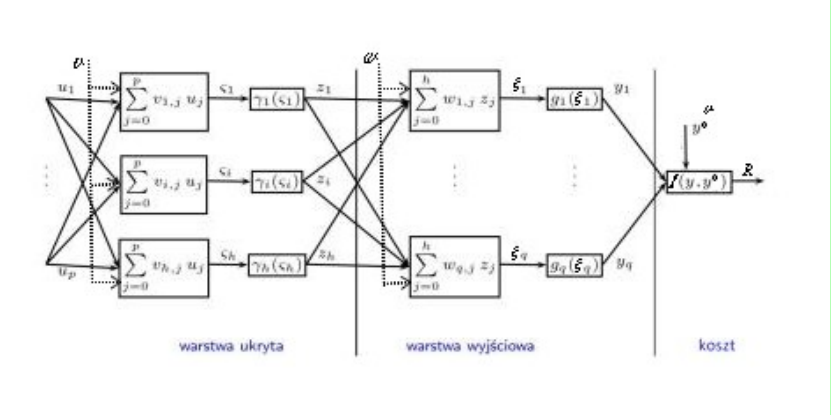
Ⓔ BP dla sieci dwuwarstwowej

- sieć dwuwarstwowa:

$$\zeta_j = \sum_{i=1}^{p} v_{j,i} u_i, \quad s_j = \gamma_j(\zeta_j), \quad z_k = \sum_{j=1}^{h} w_{k,j} s_j, \quad y_k = g_k(z_k), \quad R = \sum_{k=1}^{q} \ell(y_k, y_k^o)$$

- algorytm BP (ścieżka wstecz):

$$D_R = 1$$
$$D_{y_k} = D_R \ell'_{y_k}(y_k, y_k^o), \quad k = 1, \ldots, q$$
$$D_{z_k} = D_{y_k} g'_k(z_k), \quad k = 1,.., q$$
$$D_{w_{k,j}} = D_{z_k} s_j, \quad k = 1,.., q,$$
$$j = 1,.., h$$
$$D_{s_j} = \sum_{k=1}^{q} D_{z_k} w_{k,j}, \quad j = 1,.., h$$
$$D_{\zeta_j} = D_{s_j} \gamma'_j(\zeta_j), \quad j = 1,.., h$$
$$D_{v_{j,i}} = D_{\zeta_j} u_i, \quad j = 1,.., h, i = 1,.., p$$
$$D_{u_i} = \sum_{j=1}^{h} D_{\zeta_j} v_{j,i}, \quad i = 1,.., p$$



- w rezultacie
  - pochodne kosztu względem wag: $\dfrac{\partial R}{\partial v_{i,j}} = D_{v_{i,j}}, \ \dfrac{\partial R}{\partial w_{i,j}} = D_{w_{i,j}}$
  - pochodna kosztu względem wejścia sieci: $\dfrac{\partial R}{\partial u_j} = D_{u_j}$

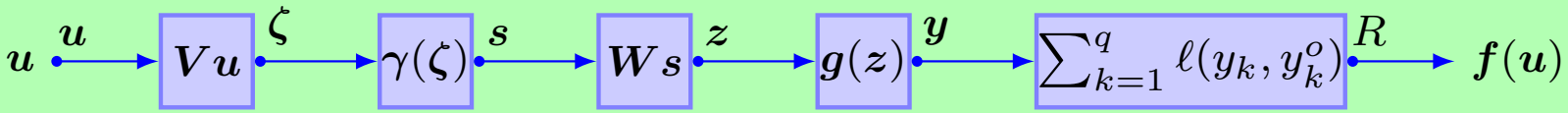
Propagacja wsteczna kosztu dla perceptronu dwuwarstwowego

Zapis wektorowy

Ⓔ BP: pochodna kosztu dla sieci dwuwarstwowej - Zapis wektorowy

- sieć dwuwarstwowa

$$\zeta = \boldsymbol{V}\boldsymbol{u}, \qquad \boldsymbol{s} = \gamma(\zeta), \qquad \boldsymbol{z} = \boldsymbol{W}\boldsymbol{s}, \qquad \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{z}), \qquad R = \sum_{k=1}^{q} \ell(y_k, y_k^o),$$



- algorytm BP

$D_R = 1$
$D_{\boldsymbol{y}} = D_R \ell'_{\boldsymbol{y}}(\boldsymbol{y}, \boldsymbol{y}^o)$
$D_{\boldsymbol{z}} = D_{\boldsymbol{y}} \cdot \boldsymbol{g}'(\boldsymbol{z})$
$D_{\boldsymbol{W}} = D_{\boldsymbol{z}}\, \boldsymbol{s}^\mathsf{T}$
$D_{\boldsymbol{s}} = \boldsymbol{W}^\mathsf{T} D_{\boldsymbol{z}}$
$D_\zeta = D_{\boldsymbol{y}} \cdot \gamma'_i(\zeta)$
$D_{\boldsymbol{V}} = D_\zeta\, \boldsymbol{u}^\mathsf{T}$
$D_{\boldsymbol{u}} = \boldsymbol{V}^\mathsf{T} D_\zeta$

- stąd
  - pochodne kosztu względem wag:
    $\dfrac{\partial R(\boldsymbol{s})}{\partial \boldsymbol{V}} = D_{\boldsymbol{V}}, \qquad \dfrac{\partial R(\boldsymbol{u})}{\partial \boldsymbol{W}} = D_{\boldsymbol{W}}$
  - pochodna kosztu względem wejść sieci:
    $\dfrac{\partial R(\boldsymbol{u})}{\partial \boldsymbol{u}} = D_{\boldsymbol{u}}$

⊕ BP dla sieci podstawowej

Ⓔ BP przez podstawową sieć dwuwarstwową

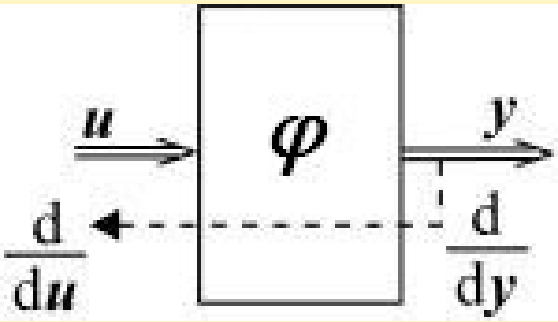
dla sieci podstawowej

warstwa ukryta	$\gamma_i(\zeta) = 1/(1+\exp(-\alpha\zeta))$	
	$\gamma_i'(\zeta) = \alpha\zeta(1-\zeta),\ i = 1,..,h$	$\boldsymbol{\gamma}'(\boldsymbol{\zeta}) = \alpha\,\boldsymbol{.}\,\boldsymbol{s}\,\boldsymbol{.}(1-\boldsymbol{s})$
warstwa wyjściowa	$g_i(z) = z$	$\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{z}$
funkcja strat	$\ell(y_k, y_k^o) = \frac{1}{2}\|y_k - y_k^o\|^2$	$\ell(\boldsymbol{y}, \boldsymbol{y}^o) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{y}^o\|^2$
	$\ell_k' = y_k - y_k^o$	$\ell' = \boldsymbol{y} - \boldsymbol{y}^o$
elementy BP		
	$D_{y_i} = y_i - y_i^o$	$D_{\boldsymbol{y}} = \boldsymbol{y} - \boldsymbol{y}^o$
	$D_{z_i} = D_{y_i}$	$D_{\boldsymbol{z}} = D_{\boldsymbol{y}}$
	$D_{\zeta_j} = \alpha D_{s_j}$	$D_{\boldsymbol{\zeta}} = \alpha D_{\boldsymbol{s}}\,\boldsymbol{.}\,\boldsymbol{s}\,\boldsymbol{.}(1-\boldsymbol{s})$

Moduł BP dla warstwy

Postać macierzowa

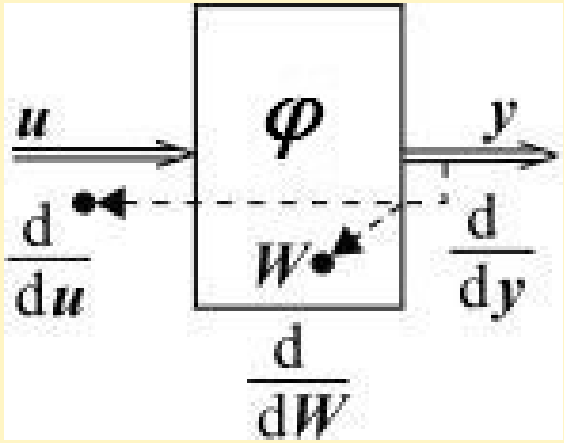
BP przez warstwę do wejść



$$D_z = D_y \,.\, g'(z)$$
$$D_u = W^\top D_z$$

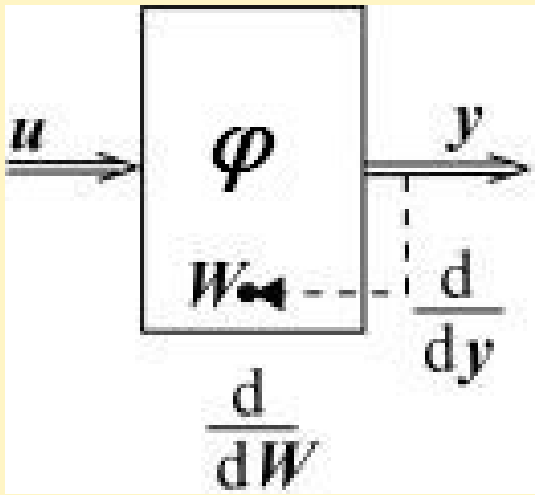
$$z = W u$$
$$y = g(z)$$

Moduł BP dla warstwy



$$D_z = D_y \,.\, g'(z)$$
$$D_W = D_z\, u^\top$$
$$D_u = W^\top D_z$$

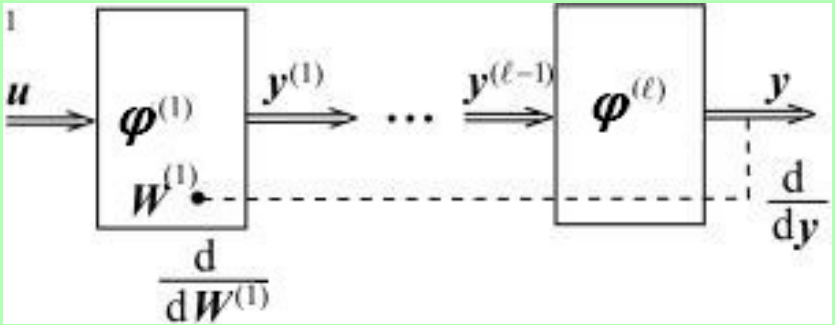
BP przez warstwę do wag



$$D_z = D_y \,.\, g'(z)$$
$$D_W = D_z\, u^\top$$

⊕ Propagacja wsteczna przez sieć

Ⓔ BP przez sieć wielowarstwową do wag



dane : $\boldsymbol{y}, \boldsymbol{y}^o, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L)},$

$\boldsymbol{D}_y = \ell'(\boldsymbol{y})$

for $k = L$ **step** -1 **to** 1

$\quad \boldsymbol{D}_z = \boldsymbol{D}_y \,.\, \boldsymbol{g}'(\boldsymbol{z})$

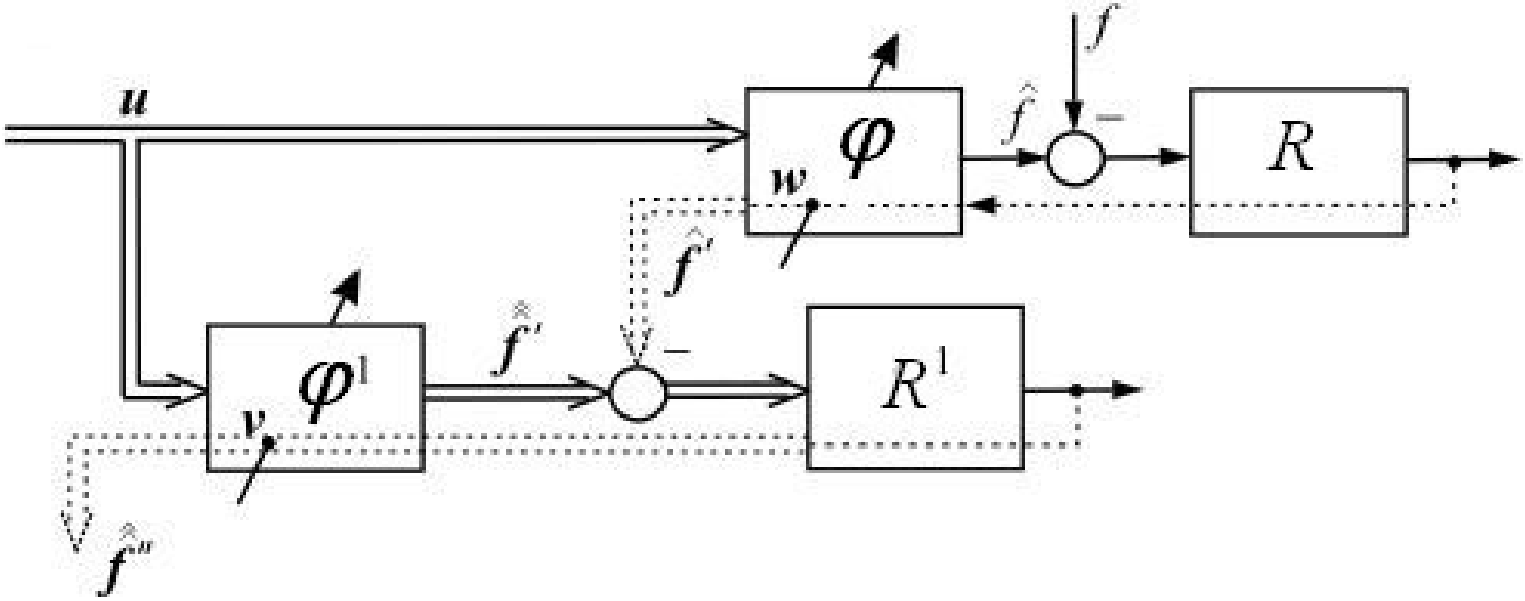
$\quad \boldsymbol{D}_{\boldsymbol{W}^{(k)}} = \boldsymbol{D}_z \boldsymbol{u}^\mathsf{T}$

$\quad \boldsymbol{D}_y = \boldsymbol{W}^{(k)^\mathsf{T}} \boldsymbol{D}_z$

endfor

Drugie pochodne

- $y^o = f(\boldsymbol{u})$ jest wartością pożądaną dla sieci ϕ z wejściem $\boldsymbol{u}$
- ϕ aproksymuje funkcję f: $\phi(\boldsymbol{u}) = \widehat{f}(\boldsymbol{u})$
- BP przez sieć ϕ oblicza gradient $\widehat{f}$: $\widehat{f}'(\boldsymbol{u}) = \dfrac{\mathrm{d}\phi(\boldsymbol{u})}{\mathrm{d}\boldsymbol{u}} \approx f'(\boldsymbol{u})$
- $\widehat{f}'(\boldsymbol{u})$ są wartościami pożądanymi dla sieci ϕ^1 z wejściem $\boldsymbol{u}$
- ϕ^1 aproksymuje funkcję $\widehat{f}'$: $\phi^1(\boldsymbol{u}) = \widehat{\widehat{f}'}$
- BP przez sieć ϕ^1 aproksymuje wartości aproksymowanego hessianu $\widehat{\widehat{f}'}'$



Literatura

Materiały źródłowe

Chen, T. and H. Chen (1993). "Approximations of Continuous Functionals by Neural Networks with Applications to Dynamic Systems". In: *IEEE Transactions on Neural Networks* 4.6, pp. 910–918.

Chen, T., H. Chen, and R. Liu (1995). "Approximation capability in $C(\bar{\mathbb{R}}^n)$ by multilayer feedforward networks and related problems". In: *IEEE Transactions on Neural Networks* 6.1, pp. 25–30.

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2, pp. 303–314.

Funahashi, K. (1989). "On the approximate realizations of continuous mappings by neural networks". In: *Neural Networks* 2.3, pp. 183–192.

Hornik, K., M. B. Stinchcombe, and H. White (1989). "Multilayer feedforward networks are univeral approximators". In: *Neural Networks* 2.5, pp. 359–366.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning internal representations by error propagation". In: *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*. Ed. by D. E. Rumelhart and J. McClelland. Vol. 1. Cambridge, MA: MIT Press. Chap. 8.

Sprecher, D. A. (Mar. 1965). "On the structure of continuous functions of several variables". In: *Trans. Amer. Math. Society* 115, pp. 340–355.

Werbos, P. J. (1974). "Beyond regression: New tools for prediction and analysis in the behavioral sciences". Ph.D. Thesis. Cambridge, MA: Harvard University.

Literatura (cd.)

— (1994). *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. New York: Wiley.

Materiały pomocnicze

Fine, T. L. (1999). *Feedforward Neural Network Methodology*. New York: Springer Verlag.

Haykin, S. (1999). *Neural Networks. A Comprehensive Foundation*. Second Edition. Upper Saddle River, New Jersey, USA: Prentice Hall, Inc.

Hecht-Nielsen, R. (1987). "Kolmogorov's mapping neural network existence theorem". In: *Proceedings of the IEEE International Conference on Neural Networks*. Vol. I, pp. 11–13.

— (1990). *Neurocomputing*. Reading, Mass.: Addison-Wesley.

Rumelhart, D. E. and J. McClelland, eds. (1986). *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*. Cambridge, MA: MIT Press.

Sandberg, I. W. (May 1991). "Approximation theorems form discrete-time systems". In: *IEEE Trans. Circuits and Systems* 38.5, pp. 564–566.

Część IV: Głębokie sieci neuronowe

Temat 13. Obliczenia głębokie

Temat 14. Głębokie sieci splotowe

Temat 15. Generatywne sieci antagonistyczne

Temat 13: Obliczenia głębokie

13.1 Specyfika obliczeń głębokich
13.2 Obrazy jako wejścia sieci

13.1 Specyfika obliczeń głębokich
13.2 Obrazy jako wejścia sieci

⊛ Głębokość systemu obliczeniowego

- system obliczeniowy /funkcja złożona / sieć neuronowa jako skierowany *graf przepływów* (ang. *flow graph*);
  - każdy wierzchołek reprezentuje pewną *funkcję elementarną*
  - krawędzie reprezentują wartości liczbowe:

    krawędzie dochodzące do wierzchołka są wartościami argumentów tej funkcji
    krawędzie wychodzące z wierzchołka są równe wartości funkcji dla tych argumentów
  - wierzchołek bez rodziców utożsamiamy z wejściem systemu
  - wierzchołek bez potomków utożsamiamy z wyjściem systemu
- przykłady rodzin funkcji elementarnych
  - $\{+, -, *, /\}$
  - {neurony formalne tzn. funkcje aktywacji kombinacji liniowej wejść}
  - {neurony RBF, tzn. funkcje gaussowskie kombinacji liniowej wejść}
  - ...
- *głębokość grafu przepływów*: długość najdłuższej ścieżki we-wy
- głębokość sieci neuronowej: liczba neuronów podlegających treningowi na najdłuższej scieżce
- architektura głęboka: głębokość > 2

Aspekty obliczeń głębokich

Efektywność reprezentacji funkcji

- zgodnie z „neuronowymi" twierdzeniami aproksymacyjnymi, sieć o głębokości 2 i odpowiednio dużej liczbie neuronów warstwy ukrytej dowolnie dokładnie aproksymuje dowolną funkcję ciągłą / całkowalną
- problem złożoności obliczeń: niezbędny rozmiar ("szerokość") warstw sieci
- *problem efektywności:* istnieją funkcje aproksymowane dla których dla określonej dokładności niezbędny rozmiar warstwy ukrytej rośnie **wykładniczo z liczbą wejść** dla $d = 2$ warstw, ale **liniowo** dla $d > 2$ warstw

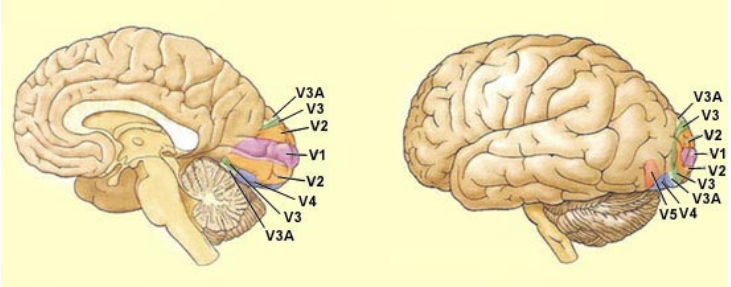
⊛ dla rodziny neuronów RBF, istnieją funkcje wymagające liczby wierzchołków rosnącej wykładniczo z liczbą wejść dla głębokości d, ale o liniowo rosnącej liczbie wierzchołków dla głębokości $d + 1$ (Hastad)

- ogólniej: **istnieją funkcje reprezentowane efektywnie dla architektur głębokich, ale nie dla płytkich**
- **no free lunch** funkcja wybrana losowo może nie być reprezentowana efektywnie, niezależnie od architektury obliczeń - brak znajomości struktury wewnętrznej względem rodziny funkcji elementarnych

Aspekty obliczeń głębokich

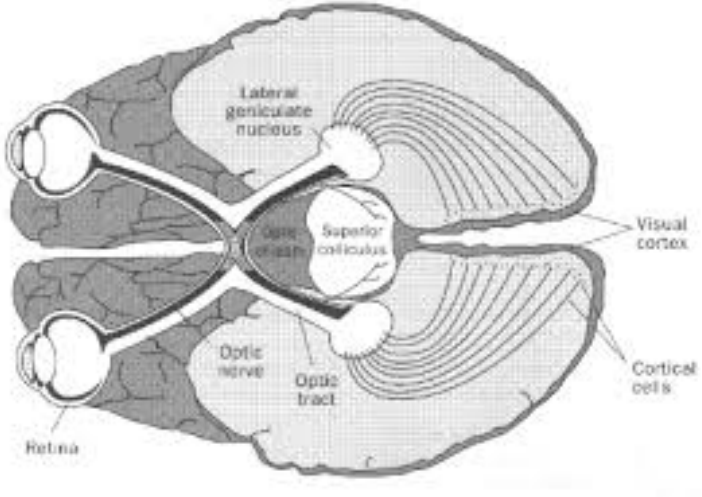
Mózg - system biologicznych sieci głębokich

- kora mózgu: system wielowarstwowy

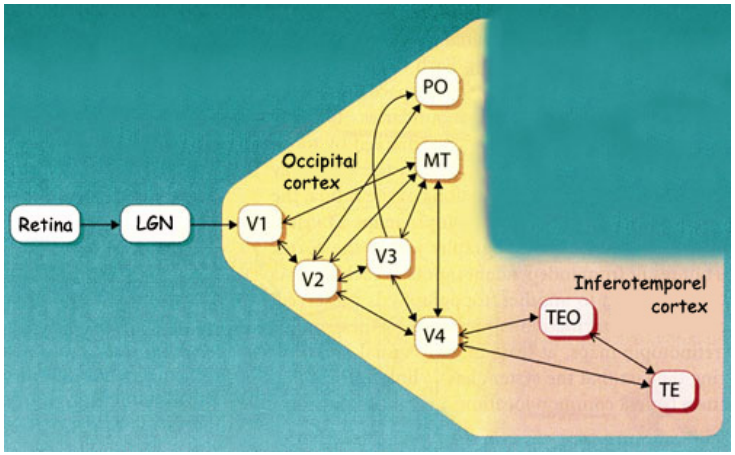


Bruno Dubuc, Douglas Hosp. Research Centre, Verdun, Quebec, Canada

- obliczenia w mózgu są wielowarstwowe (wizja: do 10 warstw)



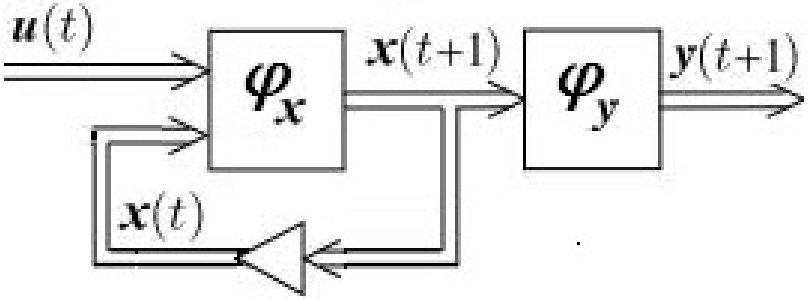
L. Simard, 3080558B Fundamentals of Computer Vision 308-558B, `www2.cs.sfu.ca/CourseCentral/821/li/material/refs/Simard-biology.pdf`

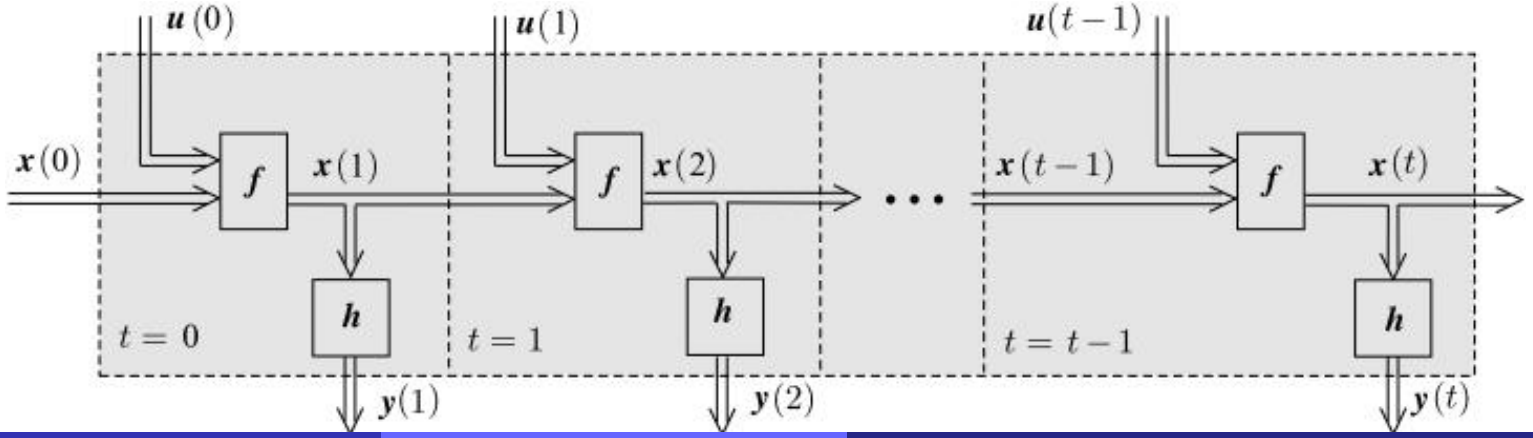


Aspekty obliczeń głębokich

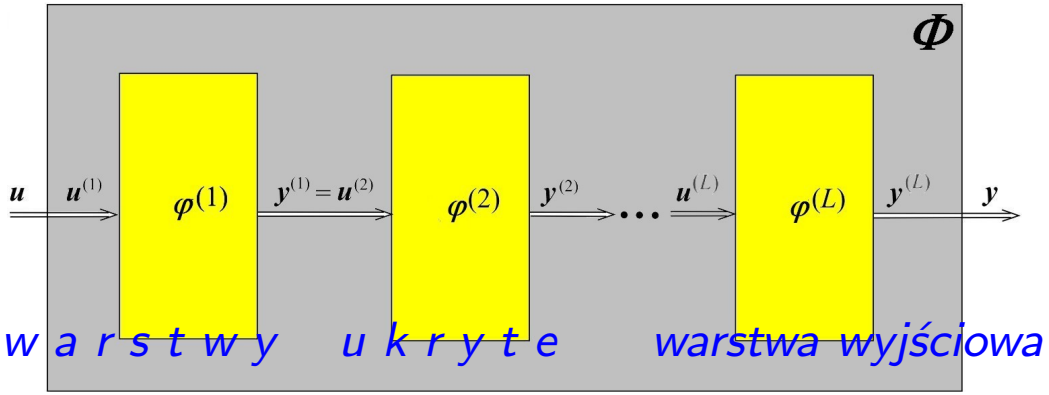
Sieci dynamiczne

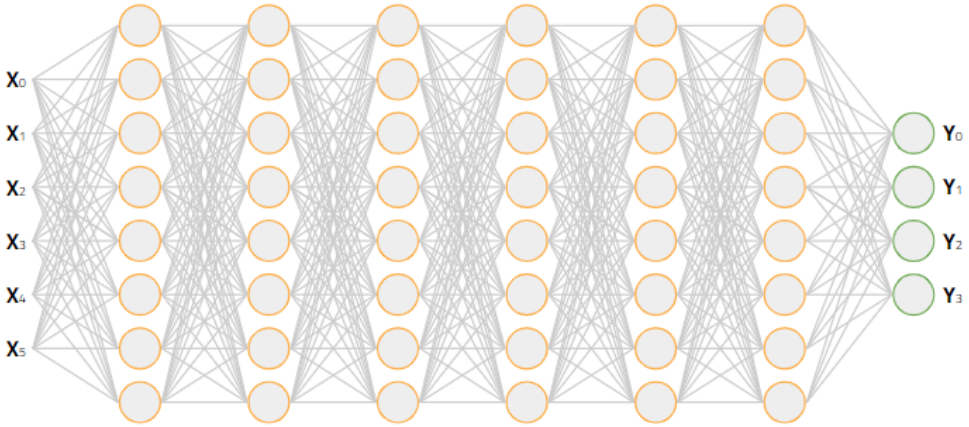
- rozwinięcie sieci dynamicznej w czasie (modelowanie, prognozowanie) prowadzi do sieci głębokiej
- kilka chwil działania sieci dynamicznej można sprowadzić do jednej chwili działania kilku segmentów sieci statycznej





Głęboki perceptron wielowarstwowy





Głębokie perceptrony wielowarstwowe

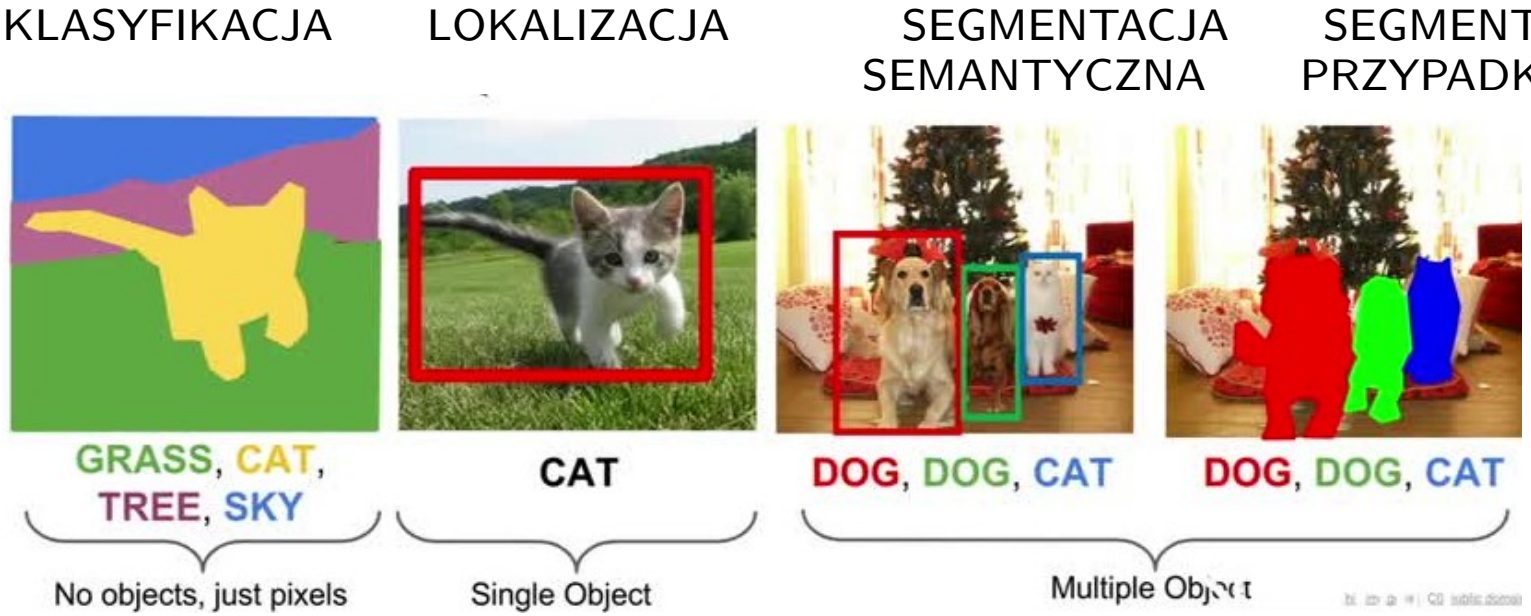
Problemy uczenia

- bardzo duża liczba wag: obraz kolorowy $1280 \times 960 \times 3 = 3\,686\,400$ wejść; tzn. jeden neuron ukryty wymaga $3\,686\,401$ wag
- "*zanikający gradient*" (ang. *vanishing gradients*): liczba wag wpływających na pochodną kosztu rośnie w niższych warstwach (tzn. bliższych warstwie wejściowej) $\Rightarrow$ gradient staje się coraz mniejszy w stosunku do wag $\Rightarrow$ w niższych warstwach wagi korygowane są bardzo powoli
- ostatnie warstwy: "zanikający gradient" może powodować wykorzystanie do uczenia tylko ostatnich warstw (tzn. bliskich warstwie wyjściowej) i sieć może uczyć się jak sieć płytka zakłócana przez pierwsze warstwy
- dostępność danych: danych etykietowanych może być zbyt mało w stosunku do liczby parametrów (problem zbyt dużego dopasowania!) - problemy generalizacji
- problemy minimalizacji kosztu: obawa przed "silnie niewypukłymi" problemami minimalizacji

13.1 Specyfika obliczeń głębokich
13.2 Obrazy jako wejścia sieci

Obrazy jako wejścia sieci

- obiekt
  - wyróżniona część obrazu
- *segmentacja* obrazu
  - podział obrazu na *regiony* o wspólnych atrybutach (poziom szarości, barwa, tekstura etc.)
- *klasyfikacja* obrazów
  - określenie przynależności obiektu na obrazie do jednej z zadanych klas
- *lokalizacja obiektu*
  - lokalizacja obiektu na obrazie np. poprzez *prostokąt ograniczający* (ang. *bounding box*)
- *detekcja obiektów*
  - wykrycie obiektów, ich lokalizacja i klasyfikacja
- *segmentacja semantyczna*
  - przypisanie etykiety klasy każdemu pikselowi obrazu, przy czym różne obiekty tej samej klasy są utożsamiane
- *segmentacja przypadków* (ang. *instance segmentation*)
  - przypisanie obiektu każdemu pikselowi obrazu, przy czym różne obiekty tej samej klasy **nie są utożsamiane**


KLASYFIKACJA
LOKALIZACJA
SEGMENTACJA SEMANTYCZNA
SEGMENTACJA PRZYPADKÓW
GRASS, CAT, TREE, SKY
CAT
DOG, DOG, CAT
DOG, DOG, CAT
No objects, just pixels
Single Object
Multiple Object

Obrazy jako wejścia sieci

Klasyfikacja

- typowe zadanie klasyfikacji: estymować prawdopodobieństwa przynależności do klas - warstwa *soft max*

$$y_i = \frac{\mathrm{e}^{\alpha s_i}}{\sum_i \mathrm{e}^{\alpha s_i}}$$

  odpowiadające rozwiązaniu zadania regresji logistycznej
- typowym wskaźnikiem chwilowego kosztu klasyfikacji jest *entropia skrośna* (ang. *cross entropy*)

$$-\sum_i p_i \ln(y_i)$$

  gdzie $p_i = 1$ jeśli $u \in \mathcal{C}_i$, $p_i = 0$ w przeciwnym przypadku

Obrazy jako wejścia sieci perceptronowych

Kodowanie wejść

- wejściami klasyfikatorów perceptronowych mogą być wektoryzowane obrazy, cechy obrazu dobierane przez eksperta lub cechy otrzymane automatycznie przy użyciu metod generycznych (SIFT, LBP, HOG, wiele innych)
  - wejście sieci w postaci wektoryzowanego obrazu "gubi" topologię obrazu i nie pozwala na efektywny dobór cech, co przedłuża czas uczenia (konieczność "odtwarzania" struktury obrazu)
  - cechy dobrane przez eksperta mogą być dobrane niewłaściwie, co obniża jakość klasyfikacji
  - generyczny dobór cech nie jest dostosowany do określonego problemu klasyfikacji
- **czy można/warto stosować obrazy jako wejścia (głębokiej) sieci perceptronowej?**
- struktury perceptronowe (również głębokie) mogą nie być dobrym narzędziem do klasyfikacji obrazów

Sieci głębokie
Źródła rozwoju

- **dostępność dużych zbiorów danych** (Google, Microsoft, Facebook) $\rightarrow$ możliwość estymacji wielu parametrów
- **szybsze obliczenia**, w tym użycie GPU
- **wydajne architektury**: współdzielenie wag, ReLU, dropout, normalizacja

Literatura

Materiały źródłowe

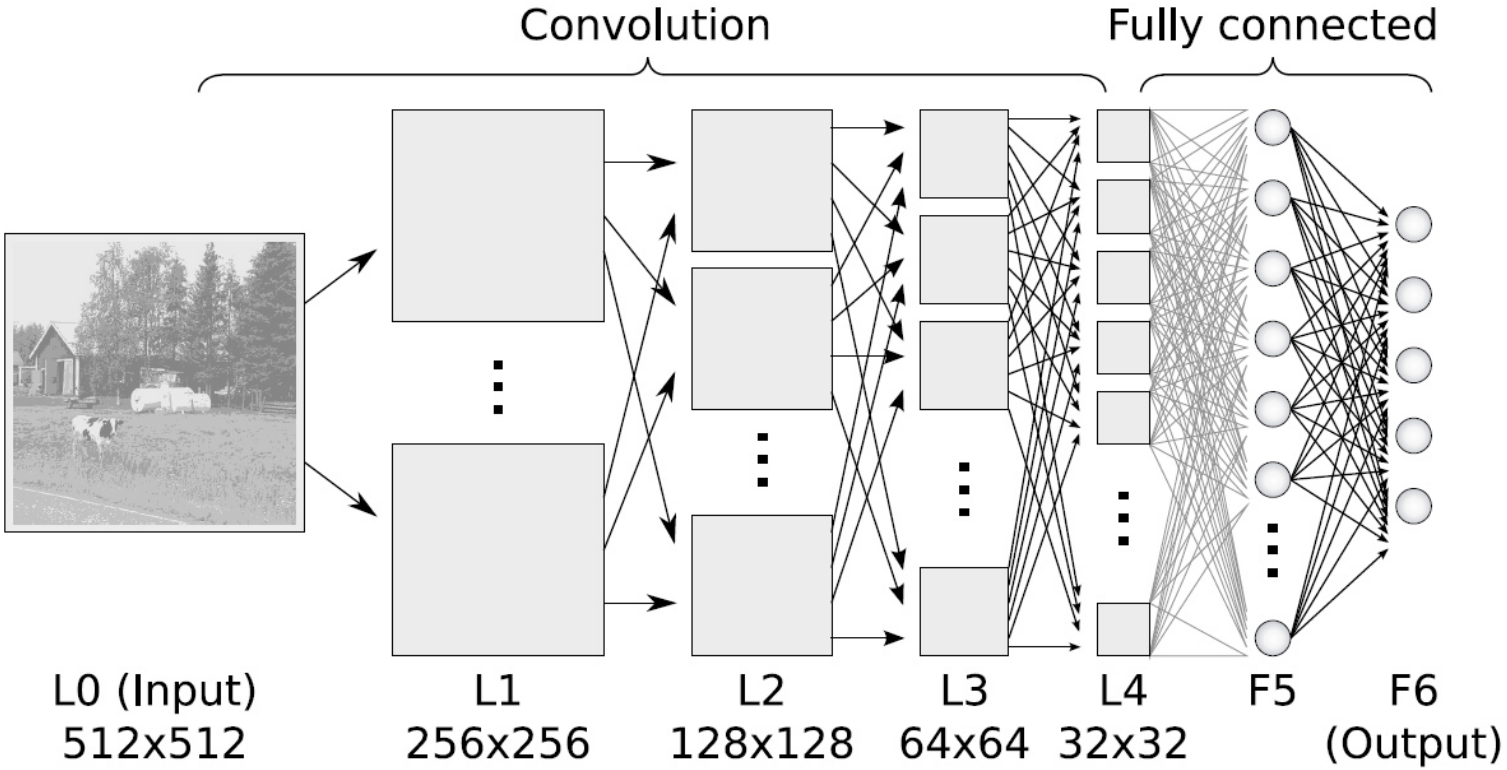
Materiały pomocnicze

Temat 14: Głębokie sieci splotowe

14.1 Struktura sieci splotowych
14.2 Przełomowe architektury splotowe
14.3 Modyfikacje procesów uczenia

14.1 Struktura sieci splotowych
14.2 Przełomowe architektury splotowe
14.3 Modyfikacje procesów uczenia

Sieci splotowe



Budowa warstwowa sieci splotowych

- sieć ma zwykle strukturę warstwową, z warstwami o strukturze tablic 3D, zakończoną warstwami wektorowymi (1D)
- warstwa wejściowa składa się najczęściej z trzech tablic 2D odpowiadających kanałom (np. R,G,B) obrazu wejściowego
- każda warstwa 3D jest złożeniem wielu *kanałów* (*map aktywacji*, *map cech*) 2D
- każdy kanał 2D jest wynikiem przekształcenia (filtracji) całej poprzedniej warstwy
- mapy aktywacji zwykle zachowują topologię obrazu wejściowego (tzn. aktywacja neuronu w danym kanale zależy od aktywacji neuronów z pewnego sąsiedztwa kanałów poprzedniej warstwy)
- rodzaje warstw:
  - *warstwy splotowe*
  - *warstwy nieliniowe*

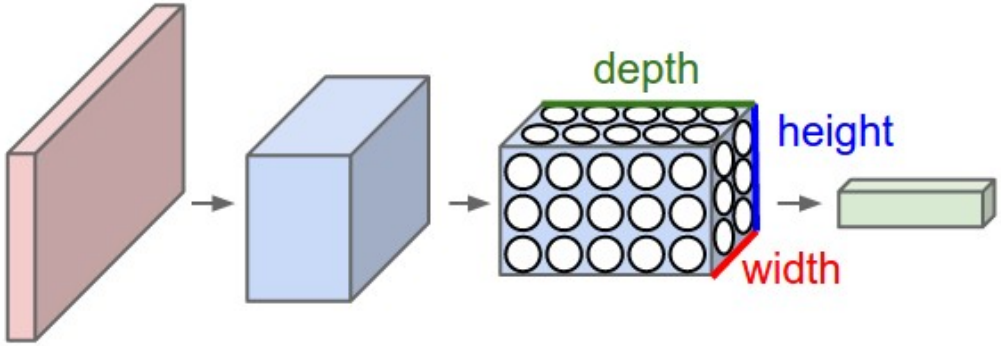
  } *nieliniowe warstwy splotowe*
  - *warstwy agregujące* (podpróbkujące) (ang. *pooling*, *subsampling*)
  - *warstwy normujące* 3D lub 1D
  - *warstwy kompletne* 1D (wektorowe)
  - warstwa wyjściowa 1D, np. typu soft-max

Budowa warstwowa sieci splotowych

Struktura warstwy splotowej

- przez x, y, z będziemy oznaczać współrzędne neuronu w pewnym kanale z, przez $X{\times}Y$ zbiór tych neuronów, a przez $|X|$, $|Y|$ jego rozmiary (szerokość, wysokość)
- warstwa 3D: tablica $X{\times}Y{\times}Z$, złożenie $|Z|$ *map aktywacji* (kanałów) $X{\times}Y$

$$\mathbf{U} = \{u_{x,y,z} : (x, y, z) \in \overbrace{\underbrace{X{\times}Y}_{\text{kanał}}{\times}Z}^{\text{warstwa}}\}$$

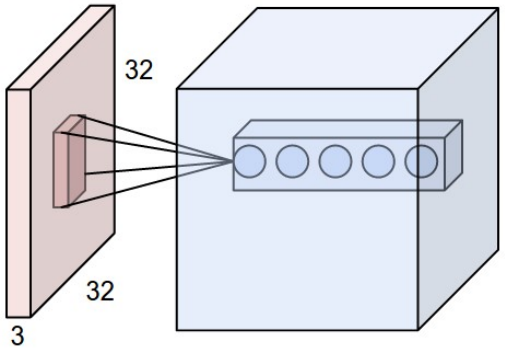


rozmiar warstwy: szerokość$\times$wysokość$\times$głębokość

Warstwy połączone lokalnie

- warstwa $X \times Y \times Z$ jest *lokalnie połączona* jeśli pola recepcyjne jej neuronów u **nie** rozciągają się na całą poprzednią warstwę, ale obejmują wszystkie jej kanały ($'$)

$$u_{x,y,z} \text{ zależy tylko od } \left(u'_{x',y',z'} : (x', y') \in R_{x,y},\ z' \in Z'\right)$$



- pola sensoryczne sąsiadujących neuronów są przesunięte o s (ang. *stride*) pozycji (w kierunku X i/lub Y); jeśli $s > 1$, to rozmiar map aktywacji poprzedniej warstwy jest ok. s razy większy w danym kierunku (zależy również od sposobu uwzględniania warunków brzegowych przez filtr)
- zastosowanie $s > 1$ jest prostą metodą s-krotnego *podpróbkowania* obrazu (ang. *downsampling*)

Warstwy liniowe połączone lokalnie

- warstwa lokalnie połączona jest *liniowa* (właściwie: afiniczna) jeśli każdy neuron u każdego kanału tej warstwy oblicza **kombinację liniową** wag i wartości neuronów u' poprzedniej warstwy w zakresie jego pola recepcyjnego, zwiększoną o obciążenie, tzn.

$$u_{x,y,z} = b_{x,y,z} + \sum_{z' \in Z'} \sum_{\substack{(x',y') \in \\ R_{x,y}}} \underbrace{w_{x'-x,y'-y,z';x,y,z}}_{\substack{\text{wagi} \\ \text{neuronu} \\ (x,y,z)}} \underbrace{u'_{x',y',z'}}_{\substack{\text{piksele} \\ \text{mapy we}}}$$

(współczynniki indeksowane przez względne położenie (x, y) i (x', y')); filtry odwrócone

- zwykle pole recepcyjne ma postać prostokąta o centrum w (x, y) i bokach $2\Delta x + 1$, $2\Delta y + 1$ i wówczas wygodnie zapisać

$$u_{x,y,z} = b_{x,y,z} + \sum_{z'} \sum_{\substack{x-\Delta x \leqslant x' \leqslant x+\Delta x \\ y-\Delta y \leqslant y \leqslant y+\Delta y}} w_{\underbrace{x'-x}_{i},\underbrace{y'-y}_{j},\underset{\substack{\uparrow \\ k}}{z'};x,y,z} \underbrace{u'_{x',y',z'}}_{\substack{\text{piksele} \\ \text{mapy we}}}$$

$$= b_{x,y,z} + \sum_{k} \sum_{\substack{|i| \leqslant \Delta x \\ |j| \leqslant \Delta y}} \underbrace{w_{i,j,k;x,y,z}}_{\substack{\text{wagi} \\ \text{neuronu} \\ (x,y,z)}} \underbrace{u'_{x+i,y+j,k}}_{\substack{\text{piksele} \\ \text{mapy we}}}$$

- wagi tworzą tensor 6 rzędu

Warstwy splotowe

Współużytkowanie wag

- *współużytkowanie wag* (ang. *weights sharing*) polega na tym, że **wagi i obciążenie neuronu są niezależne od neuronu**
- warstwę liniową ze współużytkowaniem wag nazywamy *warstwą splotową*, gdyż dla każdego kanału jej wartości są wynikiem splotu poprzedniej warstwy z filtrem liniowym, tzn. (dla prostokątnego pola recepcyjnego)

$$u_{x,y,z} = b_{\cancel{x,y},z} + \sum_{k} \sum_{\substack{|i|\leqslant\Delta x \\ |j|\leqslant\Delta y}} \underbrace{w_{i,j,k;\cancel{x,y},z}}_{\substack{\text{wagi neuronów} \\ \text{kanału } z}} \underbrace{u'_{x+i,y+j,k}}_{\substack{\text{piksele mapy we} \\ \text{kanału } z}}$$

- obciążenia i wagi neuronów *zależą* od kanału!
- współczynniki filtru tworzą tensor 4 rzędu (znacznie mniej współczynników!) $\mathbf{W}_{i,j,k,z}$
- współużytkowanie wag może być tylko lokalne, jeżeli obrazy mają specyficzną strukturę przestrzenną (np. oczy/nos/usta w obrazach twarzy)

Nieliniowe warstwy splotowe

- wyniki splatania są zwykle przekształcane przez funkcję aktywacji, tworząc *nieliniową warstwę splotową*
- często zakłada się milcząco, że każda warstwa splotowa jest nieliniowa
- tworzenie mapy aktywacji dla nieliniowej warstwy splotowej dla zadanego tensora wag $\mathbf{W}_{i,j,k,z}$
  - dla każdego kanału z
    - obliczenie splotów dla każdej mapy aktywacji k poprzedniej warstwy
    - zsumowanie wyników dla wszystkich map k ("spłaszczenie warstwy") (ang. *flattening*)
    - dodanie obciążeń b_z
    - przekształcenie przez funkcję aktywacji g

$$u_{x,y,z} = g\Big(b_z + \sum_k \overbrace{\sum_{\substack{|i|\leqslant\Delta x\\|j|\leqslant\Delta y}} \underbrace{w_{i,j,k,z}}_{\substack{\text{wagi}\\\text{neuronów}\\\text{mapy } z}} \underbrace{u'_{x+i,y+j,k}}_{\text{mapa aktywacji } k}}^{\text{splot 2D dla kanału } k\text{, ustalony kanał } z}\Big)$$

  - powtórzenie dla każdego kanału ($|Z|$ razy)
- nieliniowość g: zwykle ReLU lub podobne

Nieliniowe warstwy splotowe

Filtry o pełnym rozmiarze

- dla filtru rozmiarze całej mapy aktywacji poprzedniej warstwy i przy uzupełnianiu wartości brzegowych zerami, otrzymujemy dla ustalonego kanału z

$$u_{x,y,z} = g\Big(b_z + \overbrace{\sum_{i,j,k} \underbrace{w_{i,j,k,z}}_{\substack{\text{wagi}\\ \text{filtru } z}} \underbrace{u'_{x+i,y+j,k}}_{\text{kanał } k}}^{\substack{\text{splot 3D z uzupełnieniem zerami}\\ \text{nie zależy od } x,\, y}}\Big)$$

$$= g\Big(b_z + \overbrace{\sum_{ijk} \underbrace{w_{ijk,z}}_{\substack{\text{wagi}\\ \text{filtru } z}} \underbrace{u'_{ijk}}_{\text{warstwa we}}}^{\text{kombinacja liniowa wejść}}\Big)$$

$$u_z = \text{vec}\Big|_{\text{ustalone } z} (w_{ijk,z})^{\mathsf{T}} \text{vec}(u'_{ijk})$$

- warstwa takich filtrów jest równoważna warstwie perceptronowej (o kompletnych połączeniach)
- łatwa realizacja połączeń kompletnych dla sieci splotowych

Nieliniowe warstwy splotowe

Filtry 1×1

- dla filtru $1{\times}1$ wysokość i szerokość kanału nie ulegają zmianie, a każda kolumna (x, y, Z) jest przekształcana niezależnie, tzn.

$$u_{x,y,z} = g\Big(b_z + \sum_k \sum_{i=j=0} w_{ijk,z}\, u'_{x+i,y+j,k}\Big)$$
$$= g\Big(b_z + \sum_k w_{k,z} u'_{x,y,k}\Big)$$

- dla każdego elementu x, y, z odpowiada to działaniu neuronu perceptronowego o $|Z'|$ wejściach $u_{x,y,k}$, $k = 1, \ldots |Z'|$
- każdy kanał wyjściowy odpowiada warstwie perceptronowej o $|X \times Y|$ elementach zlokalizowanych przestrzennie
- filtry 1×1 znacznie zmniejszają liczbę współczynników w stosunku do typowych filtrów; używane są m.in. w sieciach NiN, Inception

Nieliniowe warstwy splotowe

Filtry 1×1 - przykład

Ⓔ przykład: 1. warstwa $200{\times}200{\times}50$

- 30 filtrów $5{\times}5 \Rightarrow$
  - 2. warstwa: $196{\times}196{\times}30$,
    $(5 \cdot 5 \cdot 50 + 1) \cdot 30 = 37530$ parametrów
- filtr 1×1 i 30 filtrów $5 \times 5 \Rightarrow$
  - 2. warstwa: $200{\times}200{\times}1$
    $(1 \cdot 1 \cdot 50 + 1) \cdot 1 = 51$ parametrów
  - 3. warstwa: $196{\times}196{\times}30$
    $(5 \cdot 5 + 1) \cdot 30 = 780$ parametrów
    razem: 831 parametrów

⊛ Warstwy splotowe - efekty brzegowe

Obcinanie

- *obcinanie* (ang. *cropping*) dokonuje się filtracji tylko dla tych argumentów, dla których cała maska filtru znajduje się w obrazie
  - dla filtru $f_x \times f_y$ szerokość i wysokość obrazu zmniejszają się, odpowiednio, o $f_x - 1$ i $f_y - 1$ pikseli

Ⓔ przykład: obcinanie obrazu 4×4 dla filtru 3×3;

$u_{1,1}$	$u_{1,2}$	$u_{1,3}$	$u_{1,4}$
$u_{2,1}$	$u_{2,2}$	$u_{2,3}$	$u_{2,4}$
$u_{3,1}$	$u_{3,2}$	$u_{3,3}$	$u_{3,4}$
$u_{4,1}$	$u_{4,2}$	$u_{4,3}$	$u_{4,4}$

obraz: kolor szary, obraz obcięty (po filtracji): kolor żółty

⊛ Warstwy splotowe - efekty brzegowe

Uzupełnianie zerami

- *uzupełnianie zerami* (ang. *zero padding*) obraz jest rozszerzany poprzez uzupełnianie rzędów brakujących elementów zerami - w pionie, poziomie i narożu
  - dla zachowania wymiaru obrazu, liczba zerowych kolumn/wierszy powinna być o jeden mniejsza niż rozmiar filtru w danym kierunku
  - dla filtrów o rozmiarze $2p+1 \times 2p+1$, rozmiar obrazu jest zachowany jeśli stosuje się uzupełnianie p kolumnami z lewej strony obrazu i p kolumnami z prawej strony obrazu; podobnie dla wierszy
  - stosowane często w sieciach splotowych dla zachowania rozmiaru kanałów

Ⓔ przykład: uzupełniania zerami obrazu 4×4, filtr 3×3

$$\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & 0 \\ 0 & u_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} & 0 \\ 0 & u_{3,1} & u_{3,2} & u_{3,3} & u_{3,4} & 0 \\ 0 & u_{4,1} & u_{4,2} & u_{4,3} & u_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

obraz: uzupełnianie kolumnami i wierszami zer tak, aby rozmiar obrazu (3×4) pozostał niezmieniony po splocie z filtrem 3×3 (kolor żółty)

⊛ Warstwy splotowe - efekty brzegowe

Rozszerzanie

- *rozszerzanie* (ang. *extending*) obraz jest rozszerzany poprzez uzupełnianie brakujących elementów wartościami najbliższymi sąsiadami - w pionie, poziomie lub narożu
  - realizuje najprostszą predykcję wartości obrazu poza jego brzegiem
  - liczba dodatkowych kolumn taka, by rozmiar obrazu po filtracji pozostał niezmieniony (jak w przypadku uzupełniania zerami)

Ⓔ przykład: rozszerzanie obrazu 4×4, filtr 3×3

$$\begin{array}{cccccc}
u_{1,1} & u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & u_{1,4} \\
u_{1,1} & u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & u_{1,4} \\
u_{2,1} & u_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} & u_{2,4} \\
u_{3,1} & u_{3,1} & u_{3,2} & u_{3,3} & u_{3,4} & u_{3,4} \\
u_{4,1} & u_{4,1} & u_{4,2} & u_{4,3} & u_{4,4} & u_{4,4} \\
u_{4,1} & u_{4,1} & u_{4,2} & u_{4,3} & u_{4,4} & u_{4,4}
\end{array}$$

obraz: uzupełnianie kolumnami i wierszami zer tak, aby rozmiar obrazu (3×4) pozostał niezmieniony po filtracji filtrem3×3 (kolor żółty)

⊛ Warstwy splotowe - efekty brzegowe

Przedłużanie okresowe

- *przedłużanie okresowe*, *zawijanie* (ang. *wrapping*) obraz traktowany jest jak podwójny torus, tzn. brakujące wartości są uzupełniane wartościami z przeciwnej krawędzi
  - nie stosowane w sieciach splotowych
  - analiza Fouriera zakłada, że obraz jest przedłużony okresowo do nieskończoności

(E) przykład: przedłużanie okresowe obrazu 4×4, filtr 3×3

$u_{4,4}$	$u_{4,1}$	$u_{4,2}$	$u_{4,3}$	$u_{4,4}$	$u_{4,1}$
$u_{1,4}$	$u_{1,1}$	$u_{1,2}$	$u_{1,3}$	$u_{1,4}$	$u_{1,1}$
$u_{2,4}$	$u_{2,1}$	$u_{2,2}$	$u_{2,3}$	$u_{2,4}$	$u_{2,1}$
$u_{3,4}$	$u_{3,1}$	$u_{3,2}$	$u_{3,3}$	$u_{3,4}$	$u_{3,1}$
$u_{4,4}$	$u_{4,1}$	$u_{4,2}$	$u_{4,3}$	$u_{4,4}$	$u_{4,1}$
$u_{1,4}$	$u_{1,1}$	$u_{1,2}$	$u_{1,3}$	$u_{1,4}$	$u_{1,1}$

obraz: uzupełnianie okresowe kolumn i wierszy tak, by rozmiar obrazu (3×4) pozostał niezmieniony po splocie z filtrem 3×3 (kolor żółty)

⊕ Warstwy splotowe

Efekty brzegowe - Przykłady

- (E) mapa aktywacji $\begin{bmatrix} 2 & 4 & 1 & 3 \\ 1 & 2 & 1 & 4 \\ 0 & 3 & 3 & 0 \\ 2 & 0 & 1 & 0 \end{bmatrix}$ filtr $\begin{bmatrix} 2 & 1 & 1 \\ 1 & 0 & 2 \\ 2 & 0 & 1 \end{bmatrix}$
  - (E) obcinanie; mapa wynikowa $\begin{bmatrix} 15 & 28 \\ 16 & 12 \end{bmatrix}$
  - (E) uzupełnienie kolumną zer i wierszem zer (z każdej strony obrazu)
    $\begin{bmatrix} 10 & 7 & 18 & 3 \\ 13 & 15 & 31 & 12 \\ 9 & 16 & 12 & 11 \\ 3 & 10 & 9 & 7 \end{bmatrix}$
  - (E) uzupełnienie 2 kolumnami i wierszami zer (z każdej strony obrazu)
    $\begin{bmatrix} 2 & 4 & 5 & 11 & 2 & 6 \\ 5 & 10 & 7 & 18 & 3 & 11 \\ 4 & 13 & 15 & 31 & 12 & 10 \\ 3 & 9 & 16 & 12 & 11 & 8 \\ 4 & 3 & 10 & 9 & 7 & 0 \\ 2 & 2 & 5 & 1 & 2 & 0 \end{bmatrix}$

Warstwy agregujące (pooling)

- *agregacja*, *łączenie* (ang. *pooling*) jest rodzajem podpróbkowania (ang. *subsampling*) działającego **niezależnie na każdy kanał**
  - $h(R) = \max_R u_{x,y}$ (ang. *max-pooling*)
  - $h(R) = \mathscr{A}_R u_{x,y}$ średnia
  - $h(R) = \sqrt{\sum_R u_{x,y}^2}$ L2
- liczba kanałów nie zmienia się
- nie ma trenowanych parametrów
- pole recepcyjne przesuwane jest z pewnym krokiem s (ang. *stride*)
- efekty brzegowe: pole recepcyjne może być uzupełniane zerami (np. dla zachowania rozmiaru)
- najczęściej stosuje się max-pooling,
  - pole recepcyjne $2{\times}2$, $s = 2$
  - pole recepcyjne $3{\times}3$, $s = 3$ lub $s = 2$
- warstwy agregujące stosowane są naprzemiennie z warstwami splotowymi
- większy krok $s > 1$ dla warstwy splotowej jest identyczny z agregacją z tym samym krokiem i operacją

  $h(R) = u_{1,1}$ gdzie $u_{1,1}$ jest 'lewym górnym' elementem R

Ⓔ Warstwy agregujące

Przykłady

- mapa aktywacji: $\begin{bmatrix} 2 & 4 & 1 & 3 \\ 1 & 2 & 1 & 4 \\ 0 & 3 & 3 & 0 \\ 2 & 0 & 1 & 0 \end{bmatrix}$,
- max-pooling 2×2, $s = 1$: $\begin{bmatrix} 4 & 4 & 4 \\ 3 & 3 & 4 \\ 3 & 3 & 3 \end{bmatrix}$, max-pooling 2×2, $s = 2$: $\begin{bmatrix} 4 & 4 \\ 3 & 3 \end{bmatrix}$
- średnia 2×2, $s = 1$: $\begin{bmatrix} 9/4 & 2 & 9/4 \\ 6/4 & 9/4 & 2 \\ 5/4 & 7/4 & 1 \end{bmatrix}$, średnia 2×2, $s = 2$: $\begin{bmatrix} 9/4 & 9/4 \\ 5/4 & 1 \end{bmatrix}$

⊛ Warstwy splotowe

Filtry odwrócone 2D

- zazwyczaj dla wygody zapisu wagi definiowane są w postaci filtrów "odwróconych" (ang. *flipped*) $w_{i,j} = \vartheta_{-i,-j}$ i wówczas

$$u_{x,y,z} = g\Big(b_z + \sum_{z'} \sum_{\substack{|i|\leqslant\Delta x \\ |j|\leqslant\Delta y}} \underbrace{w_{i,j,z';z}}_{\substack{\text{wagi} \\ \text{neuronu}}} \underbrace{v_{x+i,y+j,z'}}_{\substack{\text{piksele} \\ \text{mapy we}}}\Big)$$

- działanie odwróconego filtru splotowego można interpretować jako obliczanie *lokalnych korelacji*

		$\vartheta_{-1,-1}$	$\vartheta_{0,-1}$	$\vartheta_{1,-1}$	
		$\vartheta_{-1,0}$	$\vartheta_{0,0}$	$\vartheta_{1,0}$	
		$\vartheta_{-1,1}$	$\vartheta_{0,1}$	$\vartheta_{1,1}$	

Warstwy splotowe

Filtry odwrócone 2D - Przykład

Ⓔ $$\begin{bmatrix} 1 & 3 & 1 & -2 \\ 0 & 5 & 1 & 1 \\ 1 & 2 & 1 & 3 \\ 4 & 3 & 2 & 2 \end{bmatrix} * \underbrace{\begin{bmatrix} 1 & -2 & 3 \\ 2 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}}_{F} = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix}$$; obliczymy v_{12}

filtr odwrócony $F_{-} = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 1 & 2 \\ 3 & -2 & 1 \end{bmatrix}$

$$v_{12} = 3 \cdot 2 + 1 \cdot 1 - 2 \cdot 1 + 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 2 + 2 \cdot 3 + 1 \cdot -2 + 3 \cdot 1 = 20$$

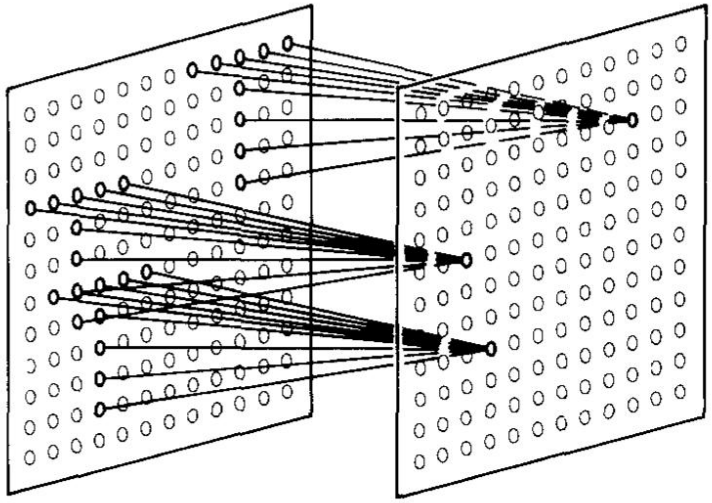
14.1 Struktura sieci splotowych
14.2 Przełomowe architektury splotowe
14.3 Modyfikacje procesów uczenia

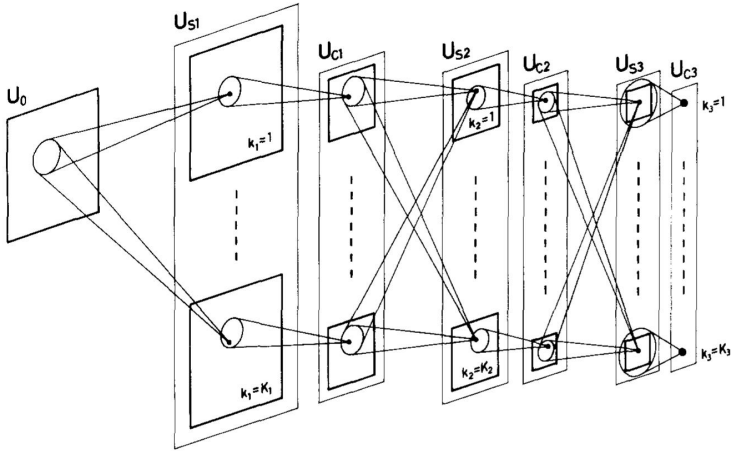
Neocognitron, Fukushima 1980

- *neocognitron* można uważać za głęboką sieć splotową, która wyprzedziła swoją epokę

  K. Fukushima (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36, pp. 193–202

- każda lokalizacja jest wynikiem filtracji pól sensorycznych neuronów poprzedniej warstwy



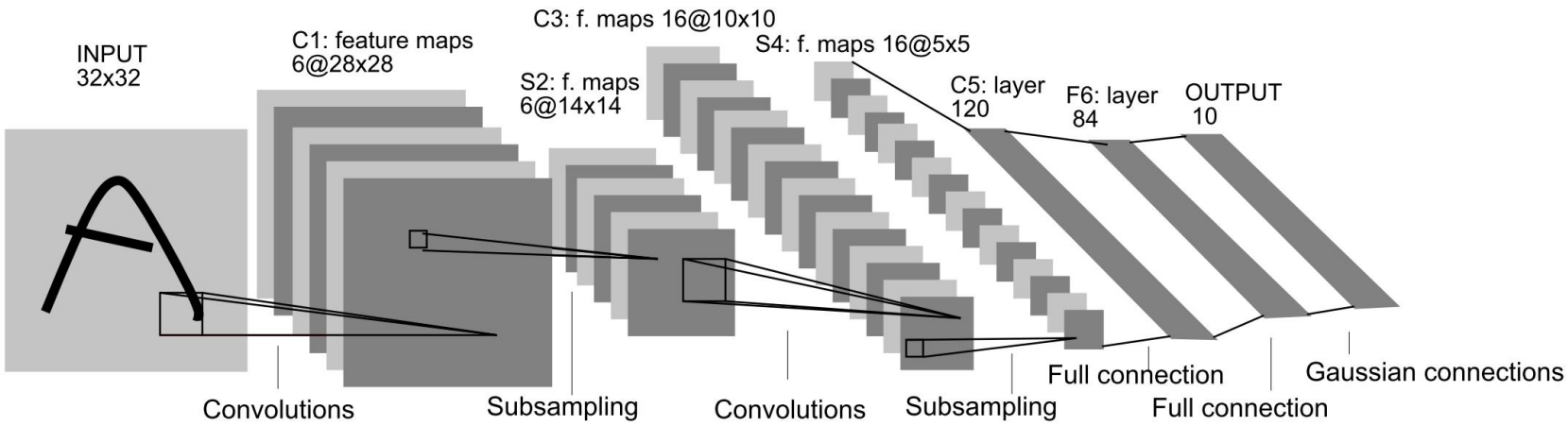


Fukushima (1980)

LeNet, Yann LeCun 1998

- LeNet - pierwsza "nowoczesna" sieć splotowa

  Y. LeCun et al. (Nov. 1998). "Gradient-Based Learning applied to Document Recognition". In: *Proceedings of the IEEE*, pp. 1–46, AT&T Labs



LeCun et al. (1998)

- zarys struktury

		obrazy 32×32
splot: 6 kanałów, filtry 5×5	$26\times6=156$ wag	warstwa $28\times28\times6$
podpróbkowanie: okno 2×2, krok 2		warstwa $14\times14\times6$
splot niekompletny: 16 kanałów, filtry 5×5,	$151\times16=1516$ wag	warstwa $10\times10\times16$
podpróbkowanie: okna 2×2, krok 2		warstwa $5\times5\times16$
splot: filtry 5×5, 120 kanałów	$401\times120=48120$ wag	warstwa $1\times1\times120$
połączenia kompletne	$121\times84=10164$ wag	warstwa 84 neurony
połączenia kompletne		wyjście: 10 neuronów

Konkurs ILSVRC

- baza *ImageNet*: 15M obrazów etykietowanych, 22K klas
- konkurs ImageNet Large-Scale Visual Recognition Challenge *ILS VRC*
- ILS VRC wykorzystuje podzbiór 1.2M obrazów do treningu, 50K do walidacji, 150K obrazów testujących
- niektóre wyniki konkursów ILS VRC

rok	błąd %	metoda	sieć	zespół
2011	25.77	FE	XRCE	Perronnin, Sanchez
2012	15.0	CNN	AlexNet	Krizhevski, Sutskever, Hinton
2013	11.7	CNN	Clarifai	Zeiler
2014	7.41	CNN	VGG Net	Simonyan, Zisserman (Oxford)
2014	6.66	CNN	GoogLeNet (Inception)	Szegedy et al.
2015	3.57	CNN	EnsembleA MSRA (Resnet)	He, Zhang, Ren, Sun
2016	2.99	CNN	Ensemble2	Trimps-Soushen
2017	2.25	CNN	WMW Ensemble C (Squeeze Excitation Net)	Hu, Shen, Sun (U. Oxford)

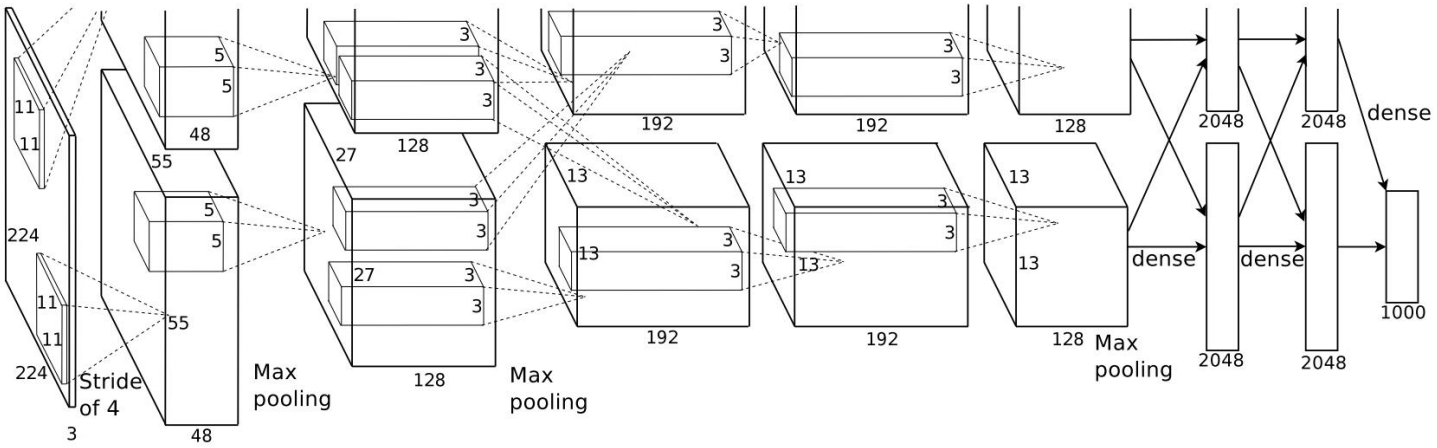
FE - wykorzystuje tworzenie cech (ang. *feature engineering*)
CNN - głębokie sieci splotowe

AlexNet, Alex Krizhevsky 2012

- konkurs ILS VRC 2012: klasyfikacja obrazów na 1000 klas
- najistotniejsza wczesna publikacja nt. sieci splotowych: A. Krizhevsky, I. Sutskever, and G. E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf, Univ. of Toronto
- *AlexNet* - zwycięzca ILSVRC 2012, błąd 5 najlepszych 15.4% (następny 26.2%)
- *Caffe Network* (convolutional architecture for fast feature embedding) późniejsza wersja z 1 GPU

AlexNet, Alex Krizhevsky 2012

Struktura



Krizhevsky, Sutskever, and Hinton (2012)

- we $224{\times}224{\times}3$
- (2)**F**$11{\times}11$s4, $\Rightarrow$ $55{\times}55{\times}48$ $(\times 2)$
- (2)MP$2{\times}2$s2, (2)LNR, (2)**F**$5{\times}5$ $\Rightarrow$ $27{\times}27{\times}128$ $(\times 2)$
- (2)MP$3{\times}3$s2, (2)**F**$3{\times}3$ $\Rightarrow$ $13{\times}13{\times}192$ $(\times 2)$
- (2)MP$3{\times}3$s2, (2)**F**$3{\times}3$ $\Rightarrow$ $13{\times}13{\times}192$ $(\times 2)$
- (2)**F**$3{\times}3$ $\Rightarrow$ $13{\times}13{\times}128$ $(\times 2)$
- (2)MPs2, połączenia gęste skrośne
- połączenia gęste skrośne, dropout 0.5 $\Rightarrow$ warstwa 2048 $(\times 2)$
- połączenia gęste $\Rightarrow$ warstwa 1000

AlexNet, Alex Krizhevsky 2012

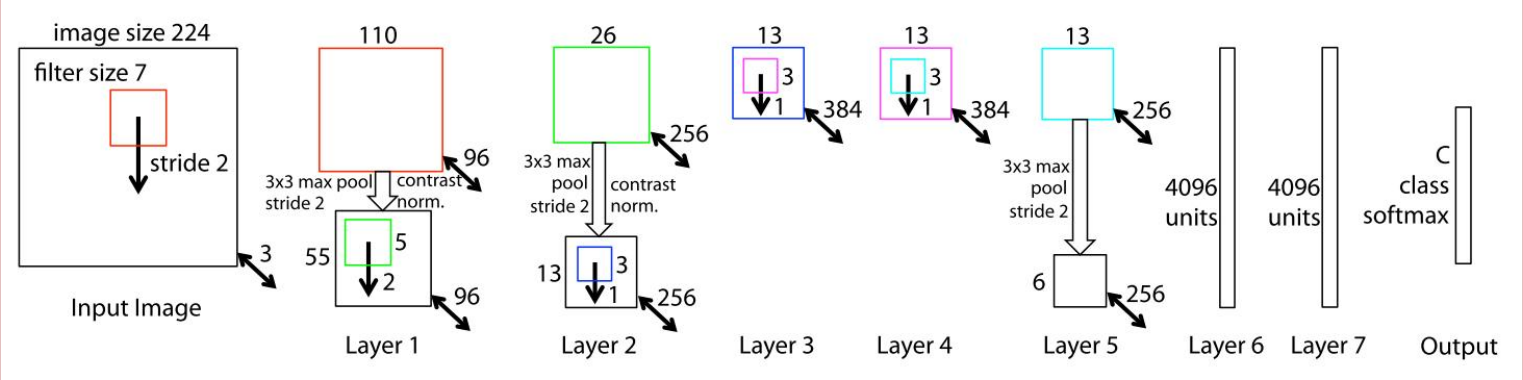
Uczenie

- dane: 1.2 M, 1000 klas
- struktura sieci
  - dwie ścieżki i komunikacja skrośna
  - 5 warstw splotowych,
  - 3 warstwy połączone kompletnie
  - 60M parametrów
  - GPU pamięć 3 GB, 2 $\times$ NVIDIA GTX 580
- technika uczenia
  - 2048-krotne zwiększenie zbioru: translacje, odbicia, wycinki
  - uczenie porcjami 128 obrazów $224\times224\times3$, 80 epok, 5-6 dni
  - inercja (momentum) 0.9
  - regularyzacja przez tłumienie (decay) 0.0005,
  - szybkość uczenia 0.01 redukowana 10-krotnie gdy błąd walidacji przestaje maleć (3 razy)
  - ReLU; sześć razy szybciej niż tgh prowadziła do błędu 25%
  - lokalna normalizacja odpowiedzi LRN $27\times27\times96$
  - dropout 0.5 w warstwach gęstych
  - uśrednianie predykcji dla 5 realizacji (redukcja 16.4%)

⊛ ZFNet, Zeiler, Clarifai, 2013

M. D. Zeiler and R. Fergus (2013). "Visualizing and Understanding Convolutional Networks". In: *CoRR* abs/1311.2901. URL: http://arxiv.org/abs/1311.2901

- *ZFNet*: zwycięzca ILS VRC 2013, 11.2%

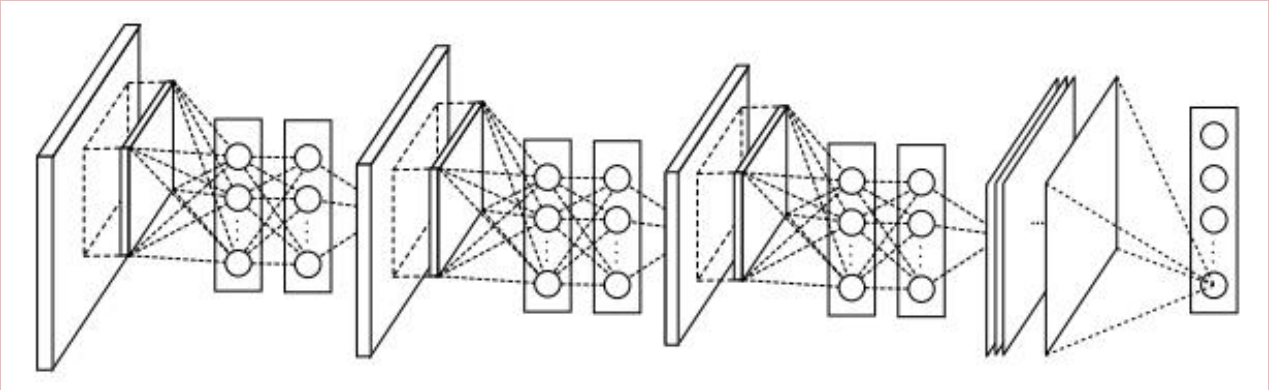


Zeiler and Fergus (2013)

- zmodyfikowana struktura AlexNet, trening 1.3 M obrazów
- filtry 7×7 w pierwszej warstwie splotowej (zamiast 11x11 w AlexNet), zmniejszony krok filtru zachowuje lepiej informacje we; w dalszych warstwach 5×5, 3×3 (3 warstwy)
- uczenie porcjami 128 obrazów, ReLU, entropia skrośna
- trening na GTX 580, 12 dni

⊛ Sieć w sieci NiN, Lin, Chen, Yan 2014

M. Lin, Q. Chen, and S. Yan (2014). "Network In Network". In: *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*



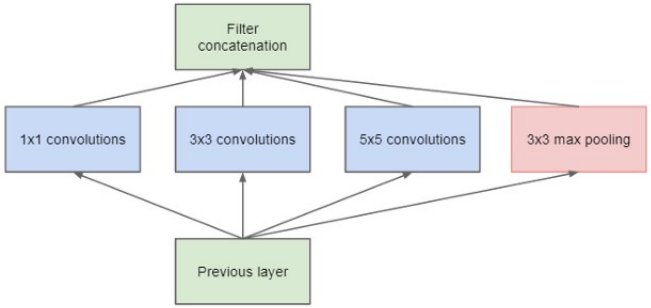
M. Lin, Q. Chen, and S. Yan (2014). "Network In Network". In: *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*

- architektura warstwy NiN
  - warstwy MLPconv (np. 5×5)
    - warstwa splotowa
      (np. $11 \times 11 \times 96$, $5 \times 5 \times 256$, $5 \times 5 \times 384$, $3 \times 3 \times 10$ dla 10 klas)
    - 2 warstwy splotowe 1×1; funkcje aktywacji ReLU
    - agregacja *maxpool* $s = 2$
    - *dropout* na wyjściach warstw MLPconv (z wyjątkiem ostatniej)
  - *globalna agregacja uśredniająca* (global average pooling) GAP - uśrednianie w ramach kanałów (jeden na klasę)
  - softmax
- uczenie
  - regularyzacja przez tłumienie wag (*weight decay*)

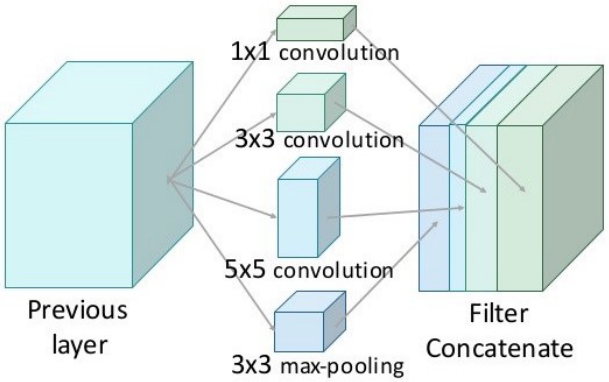
Moduły *inception, Google, 2014*

C. Szegedy et al. (2014). *Going deeper with convolutions*. Tech. rep. arXiv:1409.4842v1

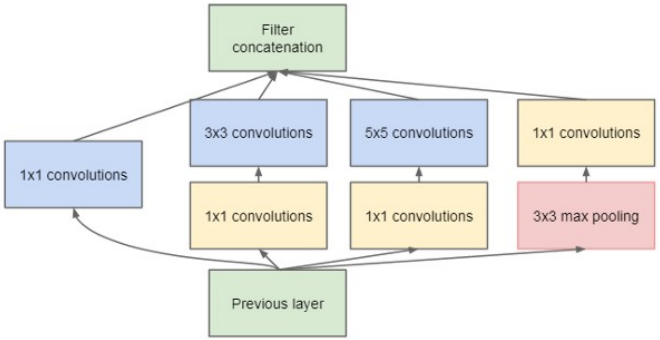
- *moduł Inception* ("pierwotny") jest równoległym złożeniem kilku filtrów o różnych rozmiarach (1×1, 3×3, 5×5, max pooling 3×3)
- rozbudowa sieci głębokiej wszerz pozwala na oszczędzenie mocy obliczeniowej i budowę głębszej sieci



"naiwny" moduł Inception, Szegedy et al. (2014)



złożenie kanałów w module inception, Xu (2017)



moduł Inception w sieci GoogLeNet, v1Szegedy et al. (2014)

we need to go deeper

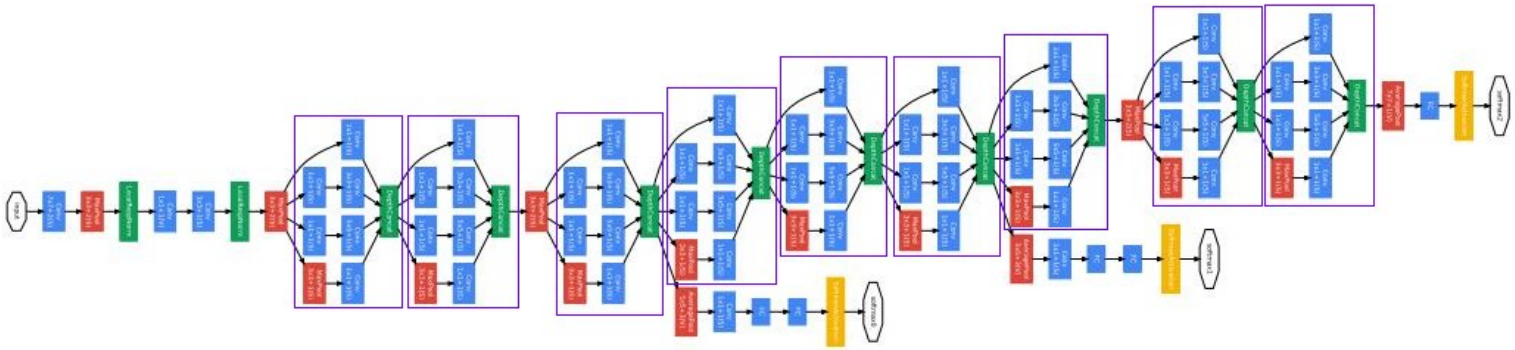
film s.f. *Inception* (2010)
zagnieżdżanie myśli głębiej w mózgu

Sieć *Inception v1, Google, 2014*

GoogLeNet

C. Szegedy et al. (2014). *Going deeper with convolutions*. Tech. rep. arXiv:1409.4842v1

- *sieć GoogLeNet* czyli *Inception v1*: zwycięzca ILSVRC 2014
- uwagi o strukturze
  - 22 warstwy splotowe
  - w tym 9 modułów *inception* złożonych z (nieliniowych) równoległych warstw splotowych
    $1\times1 \quad \| \quad 1\times1 \wedge 3\times3 \quad \| \quad 1\times1 \wedge 5\times5 \quad \| \quad 3\times5 \wedge$ max-pooling 3×3
    pozwala na równoczesną analizę segmentów o różnych rozmiarach
  - złożenie (ang. *concatenation*) kanałów w ramach warstwy inception
  - Inception można traktować jako sprawną realizację idei NiN
  - *klasyfikatory pomocnicze* (ang. *auxiliary*): zastosowanie warstw softmax do dwóch pośrednich modułów i obliczanie strat pomocniczych, dodawanych (tylko podczas uczenia) z wagami do funkcji strat.



na podstawie Szegedy et al. (2014)

Sieć *Inception v2 i v3, Google, 2015*

Christian Szegedy, Google Inc. i inni

- Inception v2 i v3: główne modyfikacje w stosunku do Inception v1:
  - *faktoryzacja splotów*
    złożenie dwóch filtrów 3×3 zastępuje filtr 5×5 zwiększając szybkość ok. 2.8=krotnie
  - *rozszerzenie splotów* (ang. *expansion*): równoległe filtry 3×1 i 1×3 zastępują filtry 3×3

Sieć VGG, Simonyan, 2014

Idea

Karen Simonyan, Andrew Zisserman, Visual Geometry Group, University of Oxford, GB, 2014

- podstawowe idee sieci VGG:
  - niewielkie rozmiary filtrów pozwalają na zmniejszenie liczby parametrów w stosunku do filtrów o większych rozmiarach:
    - liczba parametrów jednego kanału "dużych" filtrów $(3+2k)\times(3+2k)$ w stosunku do liczby parametrów k jednego kanału filtrów 3×3:
      k=1: 25 vs. 9,
      k=2: 49 vs. 18,
      k=3: 81 vs. 27
  - zwiększona liczba filtrów zwiększa liczbę nieliniowości
  - implementacja: po każdej warstwie agregującej maleje rozmiar przestrzenny (szerokość $\times$ wysokość) następnej warstwy, ale podwaja się jej głębokość (liczba filtrów)
- ImageNet Large-Scale Visual Recognition Challenge ILSVRC2015 (pierwsze miejsca - lokalizacja (regresja) i klasyfikacja)

Sieć VGG, Simonyan 2014

Szczegóły architektury

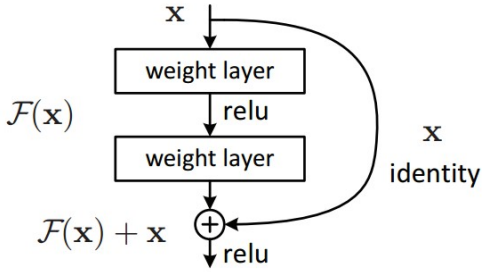
- szczegóły architektury
  - 8 do 16 warstw splotowych, 2-3 warstwy o kompletnych połączeniach, filtry 3×3, krok 1
  - agregacja max-pooling 2×2, krok 2
  - dropout 0.5 dla dwu pierwszych warstw kompletnie połączonych
  - zastosowano lokalną normalizację odpowiedzi LRN
- uczenie
  - mini-porcja 256 obrazów, 370 K iteracji (74 epoki)
  - inercja (momentum) 0.9
  - regularyzacja przez tłumienie L2 wag ($\lambda = 0.5\,10^4$)
  - prędkość uczenia 0.01 zmniejszana 10-krotnie gdy koszt walidacji przestaje się zmniejszać (powtórzone 3-krotnie)
  - 4 Nvidia Titan Black GPUs - trening 2-3 tygodnie
- najbardziej popularne realizacje: VGG-16 i VGG-19 (indeks oznacza liczę uczących się warstw)

ResNet, Microsoft, 2015

Idea rozwiązania

K. He, X. Zhang, et al. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. Tech. rep. Microsoft Research. URL: //arxiv.org/pdf/1502.01852.pdf

- problem zanikającego gradientu w sieciach o dużej liczbie warstw
  - aby ułatwić uczenie warstw, zastępuje się uczenie funkcji $H(x;\boldsymbol{\vartheta})$ przez uczenie różnicy wyjście-wejście (ang. *residual mapping*) $F(x;\boldsymbol{\vartheta}) = H(x;\boldsymbol{\vartheta}) - x$ i dodanie do niej wejścia x danej warstwy
  - propagacja wsteczna gradientu w ramach każdej pary warstw odbywa się poprzez ścieżkę sparametryzowaną $F(x)$ i poprzez ścieżkę niesparametryzowaną x, omijając zanikający gradient części sparametryzowanej; umożliwia to sprawne uczenie sieci o tysiącach warstw
  - w miarę uczenia (rozpoznawania funkcji F dla każdej warstwy) adaptują się parametry "ominiętych" ścieżek

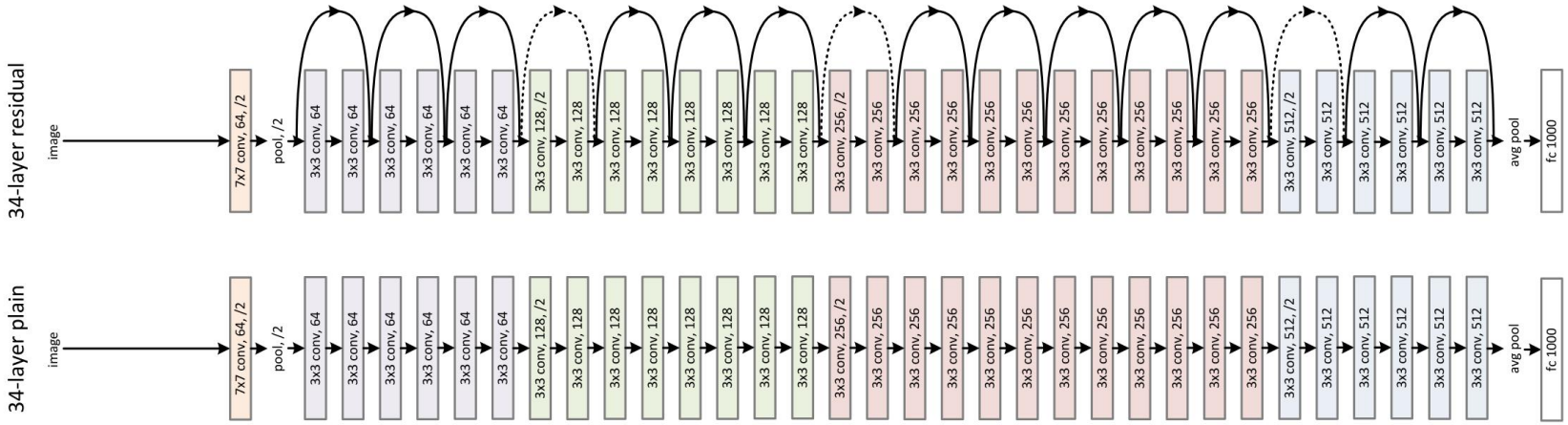


He, Zhang, et al. (2015)

ResNet, Microsoft 2015

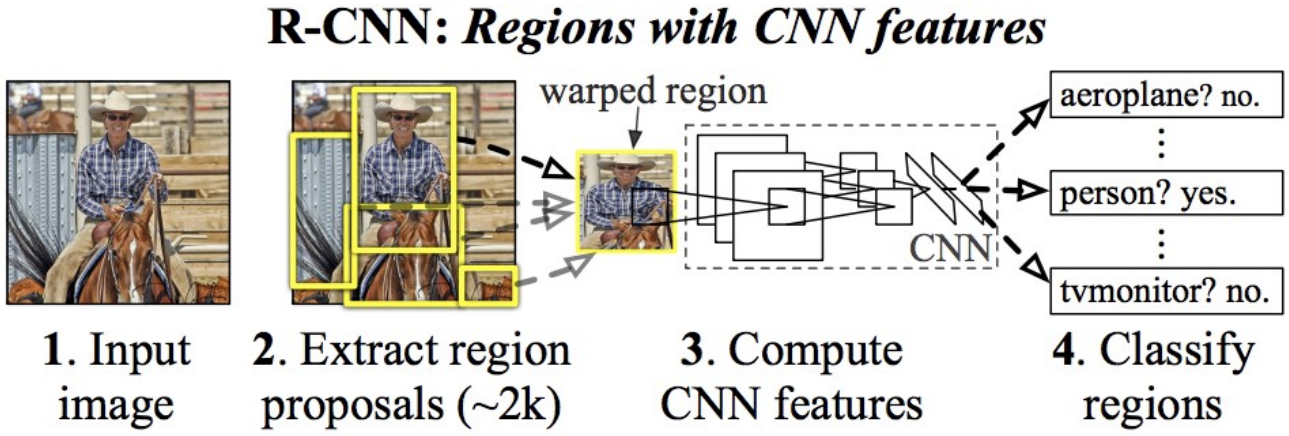
Realizacja

- zwycięzca ILSVRC 2015, błąd 3.6% (człowiek: 5-10%)
- trening (obraz 224×224 na 8 GPU, 2-3 tygodni)
- do 152 warstw splotowych ReLU (później nawet 4096)



R-CNN: Regiony z cechami CNN

- detekcja obiektów; liczba obiektów nieznana
- sieć splotowa wykorzystująca regiony R. Girshick et al. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation*. Tech. rep. arXiv:1311.2524. arXiv.org



Girshick et al. (2014)

- zadania: lokalizacja, detekcja, klasyfikacja obiektów
- *state-of-the-art* na konkursach VOC-2012 i ILSVRC-2013 (200 klas)
- architektura
  - propozycje regionów (niezależnych od klas)
    - Girshick et al. (2014): *przeszukanie selektywne* (ang. *selective search*); ok. 2000 propozycji regionów
  - *ekstrakcja cech*: wyznaczenie wektora cech dla każdego regionu
    - R. Girshick et al. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation*. Tech. rep. arXiv:1311.2524. arXiv.org: (jedna) głęboka sieć splotowa, 4096 cech
  - klasyfikacja regionów

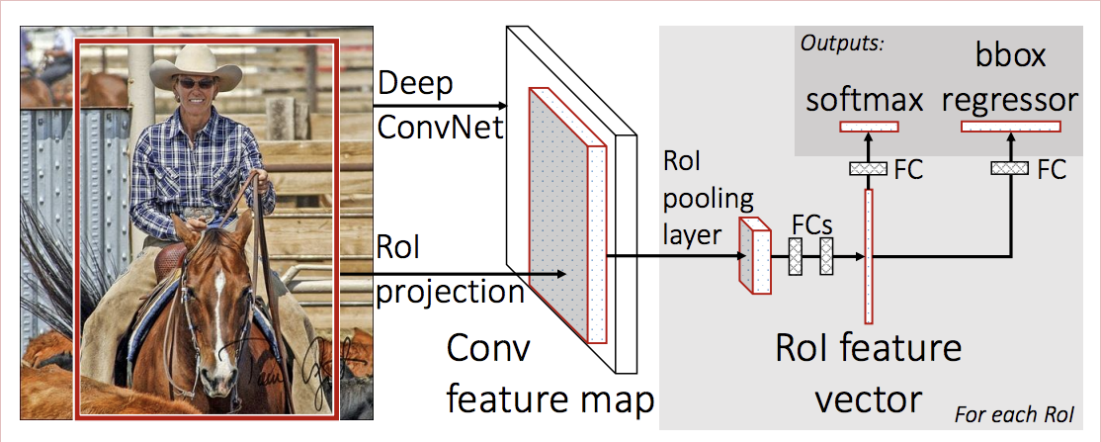
R-CNN

Trening

- trening detektora
  - stosowanie okien odkształconych (ang. *warped*) do stałego rozmiaru
  - SGD, szybkość uczenia 0.001 (1/10 szybkości przy treningu wstępnym)
  - okna pozytywne są bardzo rzadkie
  - próbkowanie warstwowe (ang. *biased (stratified) sampling*): warstwa tła i warstwa pozytywna (dowolnej klasy)
  - mini-porcje 128 okien wylosowanych warstwowo (jednostajnie w warstwie) np. z 32 pozytywnych i 96 okien tła

⊛ Fast-R-CNN, Microsoft Research, 2015

R. Girshick (2015). *Fast R-CNN*. Tech. rep. arXiv:1504.08083



Girshick (2015)

- *Fast R-CNN* model zintegrowany (zamiast struktury modułowej)
  - ekstrakcja cech przez sieć splotową (VGG-16)
  - łączenie RoI (ang. *RoI Pooling*): warstwa określa cechy dla proponowanych regionów
  - wyjście sieci jest analizowane przez sieć perceptronową
    - wyjścia klasyfikujące (użycie softmax)
    - wyjścia liniowe dla regresji BB (pozycja ramek)
  - proces powtórzony wielokrotnie dla każdego RoI
- trening znacznie szybszy (choć wymaga kandydatów RoI)
- kody źródłowe Python i C++ (Caffe) dostępne na GitHub

⊛ Fast R-CNN

Klasyfikacja i regresja BB

- RoI jest określany przez współrzędne środka p_x, p_y i rozmiary p_h, p_w
- adjustacja: współczynniki p_w, p_x, p_h, p_y

$$\begin{aligned}\widehat{g}_x &= p_w d_x(\boldsymbol{p}) + p_x\\ \widehat{g}_y &= p_h d_y(\boldsymbol{p}) + p_y\\ \widehat{g}_w &= p_w \exp\big(d_w(\boldsymbol{p})\big)\\ \widehat{g}_h &= p_h \exp\big(d_h(\boldsymbol{p})\big)\end{aligned}$$

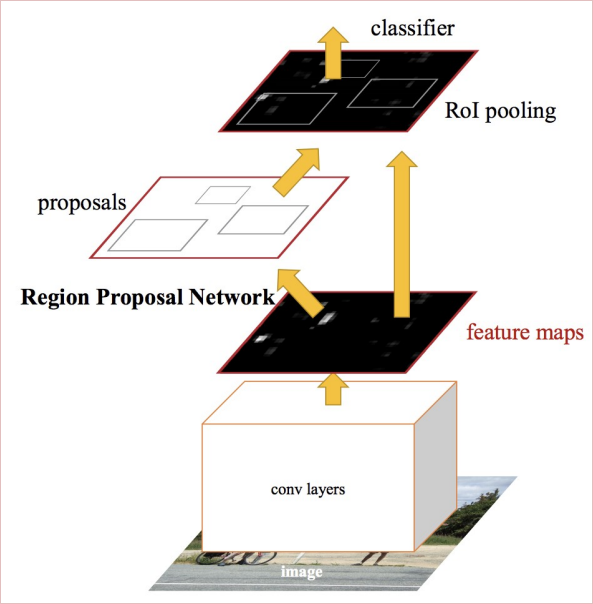
- funkcja kosztu

$$\ell_{\text{reg}} = (t_x - (g_x - p_x)/p_w)^2 + (t_y - (g_y - p_y)/p_h)^2 + (t_h - \ln(g_h/p_h))^2 (t_w - \ln(g_w/p_w))^2$$

- *tłumienie nie-maksimów* (ang. *non-maximum suppression*): dla wielokrotnych BB tego samego obiektu
  - odrzucić te o niskim *współczynniku ufnośc* (ang. *confidence score*)
  - wybrać najlepszy
  - odrzucić te o wysokim IoU (> 0.5)
- wyszukiwanie przykładów trudnych negatywnych (ang. *hard negative mining*) do ponownego treningu

⊛ Faster-R-CNN

S. Ren et al. (2016). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. Tech. rep. arXiv: 1506.01497 [cs.CV]



Ren et al. (2016)

- modyfikacje
  - proponowanie regionów przy użyciu *sieci proponowania regionów* (ang. *Region Proposal Network*) RPN
  - RPN zintegrowany z siecią splotową i perceptronową
- zredukowana liczba operacji i przyspieszone obliczenia: 5 fps na GPU
- na ILSVRC and COCO 2015 struktura wygrywająca w kilku konkursach

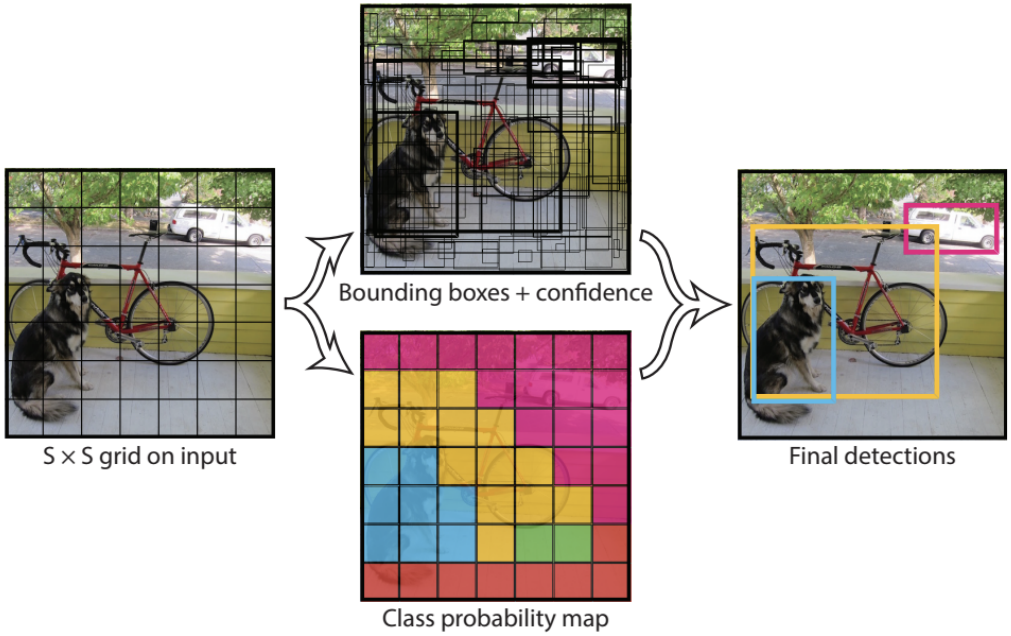
⊛ Mask R-CNN

K. He, G. Gkioxari, et al. (2018). *Mask R-CNN*. arXiv: 1703.06870 [cs.CV]

- dodatkowa funkcja segmentacji obrazu
- Python and C++ (Caffe) source code dostępny w GitHub repository

YOLO, Facebook, Redmon e.a. 2015

J. Redmon, S. Divvala, et al. (2015). *You Only Look Once: Unified, Real-Time Object Detection.* arXiv: 1506.02640 [cs.CV]



Redmon, Divvala, et al. (2015)

- *You Only Look Once* YOLO - sieć głęboka estymująca równocześnie BB i etykietę klasy, podejście "end to end"
- mniejsza dokładność niż R-CNN, ale działa z szybkością 45 fps (wersja podstawowa) - 155 fps (Fast YOLO)

YOLO, Redmon e.a. 2015/2016 (Facebook)

J. Redmon, S. Divvala, et al. (2015). *You Only Look Once: Unified, Real-Time Object Detection*. arXiv: 1506.02640 [cs.CV]

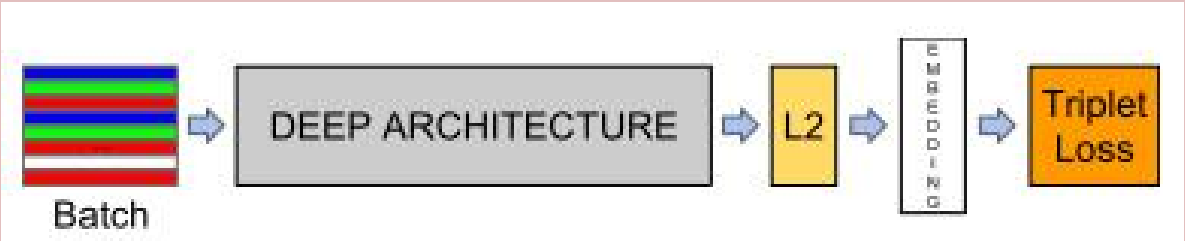
- obraz podzielony na siatkę $m \times m$ komórek
  - Redmon, Divvala, et al. (2015): komórki 7×7
- każda komórka przewiduje BB (współrzędne x, y, szerokość, wysokość, poziom ufności), i klasę jeśli centrum BB leży wewnątrz komórki
  - Redmon, Divvala, et al. (2015): każda komórka może przewidywać 2BB
  - prawdopodobieństwa klas i BB z poziomami ufności są łączone w zbiór BB i etykiet klas

⊛ YOLO v2 (YOLO 9000), YOLO v.3

- YOLO v.2, 2016 (YOLO 9000)
  - J. Redmon and A. Farhadi (2016). *YOLO9000: Better, Faster, Stronger*. Tech. rep. arXiv: 1612.08242 [cs.CV]
  - trening na dwu zbiorach równolegle, możliwe przewidywanie 9,000 klas
  - obrazy wysokiej rozdzielczości,
  - *ramki kotwiczące* (ang. *anchor boxes*) - wcześniej zdefiniowane BB z użytecznymi kształtami i rozmiarami
  - dobór BB przetwarzany zgodnie z analizą k-means na zbiorze trenującym
  - przewidywanie BB pozwala na zmniejszenie efektów małych zmian (stabilizacja): zamiast bezpośredniego określania parametrów BB, przewidywane położenia są modyfikowane (offsets)
- YOLO v.3, 2018
  - J. Redmon and A. Farhadi (2018). *YOLOv3: An Incremental Improvement*. arXiv: 1804.02767 [cs.CV]
- YOLO v.4, 2018

⊛ FaceNet (Google), 2015

F. Schroff, D. Kalenichenko, and J. Philbin (2015). "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*



Schroff, Kalenichenko, and Philbin (2015)

- elementy struktury
  - sieci głębokie: ZF lub Inception
  - normalizacja L2 i osadzenie (ang. *embedding*) na powierzchni kuli w $\mathbb{R}^d$ (np. $n = 128$)
  - trening strat trójkami
- uczenie na zbiorach od kilku do kilkuset milionów twarzy
- uczenie trójkami (ang. *triplet loss*)
- błąd 0.35% na *Labelled Faces in the Wild*
- Next Generation Identification (FBI): błąd 15%

14.1 Struktura sieci splotowych
14.2 Przełomowe architektury splotowe
14.3 Modyfikacje procesów uczenia

Transfer uczenia

- interpretacja struktur głębokich sieci neuronowych
  - pierwszą warstwę o połączeniach kompletnych można zazwyczaj traktować jako cechy obrazu
  - warstwy wcześniejsze działają jako ekstraktor cech
  - warstwy późniejsze działają jako klasyfikator
  - pełna sieć jest uczona na ogromnych zbiorach danych (np. ImageNet: 14 M obrazów) dla dużej liczby klas (INet: 1000 klas) a parametry są publicznie udostępnione
  - dla zadanego problemu uczenia zwykle mamy nieproporcjonalnie mało danych
- transfer uczenia (ang. *transfer learning*)
  - założyć, że interesujące nas cechy są funkcją zadanych klas
  - wniosek: dla nowego problemu, zamrozić wagi ekstraktora cech aby otrzymać cechy
  - uzupełnić sieć przez klasyfikator o nieznanych wagach i trenować klasyfikator przy zamrożonych wagach ekstraktora
  - w przypadku klas które nie dają się wyrazić przez funkcję istniejących cech, można również trenować najwyższe warstwy ekstraktora cech (zbyt dopasowanych do oryginalnego zadania)

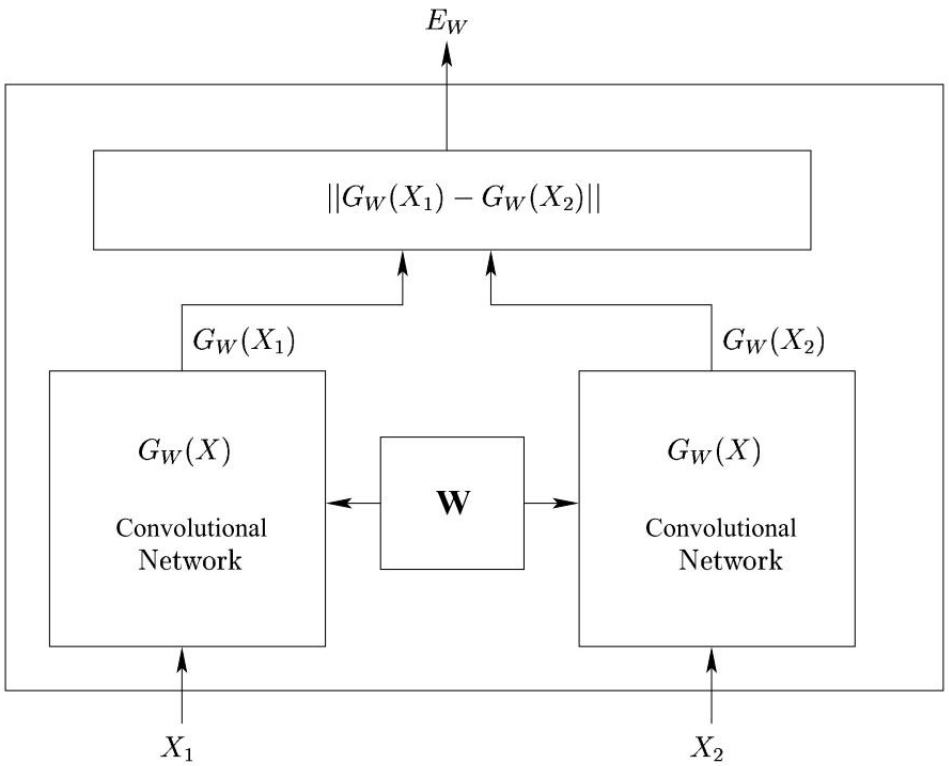
Uczenie metryki

- problem uczenia metryki: dane są
  - funkcja d wyznaczająca odległość $d(x, y)$ pomiędzy obiektami x i y
  - informacja dotycząca "pożądanej odległości"

  określić funkcję f taką, że w odległość $\widetilde{d}(x, y) = d\big(f(x), f(y)\big)$ lepiej określa pożądaną metrykę
  - przypadek liniowy: $f(x) = Gx$
  - sieć neuronowa: $f(x) = G(x)$ gdzie G jest neuronowym ekstraktorem cech

Uczenie metryki

Uczenie syjamskie



Chopra, Hadsell, and LeCun (2005)

Uczenie metryki

Uczenie trójkami

- *przykład kotwiczący* (ang. *anchor example*) $\boldsymbol{u}_a$, przykłady pozytywne $\boldsymbol{u}^+$ (tzn. z klasy tej co kotwiczący, negatywne $\boldsymbol{u}^-$ (tzn. z pozostałych klas)
- cel: zbliżenie przykładu dodatniego i odsunięcie przykładu ujemnego od przykładu kotwiczącego
- *strata trójkami* (ang. *triplet loss*) $\|\boldsymbol{u}_i^a - \boldsymbol{u}_i^+\|^2 + \alpha < \|\boldsymbol{u}_i^a - \boldsymbol{u}_i^-\|^2$, gdzie α - margines błędu
- koszt

$$R_{\mathcal{S}}(\boldsymbol{u}_i^a, \boldsymbol{u}_i^+, \boldsymbol{u}_i^-) = \min \sum_{i=1}^{N} \left[\|G(\boldsymbol{u}_i^a) - G(\boldsymbol{u}_i^+)\|^2 + \alpha - \|G(\|\boldsymbol{u}_i^a) - G(\boldsymbol{u}_i^-)\|^2 \right]_+$$

Schroff, Kalenichenko, and Philbin (2015)

- dobór elementów uczących w porcjach (ang. *mini-batches*) rzędu dziesiątek/setek elementów
- problem: dobór "trudnych" elementów tej samej i różnych klas

Literatura

Materiały źródłowe

Deshpande, A. (2018). *The 9 Deep Learning Papers You Need To Know About*. Tech. rep. UCLA. URL: https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html.

Fukushima, K. (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36, pp. 193–202.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

LeCun, Y. et al. (Nov. 1998). "Gradient-Based Learning applied to Document Recognition". In: *Proceedings of the IEEE*, pp. 1–46.

Zeiler, M. D. and R. Fergus (2013). "Visualizing and Understanding Convolutional Networks". In: *CoRR* abs/1311.2901. URL: http://arxiv.org/abs/1311.2901.

Materiały pomocnicze

Chopra, S., R. Hadsell, and Y. LeCun (2005). "Learning a Similarity Metric Discriminatively, with Application to Face Verification". In: *IEEE Computer Society Conference Computer Vision and Pattern Recognition CVPR*.

Literatura (cd.)

Cireşan, D. C. et al. (2011). "Flexible, High Performance Convolutional Neural Networks for Image Classification". In: *Proc. of the 22-nd Intern. Joint Conf. on Artificial Intelligence*. DSIA, USI and SUPSI. Manno-Lugano, Switzerland, pp. 1237–1242.

Girshick, R. (2015). *Fast R-CNN*. Tech. rep. arXiv:1504.08083.

Girshick, R. et al. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation*. Tech. rep. arXiv:1311.2524. arXiv.org.

He, K., G. Gkioxari, et al. (2018). *Mask R-CNN*. arXiv: 1703.06870 [cs.CV].

He, K., X. Zhang, et al. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. Tech. rep. Microsoft Research. URL: //arxiv.org/pdf/1502.01852.pdf.

Lebret, R., P. O. Pinheiro, and R. Collobert (2015). *Phrase-based Image Captioning*. Tech. rep.

Lin, M., Q. Chen, and S. Yan (2014). "Network In Network". In: *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*.

Redmon, J., S. Divvala, et al. (2015). *You Only Look Once: Unified, Real-Time Object Detection*. arXiv: 1506.02640 [cs.CV].

Redmon, J. and A. Farhadi (2016). *YOLO9000: Better, Faster, Stronger*. Tech. rep. arXiv: 1612.08242 [cs.CV].

— (2018). *YOLOv3: An Incremental Improvement*. arXiv: 1804.02767 [cs.CV].

Literatura (cd.)

Ren, S. et al. (2016). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. Tech. rep. arXiv: 1506.01497 [cs.CV].

Schroff, F., D. Kalenichenko, and J. Philbin (2015). "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Szegedy, C. et al. (2014). *Going deeper with convolutions*. Tech. rep. arXiv:1409.4842v1.

Taigman, Y. et al. (2014). "DeepFace: Closing the Gap to Human-Level Performance in Face Verification". In: Conference on Computer Vision and Pattern Recognition (CVPR).

Uijlings, J. et al. (2012). *Selective Search for Object Recognition*. Tech. rep. submitted to IJCV.

Wurtz, R. H. (2015). "Using perturbations to identify the brain circuits underlying active vision". In: *Philosophical Transactions of the Royal Society B* 370. URL: http://rstb.royalsocietypublishing.org/content/roypth/370/1677/20140205.full.pdf.

Xu, J. (2017). *Deep learning. Natural language processing*. Tech. rep. Stanford Uni.

Zeiler, M. D., G. W. Taylor, and R. Fergus (2011). "Visualizing and Understanding Convolutional Networks". In: *ECCV 2014 part 1*. Ed. by D. F. et al. LNCS 8689. Springer, pp. 818–833.

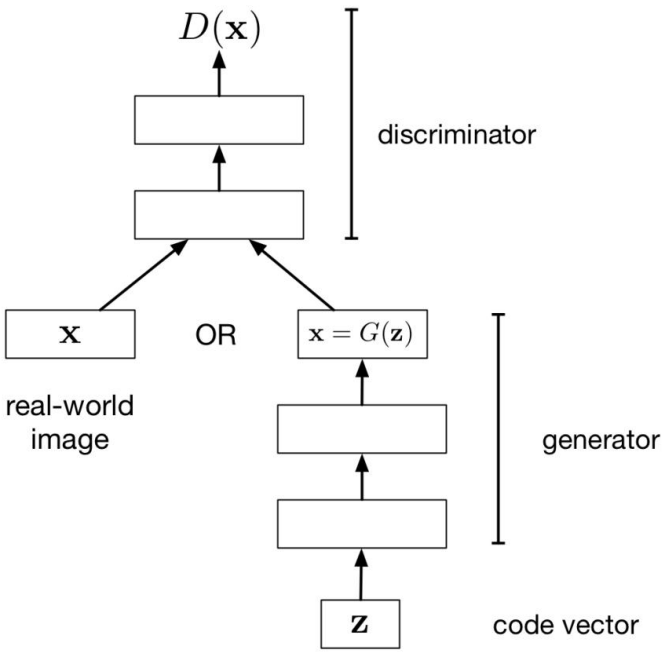
Temat 15: Generatywne sieci antagonistyczne

Generatywne sieci antagonistyczne

Schemat działania

I. J. Goodfellow et al. (2014b). *Generative Adversarial Networks*. Tech. rep.
Le Cun (2016)

- zbiór "obrazów prawdziwych" $\mathcal{T}$ - posiadających pewną wspólną cechę
- *generator* losuje *wektory kodowe* (ang. *code vectors*) $z \sim p_z$; np.
  - $z \in Z = \mathbb{R}^{100}$
  - p_z - rozkład normalny
- zbiór "obrazów fałszywych" $\mathcal{F}$ zawiera obrazy $G(z)$ utworzone z przekształconych wektorów kodowych z
- zadaniem *generatora* jest tworzenie obrazów "sztucznych" nieodróżnialnych od obrazów prawdziwych
- zadaniem *dyskryminatora* jest prawidłowe rozróżnienie obrazów prawdziwych $x \in \mathcal{T}$ od obrazów sztucznych (generowanych) $G(z) \in \mathcal{G}$
- dyskryminator szacuje prawdopodobieństwo $D(x)$ że analizowany obraz jest prawdziwy



Goodfellow et al. (2014b)

Generatywne sieci antagonistyczne GAN

Gra minimaksowa

- $D(\boldsymbol{x})$ - prawdopodobieństwo rozpoznania obrazu jako prawdziwy
- celem dyskryminatora jest **maksymalizacja** prawidłowego rozpoznawania obrazu

$$V_D = \underbrace{\mathscr{E}_{\boldsymbol{x}\sim p_x} \log D(\boldsymbol{x})}_{\substack{\text{obrazy} \in X \\ \text{rozpoznane jako prawdziwe}}} + \underbrace{\mathscr{E}_{\boldsymbol{z}\sim p_z} \log(1 - D(G(\boldsymbol{z})))}_{\substack{\text{obrazy} \in \mathcal{F} \\ \text{rozpoznane jako fałszywe}}}$$

- celem generatora jest maksymalizacja generacji obrazu "podobnego" do naturalnego, tzn. **minimalizacja** prawdopodobieństwa prawidłowego rozpoznania obrazu

$$V_G = \underbrace{\mathscr{E}_{\boldsymbol{z}\sim p_z} \log(1 - D(G(\boldsymbol{z})))}_{\substack{\text{obrazy} \in \mathcal{F} \\ \text{rozpoznane jako fałszywe}}}$$

- ponieważ pierwszy człon dla V_D nie zależy od G, zadanie można zapisać jako *grę minimaksową* graczy G i D

$$\min_G \max_D V_{G,D} \quad \text{gdzie}$$

$$V_{G,D} = \mathscr{E}_{\boldsymbol{x}\sim p_x} \log D(\boldsymbol{x}) + \mathscr{E}_{\boldsymbol{z}\sim p_z} \log(1 - D(G(\boldsymbol{z})))$$

Generatywne sieci antagonistyczne

Sieci neuronowe

- generator G i dyskryminator D zrealizowane w postaci sieci neuronowych

$$G(\boldsymbol{x}) = G_{\boldsymbol{g}}(\boldsymbol{x}) \qquad \text{sieć generatora o wagach } \boldsymbol{g}\text{: wektor} \mapsto \text{obraz}$$
$$D(\boldsymbol{z}) = D_{\boldsymbol{d}}(\boldsymbol{z}) \qquad \text{sieć dyskryminatora o wagach } \boldsymbol{d}\text{: obraz} \mapsto (0,1)$$

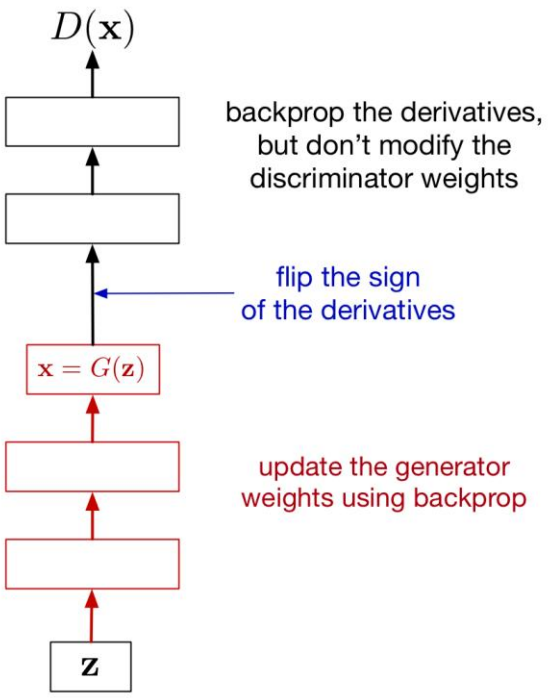
- wskaźniki empiryczne dla sieci neuronowych

$$V_G(\boldsymbol{g},\boldsymbol{d}) = \mathscr{A}_{\boldsymbol{z}\in\boldsymbol{Z}} \log\Big(1 - D_{\boldsymbol{d}}(G_{\boldsymbol{g}}(\boldsymbol{z}))\Big)$$
$$V_D(\boldsymbol{d},\boldsymbol{g}) = \mathscr{A}_{\boldsymbol{x}\in\boldsymbol{X}} \log D_{\boldsymbol{d}}(\boldsymbol{x}) + \mathscr{A}_{\boldsymbol{z}\in\boldsymbol{Z}'} \log\Big(1 - D_{\boldsymbol{d}}(G_{\boldsymbol{g}}(\boldsymbol{z}))\Big)$$

gdzie $\mathscr{A}_{\boldsymbol{z}\in\boldsymbol{Z}'}$ jest średnią wyliczoną dla próby $\boldsymbol{Z}'$ innej niż $\boldsymbol{Z}$

Generatywne sieci antagonistyczne

Uczenie sieci generatora

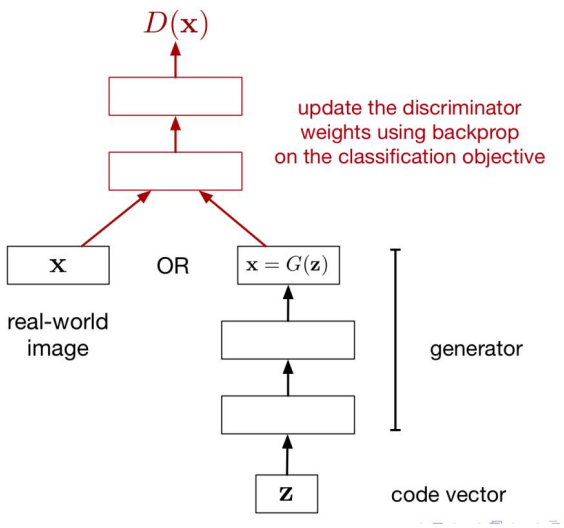


Grosse (2018)

- generacja obrazu
  - losowanie elementu z
  - utworzenie obrazu $x = G(z)$ przez sieć generatora
- oszacowanie przez sieć dyskryminatora prawdopodobieństwa D, że obraz $G(z)$ prawdziwy
- propagacja wsteczna gradientu funkcji D
  - propagacja przez sieć dyskryminatora do sieci generatora **bez zmiany wag dyskryminatora**
  - zmiana znaku gradientu (minimalizacja D)
  - kontynuacja propagacji wstecznej przez sieć generatora G_{gv} i modyfikacja wag generatora (kara za nieprecyzyjną generację)

Generatywne sieci antagonistyczne

Uczenie sieci dyskryminatora



Grosse (2018)

obliczenie gradientów wag dyskryminatora:

- losować obraz x ze zbioru "prawdziwych" obrazów $\mathcal{T}$
- x jest klasyfikowany przez dyskryminator D
- sieć dyskryminatora oblicza prawdopodobieństwo, że obraz jest rzeczywisty i funkcję strat dyskryminatora L_D
- **sieć dyskryminatora** jest karana za błędne rozróżnianie obrazów generowanych i prawdziwych
  - propagacja wsteczna gradientu funkcji strat dyskryminatora do wag sieci dyskryminatora
  - kontynuacja przez sieć dyskryminatora wejścia sieci dyskryminatora
  - kontynuacja propagacji wstecznej do wag generatora
- wynik: gradienty chwilowe wag dyskryminatora i wag generatora

Generatywne sieci antagonistyczne

Algorytm GAN

Algorytm GAN

require learning rate α, batch size m, number of discriminator iterations per generator iteration n_{disc}

require initial discriminator parameters $\boldsymbol{d}$, initial parameters $\boldsymbol{g}$

% *discriminator parameters*

while $\boldsymbol{g}$ has not converged **do**

  for $t = 0, \ldots, n_{\text{disc}}$ **do**

    sample $\{\boldsymbol{x}^{(i)}, i = 1, \ldots, m\}$ from p_x

    sample $\{\boldsymbol{z}^{(i)}, i = 1, \ldots, m\}$ from p_z

    $\nabla_{\boldsymbol{d}} = \mathrm{BP}_{\boldsymbol{d}}\Big(\mathscr{A} \log D_{\boldsymbol{d}}(\boldsymbol{x}^{(i)})\Big) + \mathscr{A} \log(1 - D_{\boldsymbol{d}}(G_{\boldsymbol{g}}(\boldsymbol{z}^{(i)})))\Big)$

    $\boldsymbol{d} \leftarrow \boldsymbol{d} + \alpha\, \mathrm{step}(\boldsymbol{d}, \nabla_{\boldsymbol{d}})$

  end for

  % *generator parameters*

  sample $\{\boldsymbol{z}^{(i')}, i = 1, \ldots, m\}$ from p_z

  $\nabla_{\boldsymbol{g}} = -\mathrm{BP}_{\boldsymbol{g}} \mathscr{A} \log\Big(1 - D_{\boldsymbol{d}}(G_{\boldsymbol{g}}(\boldsymbol{z}^{(i')}))\Big)$

  $\boldsymbol{g} \leftarrow \boldsymbol{g} - \alpha\, \mathrm{step}(\boldsymbol{g}, \nabla_{\boldsymbol{g}})$

end while

Generatywne sieci antagonistyczne

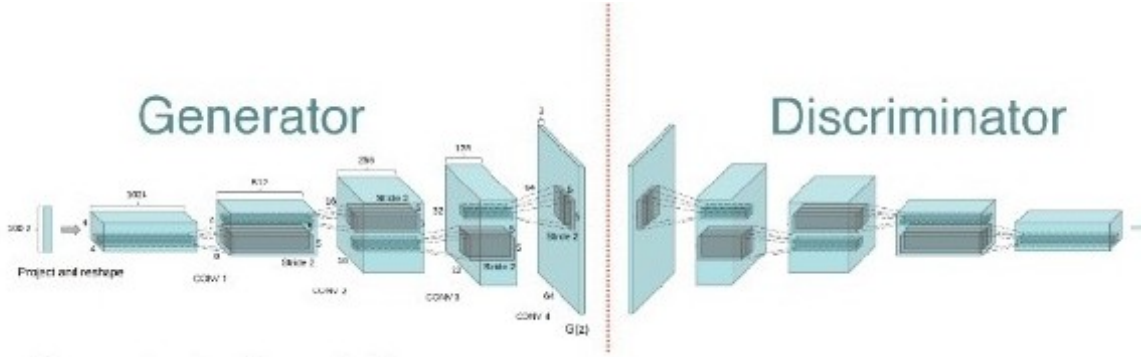
Niektóre zastosowania

- tworzenie realistycznych sztucznych obrazów (twarze, krajobrazy, etc.)
- generacja nowych danych
- zwiększanie precyzji obrazów (astronomia)
- rekonstrukcja 3D z obrazów 2D i wzorców ruchu wideo
- postarzanie portretów twarzy
- rekonstrukcja twarzy na podstawie głosu (Speech2Face)
- generacja głosu

Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

DCGAN (Deep Convolutional GAN)

A. Radford, L. Metz, and S. Chintala (2016). "DCGAN: Adversarial training to generate images". In: *International Conference on Learning Representation ICLR*



- zastąpić warstwy agregujące w sieci dyskryminatora przez *sploty kroczące* (ang. *strided convolutions*)
- zastąpić warstwy agregujące w sieci generatora przez *sploty ułamkowe* (sploty odwrotne, sploty transponowane),(ang. *fractional-strided convolutions*)
  - splot można rozumieć jako mnożenie wektoryzowanego obrazu przez odpowiednią *macierz splotu* C, a *splot ułamkowy* jako mnożenie przez macierz C^{T}
  - zadanie to może być realizowane przez splot z *krokiem ułamkowym*, sprowadzającym się do przedzielania pikseli obrazu zerami
- zastosować normalizację (ang. *batch norm*) w sieciach generatora i dyskryminatora
- nie stosować warstw w pełni połączonych
- stosować ReLU w sieci generatora z wyjątkiem warstwy wyjściowej (tam: tanh), a LeakyReLU w sieci dyskryminatora

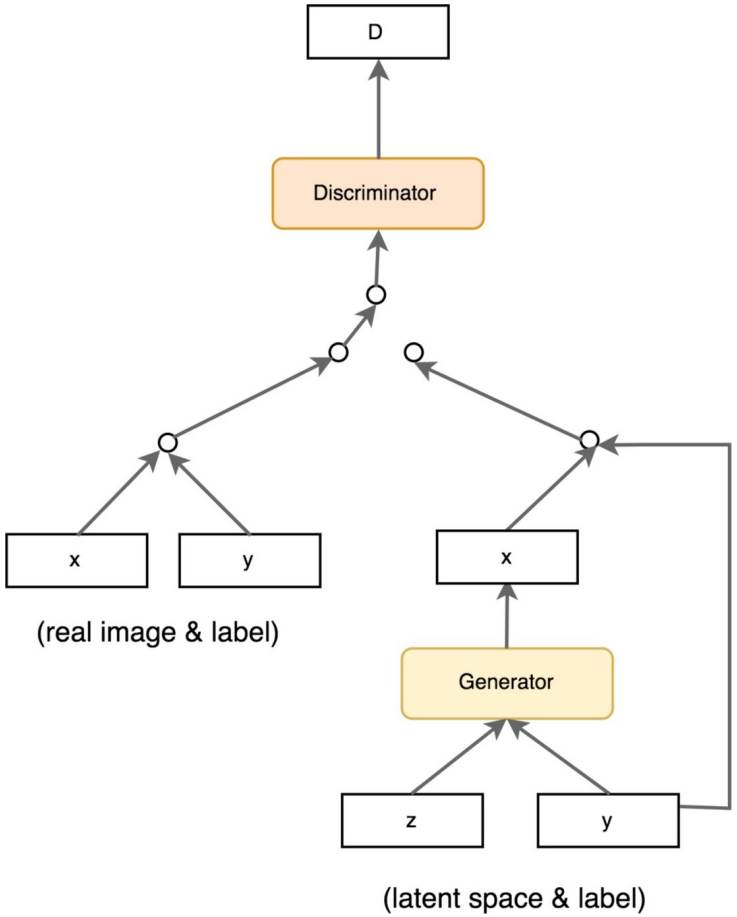
Typowe problemy uczenia sieci GAN

- minimaksowa funkcja strat może prowadzić do zatrzymania treningu na wczesnym etapie uczenia, gdy zadanie dyskryminatora jest łatwe
  - modyfikacja funkcji strat generatora: $\max \log D(G(z))$, sieć WGAN
- *zanikający gradient* (ang. *vanishing gradient*): pochodne są tak małe, że przyrosty wag są niezauważalne
  - WGAN
- *załamanie mody* (ang. *mode collapse*): generowane obrazy są znacznie mniej różnorodne, niż obrazy rzeczywiste
- normalizacja porcjami (ang. *Batch Normalization*) (Ioffe and Szegedy 2015) stabilizuje uczenie - istotne ze względu na inicjalizację sieci głębokiej
- normalizacja jest istotna przy rozpoczynaniu uczenia i zapobiega załamaniu mody do jednego punktu
- *oscylacje i niestabilność* może być spowodowana przez normalizację wszystkich warstw: nie stosować normalizacji warstwy wyjściowej generatora i wejściowej dyskryminatora
- dyskusja wpływu funkcji strat: Lucic et al. (2018)

CGAN - Warunkowe GAN (ang. *Conditional GAN*)

I. J. Goodfellow et al. (2014a). "Generative Adversarial Nets". In: *Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014)*, pp. 2672–2680

- generator i dyskryminator otrzymują dodatkowo informację o etykiecie $\boldsymbol{y}$ (np. etykieta klasy, dane)
- zastosowania: translacje obraz $\mapsto$ obraz (np. postarzanie / odmładzanie twarzy), synteza tekst $\mapsto$ obraz, generacja wideo
- generator uczy się tworzenia obrazów dla każdej etykiety
- dyskryminator uczy się odrzucać zarówno obrazy fałszywe jak i źle etykietowane

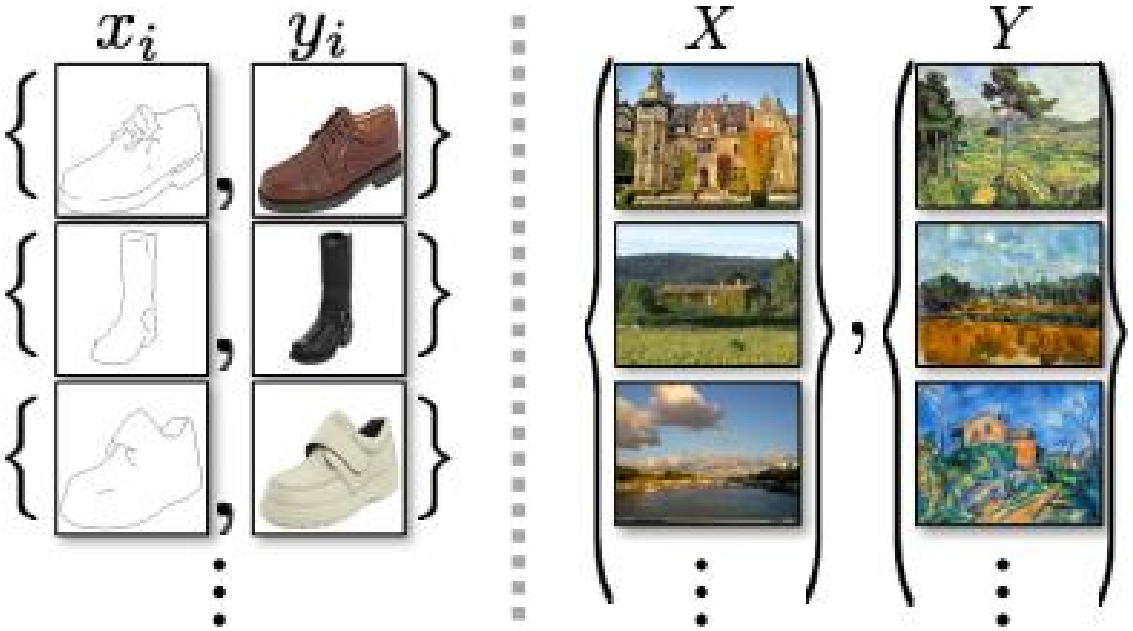


- zagadnienie

$$\min_{G} \max_{D,L} \left(\mathcal{E}_{p_{\boldsymbol{x},\boldsymbol{y}}} \log D(\boldsymbol{x},\boldsymbol{y}) + \mathcal{E}_{p_{\boldsymbol{x},\boldsymbol{z}}} \log(1 - D(\boldsymbol{x}, G(\boldsymbol{z},\boldsymbol{y}))) \right)$$

CycleGAN

Obrazy sparowane i niesparowane



J.-Y. Zhu et al. (2020). *Unpaired Image-to-Image, Translation using Cycle-Consistent Adversarial Networks*. Tech. rep. Berkeley AI Research (BAIR) laboratory, UC Berkeley. URL: %5Curl%7Bhttps://arxiv.org/abs/1703.10593v7%7D

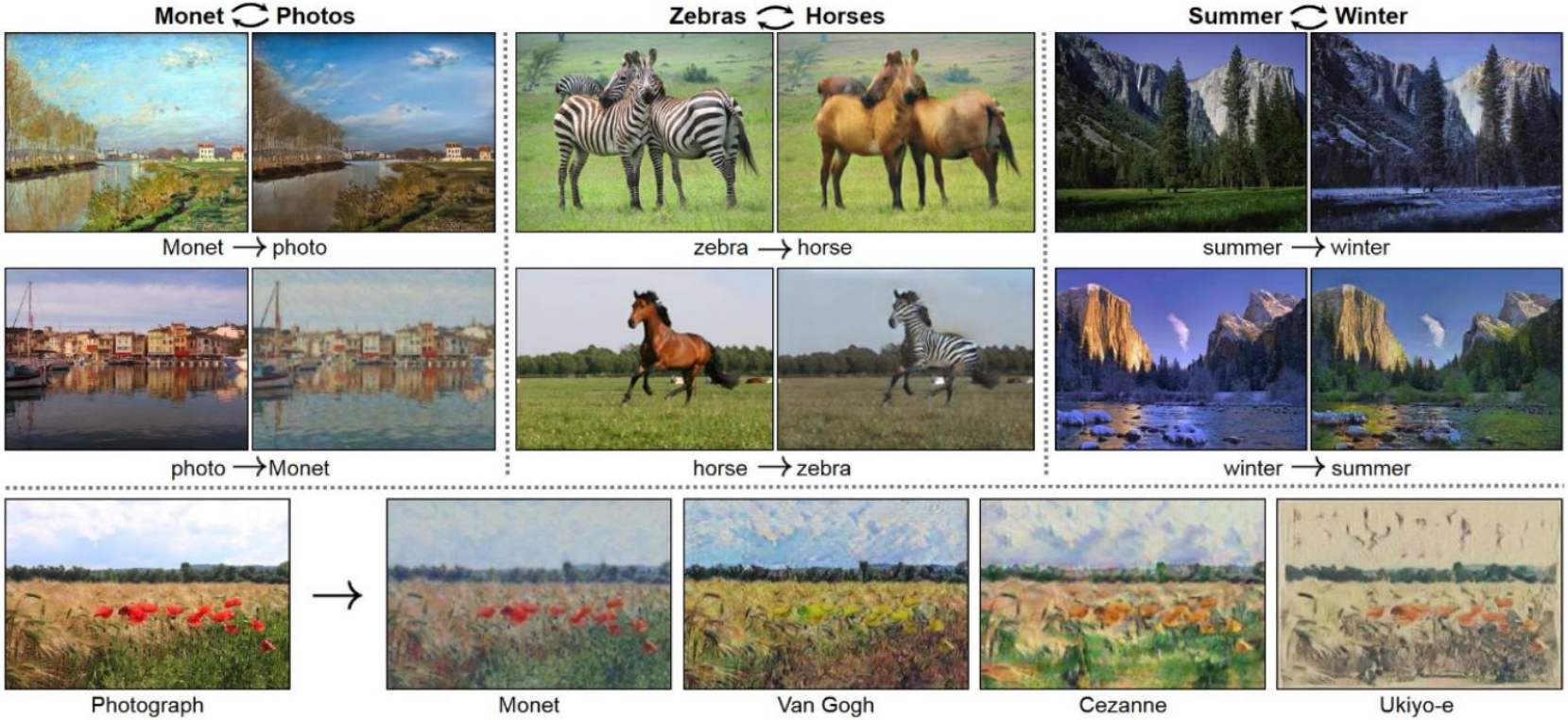
- obrazy sparowane: do uczenia niezbędne są pary pokazujące żądaną transformację obrazu (Pix2Pix)
- obrazy niesparowane: do uczenia wykorzystywane są dwa zbiory obrazów charakteryzujące pożądane transformacje

Cycle GAN - GAN cykliczny

Przykłady zastosowań

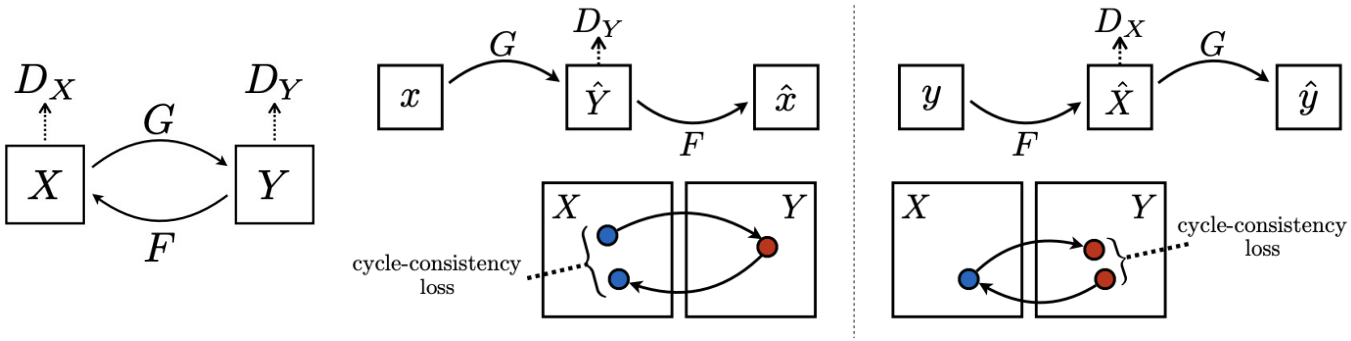
J-Y Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", 2020

- transfer stylu: zmienić styl obrazu zachowując jego zawartość
  - dwa zbiory danych (styl 1/ styl 2), np. fotografia/obrazy (Moneta, Picassa,...), obrazy/fotografie, zebry/konie, lato/zima, noc/dzień



CycleGAN

J.-Y. Zhu et al. (2020). *Unpaired Image-to-Image, Translation using Cycle-Consistent Adversarial Networks*. Tech. rep. Berkeley AI Research (BAIR) laboratory, UC Berkeley. URL: %5Curl%7Bhttps://arxiv.org/abs/1703.10593v7%7D https://arxiv.org/abs/1703.10593v7



- idea: obraz $\boldsymbol{x}$ jest przetwarzany przez generator G i $G(\boldsymbol{x})$ jest porównywany z $\boldsymbol{x}$ przez dyskryminator D_Y
- również obraz $\boldsymbol{y}$ jest przetwarzany przez generator F i $F(\boldsymbol{y})$ jest porównywany z $\boldsymbol{y}$ przez D_X
- błędy rekonstrukcji

$$L_{\text{GAN}}(G, D_X) = \mathcal{E}_{\boldsymbol{y}\sim p_y} \log D_Y(\boldsymbol{y}) + \mathcal{E}_{\boldsymbol{x}\sim p_x} \log\Big(1 - D_Y(G(\boldsymbol{x}))\Big)$$

$$L_{\text{GAN}}(F, D_Y) = \mathcal{E}_{\boldsymbol{x}\sim p_x} \log D_X(\boldsymbol{x}) + \mathcal{E}_{\boldsymbol{y}\sim p_y} \log\Big(1 - D_X(F(\boldsymbol{y}))\Big)$$

- minimalizacja *straty zgodności cykli* (ang. *cycle consistency loss*)

$$L_{\text{cyc}} = \mathcal{E}_{\boldsymbol{x}\sim p_x} \|F(G(\boldsymbol{x})) - \boldsymbol{x}\|_1 + \mathcal{E}_{\boldsymbol{y}\sim p_y} \|G(F(\boldsymbol{y})) - \boldsymbol{y}\|_1$$

- błąd CycleGAN

$$\min_{F,G} \max_{D_X, D_Y} \Big(L_{\text{GAN}}(G, D_X) + L_{\text{GAN}}(F, D_Y) + \lambda_{\text{cyc}}(G, F)\Big)$$

WGAN: Wasserstein GAN

M. Arjovsky, S. Chintala, and L. Bottou (2017). "Wasserstein Generative Adversarial Networks". In: *Proc. of the 34th International Conference on Machine Learning, PMLR 70*, pp. 214–223

- sieć Wassersteina w miejsce dyskryminatora tworzącego ocenę binarną stosowany jest *krytyk* oceniający średniokwadratowe podobieństwo w skali $[0, 1]$
- funkcja krytyka: maksymalizować $\mathcal{E}_{\boldsymbol{x}\sim p_x} C(\boldsymbol{x}) - \mathcal{E}_{\boldsymbol{z}\sim p_z} C(G(\boldsymbol{z}))$
- funkcja generatora: maksymalizować $\mathcal{E}_{z\sim p_z} C(G(z))$
- WGAN poprawia stabilność uczenia
- w szczególności nie występuje problem załamania mody

GAN Wassersteina

Algorytm

Algorytm WGAN

require learning rate α, clipping parameter c, batch size m, number of critic iterations per generator iteration n

require initial critic parameters $\boldsymbol{c}$, initial generator parameters $\boldsymbol{g}$

while $\boldsymbol{g}$ has not converged **do**

  for $t = 0, \ldots, n$ **do**

% *critic parameters*

    sample $\{\boldsymbol{x}^{(i)}, i = 1, \ldots, m\}$ from p_x

    sample $\{\boldsymbol{z}^{(i)}, i = 1, \ldots, m\}$ from p_z

    $\nabla_{\boldsymbol{c}} = \mathrm{BP}_{\boldsymbol{c}}\left(\mathscr{A}C_{\boldsymbol{c}}(\boldsymbol{x}^{(i)}) - \mathscr{A}C_{\boldsymbol{c}}(G_{\boldsymbol{g}}(\boldsymbol{z}^{(i)}))\right)$

    $\boldsymbol{c} \leftarrow \boldsymbol{c} + \alpha\,\mathrm{step}(\boldsymbol{c}, \nabla_{\boldsymbol{c}})$

    $\boldsymbol{c} \leftarrow \mathrm{clip}(\boldsymbol{c}, -c_o, c_o)$

  end for

% *generator parameters*

  sample $\{\boldsymbol{z}^{(i)}, i = 1, \ldots, m\}$ from p_z

  $\nabla_{\boldsymbol{g}} = -\mathrm{BP}_{\boldsymbol{g}}\,\mathscr{A}\left(C_{\boldsymbol{c}}(G_{\boldsymbol{g}}(\boldsymbol{z}^{(i)}))\right)$

  $\boldsymbol{g} \leftarrow \boldsymbol{g} - \alpha\,\mathrm{step}(\boldsymbol{g}, \nabla_{\boldsymbol{g}})$

end while