Sztuczne sieci neuronowe

Marek Grochowski

Wydział Fizyki, Astronomii i Informatyki Stosowanej UMK

http://www.fizyka.umk.pl/~grochu/nn
grochu@is.umk.pl

9 czerwca 2026

Plan

1. Perceptrony, sieci wielowarstwowe jednokierunkowe (MLP)
2. Metody uczenia i algorytm wstecznej propagacji błędu
3. Radialne funkcje bazowe (RBF) i metody aproksymacji
4. Samoorganizacja, sieci SOM, uczenie konkurencyjne
5. Sieci dynamiczne: model Hopfielda, maszyny Boltzmana
6. Głębokie sieci neuronowe (DNN)
7. Sieci splotowe (CNN)
8. Sieci rekurencyjne (RNN) i uczenie sekwencji
9. Autoenkodery, wykrywanie cech, DBN
10. Modele generatywne GAN, VAE
11. Transformery, mechanizm uwagi, LLM, ViT

Literatura I

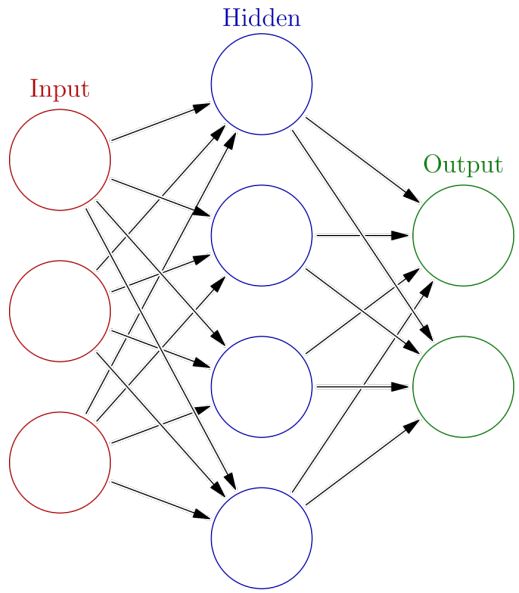
- S. Osowski, *Sieci neuronowe w ujęciu algorytmicznym*, Wydawnictwo Naukowo-Techniczne, Warszawa 1996.
- R. Tadeusiewicz, *Sieci neuronowe*, Akademicka Oficyna Wydawnicza RM, Warszawa 1993.
- R. Tadeusiewicz, T. Gąciarz, B. Borowik, B. Lepe, *Odkrywanie właściwości sieci neuronowych przy użyciu programów w języku C#*, Polska Akademia Umiejętności, 2008.
- J. Zurada, M. Barski, W. Jędruch, *Sztuczne sieci neuronowe*, Wydawnictwo Naukowe PWN, 1996.

Literatura II

- Ian Goodfellow, Yoshua Bengio i Aaron Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`
- Michael Nielsen, *Neural Networks and Deep Learning*, `http://neuralnetworksanddeeplearning.com/`
- Julien Vitay, *Neurocomputing*, `https://julien-vitay.net/course-neurocomputing/`
- Denny Britz, *Deep Learning Glossary*, `http://www.wildml.com/deep-learning-glossary/`

Sztuczne Sieci Neuronowe

Systemy obliczeniowe inspirowane biologicznymi sieciami neuronowymi, składające się ze zbioru prostych jednostek przetwarzających sygnał (neuronów), komunikujących się między sobą za pomocą połączeń (wag synaptycznych).



- model matematyczny realizujący odwzorowanie $f(\mathbf{x}; \mathbf{W}) = \mathbf{y}$
- systemy uczące się, adaptowanie wag $\mathbf{W}$, Uczenie Maszynowe
- systemy inteligentne: Inteligencja Obliczeniowa, Sztuczna Inteligencja

Inteligencja obliczeniowa

Computational Intelligence (CI) zajmuje się rozwiązywaniem problemów, które nie są efektywnie algorytmizowalne.
Cechą wielu systemów CI jest rozwiązywanie zadań na podstawie przykładów — uczenie się z empirycznych danych zamiast programowania reguł.
Problemy niealgorytmizowalne:

- zagadnienia o złożoności wykładniczej lub wyższej (np. problem komiwojażera)
- procesy z niejasnymi regułami lub silnym komponentem stochastycznym
- zjawiska trudne do opisania przez proste reguły
- środowiska zmienne, wymagające adaptacji algorytmu

Problemy efektywnie nieal gorytmizowalne

Problemy NP-trudne - złożoność obliczeniowa rośnie szybciej niż jakikolwiek wielomian liczby elementów

Przykład: problem komiwojażera



Rysunek 1.1. Najkrótsza trasa premiera przebiegająca przez wszystkie miasta wojewódzkie w podziale administracyjnym z 1975 roku (A. Adrabiński)

Tabela 1.1. Czasy znalezienia przez komputer, wykonujący 100 miliardów operacji na sekundę, najkrótszej trasy podróży premiera (zob. ćwiczenie 1.3)

Liczba województw	Czas znajdowania najkrótszej trasy
17	3.5 minuty
25	$2 \cdot 10^5$ lat
49	$4 \cdot 10^{42}$ lat

Dla 100 miejsc mamy 100! (liczba 161-cyfrowa)

Rysunek: M. Sysło, „Algorytmy”

Problemy niealgorytmizowalne - przykłady

- rozumienie sensu zdań,
- działania twórcze, decyzje intuicyjne,
- rozpoznawanie twarzy i obrazów,
- rozpoznawanie pisma ręcznego,
- rozpoznawanie mowy i sygnałów, percepcja,
- sterowanie robotem, nieliniowymi układami,
- diagnostyka medyczna, planowanie terapii.

CI i sztuczna inteligencja (AI)

Klasyczne rozumienie AI i CI:
CI - percepcja i sterowanie: zachowania sensomotoryczne – sieci neuronowe i uczenie maszynowe
AI - wyższe czynności poznawcze: logika, język, rozumowanie, planowanie

Good old-fashioned **Artificial Intelligence (GOFAI)** to część CI posługująca się symboliczną reprezentacją wiedzy, zajmuje się rozumowaniem, tworzeniem systemów ekspertowych.

CI i sztuczna inteligencja (AI)

Obecnie „sztuczna inteligencja” bywa używana do opisu szerokiego spektrum metod i algorytmów naśladujących inteligentne zachowania.

- **Strong AI, Artificial General Intelligence (AGI)** - agent świadomie działający i uczący sie działać w środowisku (podobnie do człowieka)
- **Weak AI** - systemy rozwiązujące (uczące sie rozwiązywać) konkretne zadania

Efekt AI (Larry Tesler)
„AI is whatever hasn’t been done yet”

Definicja AI jest elastyczna - granica między tym, co uważamy za AI, a tym, co jest już standardową technologią, ciągle się przesuwa.

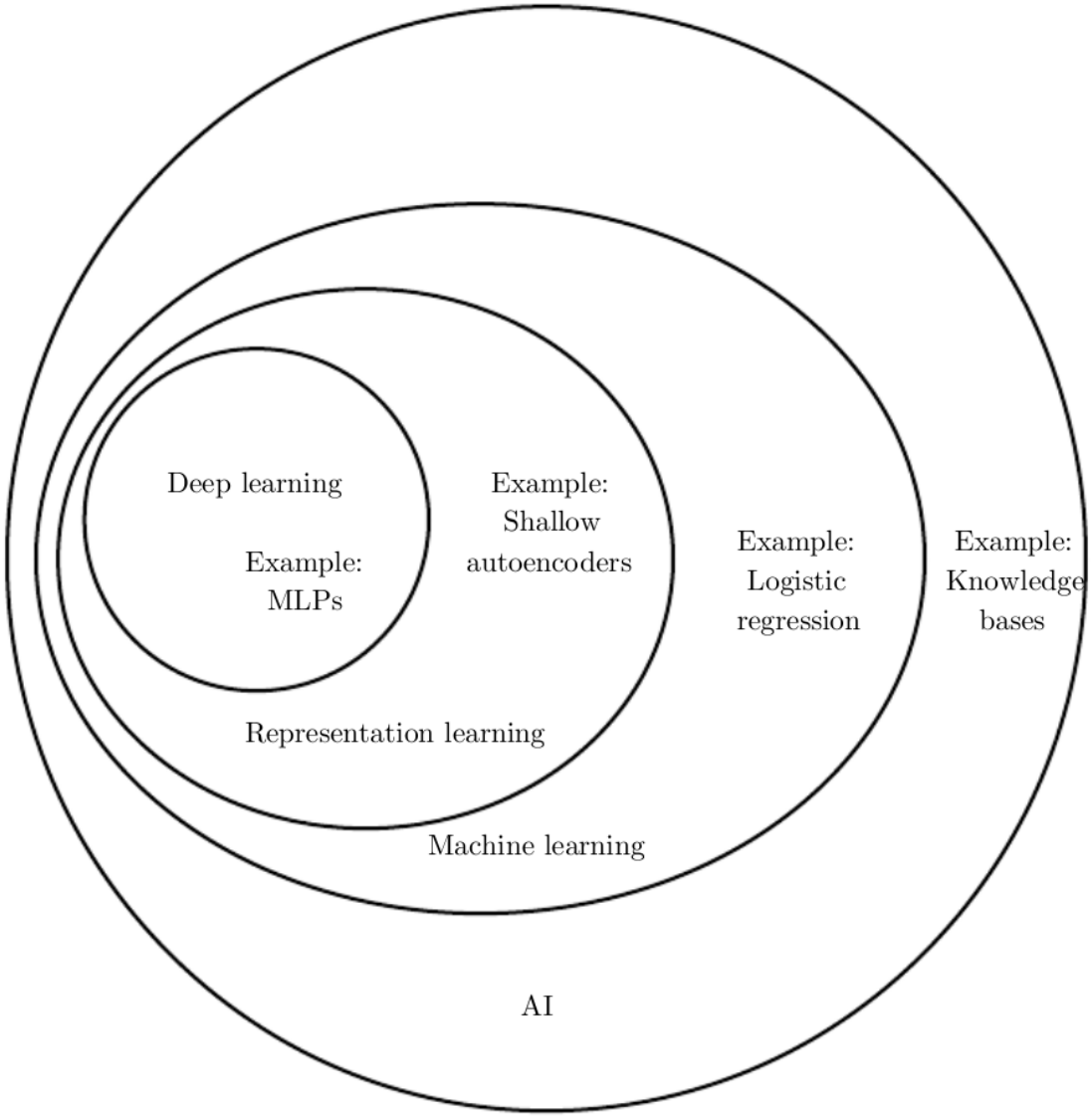
Uczenie maszynowe

- **Uczenie maszynowe** (Machine Learning, ML) - sytemy uczące się rozwiązywać problemy na podstawie przypadków uczących (danych)
- **Generalizacja** - dobrze wyuczony model działa poprawnie także dla przypadków spoza danych treningowych
- Gdy dane się zmienią - system można dostosować, trenując go na nowych danych



Rysunek: https://xkcd.com/

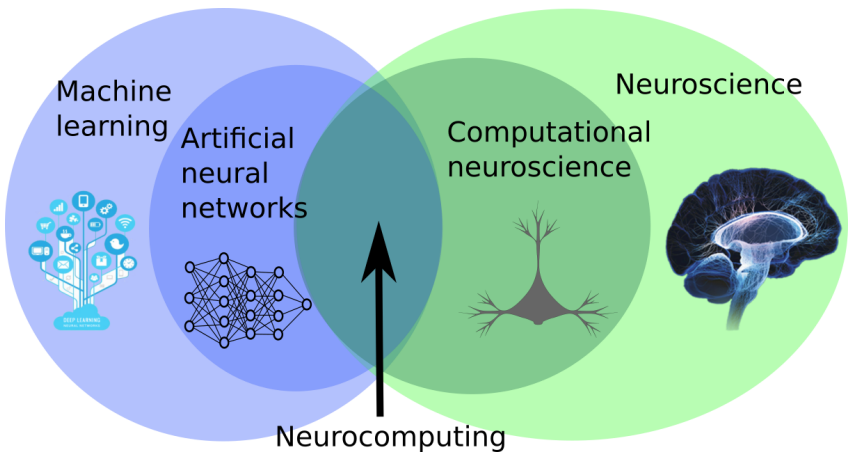
ANN i sztuczna inteligencja (AI)



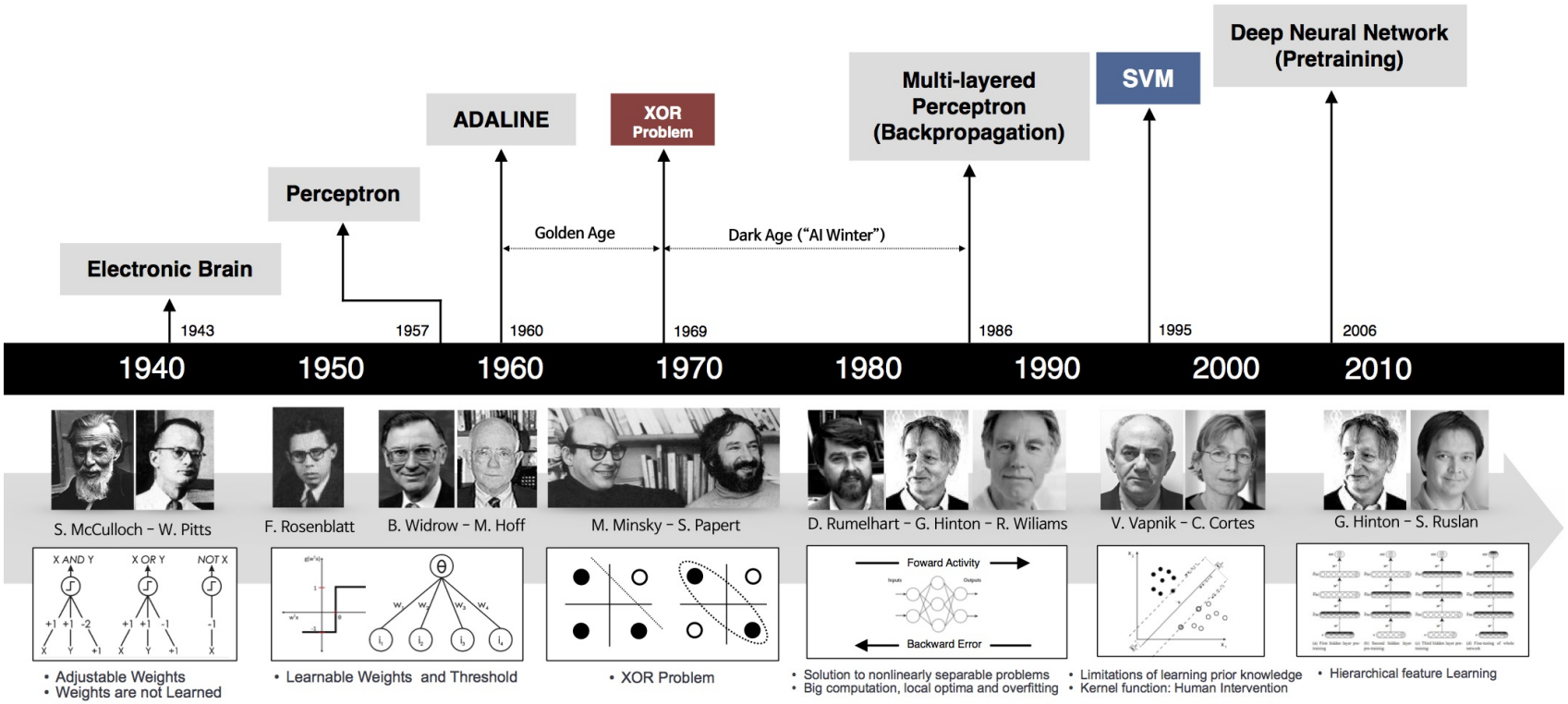
Goodfellow, 2016 [1]

Obliczenia neuronowe

- **modelowanie działania mózgu** - symulacje komputerowe, modele procesów kognitywnych
- przetwarzanie równoległe bardzo wydajne przy rozwiązywaniu niektórych problemów
- odporność na uszkodzenia, oddziaływania lokalne, uszkodzenie części sieci nie musi powodować drastycznych skutków
- rozwiązywanie praktycznych problemów metodami inspirowanymi biologicznie
  -> **Sztuczne Sieci Neuronowe (Artificial Neural Networks)**



Źródło: J. Vitay [7]



https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

Wczesny etap rozwoju (1943-1960)

1938 N. Rashevsky - neurodynamika, sieci neuronowe jako układ dynamiczny (pierwszy model sieci neuronowej)

1943 W. McCulloch, W. Pitts - sieci neuronowe jako układy logiczne

1949 D. Hebb - uczenie się na poziomie synaptycznym (pierwsza reguła uczenia)

1952 A. Hodgkin, A. Huxley - szczegółowy matematyczny model neuronu kałamarnicy olbrzymiej (nagroda Nobla 1963)

1958 F. Rosenblatt - Perceptron, sieć jako funkcja matematyczna
J. von Neuman, *The Computer and the Brain*

Pierwsze algorytmy uczące się (1960-1986)

1960 B. Widrow, M. Hoff - Adaline

1967 K. Steinbuch, E. Schmitt - macierze uczące się

1969 M. Minsky, S. Papert - książka *Perceptrony* i ograniczenia sieci liniowych

1973 Chr. von der Malsburg - samoorganizacja w układzie nerwowym

1982 J. Hopfield - model Hopfielda
T. Kohonen - mapy samoorganizujące się

1985 D. Ackley, G. Hinton, T. Sejnowski - maszyny Boltzmanna

1986 D. Rumelhart, G. Hinton, R. Williams - wsteczna propagacja błędów

1990 T. Poggio, F. Girosi - sieci RBF, teoria regularyzacji

1995 V. Vapnik - SVM i „koniec ery Data Mining"

Początki głębokiego uczenia (1980-2010)

1983 K. Fukushima, S. Miyake, T. Ito - Neokognitron, głęboka sieć neuronowa, inspiracja późniejszych sieci splotowych

1989 Yann LeCun - pierwsze praktyczne sieci splotowe (LeNet)

1997 S. Hochreiter, J. Schmidhuber - sieć rekurencyjna LSTM

1998 Y. LeCun, LeNet5, praktyczne zastosowanie CNN w analizie obrazów

2005 GPU do trenowania głęokich sieci, przyspieszenie rozwoju

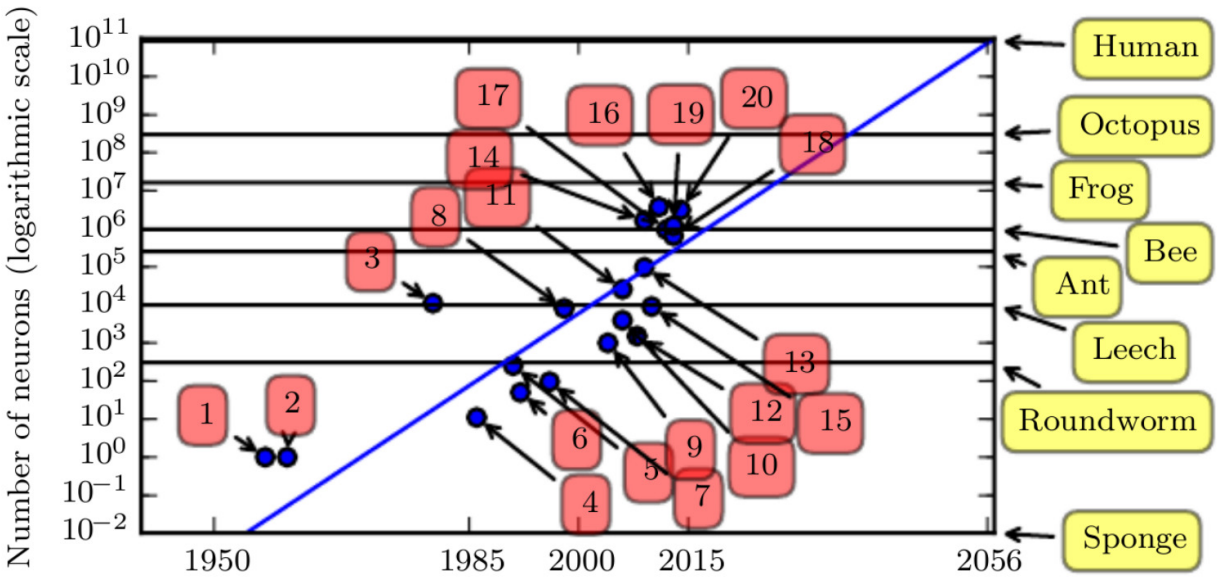
2006 Geoffrey Hinton - Deep Belief Networks (DBN) i rozwój pre-treningu

Współczesny rozwój głębokiego uczenia (2010-)

- 2012 Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton - AlexNet: znaczący spadek błędu w ImageNet (ok. 15%, wcześniej 30%), kluczowe zastosowanie GPU, bez pre-treningu
- 2013 Kingma, Welling - variational autoencoder (VAE)
- 2014 Ian Goodfellow - generative adversarial networks (GAN)
- 2015 DeepMind - Deep Q-Network (DQN), rozwój deep RL
- 2016 AlphaGo (DeepMind) - przełom w grach strategicznych, model pokonuje profesjonalnego gracza (Lee Sedol) w Go
- 2017 Ashish Vaswani (Google Brain) - transformery i rewolucja w przetwarzaniu języka naturalnego
- 2018 AlphaFold (DeepMind) - przewidywanie struktury białek
- 2020 GPT-3 (OpenAI)
- 2021 OpenAI, generator obrazów DALLE,
- 2022 Midjourney, Stable Diffusion, BARD/Gemini (Google AI)
- 2023 Llama (Meta), Grok (xAI)
- 2024 Sora (OpenAI) generowanie wideo, ...

☞ *The Evolution of Deep Learning: A Comprehensive Timeline*

Ilość neuronów w modelach neuronowych



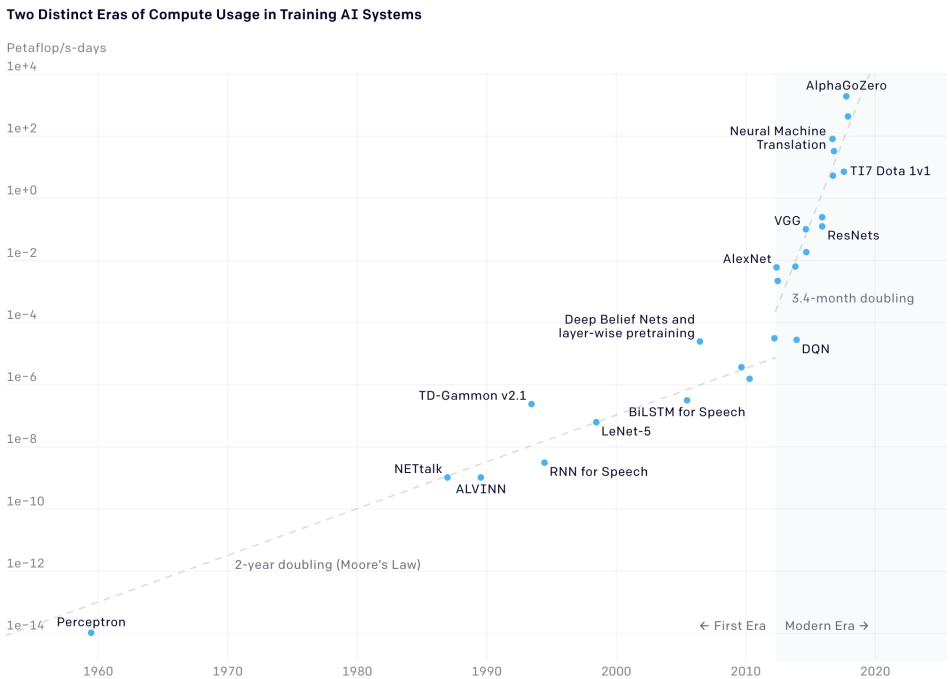
1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart *et al.*, 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallsid
6. Multilayer perceptron for speech recognition (Bengio *et al.*, 1991)
7. Mean field sigmoid belief network (Saul *et al.*, 1996)
8. LeNet-5 (LeCun *et al.*, 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton *et al.*, 2006)
11. GPU-accelerated convolutional network (Chellapilla *et al.*, 2006)
12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina *et al.*, 2009)
14. Unsupervised convolutional network (Jarrett *et al.*, 2009)
15. GPU-accelerated multilayer perceptron (Ciresan *et al.*, 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le *et al.*, 2012)
18. Multi-GPU convolutional network (Krizhevsky *et al.*, 2012)
19. COTS HPC unsupervised convolutional network (Coates *et al.*, 2013)
20. GoogLeNet (Szegedy *et al.*, 2014a)

Prawo Moore'a dla sieci neuronowych:
liczba neuronów (paramtrów) podwaja się co 2.5 roku

Goodfellow, 2016 [1]

Era głębokiego uczenia

- Modern DL era: od 2012 roku, Alexnet, dostępność wydajnego sprzętu (GPU, TPU) umożliwia trenowanie modeli z milionami parametrów
- zmiana prawa Moore'a: podwojenie wydajności co 3.4 miesiąca

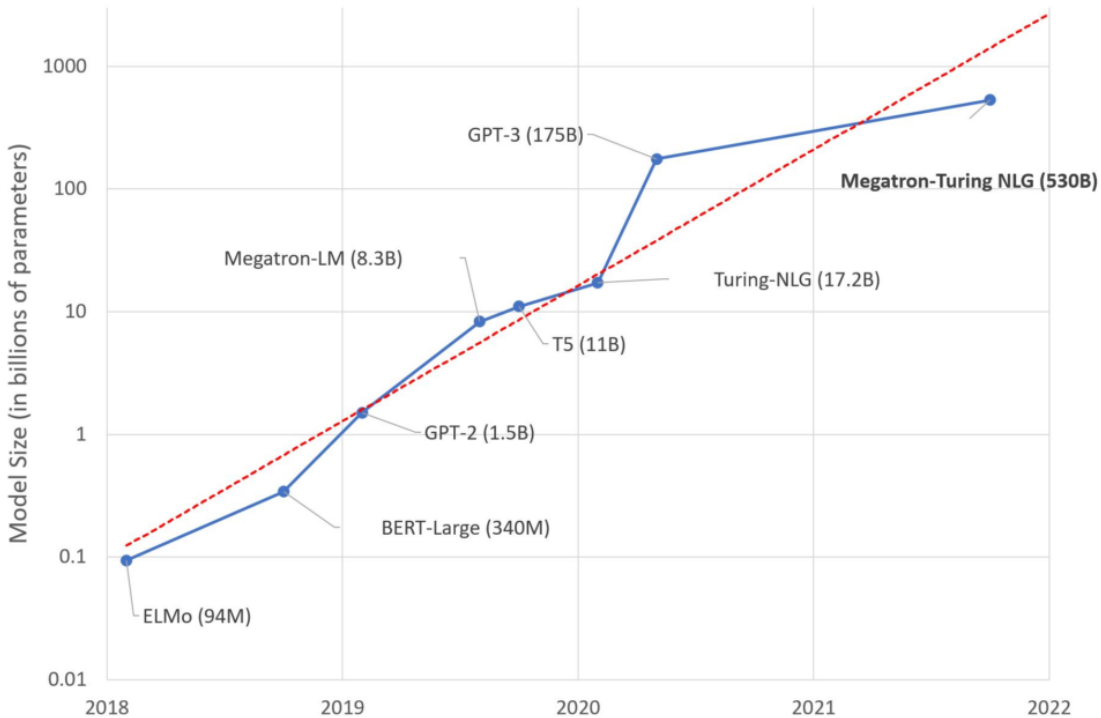


☞ *Computation used to train notable artificial intelligence systems*

☞ *AI and Scaling the Compute for the new Moore's Law*

Era dużej skali

- Large scale era: od 2016 roku, pojawienie się AlphaGo, masowe zrównoleglenie obliczeń, trenowanie na bardzo dużych zbiorach danych, algorytmy wyszukiwania architektur (NAS), expert iteration (ExIt) i inne techniki skalowania



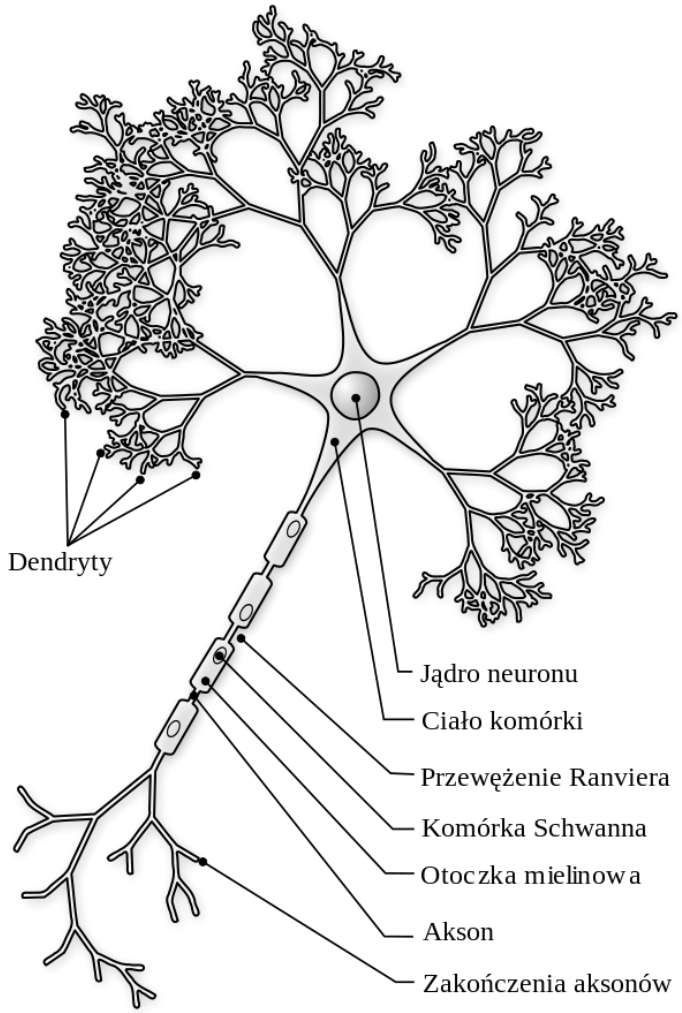
☞ *Computation used to train notable artificial intelligence systems*

☞ *AI and Scaling the Compute for the new Moore's Law*

Mózg i sieć neuronowa

- około 10^{11} neuronów (100 miliardów), do 100 Hz
- gęstość połączeń rzędu 10^4 synaps na neuron
- Receptory - dostarczają sygnały wejściowe (bodźce wewnętrzne i sygnał ze zmysłów)
- Sygnał wyjściowy z układu nerwowego steruje efektorami (reakcja na bodziec)
- obliczenia równoległe
- układy biologiczne są zbyt złożone do bezpośredniego modelowania - jedynie ogólne zasady organizacji i sposobu funkcjonowania układów biologicznych neuronów mogą służyć za inspirację
- sztuczne sieci neuronowe to uproszczone modele matematyczne, które można widzieć jako złożenie wielu (nieliniowych) funkcji i często nie mają nic wspólnego z sieciami biologicznymi

Neuron biologiczny



Rysunek: wikipedia.org

Neuron biologiczny

- **Dendryty** - wejście neuronu, odbierają sygnał od sąsiadujących neuronów, impulsy mogą być pobudzające lub hamujące
- **Ciało komórki (soma)** - akumuluje napływające w danym momencie sygnały
- **Akson** (wyjście) - emituje sygnał, gdy potencjał somy osiągnie próg
- **Synapsy** - styki pomiędzy dendrytami i aksonami; pobudzenie z aksonu powoduje uwolnienie chemicznych neurotransmiterów, które wpływają na zachowanie połączeń neuronów
- Działanie neuronów biologicznych jest bardzo złożonym procesem; neurony w sztucznych sieciach są dużym uproszczeniem

Logika progowa neuronu

Neuron zbiera dochodzące do niego sygnały x_i obliczając sumę ważoną tych sygnałów (wagi w_i określane są przez procesy synaptyczne), tworząc z nich sumaryczny sygnał wejściowy:

$$I(t) = \sum_i w_i x_i(t)$$

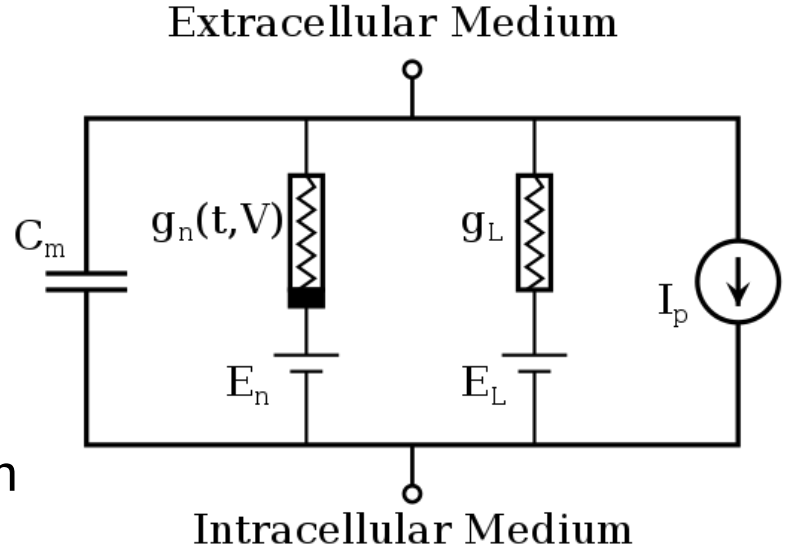
Sumowanie przestrzenne i czasowe potencjałów zgromadzonych na błonie komórkowej daje całkowitą aktywację elementu:

$$a(t) = F\big(I(t), a(t-1)\big)$$

Aktywacja, razem z progiem wzbudzenia T, określa sygnał wyjściowy

$$o(t) = f(a(t), T)$$

Model Hodgkin–Huxley (1952)



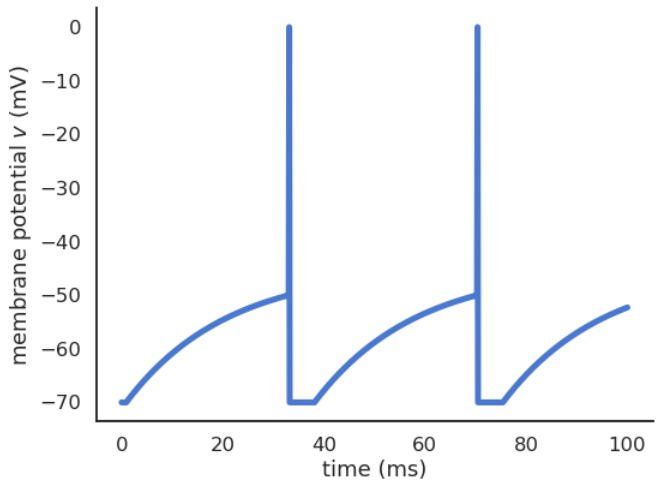
- pierwszy szczegółowy model matematyczny neuronu oparty na przewodnictwie jonowym
- nieliniowe równania różniczkowe opisujące zmianę potencjału błonowego $V(t)$, z uwzględnieniem prądów jonowych sodu (Na), potasu (K) i prądu upływu (L) oraz prądu zewnętrznego $I(t)$.

$$C_m \frac{\mathrm{d}V(t)}{\mathrm{d}t} = -g_K\big(V(t) - V_K\big) - g_{Na}\big(V(t) - V_{Na}\big) - g_L\big(V(t) - V_L\big) - I(t)$$

https://en.wikipedia.org/wiki/Hodgkin-Huxley_model

Neurony impulsujące (*spiking neurons*)

Sumuj i odpal: *Leaky integrate-and-fire* (LIF; Lapicque, 1907) - model generujący impuls, gdy sumaryczny potencjał membrany przekroczy próg



$$C_m \frac{\mathrm{d}V(t)}{\mathrm{d}t} = -g_L\big(V(t) - V_L\big) + I(t)$$

Gdy $V > V_T$ (próg wzbudzenia), neuron emituje impuls i następnie przechodzi w okres refrakcji (odpoczynku)

Źródło: Vitay, Neurocomputing [7]

Neurony kodowane częstością wzbudzeń

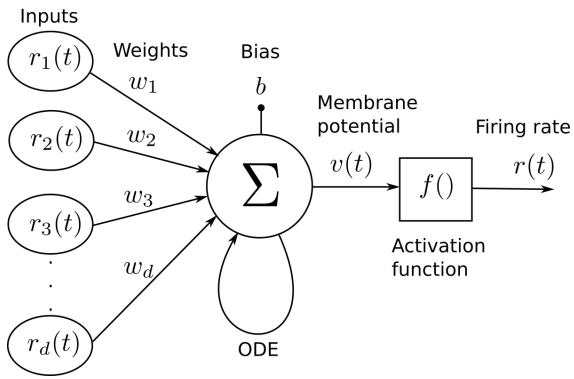
Rate-coded neurons

- potencjał membrany $V(t)$

$$\tau \frac{\mathrm{d}V(t)}{\mathrm{d}t} + V(t) = \sum_{i=1}^{d} w_{i,j} r_i(t) + b$$

- częstość wzbudzeń (*firing rate*) $r(t)$ odwzorowuje potencjał membrany w pojedynczą ciągłą wartość za pomocą **funkcji aktywacji**

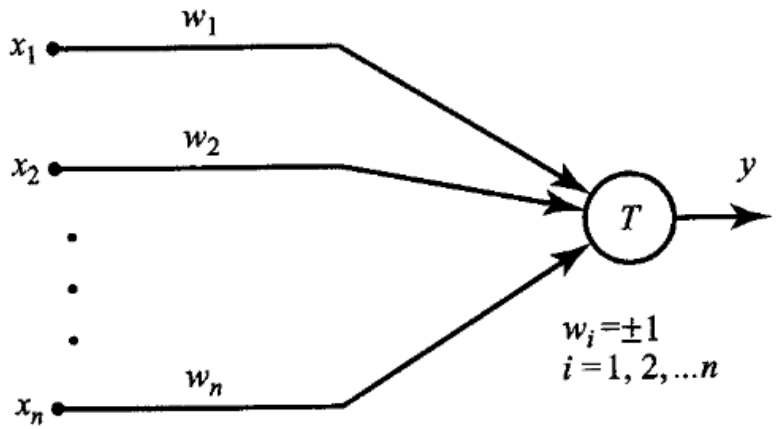
$$r(t) = f(V(t))$$



- nie symuluje pojedynczych impulsów (jak np. LIF), ale skupia się na średniej aktywności

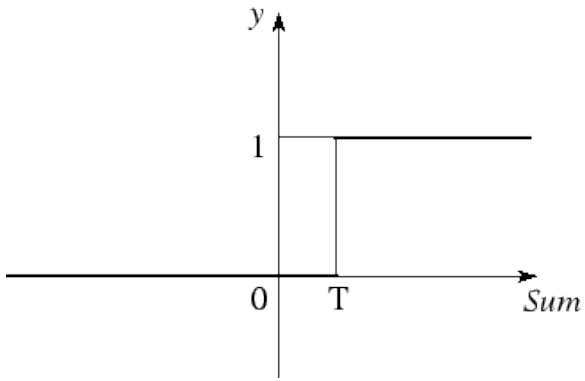
Źródło: Vitay, Neurocomputing [7]

Neuron McCulloch, Pitts (1943-49)



stan wyjściowy neuronu

$$o(\mathbf{x}) = \begin{cases} 1 & \text{gdy} \quad \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{gdy} \quad \sum_{i=1}^{n} w_i x_i < T \end{cases}$$



grafika: www.wikipedia.org

Neuron McCulloch, Pitts (1943-49)

- neurony mogą być tylko w dwóch stanach, aktywne albo nieaktywne, sygnał wejściowy i wyjściowy binarny

$$x_i \in \{0, 1\} \qquad o(\mathbf{x}) \in \{0, 1\}$$

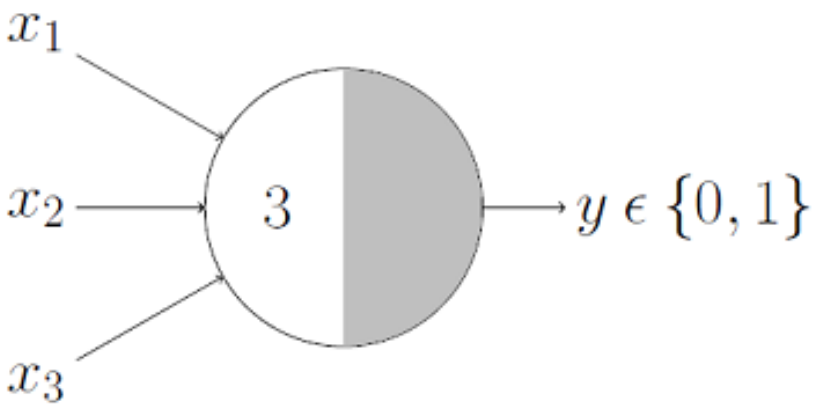
- sygnały dochodzą przez synapsy pobudzające i hamujące

$$w_{ij} \in \{-1, +1\}$$

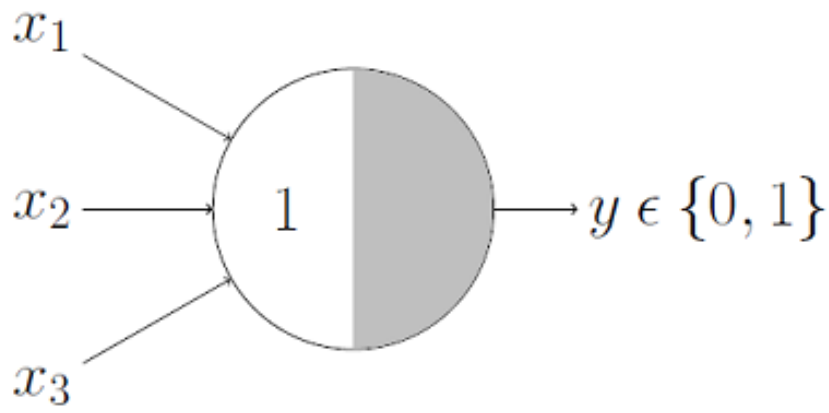
- każdy neuron ma ustalony próg pobudzenia $T > 0$
- sygnały sumują się w pewnym kwancie czasu i gdy przekroczą próg T to neuron się aktywuje
- brak uczenia się - wagi i próg mają stałą wartość
- odpowiedni dobór wag pozwala zrealizować bramki logiczne: NOT, OR, AND bądź NOR i NAND $\rightarrow$ sieć neuronów może implementować dowolną funkcję logiczną

Realizacja funkcji logicznych

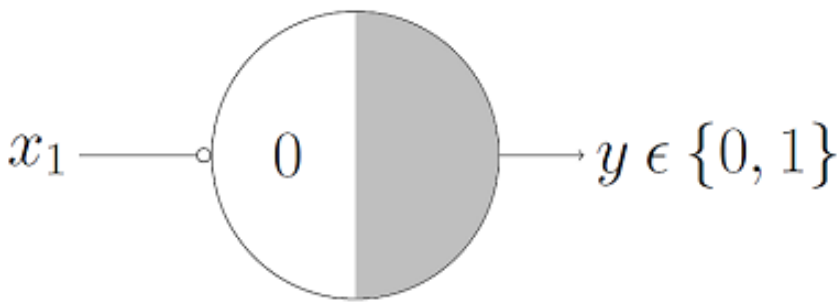
AND



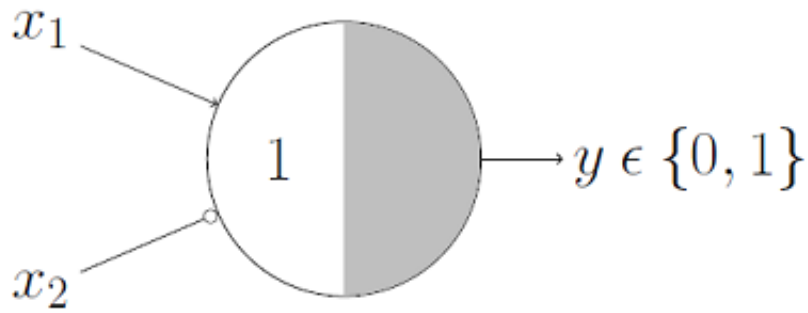
OR



NOT



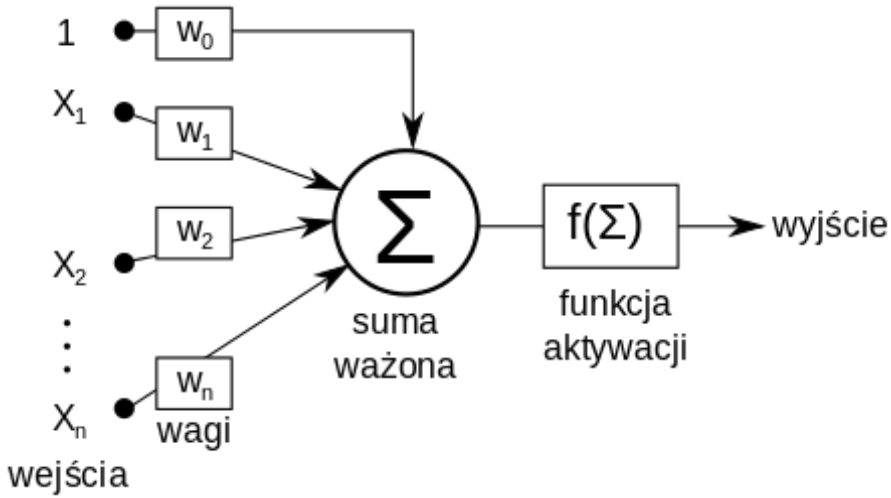
hamowanie sygnału wejściem x_2



$x_1\ AND\ !x_2$*

Źródło: Akshay L. Chandra, ☞ McCulloch-Pitts Neuron

Ogólny schemat neuronu



$$o(\mathbf{x}) = f\left(\sum_{i=1}^{n} w_i x_i + w_0\right) = f(\mathbf{w}^\top \mathbf{x})$$

gdzie

$\mathbf{x} = [1, x_1, x_2, \ldots, x_n]^\top$ wielowymiarowy sygnał wejściowy

$\mathbf{x} \in \mathbb{R}^{n+1}$

$\mathbf{w} = [w_0, w_1, w_2, \ldots, w_n]^\top$ wagi połączeń $\mathbf{w} \in \mathbb{R}^{n+1}$

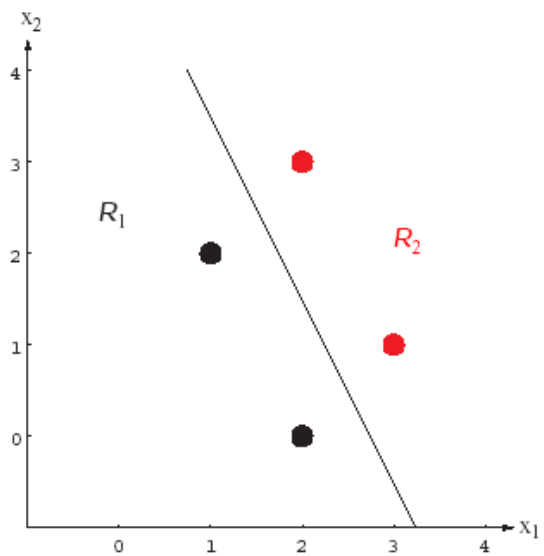
w_0 - wyraz wolny (bias), stały sygnał wejściowy $1 \cdot w_0$

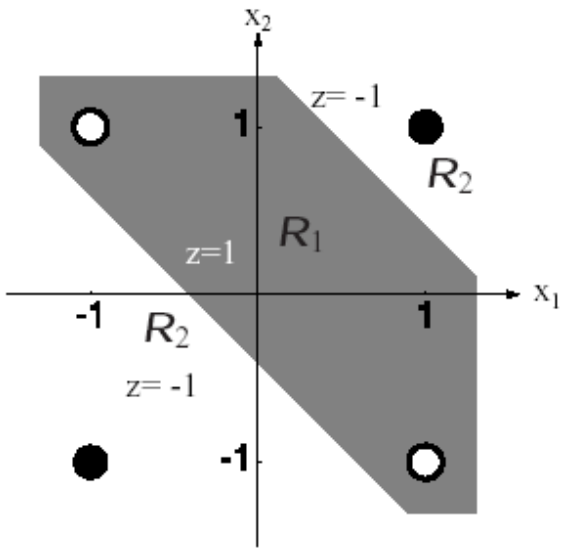
Interpretacja geometryczna - separowalność

Neuron progowy realizuje separowalność liniową: hiperpłaszczyzna dzieli przestrzeń wejściową $\mathbb{R}^n$ na dwie części

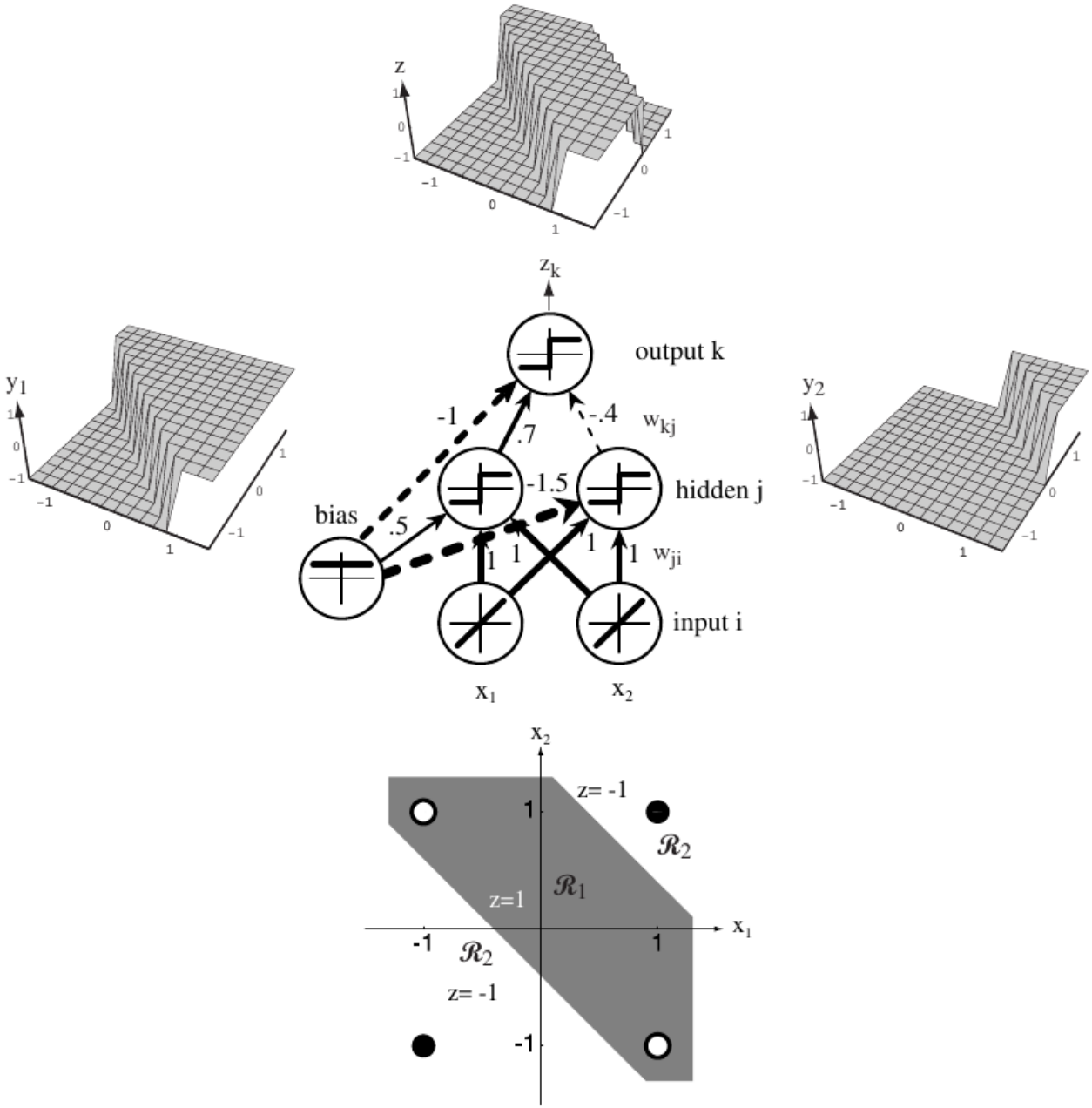
Przykład neuronu z 2 wejściami

$$o(\mathbf{x}) = \begin{cases} 1 & \text{gdy} \quad w_1 x_1 + w_2 x_2 \geq \theta \\ 0 & \text{gdy} \quad w_1 x_1 + w_2 x_2 < \theta \end{cases}$$





XOR


z
y1
y2
zk
output k
-1
.7
-.4
wkj
-1.5
hidden j
bias
.5
1
1
1
1
wji
input i
x1
x2
x2
z= -1
1
R2
R1
z=1
x1
-1
1
R2
z= -1
-1

Reguła Hebba (1949)

$$\Delta w_i = \eta \cdot o(\mathbf{x}) \cdot x_i = \eta f(\mathbf{w}^\top \mathbf{x}) x_i$$

„Kiedy akson komórki A jest dostatecznie blisko by pobudzić komórkę B i wielokrotnie w sposób trwały bierze udział w jej pobudzaniu, procesy wzrostu lub zmian metabolicznych zachodzą w obu komórkach tak, że sprawność neuronu A jako jednej z komórek pobudzających B, wzrasta."

,,neurons that fire together wire together"

Plastyczność układu (uczenie)

Uczenie się: zmiana wag synaptycznych w_i w czasie

Reguła Hebba

$$\Delta w_i = \eta o(\mathbf{x}) x_i = \eta f(\mathbf{w}^T \mathbf{x}) x_i$$

Reguła delta (Widrowa-Hoffa)

$$\Delta w_i = \eta (y - o(\mathbf{x})) x_i$$

gdzie y to spodziewana odpowiedź dla sygnału wejściowego $\mathbf{x}$

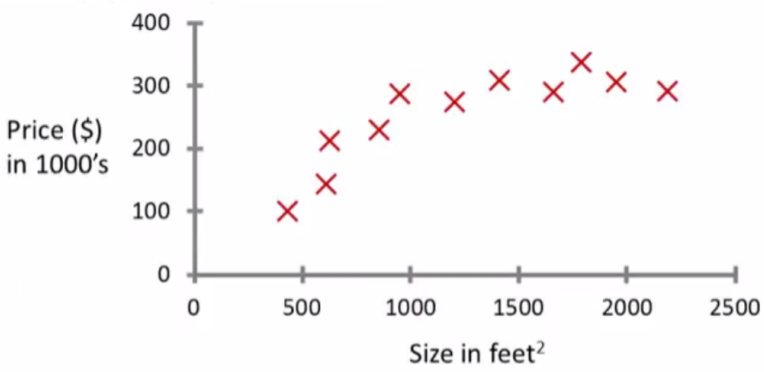
Rodzaje ucznia

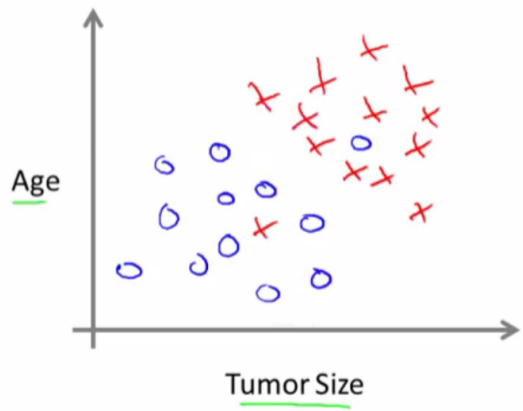
- **Uczenie bez nadzoru** (*unsupervised learning*)
  uczenie wyłącznie na podstawie sygnału wejściowego **X** (bez etykiet **y**), odkrywanie struktur i zależności w danych.
  Uczenie spontaniczne, odpowiada uczeniu w okresie niemowlęcym.
- **Uczenie nadzorowane** (*supervised learning*) - dla każdego sygnału wejściowego **x** dana jest pożądana odpowiedź **y**.
  Uczenie „szkolne".
- **Uczenie z krytykiem**, ze „wzmocnieniem" (*reinforcement learning*) - uczenie się zachowań, które przynoszą zysk po dłuższym czasie (np. autonomiczne roboty, gra z przeciwnikiem).
  Uczenie dojrzałe (nabieranie „mądrości").

Uczenie nadzorowane

Każdy przypadek **x** ze zbioru treningowego posiada przypisaną wartość **y**

- **Regresja (aproksymacja)** - obiekt wyjściowy **y** jest liczbą rzeczywistą lub wektorem liczb (np. temperatura)
- **Klasyfikacja** - obiekt wyjściowy **y** jest etykietą klasy (np. nazwa obiektu na zdjęciu)





Rysunek: Mohamad Ghassany, http://www.mghassany.com/MLcourse/introduction.html

Uczenie nadzorowane

- Model

$$f(\mathbf{x}; \mathbf{W}) = \mathbf{y}$$

  realizuje mapowanie zdefiniowane przez parametry sieci $\mathbf{W}$ (wagi połączeń) wektora wejściowego $\mathbf{x}$ do wektora wyjściowego $\mathbf{y}$ (pożądanego wyjścia)
- Uczenie najczęściej polega na takim doborze parametrów $\mathbf{W}$ aby zminimalizować różnice pomiędzy wyjściem sieci $f(\mathbf{x})$ a wartością pożądaną $\mathbf{y}$, np. błąd kwadratowy:

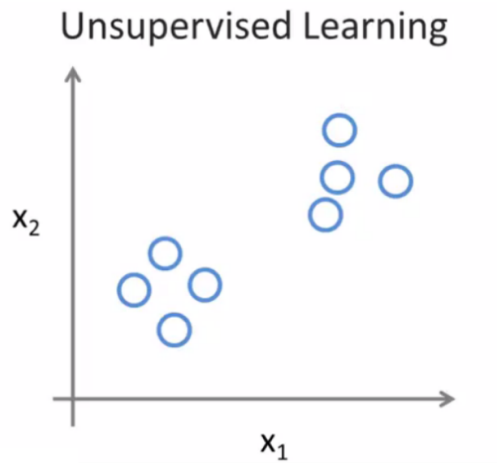
$$\|f(\mathbf{x}; \mathbf{W}) - \mathbf{y}\|^2$$

- Celem uczenia nadzorowanego nie jest uczenie „na pamięć”, lecz **generalizacja**.

Uczenie nienadzorowane

Uczenie wyłącznie na podstawie sygnału wejściowego **X**, brak pożądanego sygnału wyjściowego. Przykłady:

- **klasteryzacja** (analiza skupień), automatyczne wykrywanie grup o podobnych właściwościach

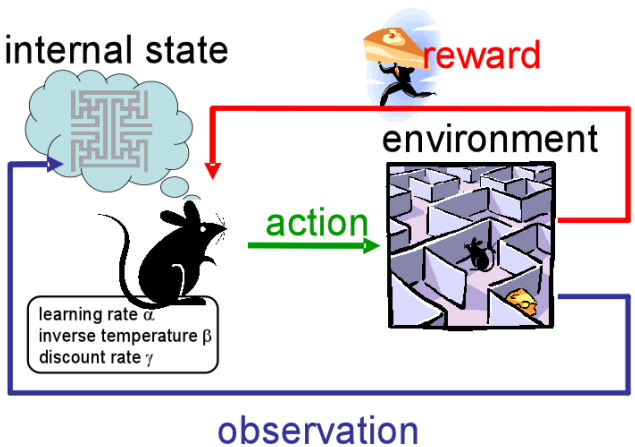


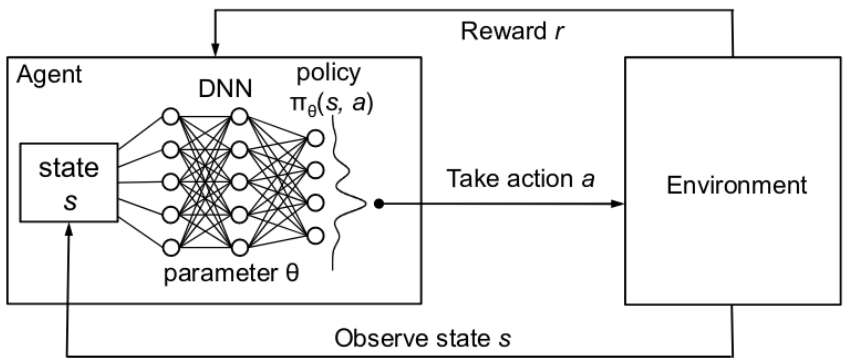
- wykrywanie cech i regularności, tworzenie przydatnych reprezentacji danych, kompresja (kodowanie) sygnału, redukcja wymiarowości, modelowanie sygnału, redukcja szumu

Rys.: Mohamad Ghassany, http://www.mghassany.com/MLcourse/introduction.html

Uczenie z krytykiem

- Sygnał wyjściowy to zazwyczaj akcja (sekwencja akcji) podejmowana przez agenta
- Nagroda (lub kara) jest dostarczana z opóźnieniem (np. przegrana/wygrana partia w szachy)

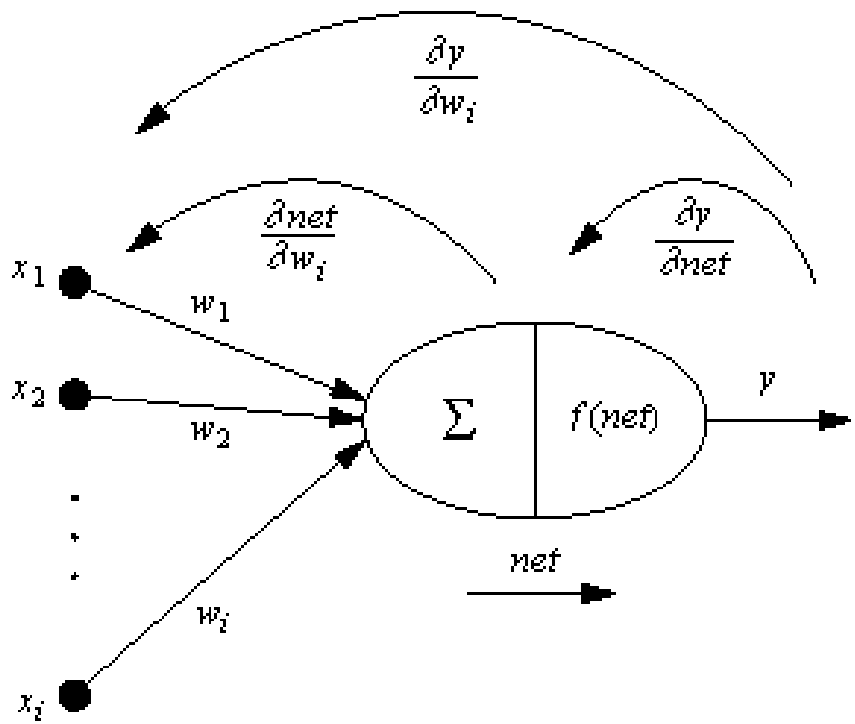




Rysunek: Hongzi Mao, „Resource Management with Deep Reinforcement Learning"
Aneek Das, The very basics of Reinforcement Learning

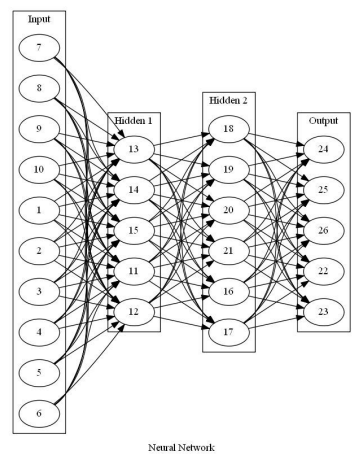
Główne aspekty modeli neuronowych I

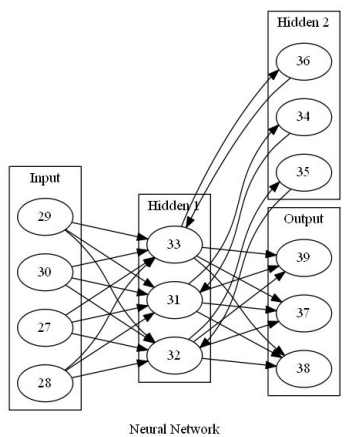
1. Sposób modelowania pojedynczego neuronu
   - stan aktywacji (wzbudzenia) poszczególnych neuronów, sposób aktywacji tych neuronów, funkcja opisująca sygnał wyjściowy elementu
   - neurony: progowe, liniowe, nieliniowe (np. sigmoidalne), ...

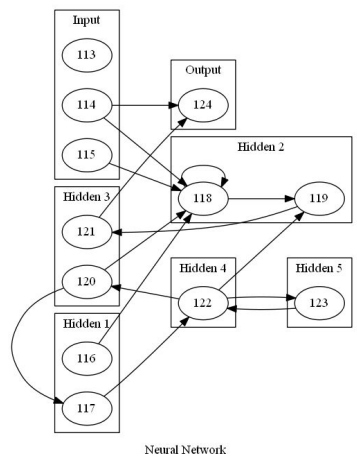


Główne aspekty modeli neuronowych II

2. Sposób propagacji sygnałów przez sieć neuronową
   - sieci jednokierunkowe (feedforward)
   - sieci ze sprzężeniami zwrotnymi, sieci rekurencyjne (recurrent), sieci dynamiczne
3. Topologia połączeń elementów (architektura sieci):
   - budowa warstwowa (warstwy wejściowe, ukryte, wyjściowe), struktura hierarchiczna
   - sieci jednowarstwowe, wielowarstwowe, płytkie, głębokie
   - warstwy w pełni połączone, splotowe, rzadkie połączenia, współdzielenie połączeń
   - sieci o stałej architekturze, sieci ontogeniczne (zmieniające strukturę), rozrastające się, kurczące się,







Główne aspekty modeli neuronowych III

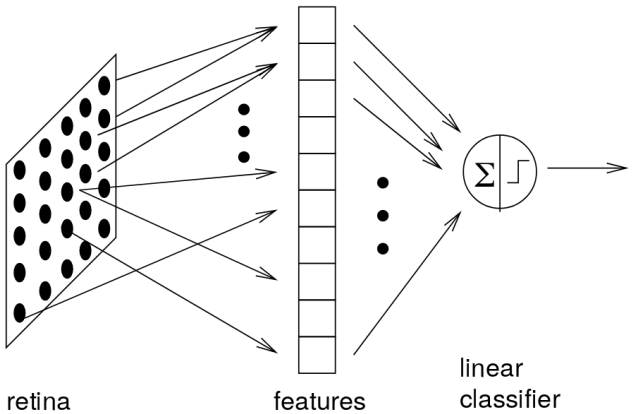
4. Uczenie sieci
   - reguły uczenia, reguły modyfikacji parametrów opisujących neurony i połączeń pomiędzy nimi
   - algorytm optymalizacji, funkcja kosztu
   - regularyzacja
5. Otoczenie, w którym działa sieć neuronowa: reprezentacja danych, sposób prezentacji danych, jakość danych, dostępność danych, sposób oceny jakości
6. Realizacja techniczna: język programowania, wykorzystanie GPU, zrównoleglenie obliczeń

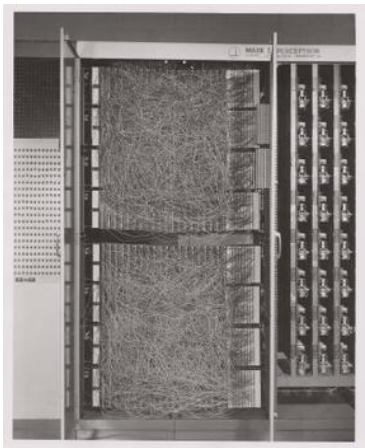
Perceptron

Reguły uczenia

Perceptron Rosenblatta (1958)

Klasyfikator neuronowy Mark I do rozpoznawania znaków alfanumerycznych, wzorowany na biologicznej percepcji (układ wzrokowy)





Trzy warstwy:

- **S-units:** wejście, siatkówka oka, 400 fotokomórek (20x20)
- **A-units:** asocjacyjne, zbierające dane z większych obszarów (ekstrakcja cech), 512 jednostek
- **R-units:** liniowy klasyfikator uczący, 8 wyjść

Rys: wikipedia.org
Martin Riedmiller, Machine Learning (lectures)

- $S_i = \{-1, +1\}$ sygnał docierający do elementów sensorycznych
- połączenia $c_{ij} = \{-1, 0, +1\}$ elementów S_j i A_i, przypadkowo rozrzucone w pewnym obszarze, nie ulegają zmianom, realizują wstępne przetwarzanie

$$A_i = \begin{cases} +1, & \text{dla} \quad \sum_j c_{ij} S_j \geq \Theta_i \\ -1, & \text{dla} \quad \sum_j c_{ij} S_j < \Theta_i \end{cases}$$

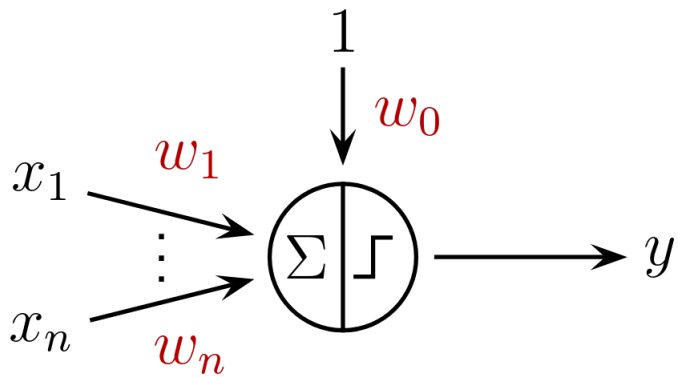
- sygnał wyjściowy

$$R_i = \begin{cases} +1 & \sum_j w_{ij} A_j > N\kappa \\ -1 & \sum_j w_{ij} A_j < -N\kappa \\ 0 & \text{w pozostałych przypadkach} \end{cases}$$

- wagi w_{ij} realizowane przez potencjometry
- proces uczenia - mechaniczna regulacja wartości potencjometrów za pomocą silników elektrycznych

Perceptron współczesny

- Obecnie pod pojęciem „perceptronu” rozumie się pojedynczy neuron z wieloma wejściami (bez jednostek S).
- **Perceptron prosty**



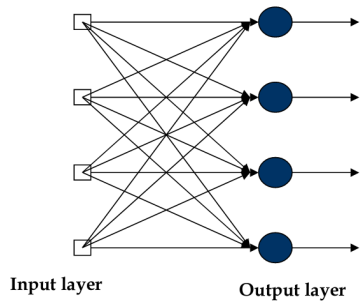
$$y = f\left(\sum_i w_i x_i + w_0\right)$$

Rys: Martin Riedmiller, Machine Learning (lectures)

Perceptron współczesny

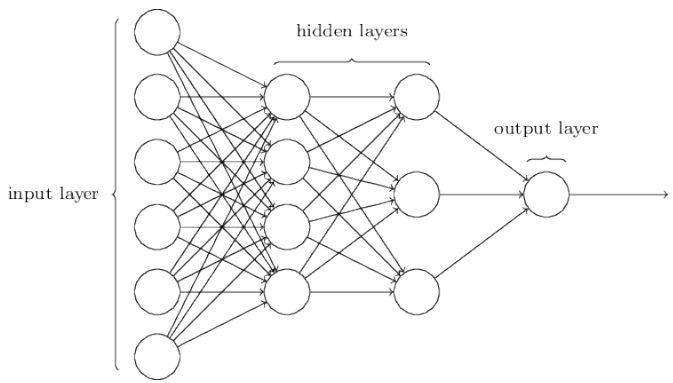
Single layer perceptron

jednowarstwowa sieć jednokierunkowa w pełni połączona



Multilayer perceptron (MLP)

wielowarstwowa sieć jednokierunkowa w pełni połączona

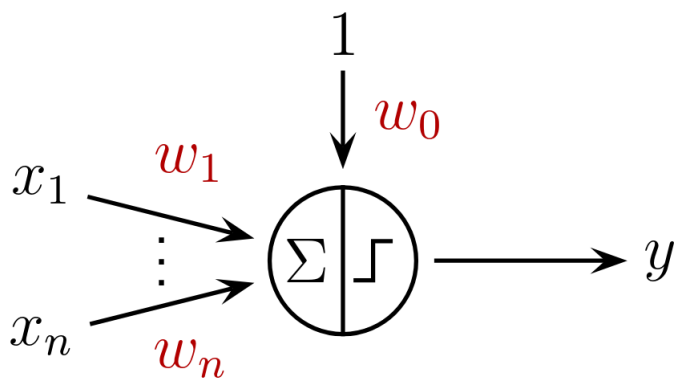


Rys: M. Bennamoun, Neural Computation (lecture)
computersciencewiki.org

Neuron binarny

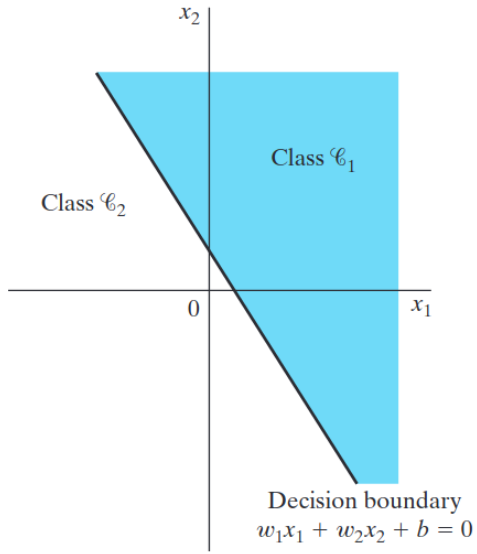
Perceptron prosty (binarny)

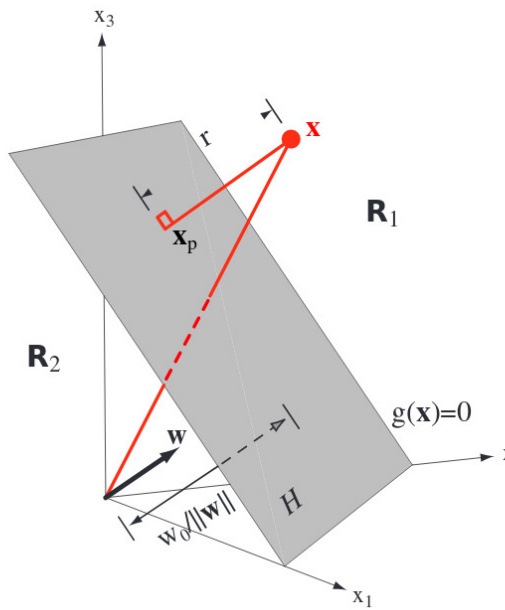
$$y = \begin{cases} 1, & \text{gdy} \quad \sum_i w_i x_i + w_0 \geq 0 \\ 0, & \text{gdy} \quad \sum_i w_i x_i + w_0 < 0 \end{cases}$$



Interpretacja geometryczna

- $x_1, \ldots, x_n$ to współrzędne punktu w n-wymiarowej przestrzeni ($\mathbf{x} \in \mathbb{R}^n$)
- równanie $\sum x_i w_i + w_0 = 0$ definiuje płaszczyznę (hiperpłaszczyznę) rozdzielającą przestrzeń wejściową na dwie części R_1 i R_2
- wagi **w** tworzą **wektor normalny** prostopadły do hiperpłaszczyzny decyzyjnej, w_0 to przesunięcie
- perceptron prosty dzieli zbiór wektorów $\mathbf{x}_i$ na dwa podzbiory: te leżące poniżej płaszczyzny decyzyjnej ($y(\mathbf{x}_i) = 0$) oraz leżące powyżej płaszczyzny ($y(\mathbf{x}_i) = 1$)





Rys: Duda and Hart, Pattern Classification

Zadanie klasyfikacji binarnej

- **Klasyfikacja binarna** - przypisanie obiektów do jednej z 2 klas
- **Dane treningowe**: zbiór n przypadków $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, każdy przypisany do jednego z dwóch zbiorów $\mathbb{P}$ lub $\mathbb{N}$
- **Uczenie**: procedura doboru wag $\mathbf{w}$ i wartości progowej w_0 tak aby perceptron zwracał wartość 1 dla wszystkich $\mathbf{x}_i$ ze zbioru $\mathbb{P}$, zaś wartość 0 dla wszystkich $\mathbf{x}_i \in \mathbb{N}$
- założenie: dane są spójne, tzn. $\mathbb{P} \cap \mathbb{N} = \emptyset$

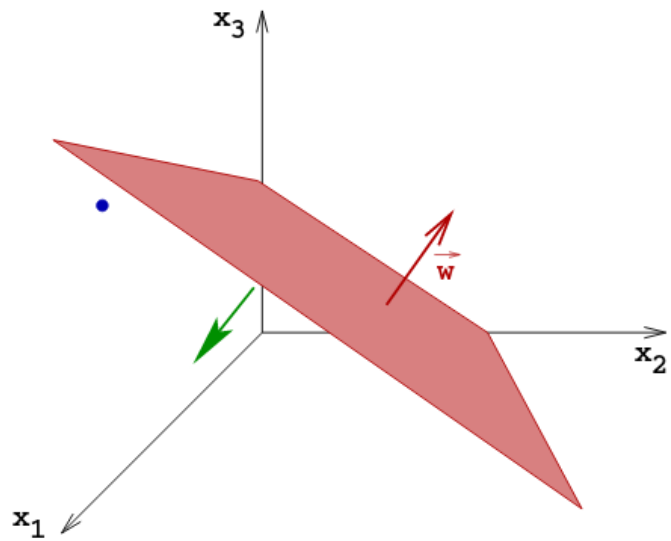
Uczenie perceptronu - idea

Niech **x** należy do zbioru $\mathbb{P}$. Jeżeli perceptron popełnia błąd to

$$\sum w_i x_i + w_0 < 0$$

Jak zmienić **w** i w_0 aby zniwelować błąd?

- zwiększyć w_0



przesunięcie płaszczyzny ze zwiększeniem w_0

Rys: Riedmiller, Machine Learning (lectures)

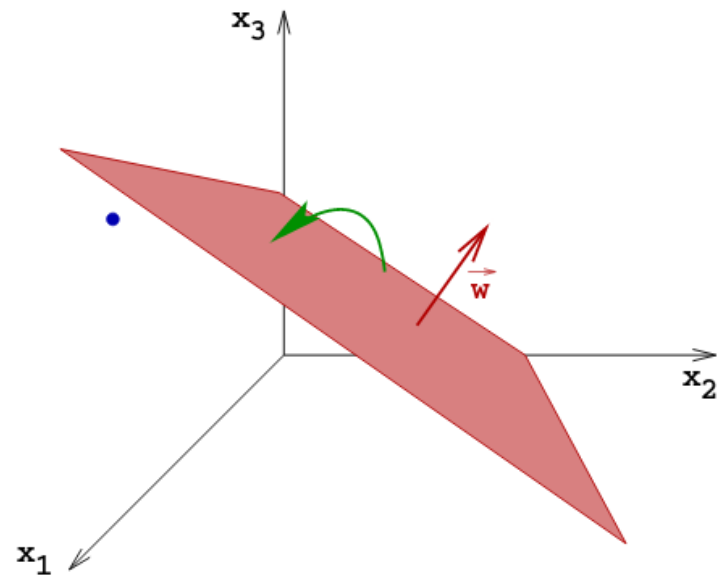
Uczenie perceptronu - idea

Niech **x** należy do zbioru $\mathbb{P}$. Jeżeli perceptron popełnia błąd to

$$\sum w_i x_i + w_0 < 0$$

Jak zmienić **w** i w_0 aby zniwelować błąd?

- zwiększyć w_0
- jeśli $x_i > 0$ to zwiększyć w_i
- jeżeli $x_i < 0$ to zmniejszyć w_i



obrót płaszczyzny ze zmianą **w**

Rys: Riedmiller, Machine Learning (lectures)

Algorytm uczenia perceptronu

Algorithm Algorytm uczenia perceptronu

Input: zbiór wektorów należących do jednego z dwóch zbiorów $\mathbb{P}$ i $\mathbb{N}$

Output: perceptron klasyfikujący wszystkie przypadki (jeżeli istnieje)

1: zainicjuj losowo wagi $\mathbf{w}$ oraz wartość progową w_0
2: **while** istnieje błędnie klasyfikowany $\mathbf{x}$ **do**
3: **if** $\mathbf{x} \in \mathbb{P}$ **then**
4: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}$
5: $w_0 \leftarrow w_0 + 1$
6: **else**
7: $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}$
8: $w_0 \leftarrow w_0 - 1$
9: **return** $\mathbf{w}$, w_0

Kodując etykiety klas $y = \{-1, +1\}$ reguła uczenia sprowadza się do $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$

☞ Perceptron visualization

Algorytm uczenia perceptronu

- Twierdzenie o zbieżności perceptronu: jeżeli zbiór wektorów $\mathbf{x}_i$ jest **liniowo separowalny** to algorytm uczenia perceptronu odnajdzie rozwiązanie w skończonej liczbie kroków
- Rozwiązanie nie jest unikatowe, kolejne uruchomienie algorytmu może znaleźć inne rozwiązanie (rezultat inicjalizacji wag i kolejności prezentacji danych treningowych)
- Możliwe cykle, gdy wektor wag się powtórzy w sekwencji uczenia to problem jest nierozwiązywalny
- Liczba błędów nie maleje monotonicznie - każda modyfikacja wag może popsuć klasyfikację poprzednio nauczonych przypadków
- Jak znaleźć najlepsze możliwe rozwiązanie nawet gdy problem nie jest liniowo separowalny?
- **Algorytm kieszeniowy** - uczenie perceptronu z zapamiętaniem wag dla których popełniono najmniej błędów

Algorithm Algorytm kieszonkowy (*pocket algorithm*)

Input: zbiór wektorów należących do jednego z dwóch zbiorów $\mathbb{P}$ i $\mathbb{N}$
Output: perceptron klasyfikujący przypadki do dwóch klas

1: zainicjuj losowo wagi $\mathbf{w}$ i w_0
2: $t \leftarrow 0, \quad t' \leftarrow t, \quad \mathbf{w}' \leftarrow \mathbf{w}, \quad w_0' \leftarrow w_0$
3: **for** losowo wybranego $\mathbf{x} \in \mathbb{P} \cup \mathbb{N}$ **do**
4: **if** $\mathbf{x}$ poprawnie klasyfikowany **then**
5: $t \leftarrow t + 1$
6: **else**
7: **if** $t > t'$ **then**
8: $t' \leftarrow t, \quad \mathbf{w}' \leftarrow \mathbf{w}, \quad w_0' \leftarrow w_0$
9: $t \leftarrow 0$
10: wykonaj aktualizację wag perceptronu
11: **return** $\mathbf{w}', w_0'$

Algorytm kieszonkowy

- Algorytm kieszonkowy ma za zadanie znaleźć najlepsze możliwe rozwiązanie nawet w przypadku problemów, które nie są liniowo separowalne
- Zapamiętuje wagi rozwiązania o najdłuższej sekwencji poprawnych klasyfikacji ale istnieje możliwość zignorowania lepszego rozwiązania o mniejszej liczbie błędów. Przeciwdziała temu alg. z zapadką, jednak wymaga on więcej nakładów obliczeniowych.
- **Algorytm kieszeniowy z zapadką** - modyfikacja algorytmu kieszeniowego, gdzie zapamiętywany jest zwycięzca tylko wtedy, gdy klasyfikuje poprawnie więcej przypadków treningowych

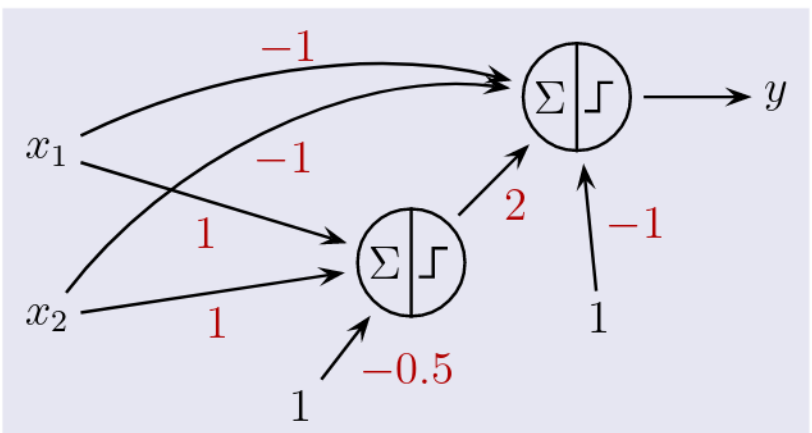
Algorithm Algorytm kieszonkowy z zapadką

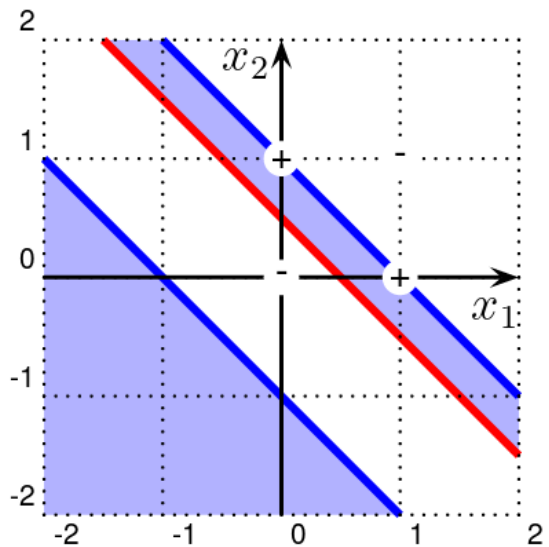
Input: zbiór wektorów należących do jednego z dwóch zbiorów $\mathbb{P}$ i $\mathbb{N}$
Output: perceptron klasyfikujący przypadki do dwóch klas

1: zainicjuj losowo wagi $\mathbf{w}$ i w_0
2: $t \leftarrow 0, \quad t' \leftarrow t, \quad \mathbf{w}' \leftarrow \mathbf{w}, \quad w_0' \leftarrow w_0$
3: **for** losowo wybranego $\mathbf{x} \in \mathbb{P} \cup \mathbb{N}$ **do**
4: **if** $\mathbf{x}$ poprawnie klasyfikowany **then**
5: $t \leftarrow t + 1$
6: **else**
7: **if** $t > t'$ **then**
8: **if** $\mathbf{w}$ i w_0 klasyfikują poprawnie więcej przypadków niż $\mathbf{w}'$ i w_0' **then**
9: $t' \leftarrow t, \quad \mathbf{w}' \leftarrow \mathbf{w}, \quad w_0' \leftarrow w_0$
10: $t \leftarrow 0$
11: wykonaj aktualizację wag perceptronu
12: **return** $\mathbf{w}', w_0'$

Czego może się nauczyć perceptron?

- Perceptron binarny potrafi rozwiązać wyłącznie problemy liniowo separowalne
- Przykład XOR - nie jest liniowo separowalny
- Sieci połączonych neuronów mają większe możliwości
- Problem XOR można rozwiązać za pomocą 2 perceptronów prostych



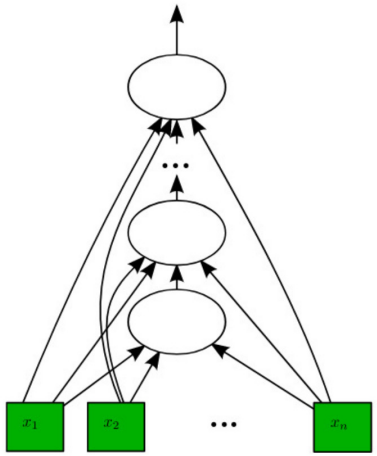


Rys: Riedmiller, Machine Learning (lectures)

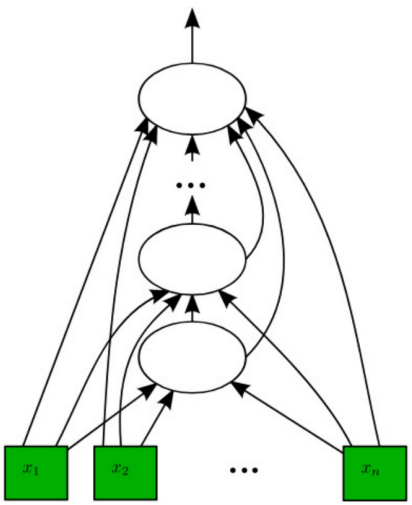
Algorytm wieżowy i piramidalny

Przykład algorytmów rozrastających się

1. Wytrenuj pojedynczy perceptron (np. algorytmem kieszonkowym) na danych treningowych
2. Jeżeli nie uzyskano zadowalającego rezultatu to dodaj perceptron, na którego wejście podany jest sygnał
   - z poprzedniego perceptronu (algorytm **wieżowy**)
   - ze wszystkich poprzednich perceptronów (algorytm **piramidalny**)
3. Wróć do punktu 1.



(a) Sieć wieżowa.

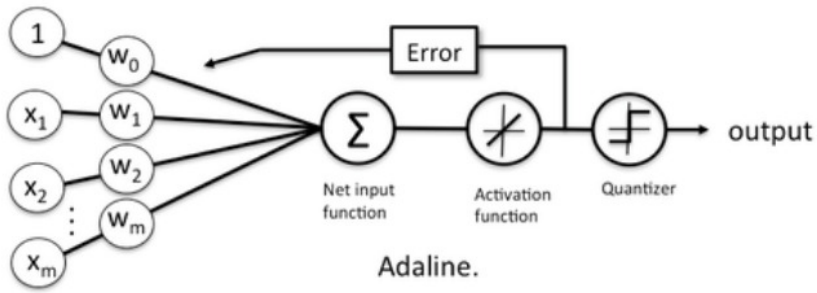


(b) Sieć piramidalna.

Maja Czoków, Jarosław Piersa, Tomasz Schreiber, Wstęp do Sieci Neuronowych

Adaline - Adaptive Linear Element

Adaline (Widrow, 1959) - jednowarstwowa sieć składająca się z perceptronów prostych. Realizacja sprzętowa z użyciem memistorów.



Adaline.

Reguła uczenia Widrowa-Hoffa

$$\Delta w_k = \lambda \left(y - \sum_i w_i x_i \right) x_k$$

gdzie y jest oczekiwanym sygnałem wyjściowym a $\sum_i w_i x_i$ jest aktywacją neuronu (sygnałem wyjściowym przed funkcją aktywacji), λ jest współczynnikiem uczenia

Rys: https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html

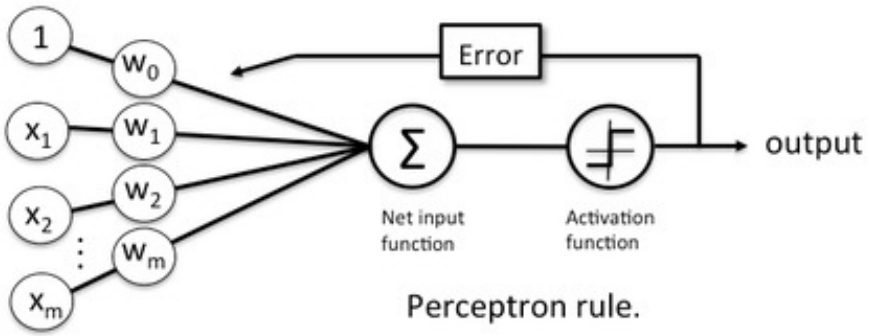
Reguła uczenia

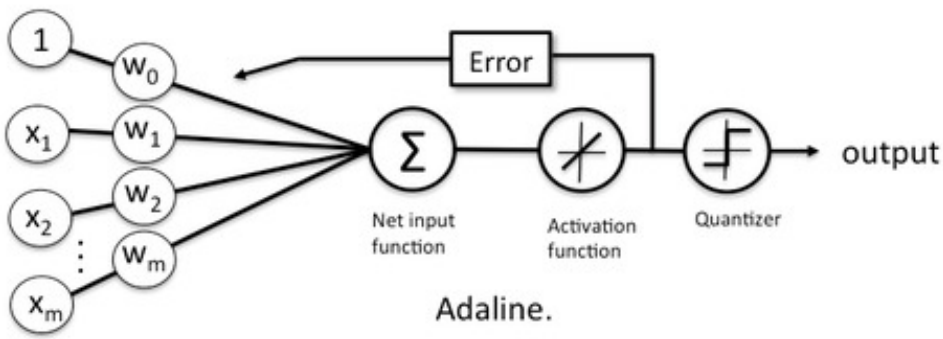
- Reguła uczenia Widrowa-Hoffa dąży do minimalizacji błędu kwadratowego aktywacji neuronu $\mathbf{w}^T\mathbf{x}$ względem pożądanego sygnału wyjściowego y

$$E(\mathbf{x}, y) = \frac{1}{2}\left(y - \sum_i w_i x_i\right)^2$$

- funkcja kosztu E jest ciągła, więc możliwe jest uczenie metodami gradientowymi
- dla neuronów liniowych reguła jest równoważna regule delta, wówczas Adaline rozwiązuje problem aproksymacji, gdzie $y_i \in \mathbb{R}$
- dla neuronów z wyjściem progowym Adaline staje się klasyfikatorem, gdzie $y_i = \{-1, +1\}$
- **Madaline** (Many Adalines) - wielowarstwowa (z warstwą ukrytą) wersja Adaline

Adaline vs. Perceptron rule





Rys: https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html

Reguła delta

- **Reguła delta** - uogólniona reguła uczenia perceptronu dla ciągłych (różniczkowalnych) funkcji aktywacji $f(x)$

$$\Delta w_i = \lambda \big(y - f(\mathbf{x})\big) f'(\mathbf{x}) x_i$$

$\lambda > 0$ współczynnik uczenia
y to pożądany sygnał dla wejścia $\mathbf{x}$
$y \in \mathbb{R}$ dla problemu aproksymacji,
$y = \{-1, +1\}$ lub $y = \{0, 1\}$ dla klasyfikacji binarnej

- minimalizacja błędu kwadratowego E w kierunku największego spadku gradientu

$$\Delta w = -\lambda \nabla_w E$$

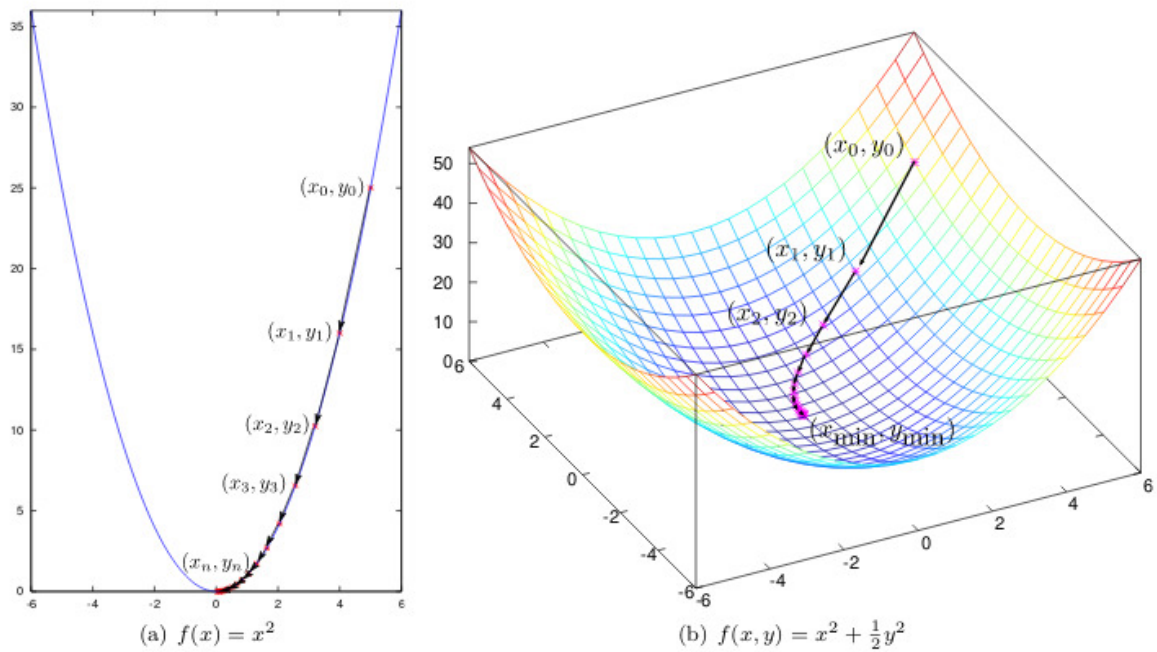
Dla błędu kwadratowego otrzymujemy

$$\Delta w_i = -\lambda \frac{\partial}{\partial w_i} E = -\lambda \frac{\partial}{\partial w_i} \frac{1}{2} \left(y - f(\mathbf{x})\right)^2 = \lambda \left(y - f(\mathbf{x})\right) f'(\mathbf{x}) x_i$$

Algorytm uczenia metodą gradientu prostego (*gradient descent*)

1. ustaw punkt startu $\mathbf{w}_0$ oraz współczynnik uczenia $\lambda > 0$
2. dopóki nie jest spełnione kryterium stopu wykonuj

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda \nabla E(\mathbf{w}_i)$$



(a) $f(x) = x^2$ (b) $f(x, y) = x^2 + \frac{1}{2}y^2$

Rys: Maja Czoków, Jarosław Piersa,Tomasz Schreiber, Wstęp do Sieci Neuronowych

Aspekty uczenia spadkiem gradientu

- Jeżeli istnieją minima lokalne to istnieje ryzyko utknięcia algorytmu w tym minimum
- Powtórzenie kilkukrotne uczenia z różnymi punktami startowymi może pozwolić uniknąć minimów lokalnych
- Trajektoria zbieżności zależy od punktu startowego
- Kryterium stopu algorytmu:
  - ustalona ilość kroków uczenia lub inne ograniczenie czasowe
  - osiągnięcie zadowalającego poziomu dokładności $E < \epsilon$
  - niewielkie zmiany w kolejnych iteracjach $||\Delta w|| < \epsilon$
- Dobór współczynnika uczenia λ jest kluczowy:
  - za duże λ: rozwiązanie może być pominięte (przeskoczone), proces uczenia może być rozbieżny
  - za małe λ: uczenie będzie powolne, algorytm może utknąć w minimum lokalnym
  - λ nie musi być stałe, współczynnik uczenia może być zmniejszany w czasie treningu lub dobierany zależnie od wypukłości funkcji błędu

Funkcje aktywacji

- **identyczność**

$$f(x) = x \qquad f'(x) = 1$$

- funkcja **liniowa** - nieograniczona $f(x) \in \mathbb{R}$

$$f(x) = ax + b \qquad f'(x) = a$$

- **progowa** unipolarna - nieciągła, nieróżniczkowalna

$$f(x) = \begin{cases} 1, & \text{gdy} \quad x \geq a \\ 0, & \text{gdy} \quad x < a \end{cases}$$

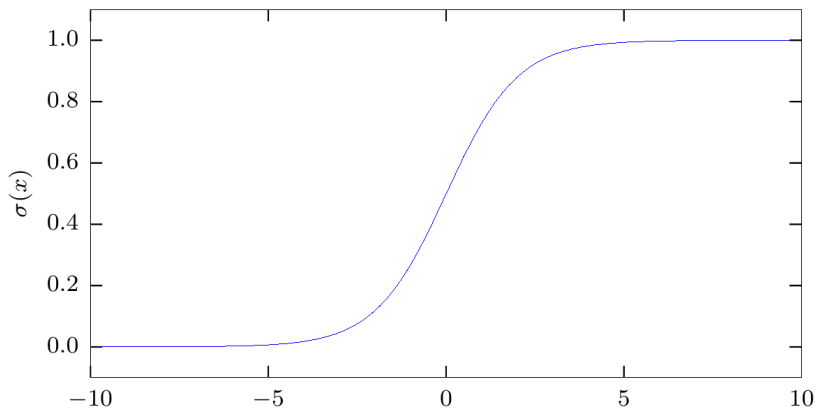
- progowa bipolarna

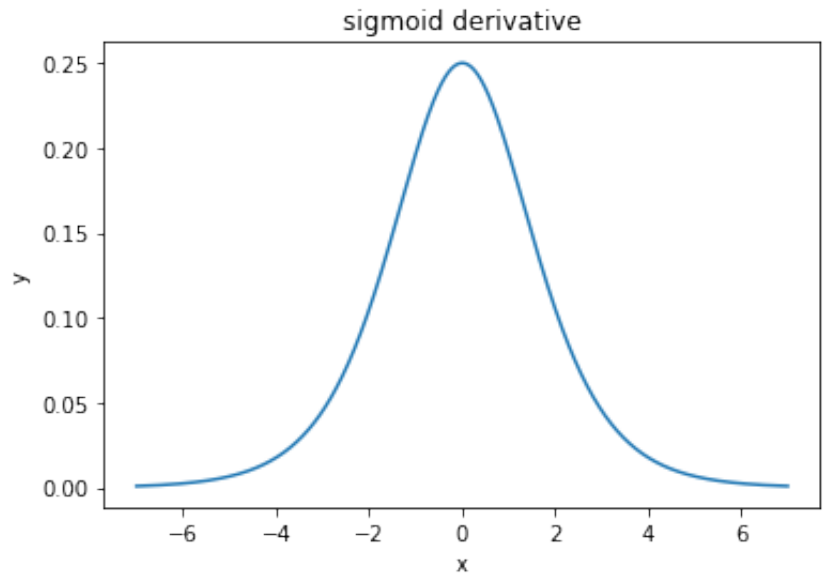
$$f(x) = \begin{cases} 1, & \text{gdy} \quad x \geq a \\ -1, & \text{gdy} \quad x < a \end{cases}$$

Funkcje aktywacji

- **sigmoidalna** (logistyczna) - ograniczona $f(x) \in (0, 1)$

$$f(x) = \frac{1}{1 + e^{-x}} \qquad f'(x) = f(x)(1 - f(x))$$



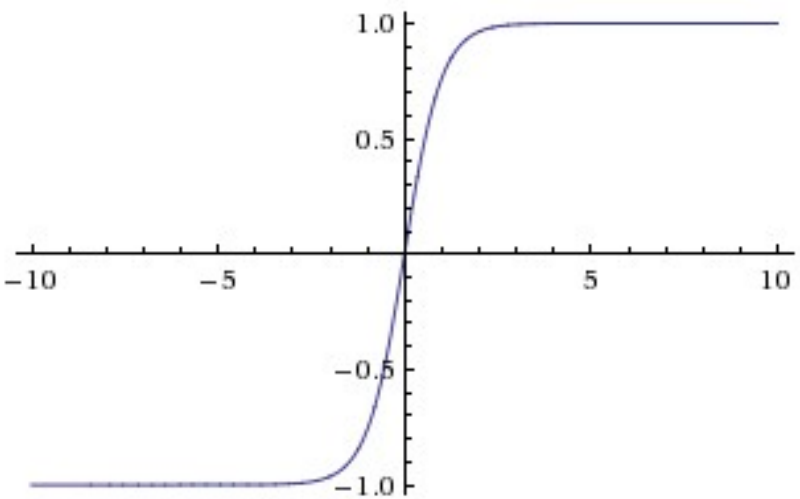


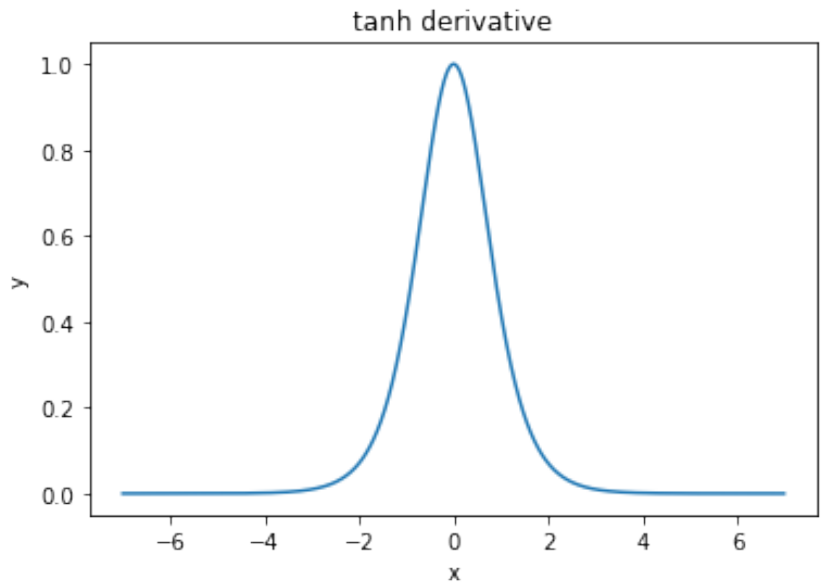
Rys: http://cs231n.github.io/

Funkcje aktywacji

- **tangens hiperboliczny** - ograniczona $f(x) \in (-1, +1)$

$$f(x) = \tanh x = \frac{1 - e^{-x}}{1 + e^{-x}} \qquad f'(x) = 1 - f^2(x)$$





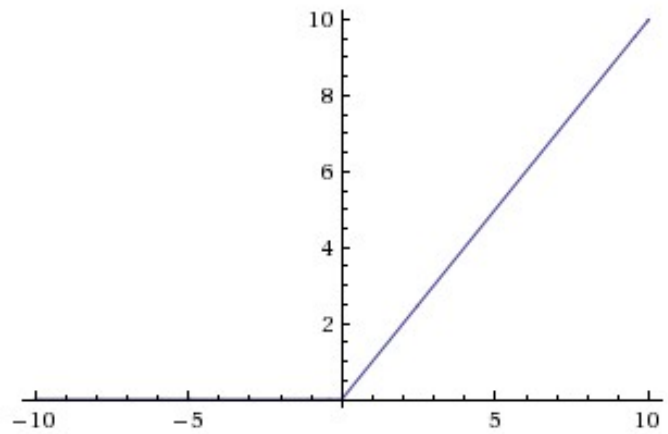
Rys: http://cs231n.github.io/,
https://www.kaggle.com/moriano/a-showcase-of-how-relus-can-speed-up-the-learning

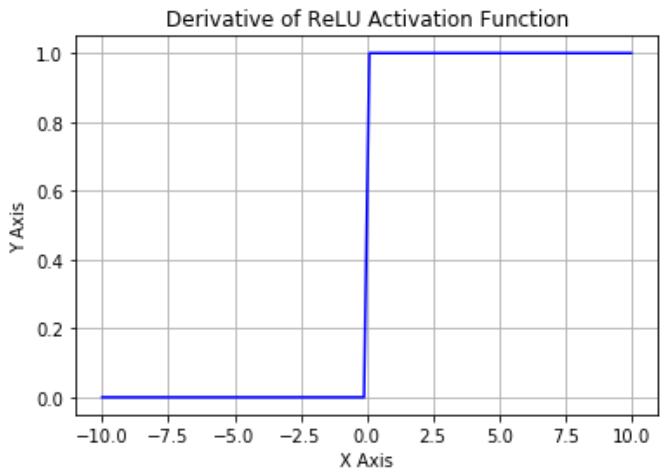
Funkcje aktywacji

- **ReLU** (Rectified Linear Unit) - fragmentami ciągła, brak pochodnej w 0

$$f(x) = \max(0, x) = \begin{cases} x, & \text{gdy} \quad x \geq 0 \\ 0, & \text{gdy} \quad x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, & \text{gdy} \quad x > 0 \\ 0, & \text{gdy} \quad x < 0 \end{cases}$$





Rys: http://cs231n.github.io/,
https://learnopencv.com/understanding-activation-functions-in-deep-learning/

Jednokierunkowe sieci wielowarstwowe

Multilayer Perceptron (MLP)

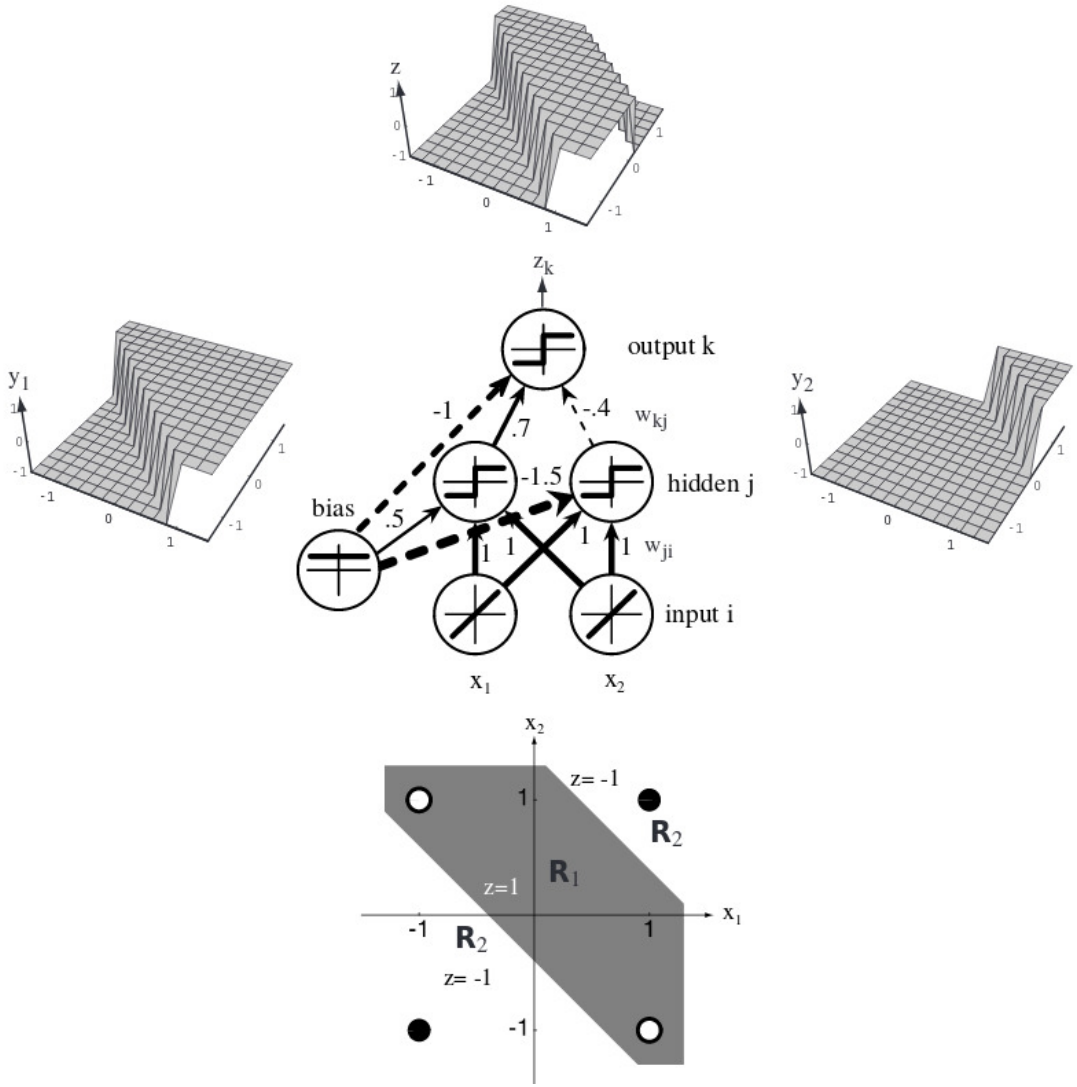
- Perceptrony jednowarstwowe potrafią rozwiązywać problemy separowalne liniowo
- Reguła delta dla sieci jednowarstwowej:

$$\Delta w_{ij} = \eta \left(y_j - f_j(\mathbf{x}) \right) f_j'(\mathbf{x}) x_i$$

  gdzie w_{ij} waga pomiędzy i-tym wejściem a j-tym wyjściem
- Problemy bardziej złożone (np. XOR) wymagają dodania kolejnej warstwy (nieliniowej) przetwarzania
- Każda kolejna warstwa tworzy nowy obraz danych, który stanowi wejście dla kolejnych warstw

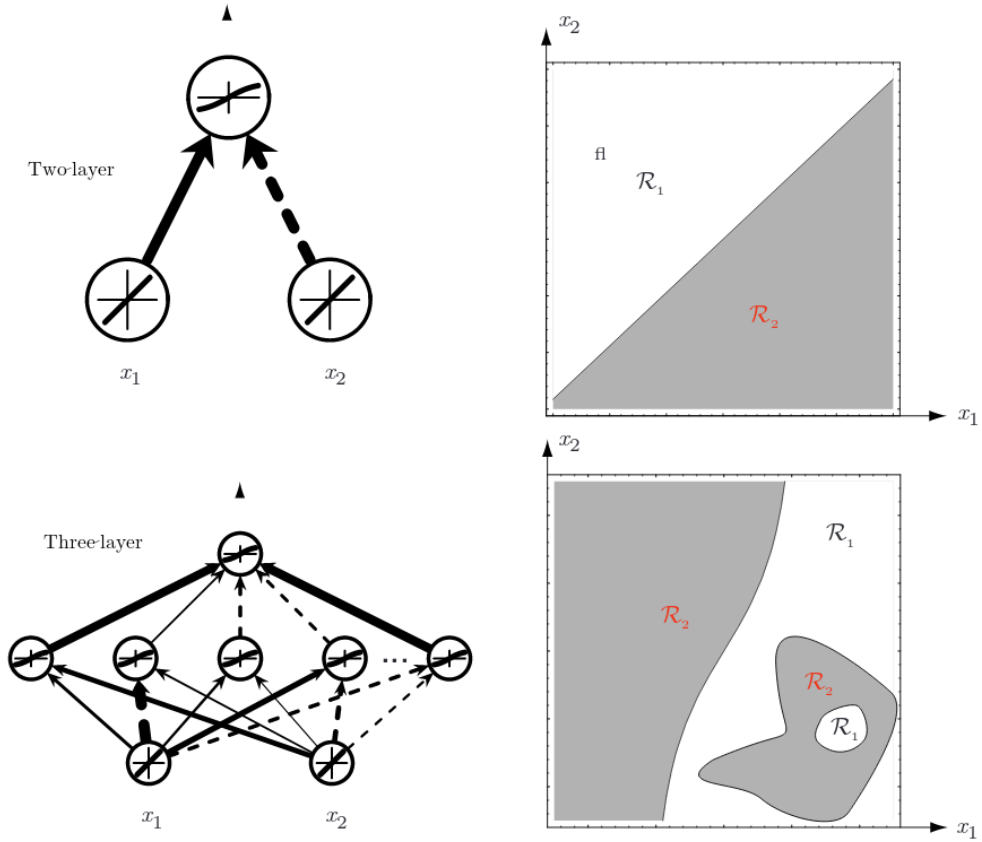
$$\mathbf{y} = f(\mathbf{x}) = f^{(M)} \left(f^{(M-1)} \left(\dots f^{(1)}(\mathbf{x}) \right) \right)$$

Jeszcze raz XOR



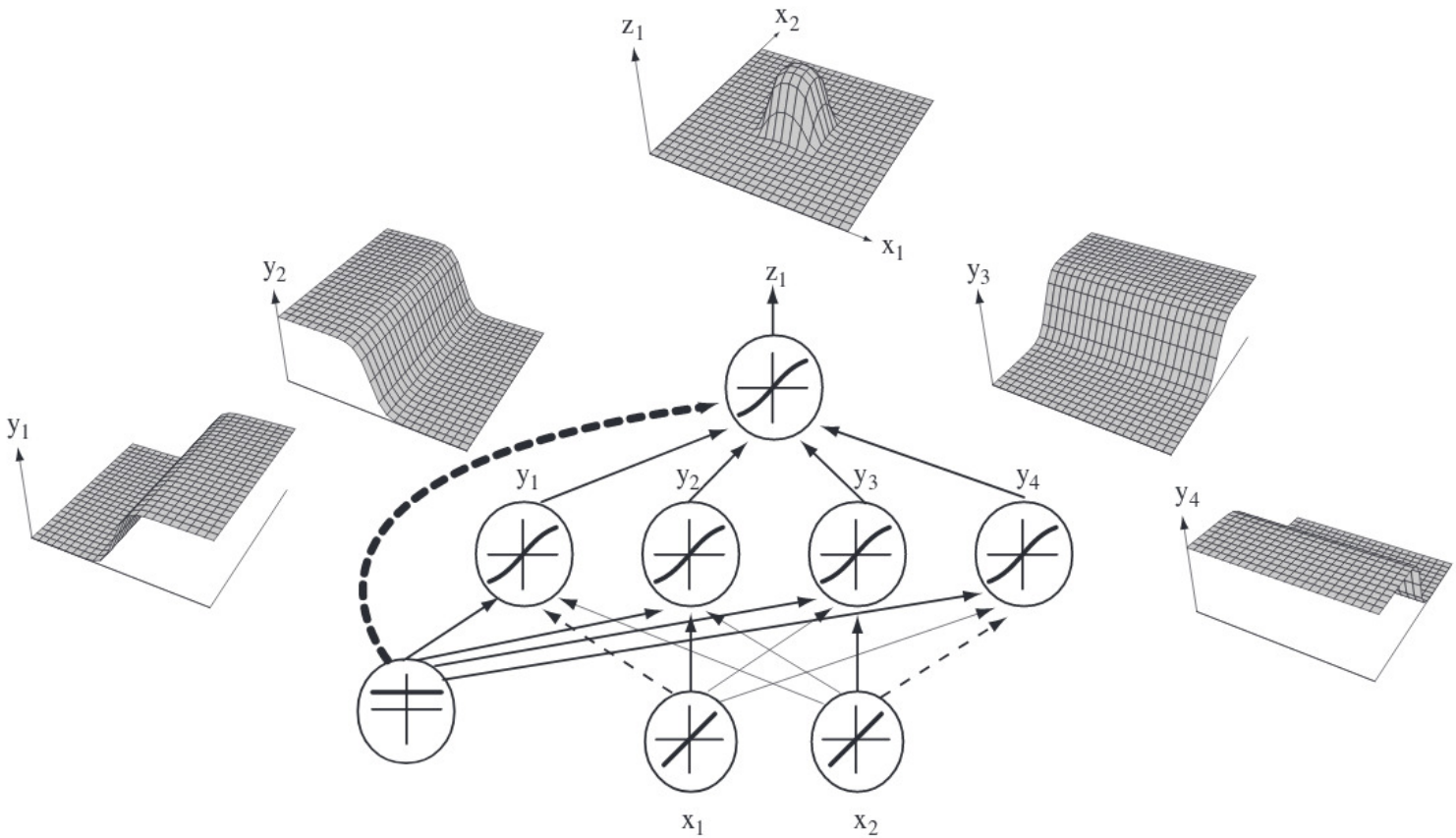
Rys: Duda, Hart, Pattern Classification

Warstwa ukryta a granice decyzji



Rys: Duda, Hart, Pattern Classification

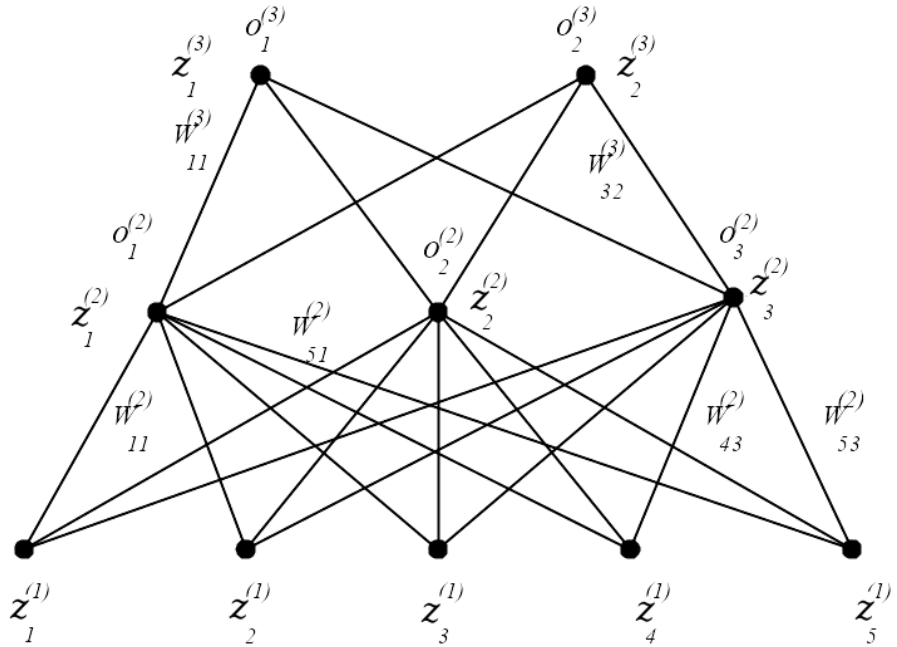
Przykład: sieć MLP 2-4-2



Rys: Duda, Hart, Pattern Classification

Perceptron 3-warstwowy (typowa sieć MLP)

- Warstwy: wejściowa, ukryta, wyjściowa

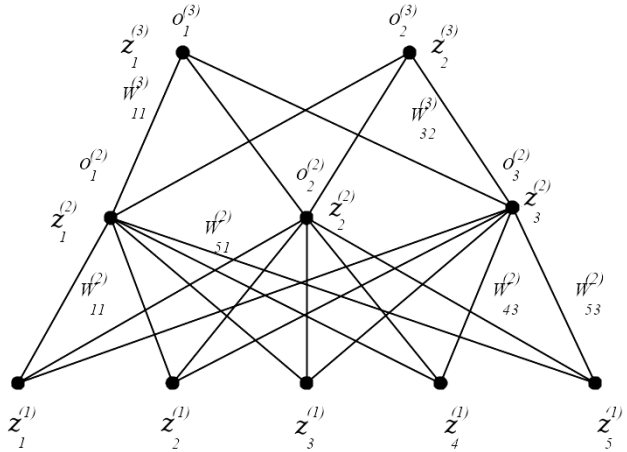


- Warstwy w pełni połączone (wagi neuronów w warstwie K tworzą macierz $\mathbf{W}^{(K)}$).
- Sygnał wyjściowy z warstwy K:

$$\mathbf{o}^{(K)} = \sigma\left(\mathbf{W}^{(K)}\mathbf{o}^{(K-1)}\right)$$

- Waga $w_{ij} = 0$ może być interpretowana jako brak połączenia

MLP - oznaczenia



M - liczba warstw ($M = 3$)

$w_{ij}^{(l)}$ - waga połaczenia neuronu i-tego z warstwy $l-1$ z neuronem j-tym z warstwy l

$z_j^{(l)}$ - aktywacja neuronu j w warstwie l

$$z_j^{(l)} = \sum_i w_{ij}^{(l)} o_i^{(l-1)}$$

$o_j^{(l)}$ - sygnał wyjściowy neuronu j należącego do warstwy l

$$o_j^{(l)} = \sigma(z_j^{(l)}) = \sigma\left(\sum_i w_{ij}^{(l)} o_i^{(l-1)}\right)$$

$f_i(\mathbf{x}) = o_i^{(M)}$ funkcja realizowana przez MLP (i-te wyjście sieci)

Algorytm wstecznej propagacji błędu

(1974, 1986)

Miara błędu sieci dla pojedynczego wzorca wejściowego $\mathbf{x}$ i pożądanej odpowiedzi $\mathbf{y} = [y_1, \ldots, y_n]$

$$E(\mathbf{x}; \mathbf{W}) = \frac{1}{2} \sum_{i=1}^{n} (y_i - f_i(\mathbf{x}; \mathbf{W}))^2$$

gdzie n to liczba wyjść

Minimalizacja błędu metodą gradientu prostego:

$$\Delta w = -\eta \frac{\partial E(\mathbf{x}; \mathbf{W})}{\partial w} = \eta \sum_{i=1}^{n} (y_i - f_i(\mathbf{x}; \mathbf{W})) \frac{\partial f_i(\mathbf{x}; \mathbf{W})}{\partial w}$$

Zmiana wag dla warstwy wyjściowej

Dla wagi $w_{jk}^{(M)}$ łączącej wyjście $o_j^{(M-1)}$ z k-tym wyjściem sieci

$$\begin{aligned}
\Delta w_{jk}^{(M)} &= \eta \sum_{i=1}^{n} \left(y_i - f_i(\mathbf{x}; \mathbf{W})\right) \frac{\partial f_i(\mathbf{x}; \mathbf{W})}{\partial w_{jk}^{(M)}} \\
&= \eta \left(y_k - f_k(\mathbf{x}; \mathbf{W})\right) \frac{\partial f_k(\mathbf{x}; \mathbf{W})}{\partial w_{jk}^{(M)}} \\
&= \eta \left(y_k - f_k(\mathbf{x}; \mathbf{W})\right) \frac{\partial \sigma(z_k^{(M)})}{\partial z_k^{(M)}} \frac{\partial z_k^{(M)}}{\partial w_{jk}^{(M)}} \\
&= \eta \left(y_k - f_k(\mathbf{x}; \mathbf{W})\right) \sigma'(z_k^{(M)}) o_j^{(M-1)}
\end{aligned}$$

$$\Delta w_{jk}^{(M)} = \eta \left(y_k - f_k(\mathbf{x}; \mathbf{W}) \right) \sigma'(z_k^{(M)}) o_j^{(M-1)}$$

Niech $\delta_i^{(M)}$ oznacza błąd „lokalny” dla warstwy M

$$\delta_k^{(M)} = \sigma'(z_k^{(M)}) \left(y_k - f_k(\mathbf{x}; \mathbf{W}) \right)$$

Zmiana wag neuronów warstwy wyjściowej:

$$\Delta w_{jk}^{(M)} = \eta \delta_k^{(M)} o_j^{(M-1)}$$

$$\Delta w_{0k}^{(M)} = \eta \delta_k^{(M)}$$

Zmiana wag dla warstwy ukrytej $(M-1)$

Dla wagi $w_{jk}^{(M-1)}$ z warstwy ukrytej $M-1$:

$$\Delta w_{jk}^{(M-1)} = -\eta \frac{\partial E(\mathbf{W})}{\partial w_{jk}^{(M-1)}} = \eta \sum_{i=1}^{n} \left(y_i - f_i(\mathbf{x};\mathbf{W})\right) \frac{\partial f_i(\mathbf{x};\mathbf{W})}{\partial w_{jk}^{(M-1)}}$$

gradient

$$\frac{\partial f_i(\mathbf{x};\mathbf{W})}{\partial w_{jk}^{(M-1)}} = \frac{\partial \sigma(z_i^{(M)})}{\partial z_i^{(M)}} \frac{\partial z_i^{(M)}}{\partial w_{jk}^{(M-1)}} = \sigma'\left(z_i^{(M)}\right) \frac{\partial \left(\sum_l w_{li}^{(M)} o_l^{(M-1)}\right)}{\partial w_{jk}^{(M-1)}}$$

$$= \sigma'\left(z_i^{(M)}\right) w_{ki}^{(M)} \frac{\partial o_k^{(M-1)}}{\partial w_{jk}^{(M-1)}}$$

$$= \sigma'\left(z_i^{(M)}\right) w_{ki}^{(M)} \sigma'\left(z_k^{(M-1)}\right) o_j^{(M-2)}$$

Otrzymujemy:

$$\frac{\partial f_i(\mathbf{x};\mathbf{W})}{\partial w_{jk}^{(M-1)}} = \sigma'\left(z_i^{(M)}\right) w_{ki}^{(M)} \sigma'\left(z_k^{(M-1)}\right) o_j^{(M-2)}$$

stąd zmiana wag dla warstwy $M-1$

$$\begin{aligned}
\Delta w_{jk}^{(M-1)} &= \eta \sum_{i=1}^{n} \left(y_i - f_i(\mathbf{x};\mathbf{W})\right) \frac{\partial f_i(\mathbf{x};\mathbf{W})}{\partial w_{jk}^{(M-1)}} \\
&= \eta \sum_{i=1}^{n} \left(y_i - f_i(\mathbf{x};\mathbf{W})\right) \sigma'\left(z_i^{(M)}\right) w_{ki}^{(M)} \sigma'\left(z_k^{(M-1)}\right) o_j^{(M-2)} \\
&= \eta \sigma'\left(z_k^{(M-1)}\right) \sum_{i=1}^{n} w_{ki}^{(M)} \delta_i^{(M)} o_j^{(M-2)} \\
&= \eta \delta_k^{(M-1)} o_j^{(M-2)}
\end{aligned}$$

$$\Delta w_{0k}^{(M-1)} = \eta \delta_k^{(M-1)} \qquad \text{gdzie} \qquad \delta_k^{(M-1)} = \sigma'\left(z_k^{(M-1)}\right) \sum_{i=1}^{n} w_{ki}^{(M)} \delta_i^{(M)}$$

Struktura wzoru w kolejnych warstwach jest taka sama

Wsteczna propagacja błędu - podsumowanie

Funkcja realizowana przez sieć wielowarstwową

$$f_i(\mathbf{x};\mathbf{W}) = \sigma\left(\sum_j w_{ji}^{(M)}\sigma\left(\sum_k w_{kj}^{(M-1)}\sigma\left(\sum_l w_{lk}^{(M-2)}\ldots\sigma\left(\sum_n w_{nm}^{(2)}x_n\right)\right)\right)\ldots\right)$$

Aktualizacja wag w dowolnej warstwie K - **uogólniona reguła delta**

$$\Delta w_{ij}^{(K)} = \eta\delta_j^{(K)}o_i^{(K-1)}$$

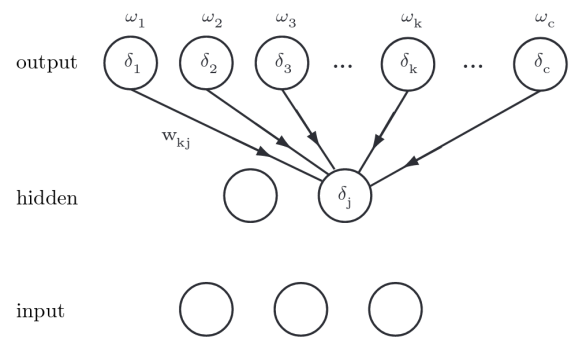
gdzie sygnał wejściowy $o_i^{(1)} = x_i$

Sygnał błędu w warstwie K

$$\delta_i^{(K)} = \sigma'\left(z_i^{(K)}\right)\sum_j \delta_j^{(K+1)}w_{ij}^{(K+1)}$$

sygnał błędu w warstwie wyjściowej :

$$\delta_i^{(M)} = \sigma'(z_i^{(M)})\,(y_i - f_i(\mathbf{x};\mathbf{W}))$$



Algorytm wstecznej propagacji

1. Zainicjuj wagi $\mathbf{W}$ małymi losowymi wartościami
2. Dopóki nie spłoniony warunek stopu wykonuj
   2.1 dla pary uczącej $(\mathbf{x}, \mathbf{y})$ oblicz sygnał uzyskany na kolejnych warstwach (od wejścia do wyjścia sieci)

$$o_i^{(K)} = \sigma\left(\sum_j w_{ji}^{(K)} o_j^{(K-1)}\right)$$

   2.2 oblicz sygnał błędu warstwy wyjściowej

$$\delta_i^{(M)} = \sigma'(z_i^{(M)})\,(y_i - f_i(\mathbf{x}))$$

   2.3 oblicz sygnał błędu wszystkich warstw propagując błąd od warstwy wyjściowej do wejściowej

$$\delta_i^{(K)} = \sigma'\left(z_i^{(K)}\right) \sum_j \delta_j^{(K+1)} w_{ij}^{(K+1)}$$

   2.4 aktualizuj wagi

$$\Delta w_{ij}^{(K)} = \eta \delta_j^{(K)} o_i^{(K-1)}$$

Algorytm wstecznej propagacji można uogólnić dla:

- sieci o innej strukturze połączeń: np. dodatkowe połączenia pomiędzy wejściem lub wyjściem, lub innymi warstwami ukrytymi. Dla sieci rekurencyjnych odpowiednikiem jest BPTT (wsteczna propagacja w czasie)
- dowolną liczbę warstw
- różne nieliniowości w warstwach
- różne funkcje aktywacji dla poszczególnych neuronów
- różne stałe uczenia dla każdej warstwy lub węzła
- inne funkcje kosztu, nie tylko MSE, np. Cross Entropy

Warianty treningu

Dla zbioru zawierającego N przypadków i sieci zawierającej M wyjść

$$\hat{E} = \frac{1}{N}\sum_{i=1}^{N} E(\mathbf{x}_i) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}(y_{ij} - f_j(\mathbf{x}_i))^2$$

$$\Delta\hat{w} = \sum_{i=1}^{N} \Delta w_i$$

Batch gradient descent

- uśredniony błąd całego zbioru treningowego
- niepraktyczne dla dużych danych
- stabilny trening
- nie zależy od kolejności prezentacji wzorców

Warianty treningu

SGD - stochastic gradient descent (on-line)

- aktualizacja dla pojedynczego przypadku treningowego ($N = 1$)
- duża szybkość działania ale i duża wariancja,
- istotna jest kolejność prezentacji przypadków

MiniBatch gradient descent

- średnia z n losowych przypadków (*mini-batch*), trening on-line,
- często utożsamiany z SGD
- mniejsza wariancja niż SGD
- operacje na macierzach, łatwiejsze zrównoleglenie GPU/CPU
- wielkość paczki można dobrać tak aby wykorzystać maksymalnie zasoby (pamięć) zwiększając szybkość przetwarzania

Własności MLP

- **Uniwersalny aproksymator:** sieć MLP z jedną warstwą ukrytą jest w stanie aproksymować dowolną funkcję ciągłą z dowolną dokładnością.
  Dwie warstwy ukryte rozszerzają możliwości na funkcje nieciągłe.
  Klasyfikator potrafi zrealizować dowolne granice decyzji (obszary nie muszą być wypukłe ani połączone)
- Neurony ukryte: transformacja nieliniowa do przestrzeni odwzorowań, tworząca nowe cechy za pomocą nieliniowych kombinacji
- Wiele zastosowań: klasyfikacja, wielowymiarowa regresja

Problemy I

- Optymalizacja nieliniowych funkcji zawsze sprawia problemy, tu mamy złożenie (czasem wielu) funkcji nieliniowych
- Dobór architektury sieci: Ile węzłów w warstwie ? Jakie funkcje aktywacji? Sieci ontogeniczne (rozrastające się) dostosowujące rozmiar do złożoności problemu
- Przeuczenie i generalizacja - zbyt duża liczba optymalizowanych parametrów powoduje przeuczenie, zbyt mała - może generować zbyt proste rozwiązania
- Regularyzacja - metody ograniczenia zjawiska przeuczenia, np. modyfikacje funkcji kosztu, ograniczenie liczby parametrów
- Inicjalizacja parametrów - szybkość treningu i wynik zależy od punktu startowego
- Minima lokalne i plateau, wąskie „rynny” - np. wielokrotny start
- Wpływ nowych wzorców na już nauczone – zapominanie

Problemy II

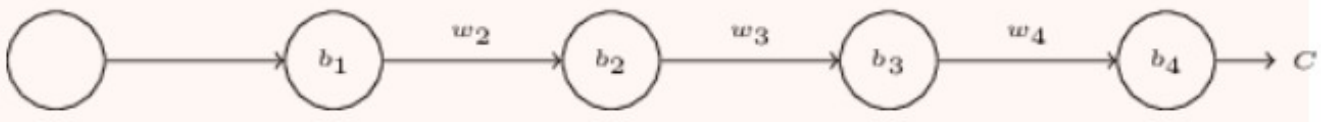
- Dobór stałej uczenia
- Znikający gradient, eksplodujący gradient
- Przygotowanie danych uczących: normalizacja, standaryzacja, kodowanie wyjść, ...
- Ocena modelu: zbiory walidacyjne, testowe, kroswalidacja

Znikający gradient

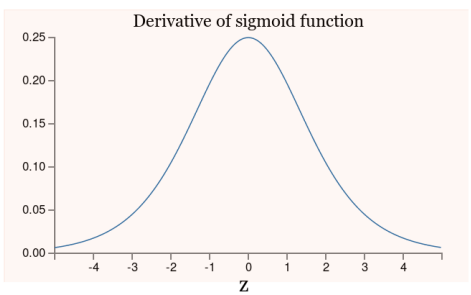
Problem **zanikającego gradientu** (ang. *vanishing gradient*) - zjawisko, w którym szybkość uczenia (zmiany wag) wsteczną propagacją drastycznie maleje w głębszych warstwach sieci - sygnał błędu zanika podczas propagacji do kolejnych warstw.

Przykład: sieć 1 x 1 x 1 x 1

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



gdy $|w_j| < 1$ wówczas $|w_j \sigma'(z_j)| < \frac{1}{4}$



Nielsen, 2016

Czynniki wpływające na zanikanie gradientu:

- gdy wagi lub pochodne przyjmują wartości mniejsze od 1, ich iloczyn maleje wykładniczo wraz z głębokością sieci
- funkcje aktywacji z nasyceniem: funkcje sigmoidalna i tanh mają małe pochodne w obszarach nasycenia
- nieprawidłowa inicjalizacja wag może prowadzić do mnożenia przez wartości < 1
- duża głębokość architektury sieci, problem zanikania gradientu nasila się wraz z liczbą warstw

Konsekwencje dla uczenia sieci

- brak aktualizacji wag w początkowych warstwach w czasie uczenia
- powolna zbieżność lub całkowite zatrzymanie procesu uczenia

Problem niestabilnego gradientu

- gradient w niższych warstwach zależy od wartości wag, aktywacji i gradientów w warstwach wyższych
- **eksplodujący gradient**
  gdy $w_i >> 1$ oraz $z_i \approx 0$ wówczas $\left|w_i \sigma'(z_i)\right| > 1$, może to owocować bardzo dużymi gradientami w początkowych warstwach
- znikający i eksplodujący gradient to przypadki szczególne **problemu niestabilnego gradientu** -> wartości gradientów w poszczególnych warstwach mogą znacznie się różnić, warstwy uczą się z różnym tempem

Specyfikacja sieci i metody treningu

- Specyfikacja modelu:
  - specyfikacja architektury
  - typ neuronów ukrytych: liniowe, sigmoidalne, ReLU, ...
  - typ neuronów wyjściowych: liniowe, sigmoidalne, softmax, ...
- Funkcja kosztu:
  - mean squared error (MSE),
  - cross-entropy (CE),
- Optymalizacja metodami spadku gradientu lub innymi:
  - SGD, RPROP, Quckprop,
  - Adam, AdaGrad, ...
- Metody regularyzacji:
  - momentum, L_1, L_2 (weight decay), ...

Funkcja kosztu MSE

Błąd średniokwadratowy (*mean squared error)*

$$J(\Theta) = \frac{1}{N} \sum (\hat{y} - y)^2$$

- średnia kwadratów różnic między wartościami prawdziwymi a przewidywanymi
- stosowana głównie w problemach regresyjnych
- gradient funkcji kosztu:

$$\frac{\partial J}{\partial \Theta} = \frac{2}{N} \sum (\hat{y} - y) \frac{\partial \hat{y}}{\partial \Theta}$$

gdzie
$\hat{y}$ - wyjścia sieci (wartość przewidywana przez model),
Θ - parametry sieci (wagi),
y - oczekiwane wyjścia sieci

Funkcja kosztu Cross Entropy

Entropia krzyżowa (*cross entropy*)

$$J(\Theta) = -\sum y \log \hat{y}$$

- mierzy dysproporcje miedzy rozkładem prawdziwym y i przewidywanym $\hat{y} \in (0, 1)$
- stosowana w problemach klasyfikacji wieloklasowej, gdzie klasy kodowane są w reprezentacji binarnej (*one-hot encoding*)
- gdy $y \approx \hat{y}$ to funkcja kosztu bliska zeru
- $J > 0$
- gradient funkcji kosztu

$$\frac{\partial J}{\partial \Theta} = -\sum y \frac{1}{\hat{y}} \frac{\partial \hat{y}}{\partial \Theta}$$

☞ Wizualizacja funkcji CE

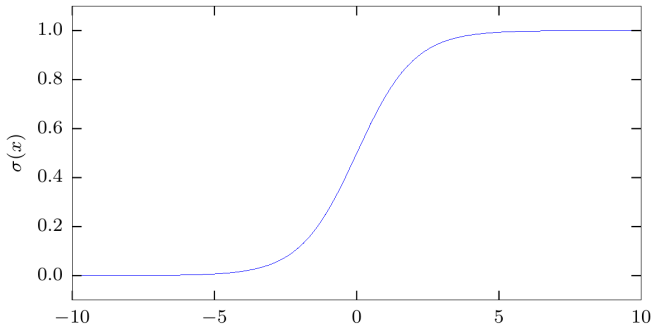
Funkcja liniowa w warstwie wyjściowej

$$z = \mathbf{w}^T \mathbf{h} + b$$

- nieograniczone wartości $(-\infty, +\infty)$
- jednostki nie nasycają się ale bardzo duże wartości aktywacji również mogą przysporzyć problemów w trakcie uczenia
- stabilniejsze w optymalizacji metodami gradientowymi od innych typowych funkcji nieliniowych stosowanych w sieciach
- zastosowanie wraz funkcją kosztu MSE do problemów regresji (dane wyjściowe ciągłe), modelowanie rozkładów gusowskich

Funkcja logistyczna w warstwie wyjściowej

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$



- wartości ograniczone $(0, 1)$
- nasycają się, aktywne uczenie tylko w okolicach $x \approx 0$
- zastosowanie: klasyfikacja binarna (2 klasy), modelowanie rozkładu Bernoullego, klasyfikacja wieloetykietowa (*multi-label classification*), gdzie na każdym wyjściu oczekiwana jest wartości binarna (0 lub 1)
- **binarna funkcja kosztu Cross-Entropy**, na każdym wyjściu oczekujemy stanu 0 lub 1

$$J_{CE} = -\sum \left[y \ln \hat{y} + (1-y) \ln(1-\hat{y}) \right]$$

Funkcja sigmoidalna w warstwie wyjściowej

Podczas minimalizacji MSE, gdy funkcja sigmoidalna σ neuronu wyjściowego się nasyca to sygnał błędu zanika, także w sytuacji niepoprawnej odpowiedzi sieci, gdy błąd $|\hat{y} - y|$ jest duży

$$\frac{\partial \hat{y}}{\partial \Theta} = \frac{\partial \sigma(z)}{\partial \Theta} = \sigma'(z)\frac{\partial z}{\partial \Theta} \approx 0$$

$$\frac{\partial J_{MSE}}{\partial \Theta} = \frac{2}{N}\sum (\hat{y} - y)\frac{\partial \hat{y}}{\partial \Theta} = \frac{2}{N}\sum (\hat{y} - y)\,\sigma'(z)\frac{\partial z}{\partial \Theta}$$

Problem znika przy minimalizacji funkcji kosztu entropii krzyżowej

$$\frac{\partial J_{CE}}{\partial \Theta} = -\sum \left(\frac{y}{\hat{y}}\frac{\partial \hat{y}}{\partial \Theta} + \frac{1-y}{1-\hat{y}}\frac{\partial (1-\hat{y})}{\partial \Theta} \right) = \sum (\hat{y} - y)\frac{\partial z}{\partial \Theta}$$

gdzie z jest liniową aktywacją (*logit*) neuronu wyjściowego $\hat{y} = \sigma(z)$, pochodna f. sigmoidalnej $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Softmax w warstwie wyjściowej

$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- wartości $(0, 1)$ unormowane

$$\sum_i softmax(\mathbf{z})_i = 1$$

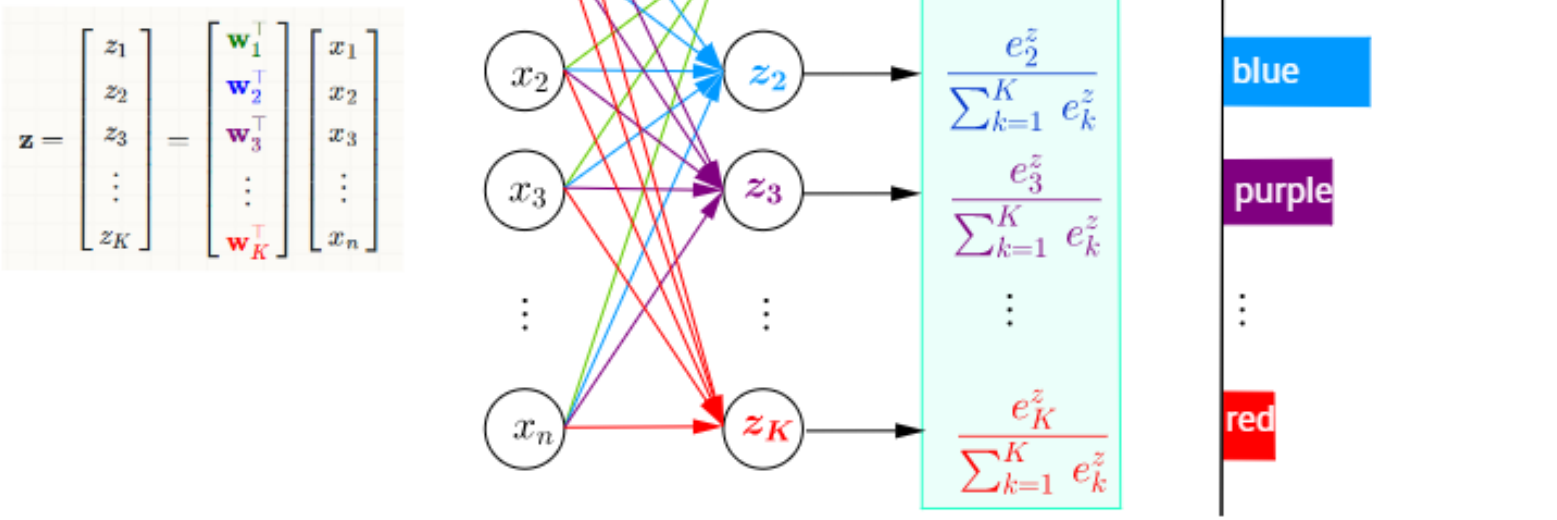
- zastosowanie: klasyfikacja n rozłącznych klas (*multi-class classification*), gdzie każde wyjście sieci związane jest z jedną klasą, etykiety klas kodowane są w postaci wektora binarnego (*one hot*)

$$[0, \ldots, 0, 1, 0, \ldots, 0]$$

- wartości wyjściowe sieci reprezentują rozkład prawdopodobieństwa przynależności do poszczególnych klas

$$\hat{y}_i = P(y = i|X), \qquad i = 1, \ldots, n$$

Multi-Class Classification with NN and SoftMax Function



Źródło: `stats.stackexchange.com/questions/265905/derivative-of-softmax-with-respect-to-weights`

- gradient

$$\frac{\partial softmax(\mathbf{z})_i}{\partial z_j} = softmax(\mathbf{z})_i(\delta_{ij} - softmax(\mathbf{z})_i)$$

- gradient funkcji kosztu entropii krzyżowej, gdy na wyjściu sieci softmax $\hat{y}_i = softmax(\mathbf{z})_i$

$$\frac{\partial J_{CE}}{\partial \Theta} = \sum(\hat{y} - y)\frac{\partial z}{\partial \Theta}$$

- wzór sie upraszcza eliminując problem zanikania sygnału błędu dla nasyconych wyjść, który występował dla funkcji kosztu MSE i wyjść sigmoidalnych
- w praktyce, podczas uczenia obliczanie logarytmu i softmax nie jest konieczne, wystarczy wartość *logit* $\mathbf{z}$

- ważne są różnice wartości pomiędzy wyjściami a nie amplitudy poszczególnych wyjść

$$softmax(\mathbf{x}) = softmax(\mathbf{x} + c)$$

- minimalizacja kosztu CE powoduje zwiększenie wartości na jednym z wyjść z_i i zmniejszenie wszystkich pozostałych $-\log \sum e^{z_k}$

$$\log softmax(\mathbf{z})_i = z_i - \log \sum_k e^{z_k}$$

- ponieważ

$$\log \sum_k e^{z_k} \approx \max_i z_i$$

  to dla poprawnej odpowiedzi $z_i \approx \max z_i$, błąd znika. Karane są najbardziej aktywne niepoprawne predykcje
- z punktu widzenia neurobiologicznego softmax może być widziany jako realizacja mechanizmów hamowania występującego w grupach neuronów w korze, podobnie do mechanizmu zwycięzca bierze wszystko (*winner takes all*)

Dobór funkcji kosztu uzależniony jest od rozkładu wartości wyjściowych i typu neuronów wyjściowych

Output Type	Output Distribution	Output Layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)
Continuous	Mixture of Gaussian	Mixture Density	Cross-entropy
Continuous	Arbitrary	See part III: GAN, VAE, FVBN	Various

Gooffellow, Deep Learning, 2016 [1]

Dobór neuronów ukrytych

- wybór funkcji realizowanych przez warstwy ukryte jest kluczowy dla przebiegu procesu uczenia
- funkcje liniowe - wiele warstw liniowych sprowadza się do pojedynczej transformacji liniowej

$$f(\mathbf{x}) = \mathbf{W}_n \mathbf{W}_{n-1} \ldots \mathbf{W}_1 \mathbf{W}_1 \mathbf{x} = \mathbf{A}\mathbf{x}$$

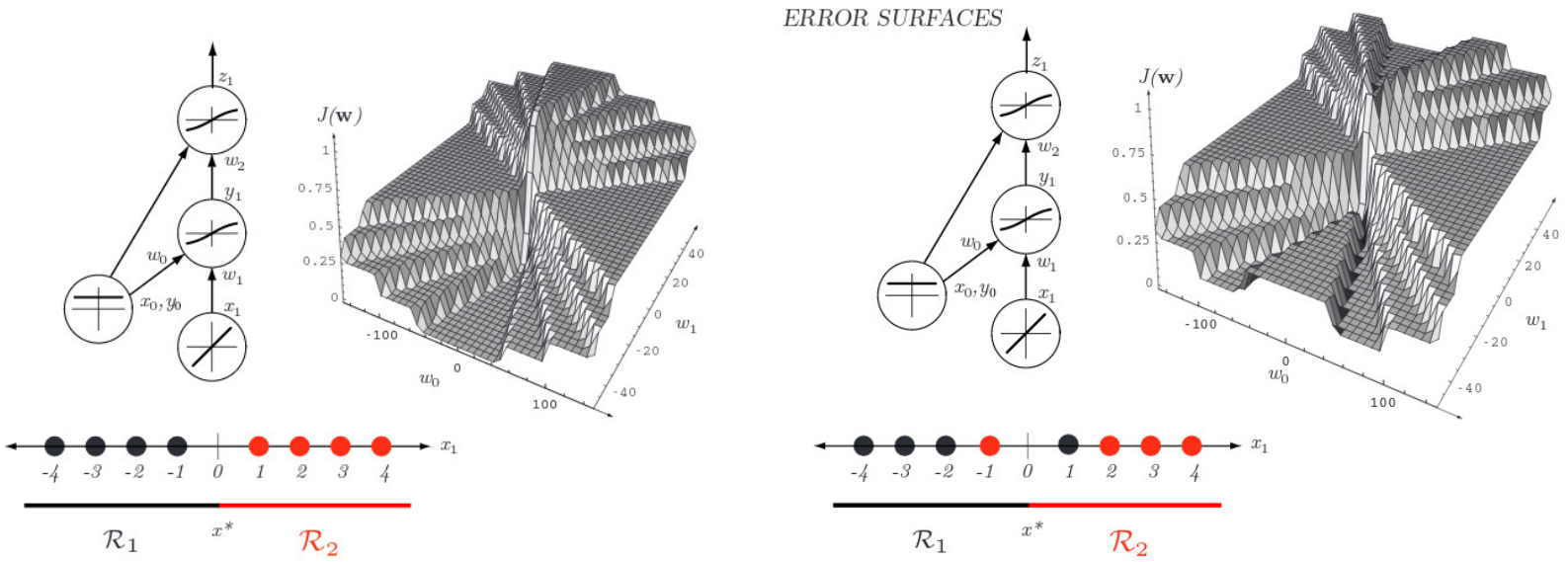
- funkcje ograniczone: sigmoidalne i tangens hiperboliczny powodują problemy z niestabilnym gradientem, uczenie może utknąć, gdy funkcja się nasyci
- obecnie najpopularniejszym typem aktywacji w sieciach jest ReLU

Demonstracje

- ☞ TensorFlow playground
- ☞ ConvNetJS - Deep Learning in your browser

Ulepszenia treningu sieci MLP

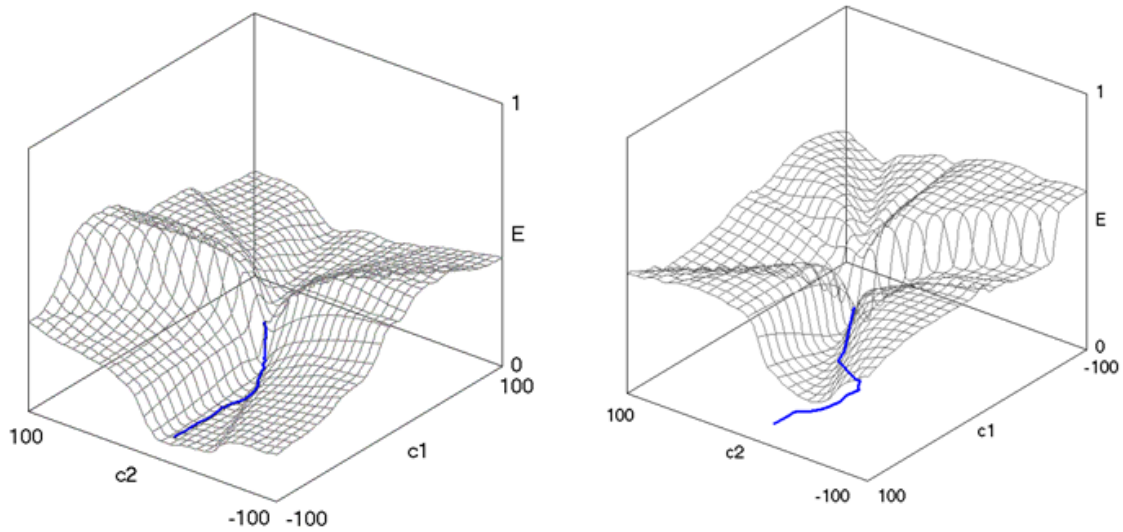
Powierzchnia błędu



- minima lokalne i minimum globalne
- „wzgórza” i „doliny” odpowiadające lokalnym minimom
- płaskie regiony, płaskowyże (*plateau*), regiony o małej zmienności błędu względem wag
- nierównomierna krzywizna, szybkie zmiany w jednych kierunkach, wolne w innych
- wysoka wymiarowość, miliony parametrów w sieciach neuronowych

Trajektorie zbieżności

Analiza PCA na macierzy kowariancji wag w kolejnych krokach iteracji



- wiele płaskowyżów i kanionów.
- dane leżące daleko od granicy mają mały wpływ na powierzchnię błędu
- przy końcu uczenia główna redukcja błędu wynika ze wzrostu wartości wag, czyli wyostrzania się sigmoidy, aż stanie się prawie skokowa

Jak omijać minima lokalne?

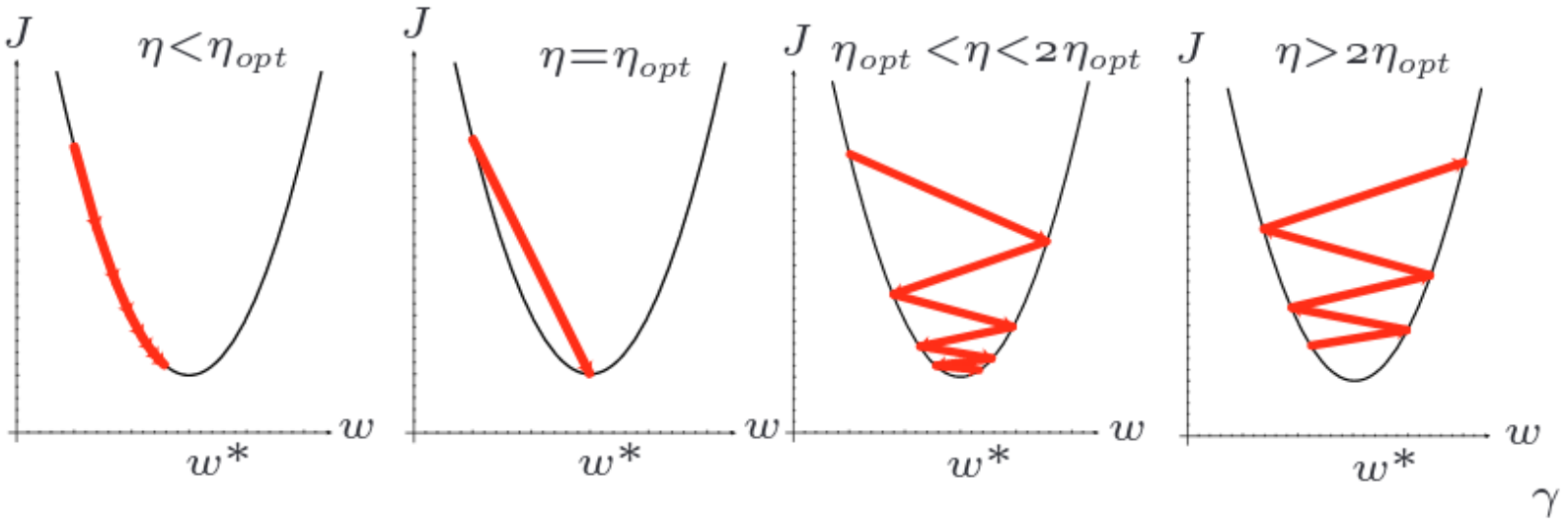
- **losowa inicjalizacja**: rozrzucanie punktów startowych w różnych regionach przestrzeni wag
- **wielokrotny start** z różnymi wartościami początkowymi - najprostsza, ale skuteczna metoda
- szum dodawany do wag lub szum dodany do danych pozwala wygładzić funkcję błędu i uciec z płytszych minimów – formalnie jest to równoważne **regularyzacji**, czyli dodaniu dodatkowego członu wygładzającego do funkcji błędu
- losowa kolejność prezentowania przypadków
- modyfikacje algorytmów gradientowych (np. momentum)
- metody optymalizacji globalnej

Metody globalnej optymalizacji

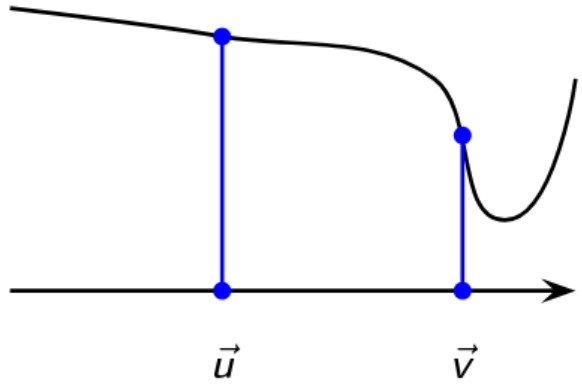
- Metody globalnej optymalizacji: wiele podejść opartych na przeszukiwaniu przestrzeni wag
- Monte Carlo, symulowane wyżarzanie, metody multisympleksowe, optymalizacja rojem cząstek (PSO), algorytmy genetyczne, minimalizacja Tabu, homotopia ...
- Dużo prac łączących algorytmy genetyczne z sieciami MLP
- Zalety: globalne, proste w realizacji, niektóre nie potrzebują gradientu, inne łączą zalety metod gradientowych z metodami globalnymi
- Wady: zwykle kosztowne obliczeniowo i czasochłonne

Dobór współczynnika uczenia η

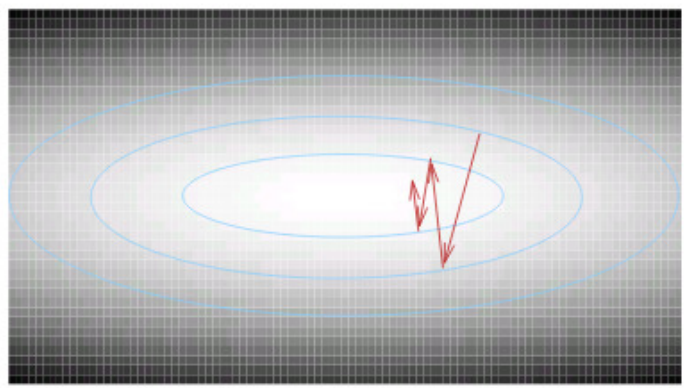
- zbyt mała wartość $\eta \rightarrow$ powolna zbieżność
- za duża wartość $\eta \rightarrow$ niestabilny proces uczenia, oscylacje
- wartość kroku może być zmieniana w czasie uczenia, różne podejścia:
  - zmniejszana w czasie treningu
  - zwiększana dla płaskich powierzchni błędu
  - dobierana indywidualnie dla każdej warstwy lub pojedynczych neuronów i wag



Płaskie powierzchnie i strome zbocza dolin



Powierzchnia błędu w wielu wymiarach ma różne nachylenie wzdłuż różnych kierunków



Rys: Riedmiller, Machine Learning

Trening z momentem

- prędkość uczenia zależna od poprzednich wartości gradientów

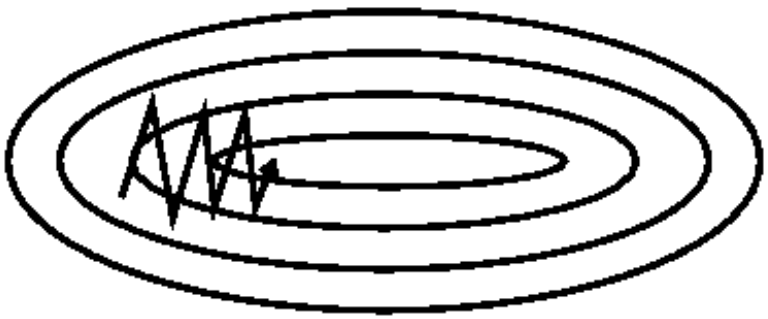
$$\Delta w(t) = -\eta \nabla E(t) + \gamma \Delta w(t-1)$$

  gdzie $0 < \gamma < 1$, typowo $\gamma = 0.9$
- czynnik *momentum* zwiększa krok uczenia, gdy gradient nie zmienia kierunku
- przyspieszenie uczenia na płaskich odcinkach (gdzie ∇E się nie zmienia) wynosi w przybliżeniu

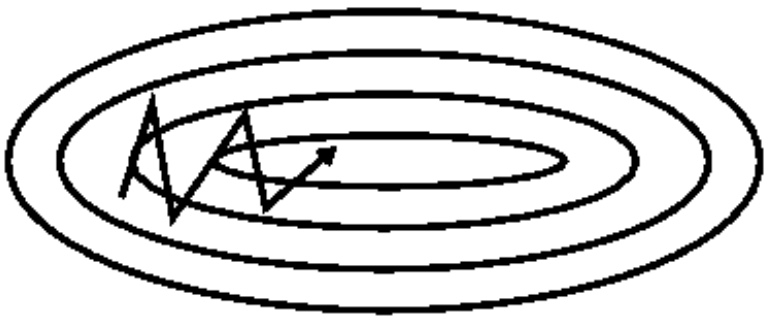
$$\Delta w(t) = -\frac{\eta}{1-\gamma} \nabla E$$

- redukuje oscylacje (spowalnia uczenie), gdy gradient zmienia kierunek
- gdy $\nabla E = 0$ wagi są nadal modyfikowane (bezwładność uczenia), może to ułatwić opuszczenie regionu „przyciągania” do minimum lokalnego

Trening z momentem



SGD bez momentu



SGD z momentem

Inicjalizacja wag

- Losowe wartości z rozkładu jednostajnego w okolicach 0
- Za duże wagi powodują duże wartości aktywacji i ryzyko nasycenia funkcji sigmoidalnych
- Dla d wejść można wybrać wartości z zakresu

$$-\frac{1}{\sqrt{d}} < w_{ij} < \frac{1}{\sqrt{d}}$$

  co przy standaryzacji danych wejściowych daje uśrednioną aktywację neuronów w zakresie liniowym $[-1, 1]$ sigmoidy
- Analogicznie można inicjować wagi w kolejnych warstwach

Skalowanie wartości wejściowych

- Odmienne zakresy wartości mierzonych cech x_i (różne skale i jednostki), rozkłady wartości zmiennych dalekie od rozkładu normalnego, wartości odstające - mogą negatywnie wpływać na proces uczenia
- Dane wejściowe $\mathbf{x}_i$ powinny mieć zbliżone zakresy wartości, aby uniknąć problemów zbieżności
- **Standaryzacja**

$$x_s = \frac{x - \mu}{\sigma}$$

  μ - wartość średnia, σ - odchylenie standardowe
- **Normalizacja** w zakresie $[-1, +1]$

$$x_n = 2\frac{x - x_{min}}{x_{max} - x_{min}} - 1$$

RPROP

Resilient BP (Riedmiller, Braun, 1992)

- heurystyczny algorytm uczenia, który ma przyspieszać proces uczenia i ograniczać problemy wynikające z bardzo małych lub bardzo dużych wartości gradientu
- aktualizacja wag $\Delta w_{ij}(t)$ zależy od znaku gradientu, a nie od jego wartości
- współczynnik uczenia η_{ij} jest dobierany dla każdej wagi w_{ij} niezależnie, jego wartość zmienia się w czasie uczenia

$$\Delta w_{ij}(t) = -\eta_{ij}(t) \cdot \mathrm{sgn}\left(\frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}}\right)$$

RPROP

- aktualizacja współczynnika zależy od zmian znaku gradientu w dwóch ostatnich krokach $t, t-1$
- η_{ij} rośnie, gdy brak zmiany znaku gradientu
- η_{ij} maleje, gdy następuje zmiana znaku gradientu

$$\Delta w_{ij}(t) = -\eta_{ij}(t) \cdot \text{sgn}\left(\frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}}\right)$$

gdzie

$$\eta_{ij}(t) = \begin{cases} \min(a \cdot \eta_{ij}(t-1), \eta_{max}) & \text{dla} \quad \frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}} \frac{\partial E(\mathbf{w}(t-1))}{\partial w_{ij}} > 0 \\ \max(b \cdot \eta_{ij}(t-1), \eta_{min}) & \text{dla} \quad \frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}} \frac{\partial E(\mathbf{w}(t-1))}{\partial w_{ij}} < 0 \\ \eta_{ij}(t-1) & \text{w pozostałych przypadkach} \end{cases}$$

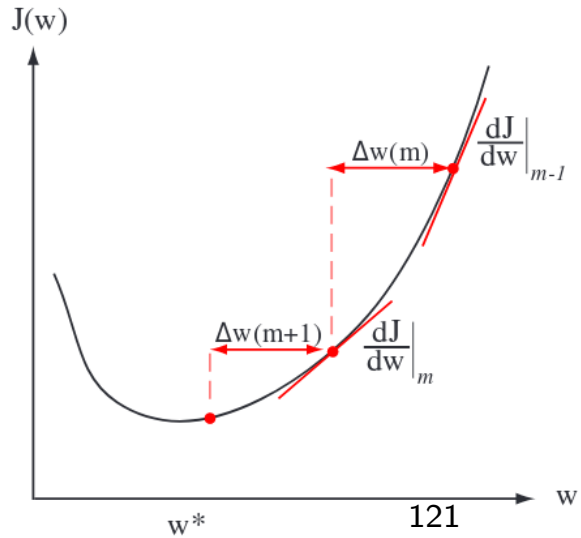
$a = 1.2, \quad b = 0.5, \quad \eta_{min} = 10^{-6}, \quad \eta_{max} = 50$

Quickprop (Fahlman, 1988)

- założenie: funkcja błędu lokalnie jest kwadratowa i może być przybliżona parabolą
- przybliżenie położenia wierzchołka paraboli za pomocą wartości wag i gradientów (nachylenia) funkcji kosztu w 2 punktach $w(t)$ i $w(t-1)$

$$\Delta w_{ij}(t) = \frac{\nabla_{ij} E(t)}{\nabla_{ij} E(t-1) - \nabla_{ij} E(t)} \Delta w(t-1)$$

- zbieżność kwadratowa (szybciej niż liniowa metoda największego spadku)
- trening może być niestabilny z powodu dużych kroków, założenia metody mogą być spełnione tylko w pobliżu minimum



Metody 2 rzędu

Rozwinięcie w szereg Taylora funkcji kosztu w sąsiedztwie $\mathbf{w}$ w kierunku $\Delta\mathbf{w}$:

$$E(\mathbf{w}+\Delta\mathbf{w}) \approx E(\mathbf{w}) + \nabla E(\mathbf{w})^T \Delta\mathbf{w} + \frac{1}{2}\Delta\mathbf{w}^T \nabla^2 E(\mathbf{w}) \Delta\mathbf{w}$$

gdzie $\nabla^2 E(\mathbf{w}) = \mathbf{H}$ jest macierzą Hessego (tzw. hesjan) zawierającą $n \times n$ pochodnych 2 rzędu

$$H_{ij} = \frac{\partial^2 E(\mathbf{x};\mathbf{w})}{\partial w_i \partial w_j}$$

stąd gradient funkcji kosztu:

$$\nabla E(\mathbf{w}+\Delta\mathbf{w})^T \approx \nabla E(\mathbf{w})^T + \Delta\mathbf{w}^T \mathbf{H}$$

Minimum funkcji wymaga aby $\nabla E(\mathbf{w}+\Delta\mathbf{w}) = 0$, stąd

$$\Delta\mathbf{w} = -\mathbf{H}^{-1} \nabla E(\mathbf{w})$$

rozwiązanie w jednym kroku dla funkcji kwadratowej, jeżeli potrafimy obliczyć $\mathbf{H}^{-1}$

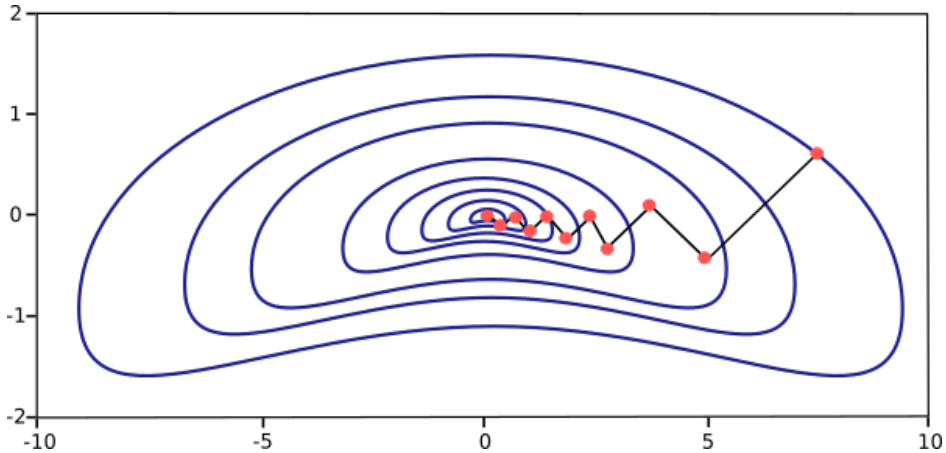
Metoda Newtona

Iteracyjna metoda 2 rzędu:

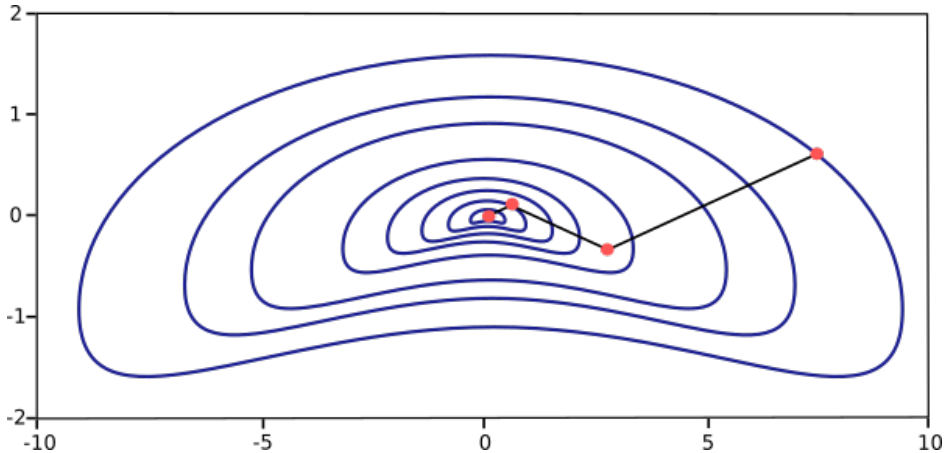
$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mathbf{H}^{-1}\nabla E(\mathbf{w}(t))$$

- metoda kosztowna czasowo $O(n^3)$, wymaga odwracania macierzy w każdej iteracji
- zbieżność kwadratowa, szybciej niż metoda gradientu prostego
- kosztowna pamięciowo, hesjan $O(n^2)$, gdzie n to liczba wag
- praktyczne zastosowania tylko dla małych sieci
- hesjan $\mathbf{H}$ musi pozostać dodatnio określony aby zapewnić zbieżność do minimum i bezpieczne odwracanie macierzy
- w praktyce nierealne jest zapewnienie dodatniej określoności hesjanu w każdym kroku iteracji, dlatego stosuje się metody przybliżone

Spadek gradientu



Metoda Newtona



Rys: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network

Metody quasi-Newtona

Przybliżenia hesjanu

- zaniedbanie pozadiagonalnych elementów

$$\Delta w_i = -\frac{\partial E}{\partial w_i} \Big/ \frac{\partial^2 E}{\partial w_i^2}$$

- metoda **zmiennej metryki** - iteracyjne przybliżenie $\mathbf{H}^{-1}$, kwadratowo zbieżna, dwie główne odmiany:
  - Davidon-Fletcher-Powell (DFP)
  - Broyden-Fletcher-Goldfarb-Shanno (BFGS)
- metoda **Levenberga-Marquardta** oparta na przybliżeniu Gaussa-Newtona

Metoda Levenberga-Marquardta

Metoda znajduje rozwiązanie zadań optymalizacji, które daje się sprowadzić do postaci

$$E = \sum_{i=1}^{p} e_i^2$$

Dla n przypadków uczących i sieci posiadającej m neuronów wyjściowych funkcja kosztu MSE zawiera sumę $p = n \cdot m$ czynników rezydualnych e_i:

$$E = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} (f_j(\mathbf{x}_i) - y_{ij})^2$$

Niech $\mathbf{e} = [e_1, e_2, \ldots, e_p]$ tworzy **wektor rezydualny** zaś **J** to macierz Jacobiego (**jakobian**) zawierająca pochodne cząstkowe wartości rezydualnych względem wag:

$$J_{ij} = \frac{\partial e_i}{\partial w_j}, \qquad i = 1, \ldots, p \qquad j = 1, \ldots, k$$

gdzie k to liczba wszystkich wag.

Metoda Levenberga-Marquardta

Gradient i hesjan funkcji błędu można wówczas przybliżyć w postaci:

$$\nabla E = 2\mathbf{J}^T\mathbf{e} \qquad \mathbf{H} = 2\mathbf{J}^T\mathbf{J}$$

Metoda zastępuje hesjan w metodzie Newtona przybliżeniem, które zawiera czynnik regularyzacyjny zapewniający dodatnią określoność hesjanu:

$$\mathbf{H} \approx 2\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}$$

gdzie parametr tłumienia $\mu > 0$ jest dostosowywany w czasie uczenia.

Reguła aktualizacji wag:

$$\Delta\mathbf{w} = -2\left(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}\right)^{-1}\mathbf{J}^T\mathbf{e}$$

Metoda Levenberga-Marquardta

$$\Delta \mathbf{w} = -2 \left(\mathbf{J}^T \mathbf{J} + \mu \mathbf{I} \right)^{-1} \mathbf{J}^T \mathbf{e}$$

- dla dużych μ czynnik kwadratowy zanika i uzyskujemy metodę gradientu prostego
- dla $\mu = 0$ mamy metodę Newtona o kwadratowej zbieżności
- algorytm uczenia startuje z dużą wartością μ, a następnie w trakcie uczenia, w miarę zbliżania się do rozwiązania, μ jest zmniejszana
- bardzo szybko zbieżna metoda dla funkcji, które są sumą kwadratów (MSE)
- nie nadaje się do zastosowań z funkcją straty Cross-Entropy
- niepraktyczne dla dużych danych, duży koszt pamięci, jakobian ma wymiar $p \times k$, gdzie $p = n \cdot m$ - liczba wektorów razy liczba wyjść, k - liczba parametrów (wag)

Metoda Levenberga-Marquardta

Algorytm doboru współczynnika μ w i-tym kroku, dla danego $\mathbf{w}(i)$:

1. oblicz wartość $\mathbf{w}(i+1)$ zgodnie z regułą uczenia LM
2. oblicz wartość błędu w punkcie $\mathbf{w}(i+1)$
3. jeśli błąd wzrósł, wróć do poprzedniej wartości $\mathbf{w}(i)$ i zwiększ wartość μ k-krotnie i wróć do kroku 1 (przybliżenie liniowe minimalizowanej funkcji w otoczeniu $\mathbf{w}(i)$ okazało się nie dość ścisłe, więc zwiększamy „wpływ" metody gradientu prostego)
4. jeśli błąd zmalał, zaakceptuj ten krok i zmniejsz wartość μ k-krotnie (założenie o liniowości minimalizowanej funkcji w otoczeniu $\mathbf{w}(i)$ okazało się wystarczająco ścisłe, więc zwiększamy „wpływ" metody Gaussa-Newtona).

Typowo $k = 10$.

Metoda gradientów sprzężonych

Metoda **gradientów sprzężonych** (*conjugate gradients*) poszukuje minimum wzdłuż kierunku $\Delta\mathbf{w}(m)$ sprzężonego (względem $\mathbf{H}$) z poprzednimi kierunkami.

Kierunki $\Delta\mathbf{w}(m)$ i $\Delta\mathbf{w}(m-1)$ są sprzężone gdy:

$$\Delta\mathbf{w}^T(m-1)\mathbf{H}\Delta\mathbf{w}(m) = 0$$

Kierunek wyszukiwania minimum w iteracji m wyznaczamy z reguły

$$\Delta\mathbf{w}(m) = -\nabla E(m) + \beta(m)\Delta\mathbf{w}(m-1)$$

gdzie **współczynnik sprzężenia** β kumuluje informacje o kierunkach z poprzednich iteracji $m-1, ..., 1$, zapewniając warunek sprzężenia.

Metoda gradientów sprzężonych

Przykładowe postaci współczynnika sprzężenia:

- reguła Fletchera-Reevesa

$$\beta(m) = \frac{\nabla E(m)^T \nabla E(m)}{\nabla E(m-1)^T \nabla E(m-1)}$$

- reguła Polaka-Ribiera

$$\beta(m) = \frac{\nabla E(m)^T (\nabla E(m) - \nabla E(m-1))}{\nabla E(m-1)^T \nabla E(m-1)}$$

Algorytm metody gradientów sprzężonych

Pierwszy kierunek

1. zainicjuj $\mathbf{w}(0)$ i znajdź pierwszy kierunek $\Delta\mathbf{w}(0)$ metodą największego spadku $\Delta\mathbf{w}(0) = -\nabla E(0)$
2. wykonaj liniowe przeszukiwanie wzdłuż $\Delta\mathbf{w}(0)$ $\alpha = \arg\min_\alpha E(\mathbf{w}(0) + \alpha\Delta\mathbf{w}(0))$
3. zaktualizuj wagi $\mathbf{w}(1) = \mathbf{w}(0) + \alpha\Delta\mathbf{w}(0)$

Kolejne kierunki $m = 1, ..., d$

1. wyznacz kierunek $\Delta\mathbf{w}(m) = -\nabla E(m)$ metodą największego spadku w punkcie $\mathbf{w}(m)$
2. wyznacz współczynnik sprzężenia $\beta(m)$
3. zaktualizuj kierunek czyniąc go kierunkiem sprzężonym $\Delta\mathbf{w}(m) = -\nabla E(m) + \beta(m)\Delta\mathbf{w}(m-1)$
4. wykonaj liniowe przeszukiwanie wzdłuż $\Delta\mathbf{w}(m)$ $\alpha = \arg\min_\alpha E(\mathbf{w}(m) + \alpha\Delta\mathbf{w}(m))$
5. zaktualizuj wagi $\mathbf{w}(m+1) = \mathbf{w}(m) + \alpha\Delta\mathbf{w}(m)$

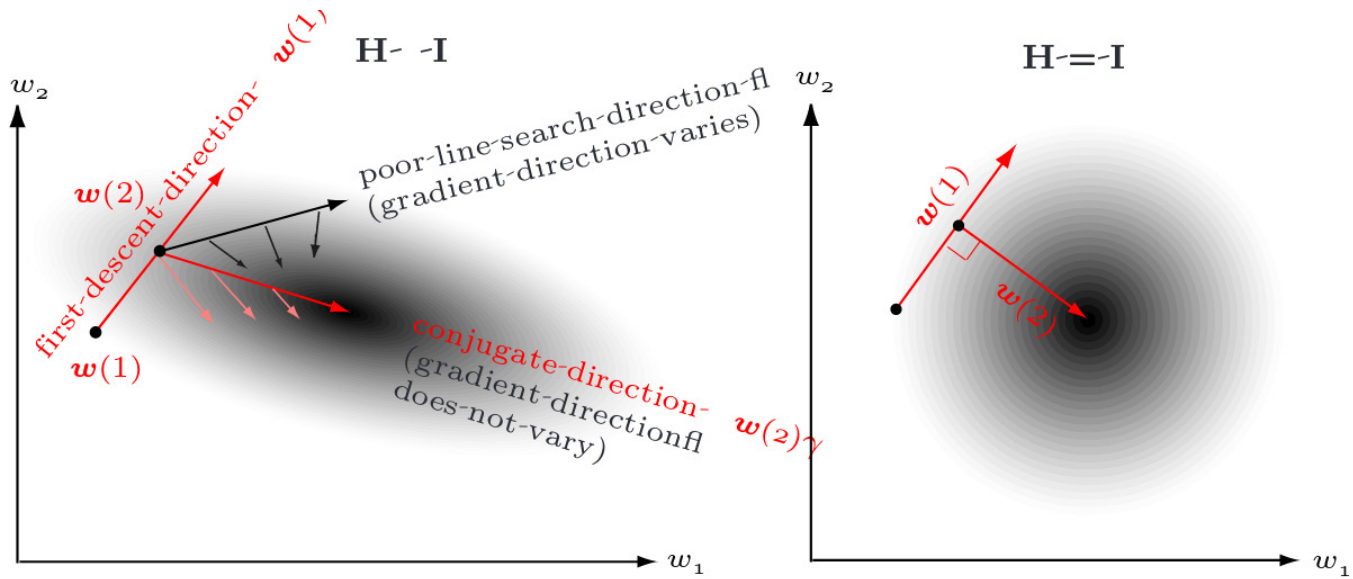
Metoda gradientów sprzężonych

- na skutek przybliżeń w kolejnych iteracjach kierunki mogą zatracić ortogonalność, dlatego po n krokach (gdzie n jest liczbą optymalizowanych parametrów) metodę inicjujemy ponownie startując z ostatnio znalezionego punktu
- dla kwadratowej funkcji kosztu w n wymiarach metoda CG osiąga minimum w n krokach
- zbieżność liniowa, lecz znacznie szybsza od spadku gradientem, brak konieczności wyznaczania odwrotności hesjanu $\mathbf{H}^{-1}$
- małe zapotrzebowanie pamięciowe, stosunkowo mała złożoność obliczeniowa pozwala stosować do bardzo złożonych problemów optymalizacyjnych

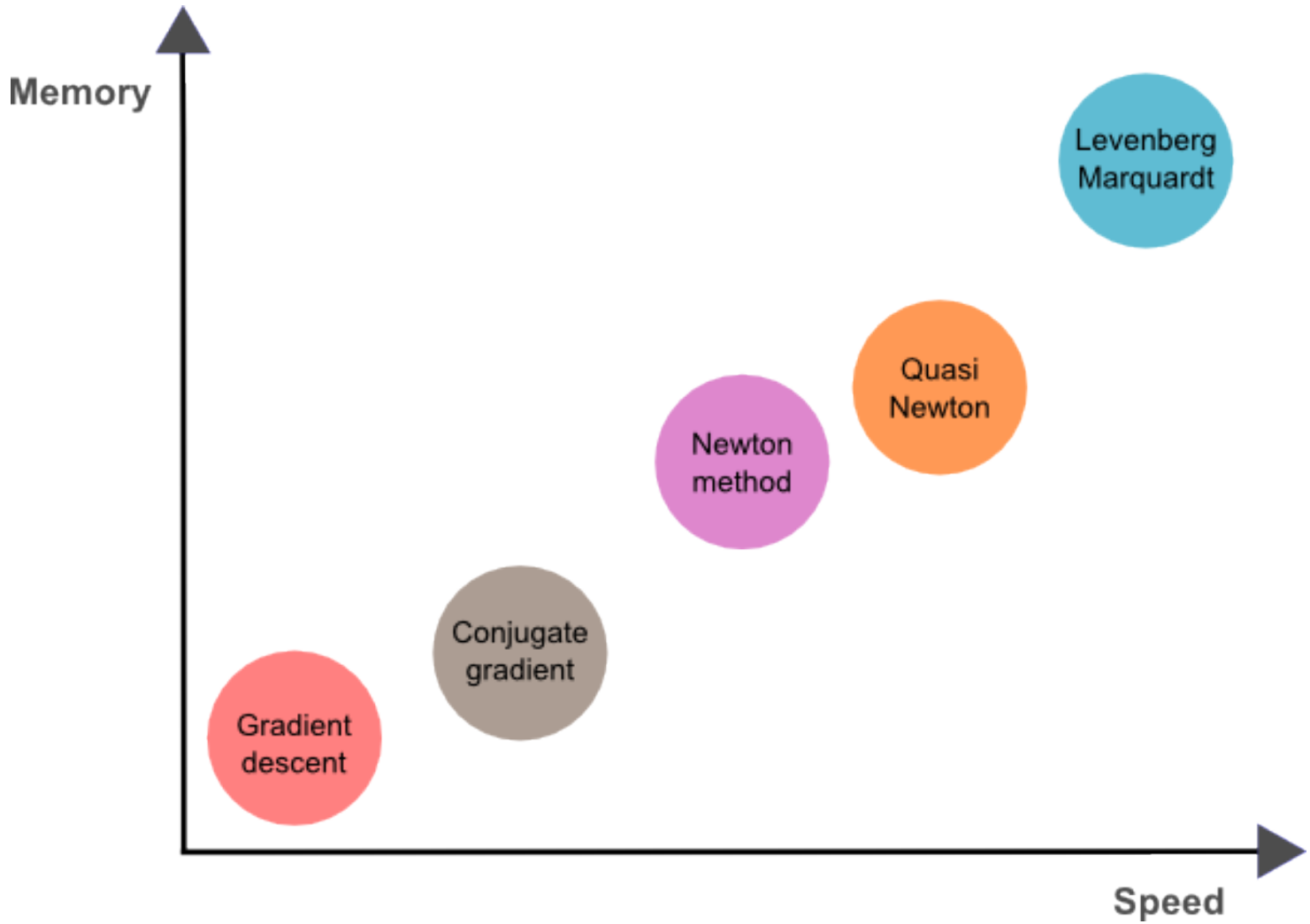
https://en.wikipedia.org/wiki/Nonlinear_conjugate_gradient_method

Minimalizacja CG

- kolejny kierunek sprzężony nie wpływa ujemnie na wartość błędu wzdłuż poprzednio wyszukanych kierunków
- dla **H** diagonalnej kolejne kierunki są ortogonalne względem siebie



Duda, Hart, Pattern recognition



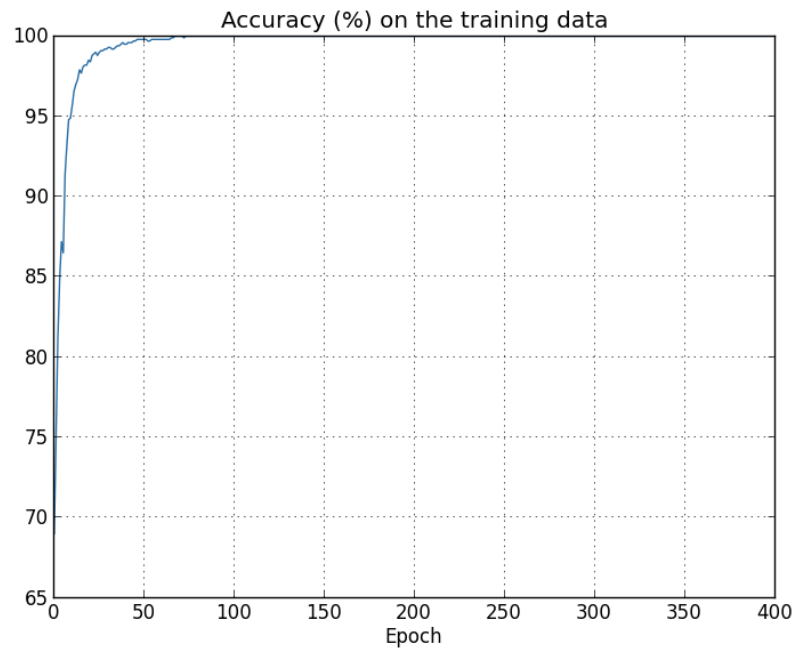
Rys: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network

Generalizacja i regularyzacja sieci MLP

Generalizacja

Wartość funkcji błędu dla zbioru treningowego może osiągnąć 0.
Czy to znaczy, że mamy najlepszy model?



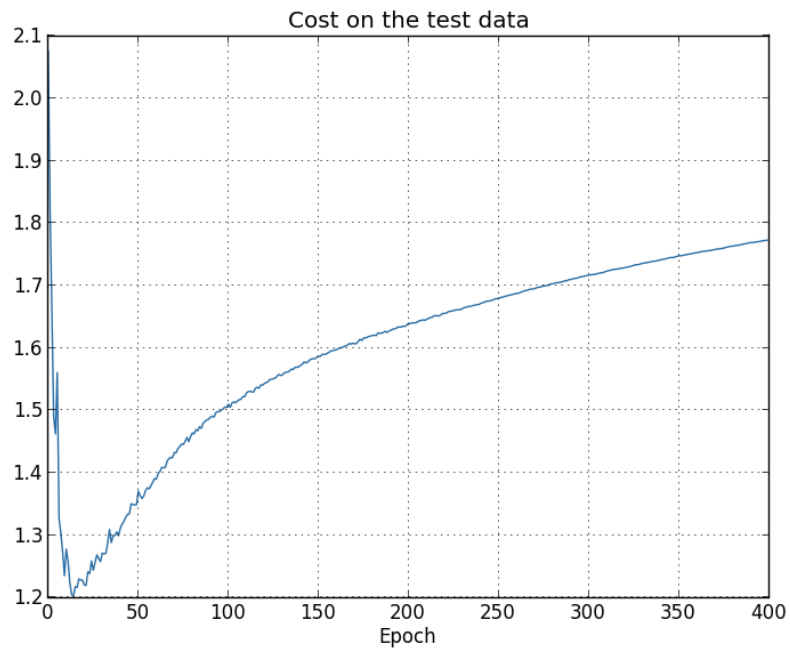


Rys: http://neuralnetworksanddeeplearning.com/

Generalizacja

Wartość funkcji błędu dla zbioru treningowego może osiągnąć 0.
Czy to znaczy, że mamy najlepszy model?

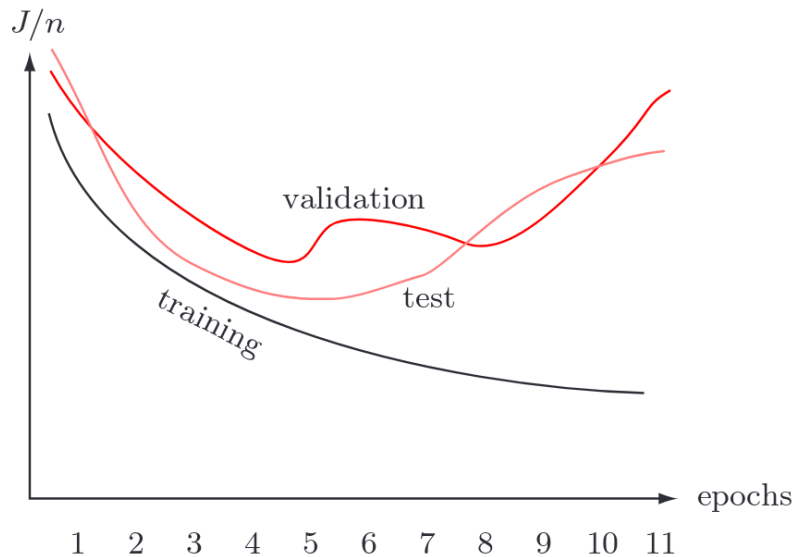




Rys: http://neuralnetworksanddeeplearning.com/

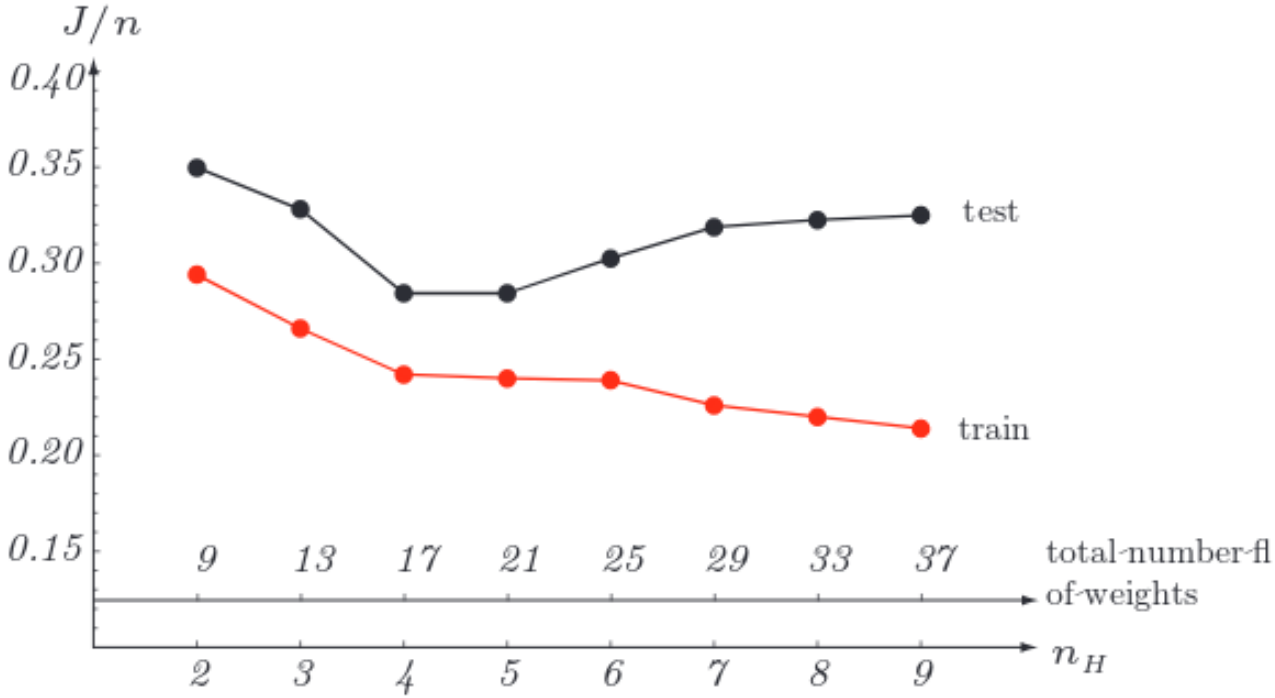
Generalizacja

- Celem uczenia jest **generalizacja**, zdolność modelu do uogólniania wiedzy pozwalającej przewidywać wyniki dla nowych, nieznanych danych
- W praktyce dążymy do minimalizacji błędu na danych, które nie zostały użyte w treningu (tzw. **zbiór testowy**)
- Zbiór **walidacyjny** to wyodrębniony fragment danych (najczęściej ze zbioru uczącego) pozwalający oszacować błąd generalizacji w trakcie procesu uczenia.
  Jest przydatny przy określaniu punktu zatrzymania treningu.



Duda, Hart, Pattern Recognition

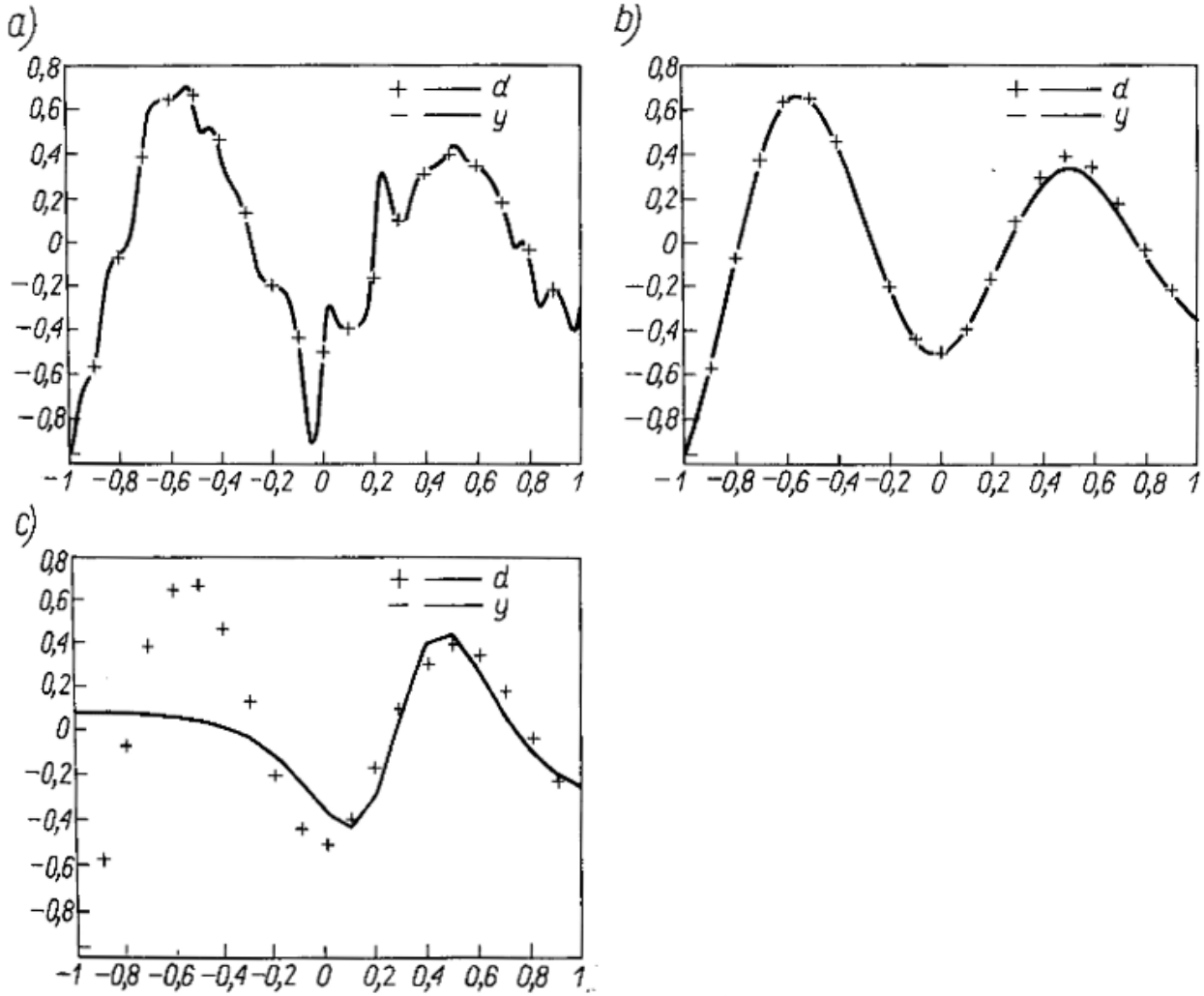
Rozmiar sieci



- liczba wag określa liczbę stopni swobody - liczba próbek uczących powinna być zwykle wielokrotnością liczby parametrów, (np. 10-krotnością liczby wag)
- za dużo wag $\rightarrow$ złożony model „uczy się na pamięć” (przeuczenie)
- za mało wag $\rightarrow$ model zbyt prosty w stosunku do problemu (niedouczony)

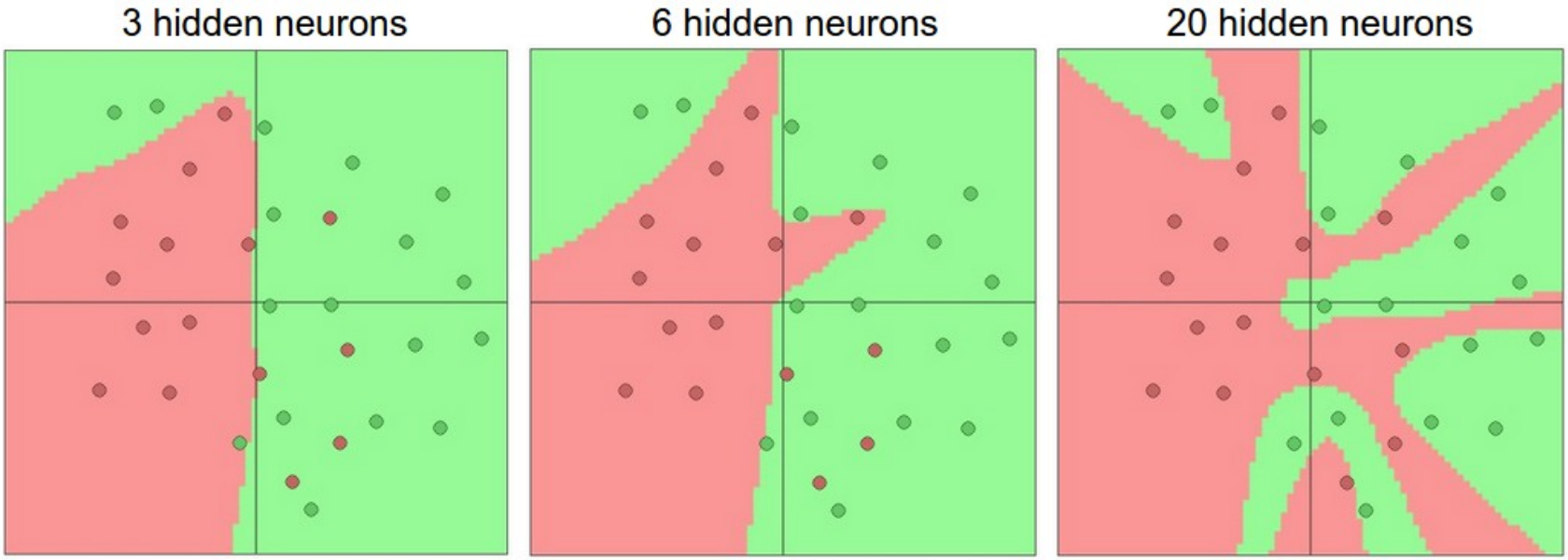
Rozmiar warstwy ukrytej

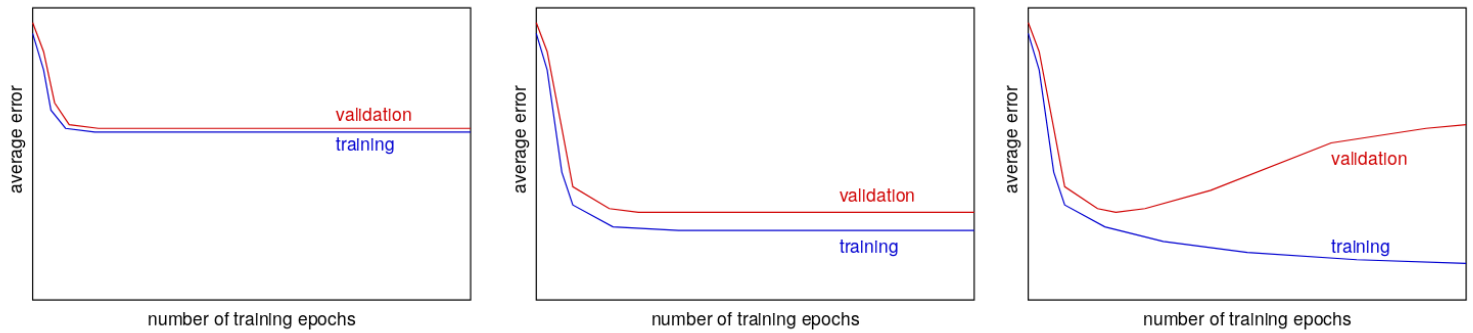
Zdolność uogólniania w problemie aproksymacji



Rozmiar warstwy ukrytej

Zdolność uogólniania w problemie klasyfikacji





- niedouczenie - błąd treningowy i walidacyjny pozostają duże
- przeuczenie - błąd walidacyjny rośnie a błąd treningowy maleje

Rys: Riedmiller, Machine Learning

Metody regularyzacji sieci neuronowych

Regularyzacja to metody, których celem jest poprawienie generalizacji

- dobór wielkości modelu (ilości neuronów i wag), preferowane są najprostsze rozwiązania (brzytwa Ockhama)
  - spośród wielu modeli wybierz ten o najmniejszym błędzie walidacyjnym i najmniejszej liczbie parametrów (neuronów)
  - **pruning** - usuwanie nadmiarowych połączeń lub neuronów z wytrenowanej sieci
  - metody **konstrukcyjne** (rosnące) - zacznij od najprostszego modelu i zwiększaj jego złożoność w kolejnych iteracjach
  - sieci **ontogeniczne** - rozrastające się i kurczące się w czasie treningu
- wymuszanie małych absolutnych wartości wag lub ich zanikanie
  - wagi zerowe oznaczają brak połączenia
  - neurony sigmoidalne z małymi wagami są w przybliżeniu liniowe

Regularyzacja

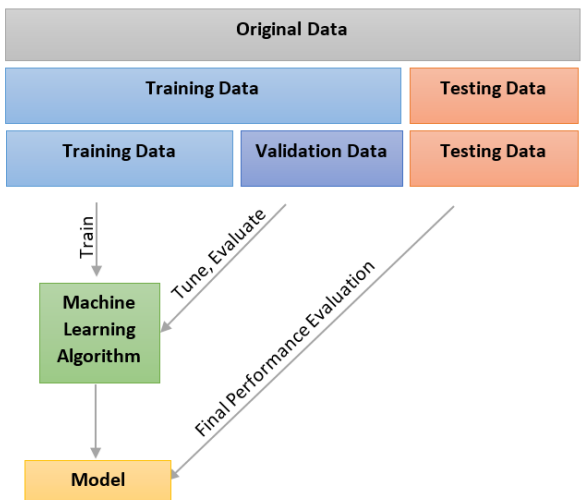
- modyfikacje funkcji kosztu wymuszające odpowiednią strukturę sieci, np. wymuszające minimalizację wartości wag

$$\bar{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda\Omega(\mathbf{w})$$

  gdzie $\lambda > 0$ steruje wielkością regularyzacji
- **wczesne zatrzymanie** (*early stopping*) - strategia polegająca na przerwaniu procesu uczenia zanim wystąpi przeuczenie (np. gdy błąd walidacyjny zacznie rosnąć)
- zwiększenie rozmiaru zbioru uczącego - więcej danych pozwala na lepsze dopasowanie modelu do problemu
- przetransformowanie danych do takiej postaci, która zwiększy szansę znalezienie optymalnego rozwiązania (ekstrakcja cech, normalizacja, itp.)
- selekcja cech - wybór tylko istotnych zmiennych do treningu
- dodanie szumu do wag lub do danych uczących
- uśrednianie wyników z wielu modeli, np.: boosting, bagging

Trening i walidacja

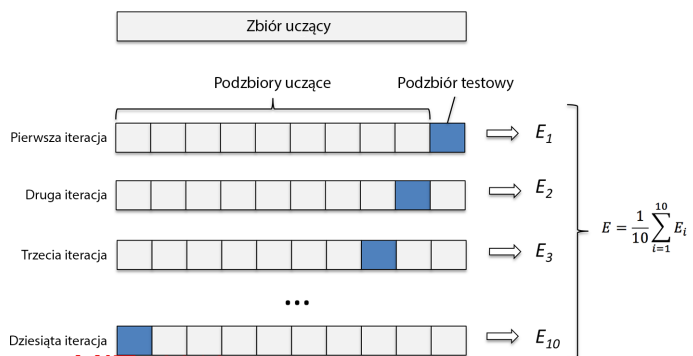
- uczenie sieci minimalizuje błąd treningowy a nie błąd generalizacji
- **zbiór walidacyjny** pozwala oszacować błąd generalizacji w trakcie treningu. Wydzielany ze zbioru treningowego, więc powoduje zmniejszenie liczby dostępnych próbek uczących.
- **zbiór testowy** używany wyłącznie do ewaluacji nauczonego modelu



Rys. S. Mostafa Eissa, ☞ Introduction to Machine Learning , 2016

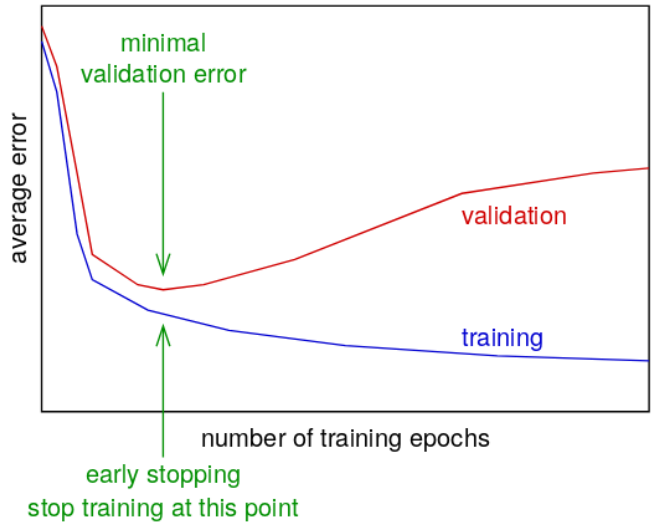
Walidacja krzyżowa (kroswalidacja)

- **Kroswalidacja** (walidacja krzyżowa) - metoda pozwalająca na oszacowanie błędu generalizacji i wariancji modelu poprzez powtarzanie treningu i ewaluacji na kolejnych podziałach zbioru uczącego
- k-krotna kroswalidacja - dzieli zbiór na k części, gdzie $k-1$ części jest używanych do treningu a reszta do testu
- kroswalidacja **leave-one-out** (LOO) - rodzaj kroswalidacji, gdzie N jest równe liczbie wektorów uczących, tj. trening odbywa się na $N-1$ przypadkach, predykcja dotyczy pojedynczego przypadku. Stosowane dla bardzo małych danych.
- średni błąd wyznaczany jest z k treningów (foldów)



Rys. S. Raschka, Python Machine Learning, MIT, 2019

Wczesne zatrzymanie



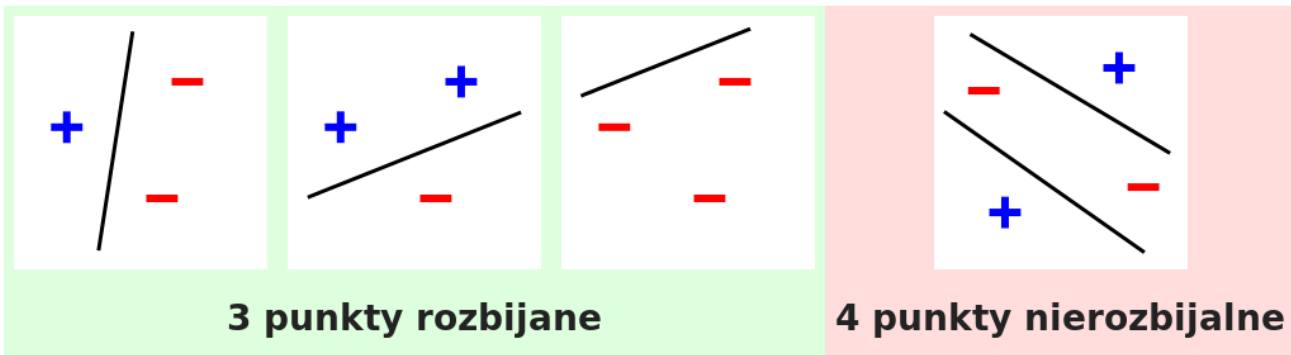
Strategie:

- przerwij trening w momencie, gdy błąd walidacyjny zacznie rosnąć (np. gdy brak poprawy przez k epok). Często wystarczy tylko kilka iteracji.
- trenuj przez N epok i zapamiętaj model o najmniejszym błędzie walidacyjnym wyznaczonym po każdej epoce

Rys: Riedmiller, Machine Learning

Wymiar Vapnika-Chervonenkisa

Wymiar $VC_{dim}(\mathbb{H})$ w klasie hipotez $\mathbb{H}$ to największa liczba przypadków uczących, którą $\mathbb{H}$ jest w stanie *rozbić*, tj. podzielić przy dowolnej kombinacji etykiet (2^N możliwości w przypadku 2 klas)



Dla perceptronu z 2 wejściami $y = w_1 x_1 + w_2 x_2 + b$

$$\mathrm{VC}_{dim}(\mathrm{Perceptron}(\mathbb{R}^2)) = 3$$

W przestrzeni $\mathbb{R}^d$ mamy $(d+1)$ parametrów stąd

$$\mathrm{VC}_{dim}(\mathit{Perceptron}(\mathbb{R}^d)) = d+1$$

Twierdzenie Vapnika-Chervonenkisa (Vapnik, 1995)

Błąd generalizacji $E_G(h)$ hipotezy h pochodzącej z klasy hipotez $\mathbb{H}$ o skończonym wymiarze VC_{dim} wytrenowany na N przypadkach uczących jest ograniczony przez sumę błędu na zbiorze uczącym $E_T(h)$ oraz wyrażenia zależnego od liczby próbek N i wymiaru VC_{dim}

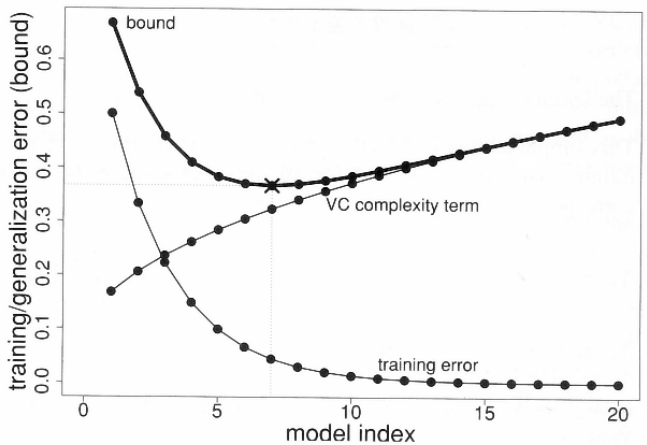
$$E_G(h) \leq E_T(h) + \sqrt{\frac{\mathrm{VC}_{dim}(\mathbb{H}) \cdot \left(1 + \log\left(\frac{2N}{\mathrm{VC}_{dim}(\mathbb{H})}\right)\right) - \log(\frac{\delta}{4})}{N}}$$

z prawdopodobieństwem $1 - \delta$ jeżeli $\mathrm{VC}_{dim}(\mathbb{H}) \ll N$.

V. Vapnik (2000) The nature of statistical learning theory

Minimalizacja ryzyka strukturalnego

Structural risk minimization (SRM) (Vapnik) zasada równoważenia złożoności modelu względem jego zdolności do dopasowywania się do danych



- **im bardziej złożony model tym więcej potrzeba danych uczących do uzyskania satysfakcjonującego błędu generalizacji**
- jeśli znamy VC_{dim} klasyfikatora to możemy oszacować ile próbek uczących jest potrzebnych aby uzyskać satysfakcjonujący błąd generalizacji

Rys: https://julien-vitay.net/lecturenotes-neurocomputing/2-linear/6-LearningTheory.html

Dobór złożoności sieci

Rule of thumb

$$E_G(\mathbf{w}) \approx \frac{\mathrm{VC}_{dim}}{N}$$

- prawdopodobieństwo przeuczenia rośnie, gdy $\mathrm{VC}_{dim} > N$
- złożoność VC_{dim} sieci neuronowej jest proporcjonalna do liczby wag N_w
  $\mathrm{VC}_{dim} \approx N_w$ dla MLP z binarnymi funkcjami aktywacji
  $\mathrm{VC}_{dim} \approx 2N_w$ dla MLP z sigmoidalnymi funkcjami aktywacji
- regularyzacja zmniejsza VC_{dim} modelu
- dla złożonych modeli wyznaczenie VC_{dim} jest trudne
- w praktyce: minimalna liczba próbek uczących powinna być co najmniej 10-krotnością wymiaru VC

$$N \geq 10 VC_{dim}$$

Regularyzacja L_2

Regularyzacja L_2 (weight decay, regularyzacja Tichonova) - dąży do zmniejszenia wartości wag

$$\hat{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\lambda \sum_{ij} w_{ij}^2$$

gdzie $\lambda > 0$ decyduje o sile regularyzacji (typowo $\lambda = 0.01$ lub 0.001)

Zmniejszane są wszystkie wagi w trakcie uczenia.
Korekta wag $\hat{w}_{ij}$ w stosunku do wersji nieregularyzowanej w_{ij}

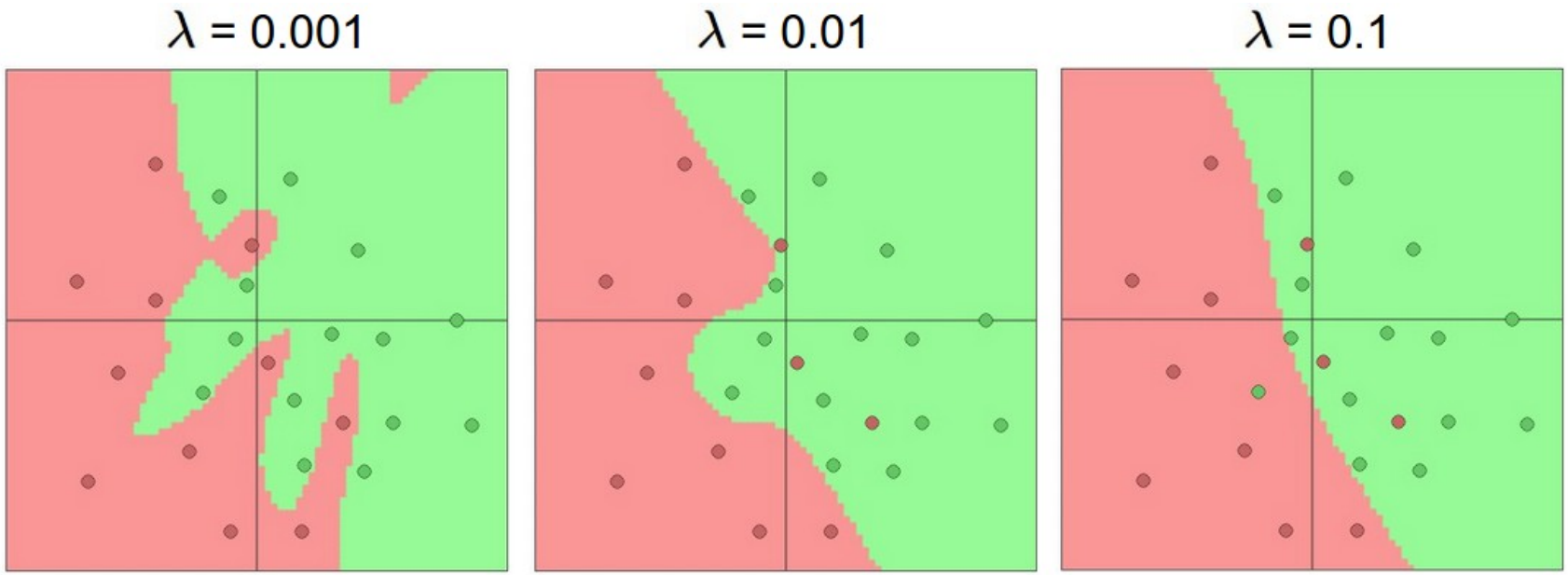
$$\hat{w}_{ij} = (1 - \lambda\eta) w_{ij}$$

$$\mathbf{w}(n+1) = \hat{\mathbf{w}}(n) - \eta \nabla E(\mathbf{w})$$

Możliwe modyfikacje:

- wagi, które zmalały poniżej pewnego progu mogą być usunięte
- usuwamy neurony dla których $\sum |w_i|$ jest bliskie zeru

Regularyzacja



Rys: http://cs231n.github.io/neural-networks-1/

Zmodyfikowana regularyzacja L_2

W przypadku regularyzacji L_2 wartości wszystkich wag maleją. Poniższa modyfikacja zachowuje wagi o dużych wartościach

$$\hat{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\lambda \sum_{ij} \frac{w_{ij}^2}{1 + w_{ij}^2}$$

Korekta wag $\hat{w}_{ij}$ w stosunku do wersji nieregularyzowanej w_{ij}

$$\hat{w}_{ij} = \left(1 - \lambda\eta \frac{1}{\left(1 + w_{ij}^2\right)^2} \right) w_{ij}$$

- dla małych wag $w_{ij} \ll 1$ równoważne z L_2
- dla dużych wag $w_{ij} \gg 1$ regularyzacja zanika

Zmodyfikowana regularyzacja L_2

Zmodyfikowany człon kary wykorzystywany do eliminacji niepotrzebnych neuronów w warstwie:

$$\hat{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\lambda \sum_{ij} \frac{w_{ij}^2}{1 + \sum_k w_{ik}^2}$$

Minimalizacja E powoduje:

- eliminacje połączeń (wagi zanikają)
- eliminację neuronów dla których $\sum_k |w_{ik}|$ jest bliskie zera

Regularyzowana zmiana wag zależna jest od wartości w_{ij}

$$\hat{w}_{ij} = \left(1 - \lambda\eta \frac{1 + 2\sum_{k \neq j} w_{ik}^2}{\left(1 + \sum_k w_{ik}^2\right)^2} \right) w_{ij}$$

- małe wartości wag w_{ik} prowadzące do i-tego neuronu zanikają
- duże wartości wag $\sum_k |w_{ik}|$ neuronu niwelują wpływ czynnika korekcyjnego

Regularyzacja L_1

$$\hat{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda \sum_{ij} |w_{ij}|$$

$$\nabla \hat{E}(\mathbf{w}) = \nabla E(\mathbf{w}) + \lambda \operatorname{sgn}(\mathbf{w})$$

regularyzacja wpływa na gradient ze stałą wartością (zależy tylko od znaku w_{ij})

$$\hat{w}_{ij} = \begin{cases} w_{ij} - \lambda\eta & \text{dla} \quad w_{ij} > 0 \\ w_{ij} + \lambda\eta & \text{dla} \quad w_{ij} < 0 \end{cases}$$

Regularyzacja L_1 daje rzadsze rozwiązanie od L_2, tzn. dąży do rozwiązania o mniejszej liczbie niezerowych wag

Metody wrażliwościowe redukcji sieci

- Eliminacja wag na podstawie ich wpływu na wartość błędu
- Wagi o małej wartości $|w_i|$ są mniej istotne od dużych wag
- Wrażliwość połączenia wyznaczana dla nauczonej sieci o błędzie E

  $$S_{ij} = E - E(w_{ij} = 0)$$

  gdzie $E(w_{ij} = 0)$ to błąd sieci po usunięciu w_{ij}
- Połączenia o najmniejszej wrażliwości usuwa się po czym sieć jest douczana
- Usuwamy neuron dla którego wszystkie dochodzące (lub wychodzące) połączenia są wyzerowane

Optimal Brain Damage (LeCun)

Założenie: hesjan $\mathbf{H}$ jest diagonalnie dominujący, więc uwzględniamy tylko składową diagonalną.

Z rozwinięcia funkcji błędu E w szereg Tylora i przyjmując $\nabla E = 0$ (punkt zbieżności dla wytrenowanej sieci) otrzymujemy oszacowanie zmiany błędu:

$$\Delta E \approx \frac{1}{2}\Delta \mathbf{w}^T \mathbf{H} \Delta \mathbf{w} = \frac{1}{2}\sum_i H_{ii}\Delta w_i^2$$

Współczynnik istotności wagi

$$S_i = \frac{1}{2}\frac{\partial^2 E}{\partial w_i^2} w_i^2$$

Wagi o najmniejszym S_i mogą być usunięte z wytrenowanej sieci.

Optimal Brain Damage

Algorytm redukcji sieci OBD

1. Wstępne uczenie: sieć jest w pełni uczona na dostępnych danych, aż osiągnie lokalne minimum funkcji celu
2. Wyznacz diagonalne elementy hesjanu oraz wartości wrażliwości S_i dla każdej wagi w_i
3. Posortuj wagi zgodnie z rosnącymi wartościami S_i i obetnij te o najmniejszych wartościach (zazwyczaj 1-5% wag)
4. Wróć do punktu 1

Redukcja neuronów o małej aktywności

- Neurony o niewielkiej aktywności (których sygnał wyjściowy nie zmienia się dla całego zbioru treningowego) można usunąć bez szkody dla generalizacji
- duża aktywność neuronu świadczy o jego przydatności
- modyfikacja funkcji kosztu z czynnikiem kary za zbyt małą aktywność neuronów (Y. Chaurin)

$$\hat{E} = E + \lambda \sum_{i=1}^{k} \sum_{j=1}^{n} e(\Delta_{ij}^2)$$

gdzie Δ_{ij} oznacza zmianę sygnału wyjściowego i-tego neuronu dla j-tego wektora treningowego

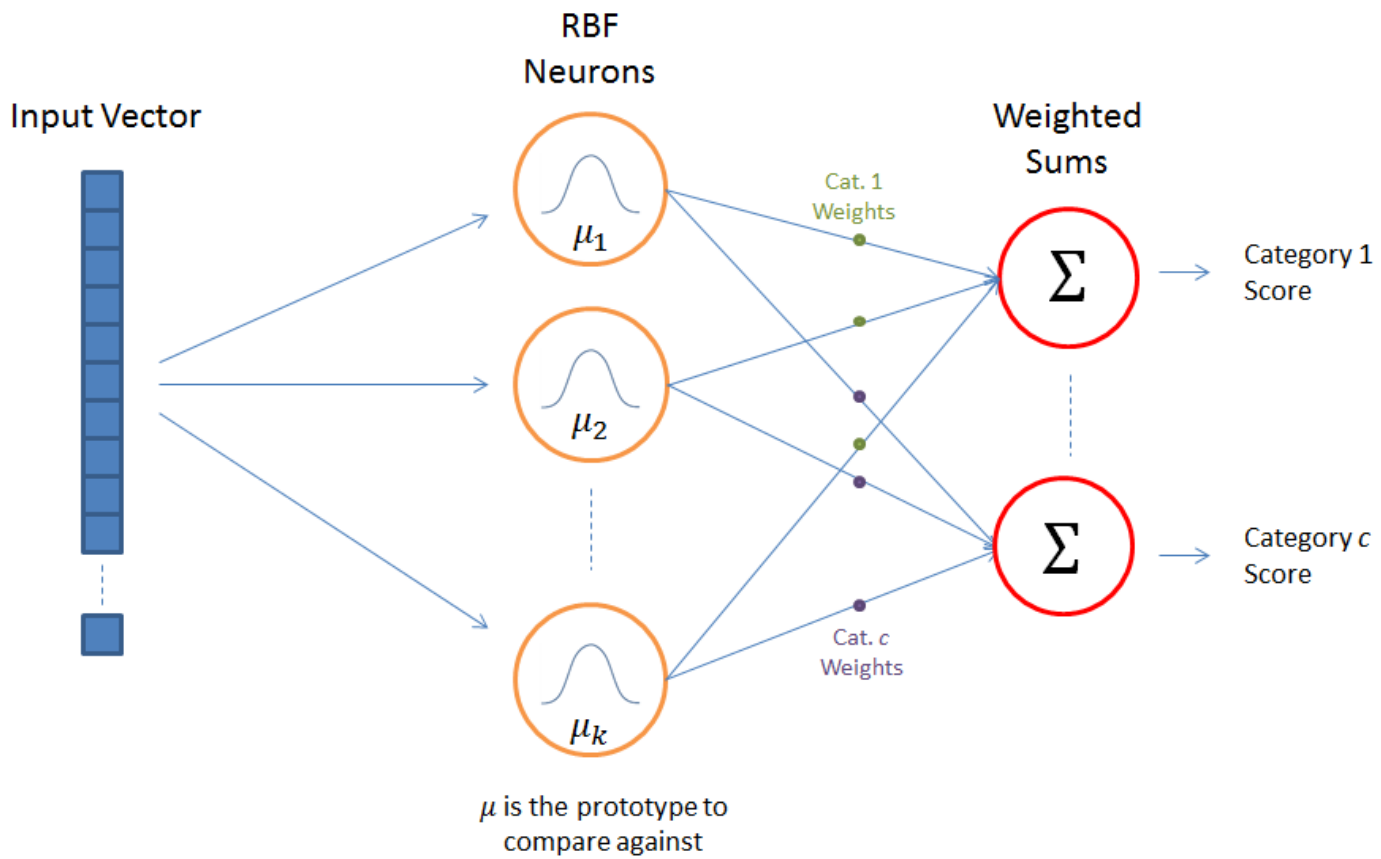
- przykładowy czynnik korelacyjny

$$e = \frac{1}{1 + \Delta_{ij}^2}$$

RBF Radial Basis Functions

Sieci neuronowe z radialnymi funkcjami bazowymi

Radial Basis Function Network



Rys: Chris McCormick, Radial Basis Function Network (RBFN) Tutorial

Sieć RBF

Funkcja realizowana przez sieć RBF z jednym wyjściem:

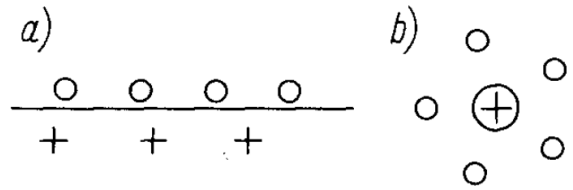
$$f(\mathbf{x}) = \sum_{i=1}^{K} w_i \varphi(\|\mathbf{x} - \mathbf{c}_i\|) + w_0$$

gdzie K - liczba funkcji bazowych z centrami w $\mathbf{c}_i$

- jedna warstwa ukryta, neurony z radialnymi funkcjami
- parametry neuronu ukrytego $\mathbf{c}_i$ definiują centrum funkcji bazowej φ_i; odpowiadają wagom połączeń neuronów ukrytych
- funkcja bazowa (radialna) φ określa podobieństwo przypadku $\mathbf{x}$ do centrum (prototypu) $\mathbf{c}_i$
- wyjścia liniowe - ważona suma pobudzeń neuronów radialnych

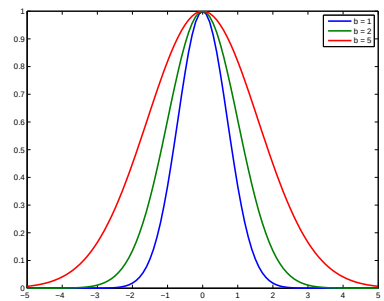
Funkcje radialne

- funkcje radialne mapują relacje lokalne, gdzie perceptrony (np. neurony sigmoidalne) mapują relacje globalne



- argumentem funkcji radialnej jest odległość od pewnego centrum (od prototypu) $\|\mathbf{x} - \mathbf{c}\|$. Zazwyczaj stosuje się metrykę euklidesową.
- najczęściej stosowane funkcje radialne $\varphi(x)$ posiadają maksimum w $x = 0$ i zanikają w obszarach dalekich od $x = 0$, np. funkcja Gaussa

$$\varphi(x) = e^{-\frac{x^2}{\sigma}}$$



- współczynnik dyspersji σ decyduje o zasięgu funkcji bazowej

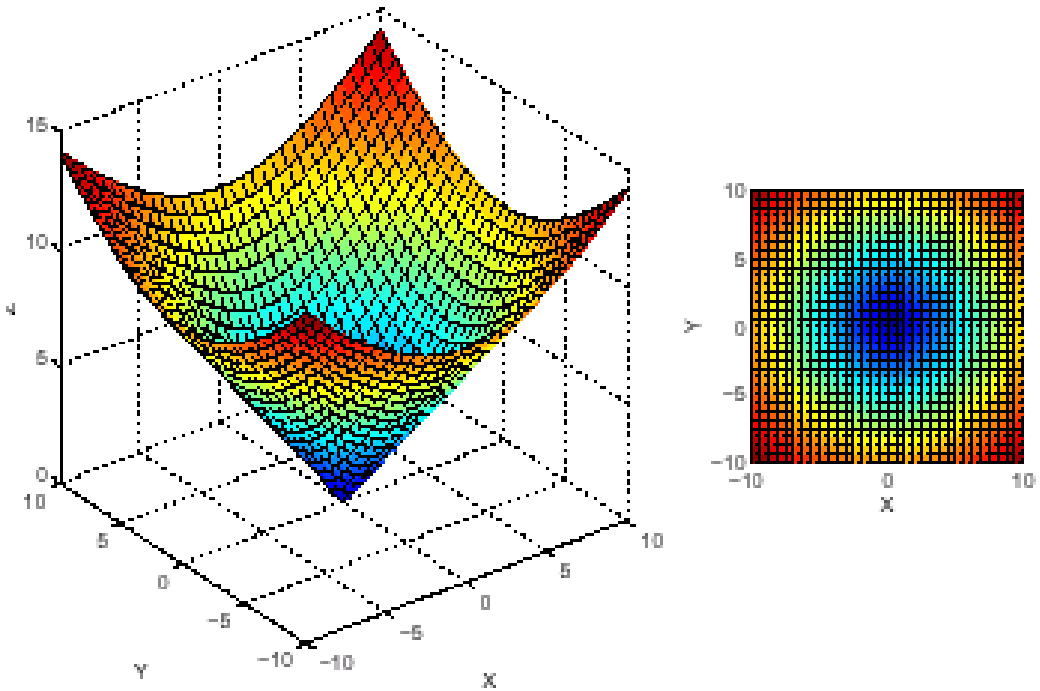
Przykłady funkcji radialnych

Niech $r = \|\mathbf{x} - \mathbf{c}_i\|$

$h(r) = r$		liniowa
$h(r) = e^{-(\frac{r}{\sigma})^2}$		gaussowska
$h(r) = (\sigma^2 + r^2)^\beta,$	$0 < \beta < 1$	*multiquadratic* (wielokwadratowa)
$h(r) = (\sigma^2 + r^2)^{-\alpha},$	$\alpha > 0$	*inverse multiquadratic*
$h(r) = (\sigma r^2) \ln(\sigma r)$		*thin-plate spline* (cienkiej płytki)
$h(r) = \sigma^2 + r^2$		wielomianowa

Funkcja liniowa współrzędnej radialnej

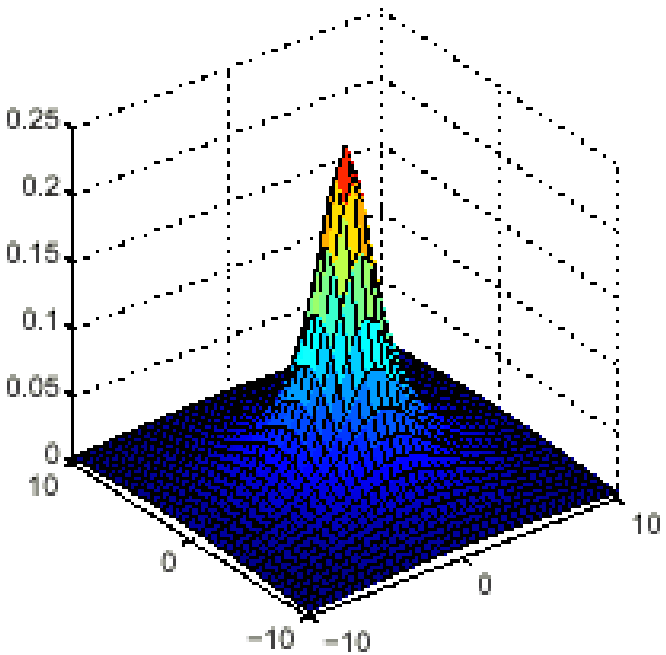
$$h_i(r) = r = \|\mathbf{x} - \mathbf{c}_i\|$$

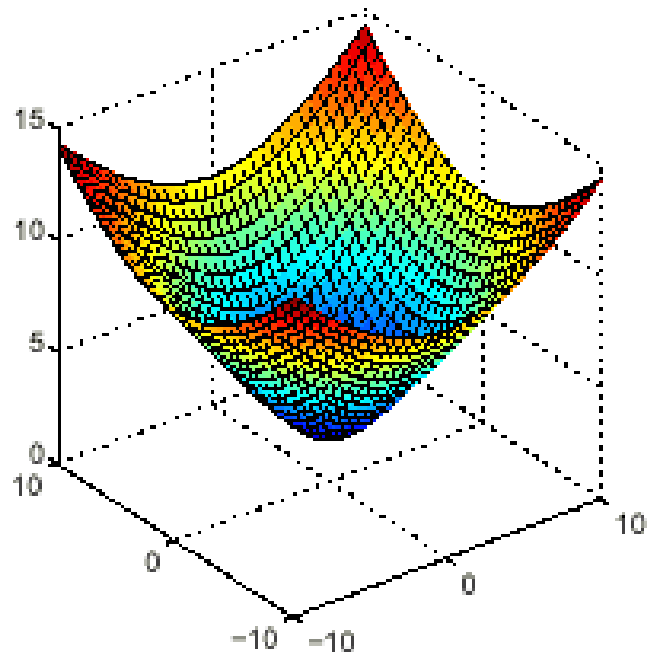


Funkcje wielokwadratowe

$$h(r) = (\sigma^2 + r^2)^{-\alpha}, \quad \alpha = 1$$

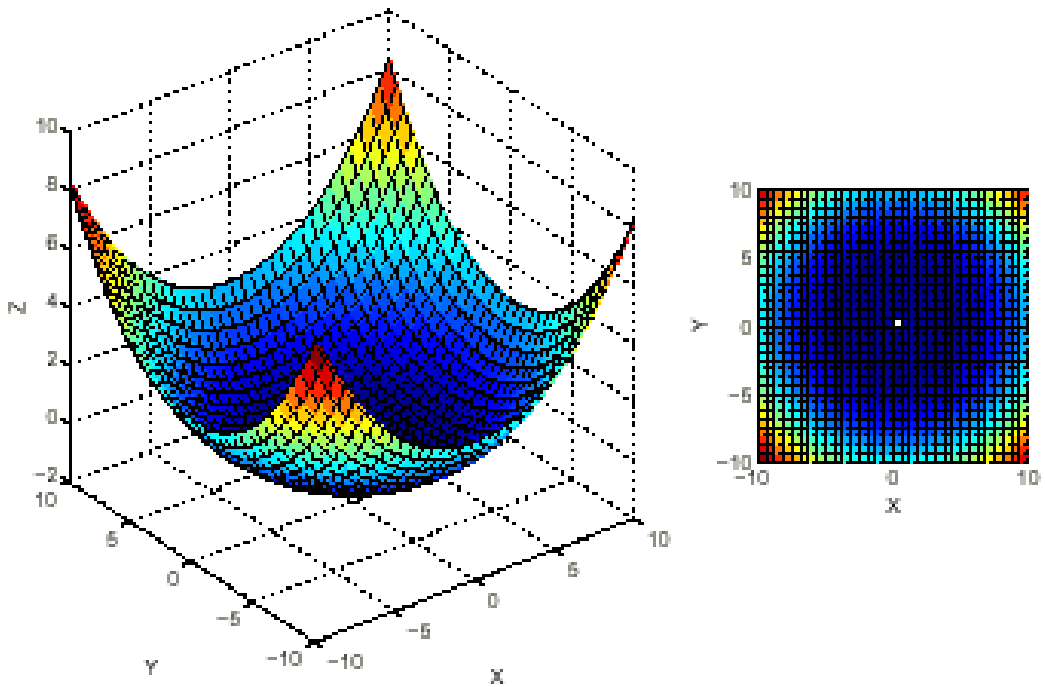
$$h(r) = (\sigma^2 + r^2)^{\beta}, \quad \beta = 0.5$$





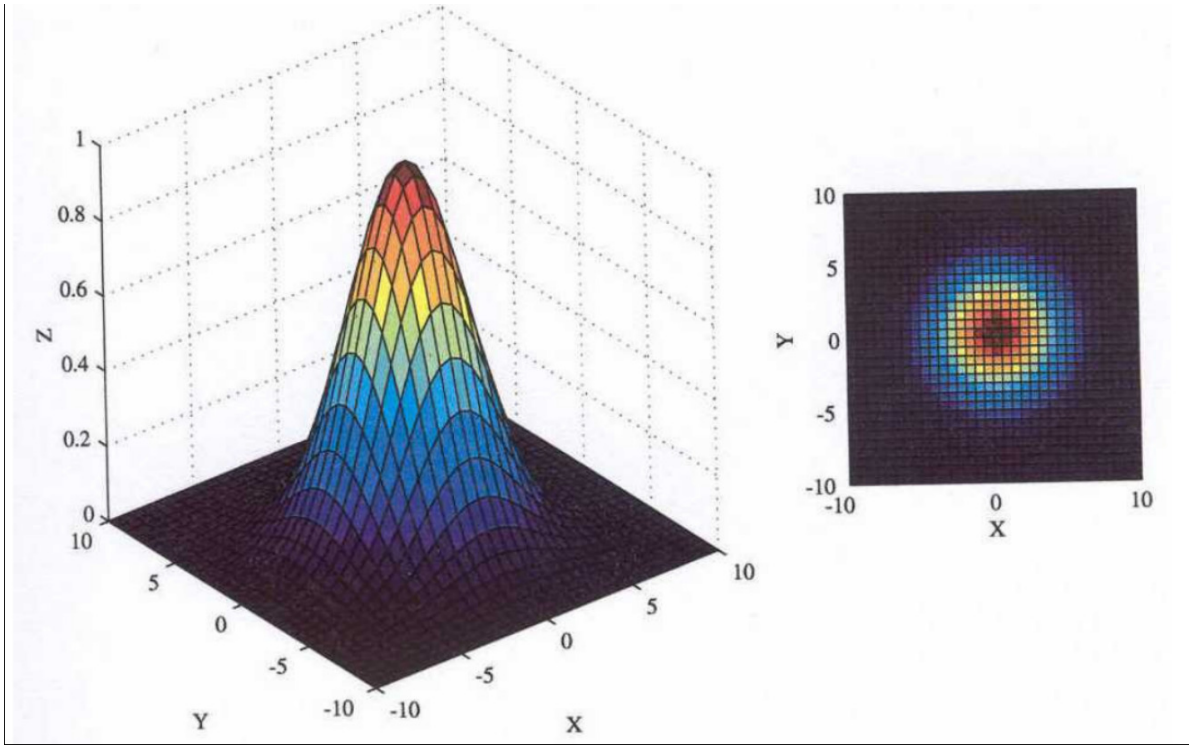
Funkcja cienkiej płytki

$$h(r) = (\sigma r^2) \ln(\sigma r)$$



Funkcja Gaussa

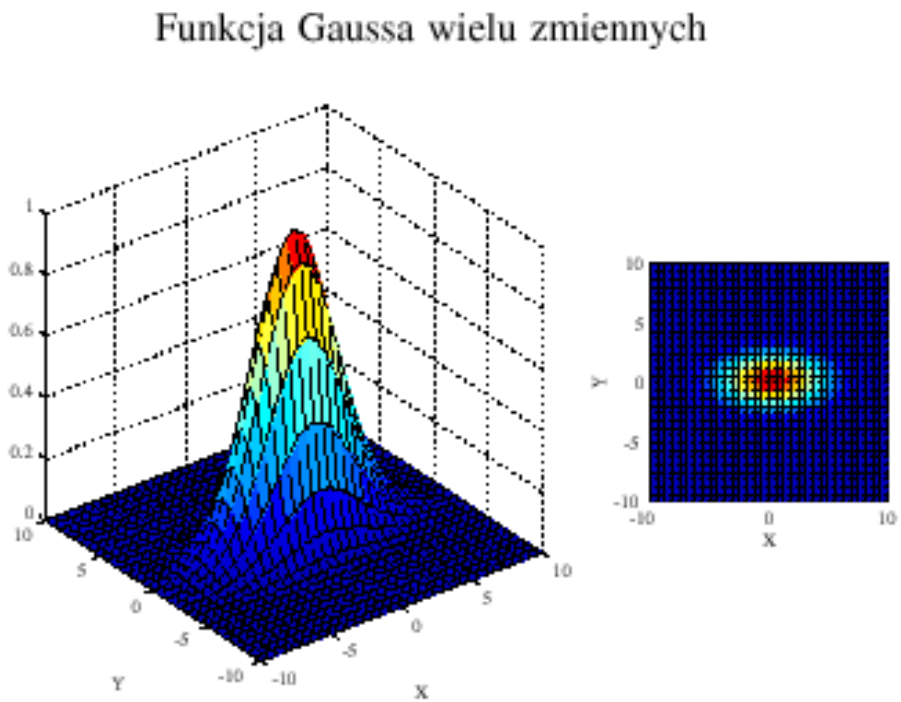
$$h(r) = e^{-\frac{r^2}{2\sigma^2}}$$



Najczęściej używana w roli funkcji bazowej RBF

Funkcja Gaussa z ogólną macierzą kowariancji

$$h(r) = e^{-\frac{1}{2}(\mathbf{x}-\mathbf{c})^T \Sigma^{-1}(\mathbf{x}-\mathbf{c})}$$



Macierz $\mathbf{\Sigma}$ określa rozmycia i rotacje funkcji w wielowymiarowej przestrzeni

Uczenie RBF jako problem aproksymacji

Dla N punktów $\mathbf{x}_i, y_i$ znajdź funkcję spełniającą

$$f(\mathbf{x}_i) = y_i \quad i = 1, \ldots, N$$

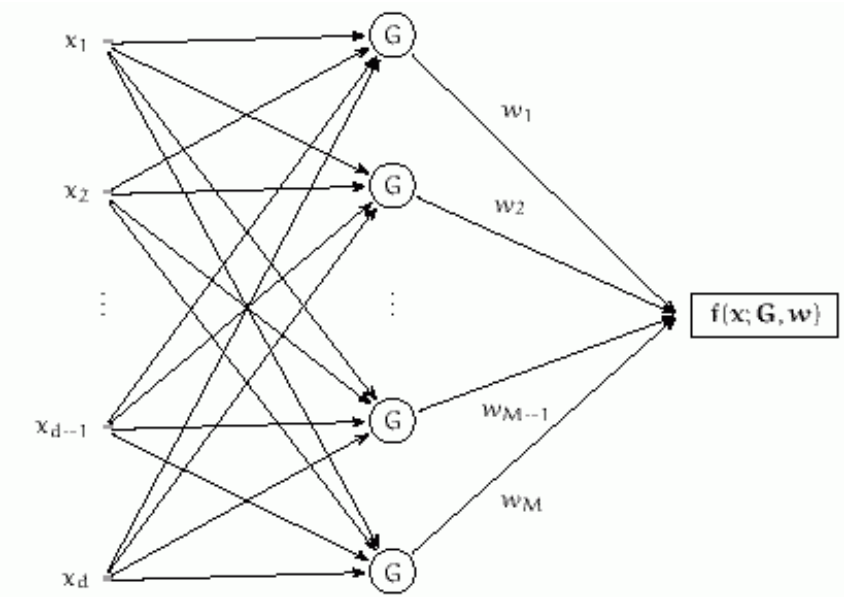
Funkcja sieci RBF

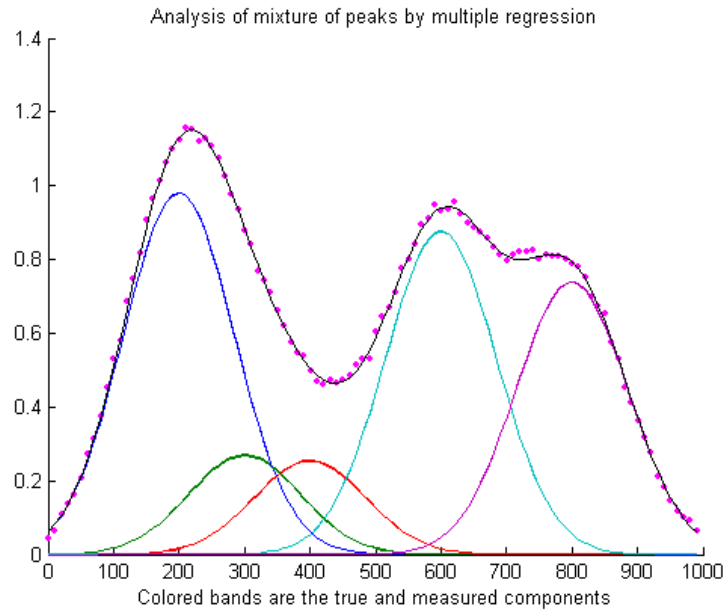
$$f(\mathbf{x}_i) = \sum_{j=1}^{K} w_j \varphi \left(\|\mathbf{x}_i - \mathbf{c}_j\| \right)$$

Szukamy minimum funkcji błędu MSE

$$E = \sum_{i=1}^{N} \left(y_i - f(\mathbf{x}_i) \right)^2$$

RBF z jedną warstwą ukrytą i linowym wyjściem jest uniwersalnym aproksymatorem





Rys: https://terpconnect.umd.edu/~toh/spectrum/CurveFittingB.html

Rozwiązanie RBF

Dla $K = N$ gdzie centra funkcji bazowych są ustalone w punktach treningowych $\mathbf{x_i} = \mathbf{c_i}$

$$\begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,N} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ h_{N,1} & h_{N,2} & \cdots & h_{N,N} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

$$h_{ij} = \varphi(\|\mathbf{x}_i - \mathbf{x}_j\|)$$

$$\mathbf{Hw} = \mathbf{y}$$

dla $\mathbf{x}_1 \neq \mathbf{x}_2 \neq \ldots \neq \mathbf{x}_N$ macierz $\mathbf{H}$ jest nieosobliwa i dodatnio określona, więc istnieje

$$\mathbf{w} = \mathbf{H}^{-1}\mathbf{y}$$

RBF jako problem aproksymacji

- dla ściśle dodatnio określonych funkcji radialnych (np. Gauss) macierz interpolacji $\mathbf{H}$ jest nieosobliwa i dodatnio określona (Light 1992)
- dla wąskich funkcji Gaussa (gdy $\mathbf{H} \approx \mathbf{I}$) istnieje rozwiązanie trywialne $\mathbf{w} = \mathbf{y}$ o niskiej generalizacji
- problem źle określony, przewymiarowanie
- hiperpłaszczyzna interpolacji nie jest gładka
- większe dyspersje σ i mniejsza liczba funkcji ($K < N$) $\rightarrow$ lepsza generalizacja

Aproksymacja RBF

Dla ustalonych $\mathbf{c}_i$ oraz σ_i gdy $K < N$

$$\begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,K} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ h_{N,1} & h_{N,2} & \cdots & h_{N,K} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

$$h_{ij} = \varphi(\|\mathbf{x}_i - \mathbf{c}_j\|)$$

$$\mathbf{Hw} = \mathbf{y}$$

rozwiązanie układu równań przez pseudoinwersję

$$\mathbf{w} = \mathbf{H}^{+}\mathbf{y}, \qquad \text{gdzie} \qquad \mathbf{H}^{+} = (\mathbf{H}^{T}\mathbf{H})^{-1}\mathbf{H}^{T}$$

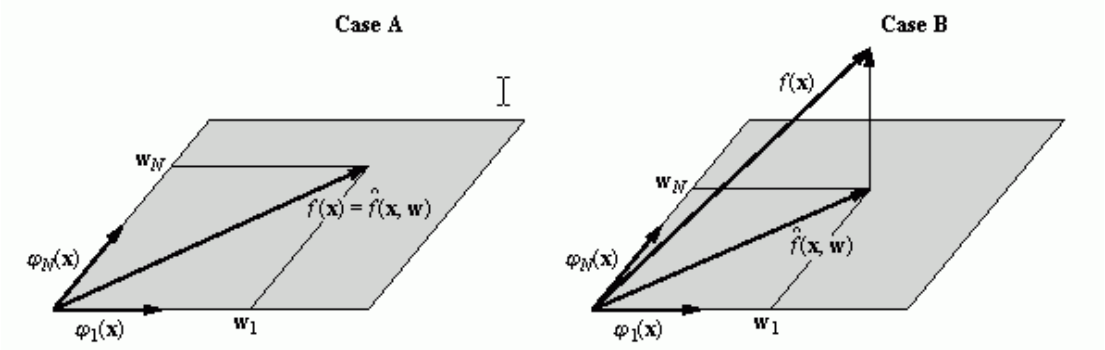
W praktyce pseudoinwersję realizuje się za pomocą dekompozycji SVD

Interpretacja geometryczna

Sieć RBF szuka funkcji będącej **liniową kombinacją** funkcji bazowych $\Phi(\mathbf{x}) = [\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \ldots, \varphi_K(\mathbf{x})]^T$

$$\hat{f}(\mathbf{x}; \mathbf{w}) = \sum_i w_i \varphi_i(\mathbf{x})$$

Jeśli prawdziwa aproksymowana funkcja $f(x)$ leży w przestrzeni rozpiętej przez wektory bazowe $\Phi(\mathbf{x})$ to możliwe jest rozwiązanie bez błędu, wystarczy dobrać odpowiednie wagi $\mathbf{w}$.
Jeżeli prawdziwa funkcja f leży poza tą przestrzenią to sieć nie jest w stanie jej odtworzyć. Minimalizacja MSE daje **prostopadłą projekcję** f. Błąd resztkowy $e = f - \hat{f}$ jest **ortogonalny** do przestrzeni rozpiętej przez wektory bazowe $\Phi(\mathbf{x})$.



Separacja w wielowymiarowej przestrzeni

Twierdzenie (Cover 1965):
Jeśli przekształcić wzorce $\mathbb{X} = \{\mathbf{x}_i\}, i = 1, \ldots, N$, nieliniową funkcją na wektory $\Phi(\mathbf{x}_i) = [\varphi_1(\mathbf{x}_i), \varphi_2(\mathbf{x}_i), \ldots, \varphi_K(\mathbf{x}_i)]^T$ gdzie $K > N$ to rośnie prawdopodobieństwo liniowej separacji, tj. istnieje płaszczyzna

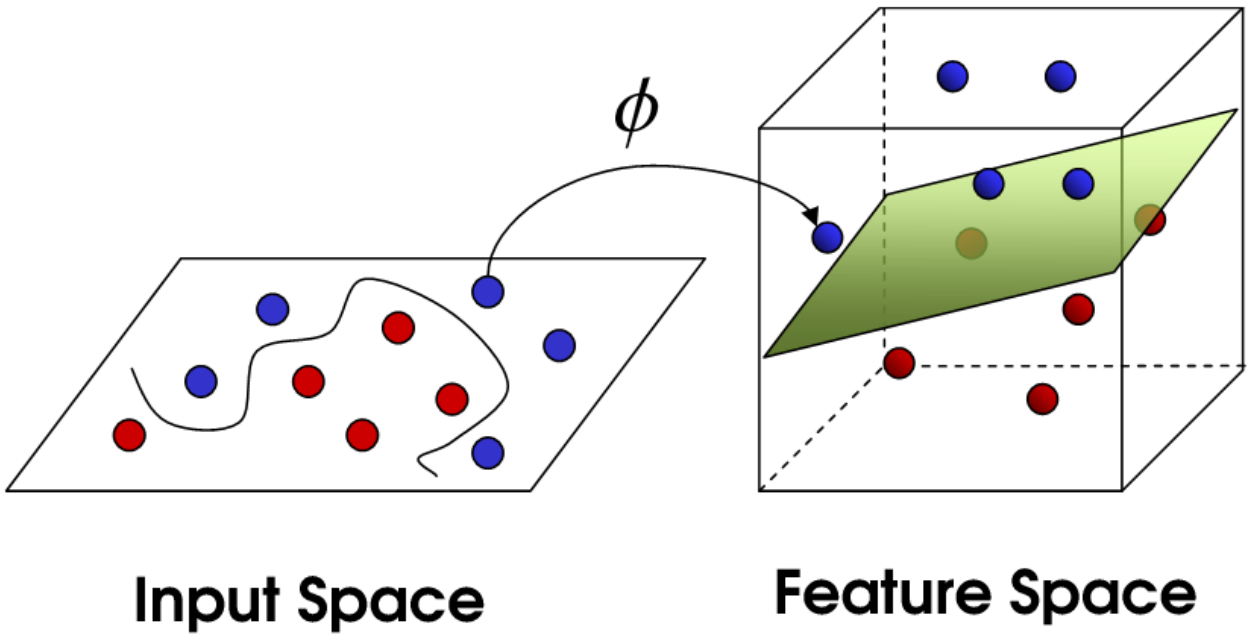
$$\begin{aligned} \mathbf{w}^T\Phi(\mathbf{x}_i) &\geq 0 \quad \text{dla} \quad \mathbf{x}_i \in \mathbb{C}_1 \\ \mathbf{w}^T\Phi(\mathbf{x}_i) &< 0 \quad \text{dla} \quad \mathbf{x}_i \in \mathbb{C}_2 \end{aligned}$$

granica decyzyjna zdefiniowana jest równaniem

$$\mathbf{w}^T\Phi(\mathbf{x}) = 0$$

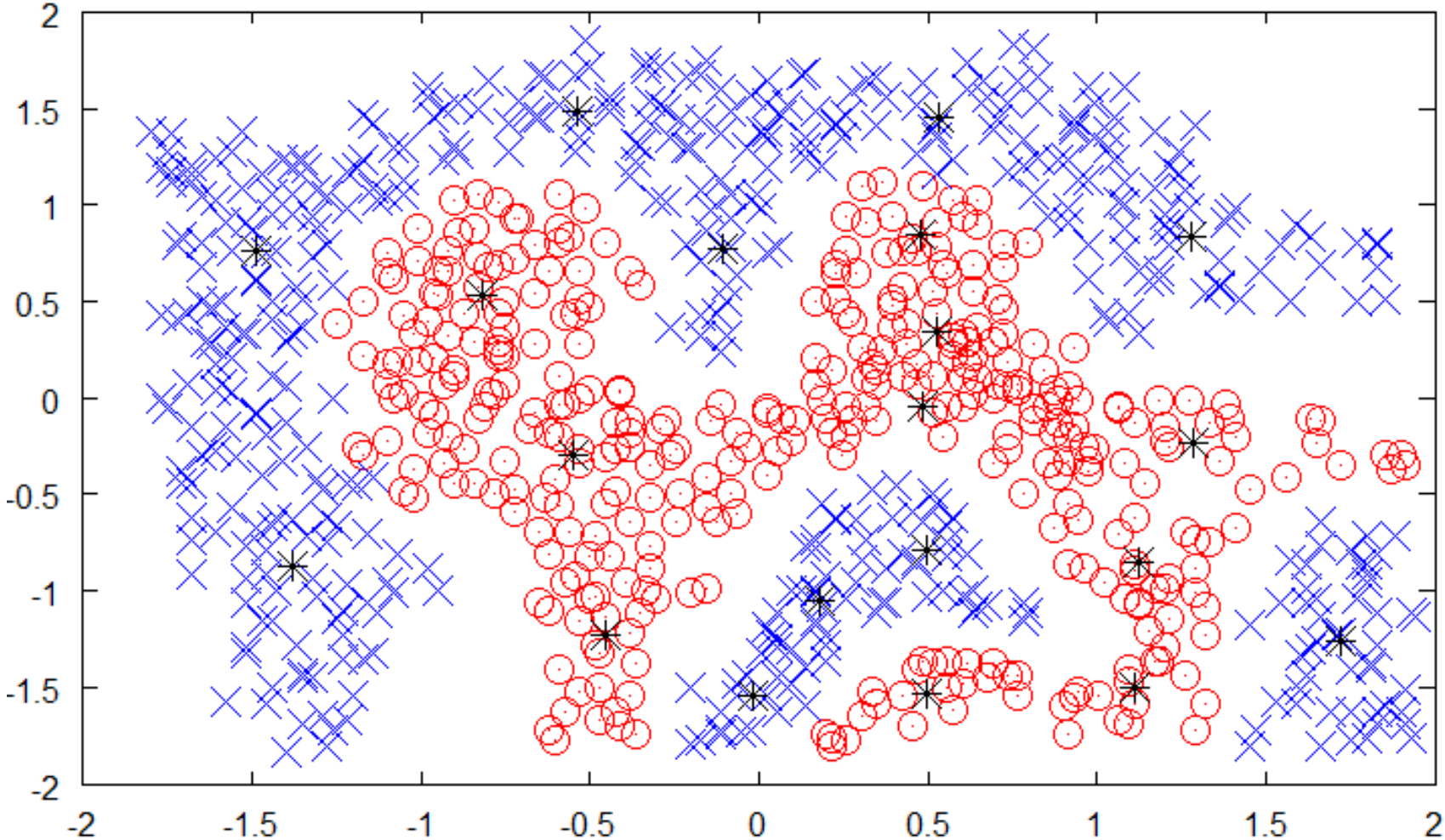
Przy dostatecznie dużej wymiarowości przestrzeni, do której rzutujemy wzorce, prawdopodobieństwo separowalności liniowej rośnie do 1.

Projekcja do przestrzeni cech



☞ *Julien Vitay, Neurocomputing*

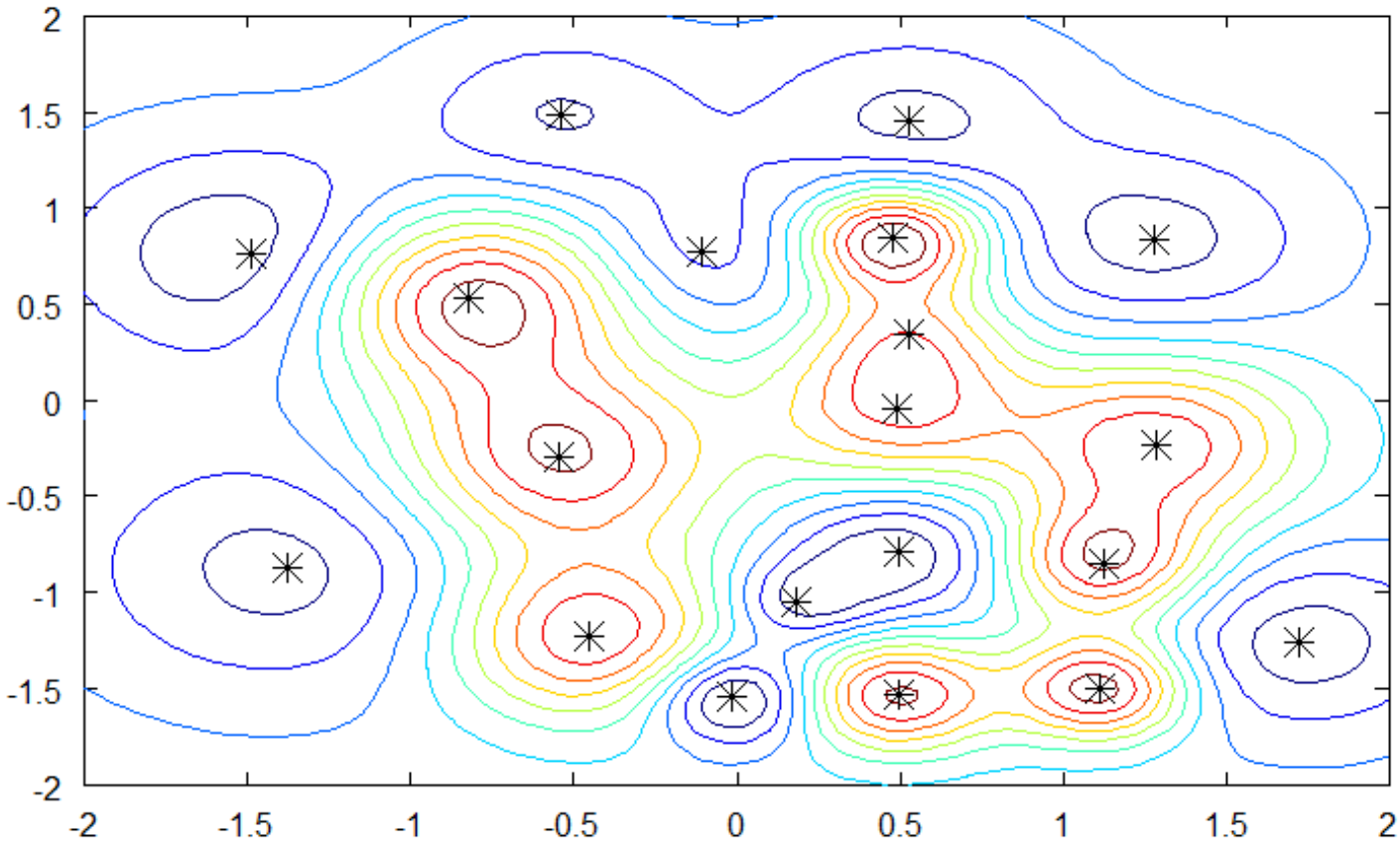
Przykład: 2 klasy



Rys: Chris McCormick, Radial Basis Function Network (RBFN) Tutorial

Przykład: 2 klasy

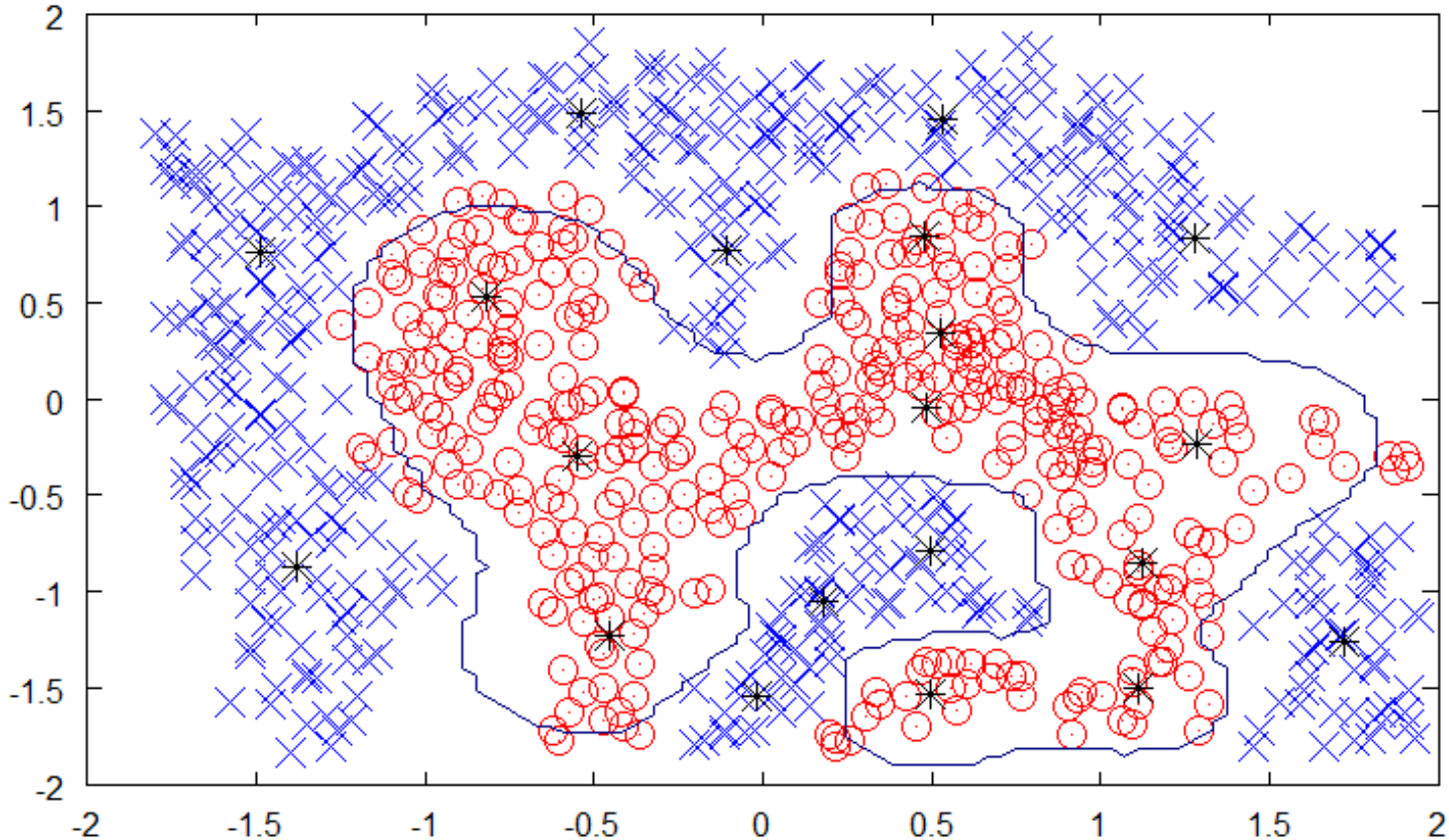
Wartości dla klasy 1, najczęściej wagi powiązane z funkcjami radialnymi dla klasy 1 mają wartości dodanie a dla klasy 2 - ujemne



Rys: Chris McCormick, Radial Basis Function Network (RBFN) Tutorial

Przykład: 2 klasy

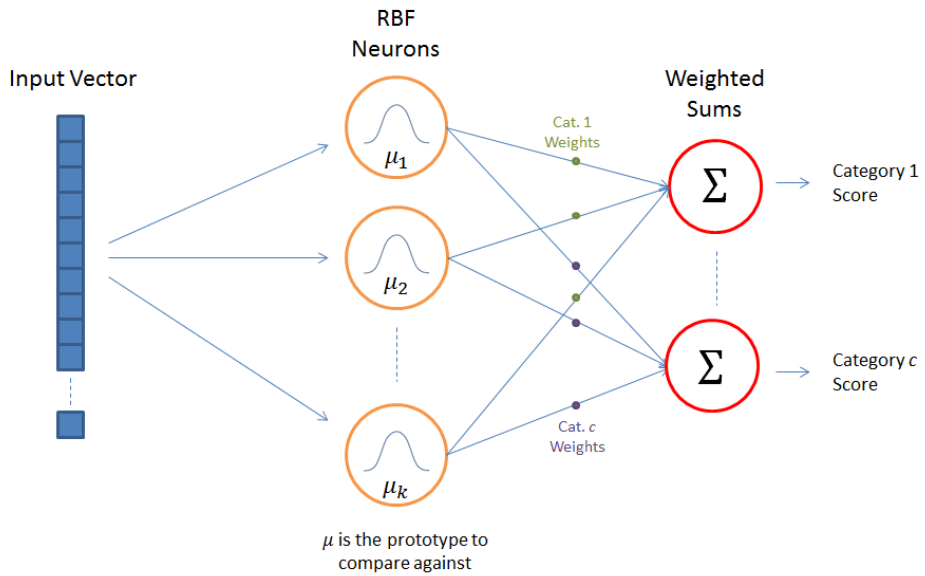
Granica decyzyjna



Rys: Chris McCormick, Radial Basis Function Network (RBFN) Tutorial

Sieć RBF

- jedna warstwa ukryta z radialnymi finkcjami + wyjśca liniowe
- parametry: wagi $\mathbf{w}$, centra $\mathbf{c}_i$, dyspersja σ_i (lub pełna macierz Σ), liczba funkcji radialnych K



Rys: Chris McCormick, Radial Basis Function Network (RBFN) Tutorial

Trening RBF

Różne strategie uczenia:

- Trening 2 etapowy:
  - najpierw ustawienie funkcji bazowych (centra i dyspersje)
  - potem rozwiązanie układu równań $\mathbf{HW} = \mathbf{Y}$
- Możliwe jednoczesne uczenie centrów, dyspersji i wag, np. za pomocą metod gradientowych (wsteczna propagacja)
- Metody inicjalizacji początkowych położeń centrów: losowa, klasteryzacja, samoorganizacja, probabilistyczna
- Metody inicjalizacji dyspersji: wartość stała, dobór zależnie od gęstości danych w różnych rejonach, średnie odległości od wektorów z innych klas, ...

Ustalenie centrów - różne strategie

- losowe równomierne pokrycie przestrzeni wejściowej - może nie oddawać specyfiki problemu
- w problemie aproksymacji ustawiamy centra w miejscach minimum i maksimum a następnie usuwamy wzorce z otoczenia tych centrów a resztę rozmieszczemy równomiernie wśród pozostałych wzorców
- dla klasyfikacji ustalamy centra funkcji bazowych w pobliżu granic decyzyjnych lub wewnątrz skupisk wektorów należących do wspólnej klasy
- wybieramy losowy podzbiór K wzorców treningowych - prosta ale skuteczna metoda
- ustalenie centrów w procesie uczenia nadzorowanego lub nienadzorowanego: klasteryzacja, samoorganicazcja, LVQ, ...

Ustalenie dyspersji - różne strategie

- wymagane gładkie odwzorowanie, rozmycie pełni rolę regularyzacji
- „pola recepcyjne" wszystkich funkcji bazowych powinny pokrywać cały obszar danych wejściowych
- pola recepcyjne sąsiadujących funkcji bazowych powinny nakrywać się w niewielkim stopniu
- rozmycie σ jednakowe dla wszystkich funkcji np. (Haykin, 1994):

$$\sigma = \frac{d}{\sqrt{2K}}$$

gdzie K ilość funkcji bazowych, d maksymalna odległość pomiędzy centrami. Funkcja Gaussa przybiera wówczas postać:

$$\varphi(\|\mathbf{x} - \mathbf{c}_i\|) = e^{-\frac{K\|\mathbf{x}-\mathbf{c}_i\|^2}{d^2}}$$

- rozmycie równe średniej odległości od sąsiadujących centrów

Ustawienie dyspersji - rózne strategie

- średnia odległość euklidesowa wektorów treningowych od najbliższego centrum

$$\sigma_k = \frac{1}{m} \sum_{i=1}^{m} \|\mathbf{x}_i - \mathbf{c}_k\|$$

gdzie m to liczba wektorów $\mathbf{x}_i$ przypisanych do centrum $\mathbf{c}_k$.

- σ_i równe odległości euklidesowej od najbliższego sąsiedniego centrum klastra (Tarasenko, 1994)
- uwzględniając odległość P najbliżej położonych centrów sąsiadujących funkcji radialnych (gdzie zwykle $P \leq 3$) (Moody, 1989)

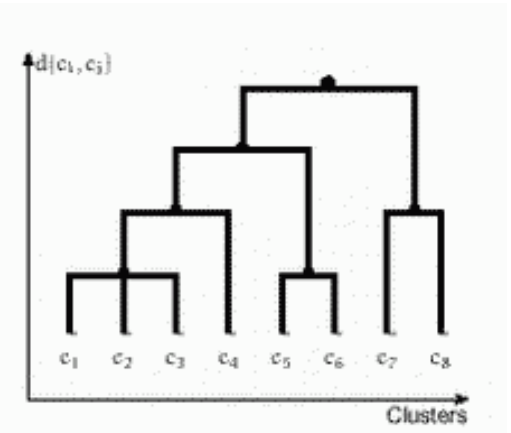
$$\sigma_i = \sqrt{\frac{1}{P} \sum_{i=1}^{P} \|\mathbf{c}_i - \mathbf{c}_k\|^2}$$

Incjalizacja RBF: klasteryzacja

Inicjalizacja centrów funkcji radialnych w centrach skupisk (klastrów) uzyskanych za pomocą algorytmów klasteryzacji (analizy skupień).
Przykład: metoda dendrogramów, algorytm k-średnich

Metoda dendrogramów:

1. przypisz każdy $\mathbf{x}_i$ do odrębnego klastra
2. połącz najbliższą parę klastrów $\mathbf{x}_i$ i $\mathbf{x}_j$ w jeden klaster
3. powtarzaj punkt 2 aż do uzyskania zadowalającej liczby klastrów lub gdy najmniejsza odległość między klastrami przekroczy ustalony poziom



Odległość między klastrami może być definiowana na wiele sposobów, np. minimalna odległość między obserwacjami z różnych klastrów

Incjalizacja RBF: klasteryzacja

Algorytm k-średnich

1. Rozmieść k centrów $\mathbf{c}_i$ równomiernie w przestrzeni wejściowej lub losowo wybierając k obserwacji ze zbioru treningowego
2. Dla każdego centrum $\mathbf{c}_i$ znajdź zbiór wszystkich punktów $\mathbf{x}_j$ leżących najbliżej tego centrum
3. Aktualizuj położenie centrum $\mathbf{c}_i$ ustawiając je w punkcie wyznaczonym przez średnią położeń punktów przypisanych do danego centrum

$$\mathbf{c}_i(k+1) = \frac{1}{N_i} \sum_{j=1}^{N_i} \overline{\mathbf{x}}_j(k)$$

4. Powtarzaj dwa ostatnie kroki aż do uzyskania zbieżności

☞ Clustering Demo

Inicjalizacja RBF: samoorganizacja

Metoda on-line k-średnich: inicjalizacja centrów funkcji radialnych przez samoorganizację metodami uczenia konkurencyjnego

1. Wybierz losowo k wektorów $\mathbf{c}_i$
2. Dla przypadku $\mathbf{x}$ ze zbioru uczącego znajdź najbliższe centrum $\mathbf{c}_j$ (zwycięzcę)
3. Zmień położenie centrum $\mathbf{c}_j$ zwycięzcy przesuwając je w kierunku punktu $\mathbf{x}$

$$\mathbf{c}_i(t+1) = \mathbf{c}_i(t) + \eta_t(\mathbf{x} - \mathbf{c}_i(t))$$

4. Powtarzaj kroki 2 i 3 określoną liczbę razy lub do uzyskania zbieżności

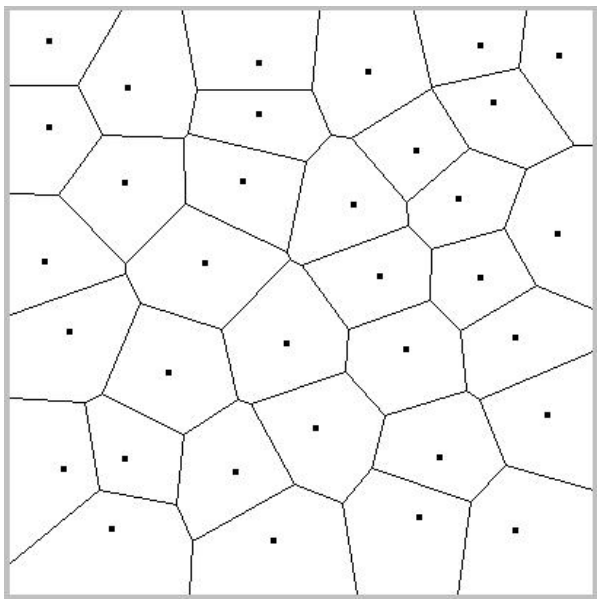
Stała uczenia η_t powinna zanikać ze wzrostem liczby iteracji t

$$\eta_t = \frac{\eta_0}{1 + \frac{t}{T}}$$

gdzie T to maksymalna liczba iteracji a η_0 - początkowa wartość

Diagram Voronoi

W drodze samoorganizacji przestrzeń dzielona jest na obszary Voronoi z centrum funkcji radialnej osadzonym w każdej komórce definiującymi teselację



Trening RBF: metoda probabilistyczna

RBF można interpretować jako model mieszaniny Gaussowskiej, gdzie funkcje radialne są komponentami tej mieszaniny

- zakładamy diagonalną macierz skalującą $\mathbf{\Sigma}$
- Gaussowska funkcja radialna $\varphi_i(\mathbf{x})$ określa miarę podobieństwa przypadku $\mathbf{x}$ do prototypu w centrum $\mathbf{c}_i$
- uczenie online EM (*Expectation-Maximization*)

$$\mathbf{c}_i(k+1) = \frac{\mathbf{c}_i(k) + \eta_k(\mathbf{x}\varphi_i(\mathbf{x}) - \mathbf{c}_i(k))}{(1-\eta_k) + \eta_k\varphi_i(\mathbf{x})}$$

$$\mathbf{\Sigma}_i(k+1) = \frac{\mathbf{\Sigma}_i(k) + \eta_k\left(\varphi_i(\mathbf{x})[\mathbf{x}-\mathbf{c}_i(k)][\mathbf{x}-\mathbf{c}_i(k)]^T - \mathbf{\Sigma}_i(k)\right)}{(1-\eta_k) + \eta_k\varphi_i(\mathbf{x})}$$

Gdy współczynnik uczenia maleje $\eta_k = \frac{\eta_0}{k}$, gdzie $\eta_0 \in [0,1]$ to algorytm zbiega do lokalnego maksimum funkcji wiarygodności dla danych treningowych

Trening wag $\mathbf{w}_i$ warstwy wyjściowej odbywa sie po ustaleniu parametrów funkcji radialnych (np. z rozwiązania układu równań)

Uczenie metodami gradientowymi

Minimalizacja funkcji błędu metodą największego spadku gradientu

$$E = \frac{1}{2} \sum_{n=1}^{N} \left(y_n - f(\mathbf{x}_n) \right)^2, \quad f(\mathbf{x}) = \sum_{j=1}^{K} w_j \, \varphi_j(\mathbf{x}),$$

dla funkcji gusowskiej i metryki euklidesowej

$$\varphi_j(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{2\sigma_j^2} \right)$$

daje reguły uczenia dla wag, centrów i rozmyć funkcji bazowych

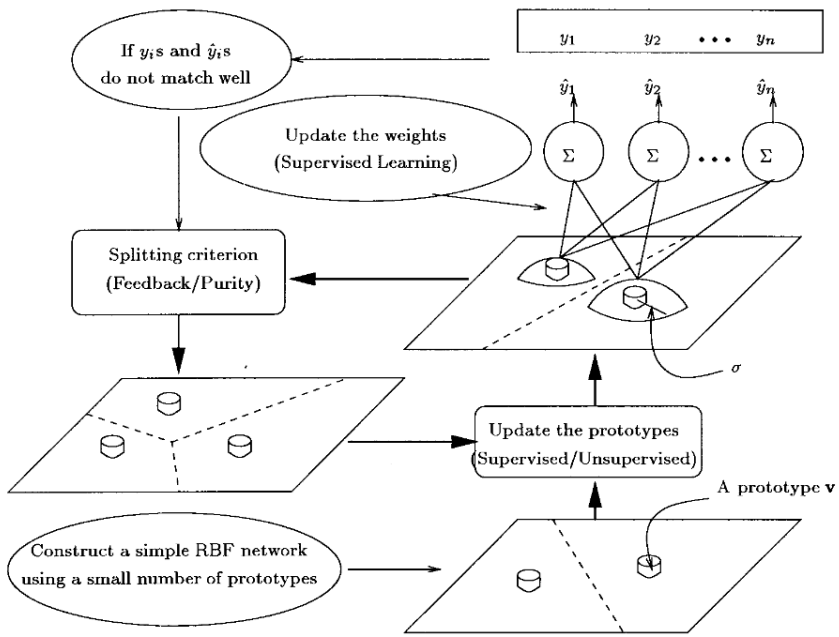
$$\Delta w_j = \eta \left(y - f(\mathbf{x}) \right) \varphi_j(\mathbf{x}),$$

$$\Delta c_{ji} = \eta \left(y - f(\mathbf{x}) \right) w_j \, \varphi_j(\mathbf{x}) \, \frac{x_i - c_{ji}}{\sigma_j^2},$$

$$\Delta \sigma_j = \eta \left(y - f(\mathbf{x}) \right) w_j \, \varphi_j(\mathbf{x}) \, \frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{\sigma_j^3}.$$

Rozrastające się architektury RBF

- GAL (*Growing and Learning*)
- GrRBF (*Growing Radial Basis Function*)
- FEN (*Function Estimation Networks*)
- GCS (*Growing Cell Structures*)
- RAN (*Resource Allocation Networks*)



Nicolaos, Growing Radial Basis Neural Networks: MergingSupervised and Unsupervised Learningwith Network Growth Techniques, 1997

Supervised Growing Cell Structures (Fritzke)

1. Zainicjuj małą sieć RBF
2. Wytrenuj sieć dowolną metodą
3. Oblicz błąd związany z każdym neuronem radialnym e_i akumulując bład MSE sieci dla najlepiej dopasowanych centrów $e_{i^*} \mathrel{+}= \left(y - f(\mathbf{x})\right)^2$ (zwycięzców położonych najbliżej przypadku uczącego $\mathbf{x}$)
4. Wstaw centrum nowej funkcji radialnej w obszarze o największym błędzie

$$\mathbf{c}_{new} = \frac{1}{2}(\mathbf{c}_m + \mathbf{c}_n)$$

gdzie $\mathbf{c}_m$ to centrum z największym błędem,
$\mathbf{c}_n$ to sąsiadujące centrum $\mathbf{c}_m$ (tj. położone w sąsiedniej komórce Voronoi) o największym błedzie wśród sąsiadów $\mathbf{c}_m$

5. Powtarzaj kroki 2-4 dopóki błąd nie zmaleje do pożądanej wartości

Resorce-Allocating Network (RAN)

- RAN (Pratt, 1991) algorytm rozrostu sieci RBF
- Startuje od pustej warstwy ukrytej i dla danego wektora $\mathbf{x}_t$ dodawany jest nowy neuron jeżeli spełnione są kryteria „odkrywczości”

$$\|\mathbf{x}_t - \mathbf{c}_i\| \geq \epsilon(t)$$

$$\|\mathbf{e}(t)\| = \|\mathbf{y}_t - f(\mathbf{x}_t)\| > e_{min}$$

  gdzie $\mathbf{c}_i$ to centrum najbliższe $\mathbf{x}_t$
- Wrtość progu $\epsilon(t)$ maleje z każdym krokiem

$$\epsilon(t) = \max\{\epsilon_{max} e^{-\frac{t}{T}}, \epsilon_{min}\}$$

- zakładając, że w kroku t istnieje k neuronów, wówczas nowy neuron dodawany jest z parametrami

$$\mathbf{c}_{k+1} = \mathbf{x}_t, \qquad w_{k+1,j} = e_j(t), \qquad \sigma_{k+1} = \alpha\|\mathbf{x}_t - \mathbf{c}_i\|$$

- gdy kryteria „odkrywczości” nie sa spełnione to błąd jest korygowany aktualizacja parametrów (spadkiem gradientu)

Porównanie MLP vs RBF

MLP	RBF
nielokalne relacje, wymagają douczania	lokalne efekty, stabilność, niewrażliwość na odstające przypadki
jeden rodzaj parametrów (wagi)	kilka rodzajów parametrów: wagi, centra, rozmycia
trudna inicjalizacja	łatwa inicjalizacja
trudna interpretacja	łatwa interpretacja (prototypy)
uczenie tylko pod nadzorem	możliwe uczenie bez nadzoru
zawsze wie	czasami nie wie (przypadki w obszarach nie objętych polami recepcyjnymi)
BP dość skomplikowane dla wielu warstw	uczenie łatwe bo 1 warstwa
granica decyzji perceptronu: hiperpłaszczyzna	granica decyzyjna funkcji radialnej: hipersfera
aktywacja: ważona suma sygnałów $\mathbf{w}^T\mathbf{x}$	odległość od centrum funkcji bazowej $\|\mathbf{x}-\mathbf{c}\|$

Samoorganizacja i uczenie konkurencyjne

SOM Self Organizing Map

Uczenie nadzorowane i nienadzorowane

Uczenie nadzorowane:

- sygnał wejściowy $\mathbf{x}_i$ posiada oczekiwaną odpowiedź $\mathbf{y}_i$
- proces uczenia dąży do minimalizacji błędu pomiędzy odpowiedzią sieci $f(\mathbf{x}_i)$ a wartością oczekiwaną $\mathbf{y}_i$
- przykłady: klasyfikacja, aproksymacja

Uczenie nienadzorowane:

- modelowanie rozkładu danych $\mathbf{x}_i$, brak sygnału zwrotnego $\mathbf{y}_i$
- samoorganizacja - prowadzi do uporządkowania sieci poprzez lokalne, niezależne oddziaływania

Mechanizmy samoorganizacji

- oparte o mechanizm konkurencji - reguła Kohonena
- oparte o regułę asocjacji Hebba (metody korelacyjne)

Aspekty samoorganizacji

- uczenie dostraja wagi neuronów (a przez to ich aktywacje) do rozkładu próbek uczących (korelacje, podobieństwo)
- pewne grupy neuronów reagują na podobne wzorce (grupowanie)
- silniejsze sygnały $\rightarrow$ większe wagi $\rightarrow$ większe aktywacje $\rightarrow$ większe różnice pomiędzy grupami neuronów reagujących na odmienne bodźce
- nadmiarowość danych (redundancja) - wielokrotne powtarzanie podobnych wzorców jest niezbędne w samoorganizacji

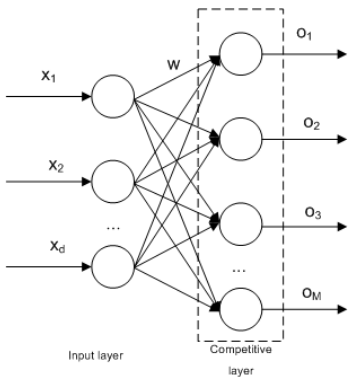
Zastosowanie samoorganizujących się sieci

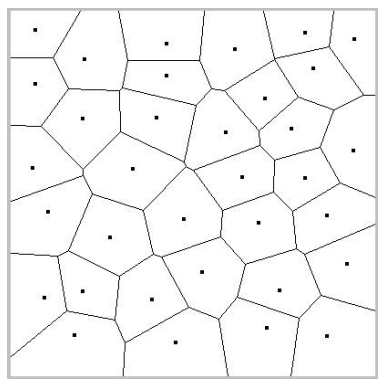
- wykrywanie istotnych cech w sygnale, podobieństw i różnic
- modelowanie rozkładu danych
- **analiza skupień** (klasteryzacja) - grupowanie wektorów podobnych
- analiza składowych głównych, **redukcja wymiarowości** do najistotniejszych cech
- **prototypy** - wykrycie typowego reprezentanta pewnej grupy wektorów treningowych
- **wektorowa kwantyzacja, kodowanie, kompresja** - opisanie zbioru danych mniejsza liczbą wektorów kodujących (prototypów)
- **mapy cech** - wizualizacja organizacji danych w niskowymiarowych przestrzeniach (1D, 2D, 3D) przy zachowaniu porządku topologicznego
- klasyfikacja

Uczenie konkurencyjne

Uczenie konkurencyjne (*competitive learning*) - w trakcie uczenia neurony konkurują ze sobą o prawo do reprezentacji danych wejściowych

- **Zwycięzca bierze wszystko WTA** (*winner takes all*) nazywane również Hard Competitive Learning, tylko sygnał neuronu najbardziej pobudzonego (zwycięzcy) jest brany pod uwagę
- **Zwycięzca bierze większość WTM** (*winner takes most*) - łagodniejsza forma (*soft competitive*) biorąca pod uwagę zwycięzcę oraz neurony o słabszym pobudzeniu, w pewnym otoczeniu $h(\mathbf{r}_i, \mathbf{r})$





Neuron zwycięzca k o największym pobudzeniu

$$\mathbf{w}_k^T \mathbf{x} > \mathbf{w_j}^T \mathbf{x}, \qquad j \neq k$$

Dla znormalizowanych wektorów $\mathbf{w}$ oraz $\mathbf{x}$

$$\hat{\mathbf{w}} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

neuron zwycięzca $\mathbf{w}_k$ ma najmniejszą odległość do $\mathbf{x}$

$$\|\mathbf{x} - \mathbf{w}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{w}\|^2 - 2\mathbf{w}^T\mathbf{x}$$

$$\arg\min_i \|\mathbf{x} - \mathbf{w}_i\| = \arg\max_i \mathbf{w}_i^T \mathbf{x}$$

stąd neuron zwycięzca

$$k = \arg\min_j \|\mathbf{w}_j - \mathbf{x}\| = \arg\min_j \sqrt{\sum_{i=1}^{d} (x_i - w_{ij})^2}, \quad j = 1, \cdots, N$$

Miary podobieństwa

- odległość Euklidesowa L_2

$$D\left(\mathbf{x}, \mathbf{w}_j\right) = \sqrt{\sum_{k=1}^{d} \left(x_k - w_{jk}\right)^2}$$

- odległość Manhattan L_1

$$D\left(\mathbf{x}, \mathbf{w}_j\right) = \sum_{k=1}^{d} \left|x_k - w_{jk}\right|$$

- iloczyn skalarny (odległość kosinusowa)

$$D\left(\mathbf{x}, \mathbf{w}_j\right) = 1 - \mathbf{w}_j^T \mathbf{x} = 1 - \|\mathbf{x}\| \|\mathbf{w}_j\| \cos(\mathbf{x}, \mathbf{w}_j)$$

- norma L_∞

$$D\left(\mathbf{x}, \mathbf{w}_j\right) = \max_k \left|x_k - w_{jk}\right|$$

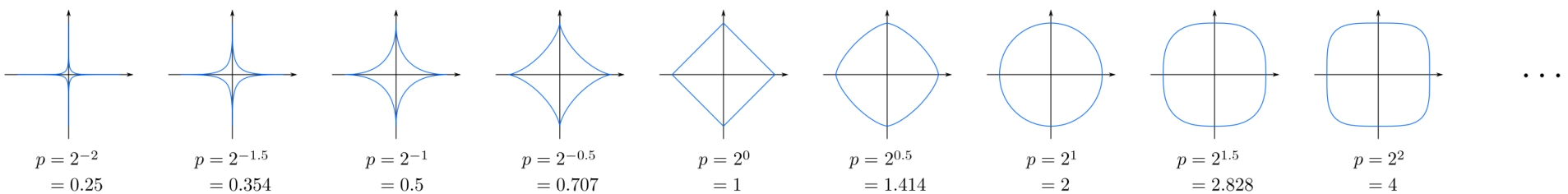
Odległość Minkowskiego

Uogólniona funkcja odległości

$$D(\mathbf{x}, \mathbf{w}) = \left(\sum_{k=1}^{d} |x_k - w_k|^p \right)^{\frac{1}{p}}$$

- gdy $p = 1$ odległość Manhattan
- gdy $p = 2$ odległość euklidesowa

Jednostkowy okrąg w przestrzeni $\mathbb{R}^2$



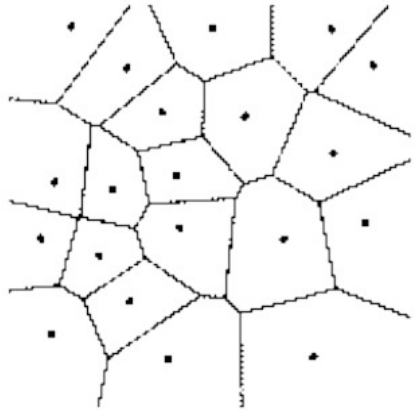
Rys: wikipedia.org

Voronoi i Delaunay

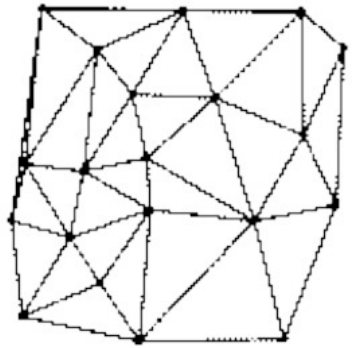
Zbiór punktów



diagram Voronoi



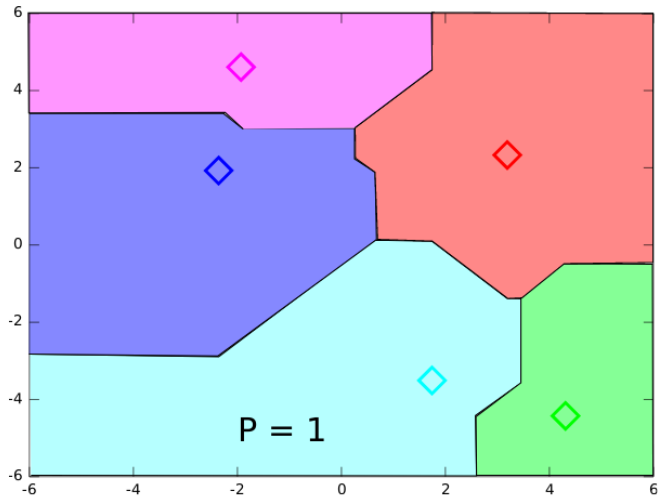
Triangulacja Delaunaya

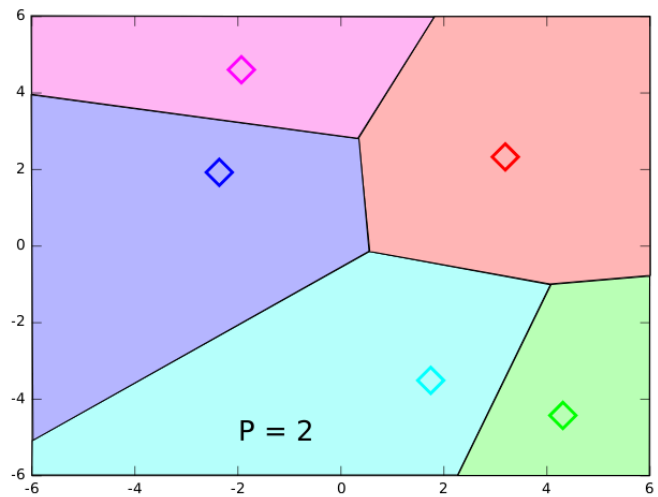


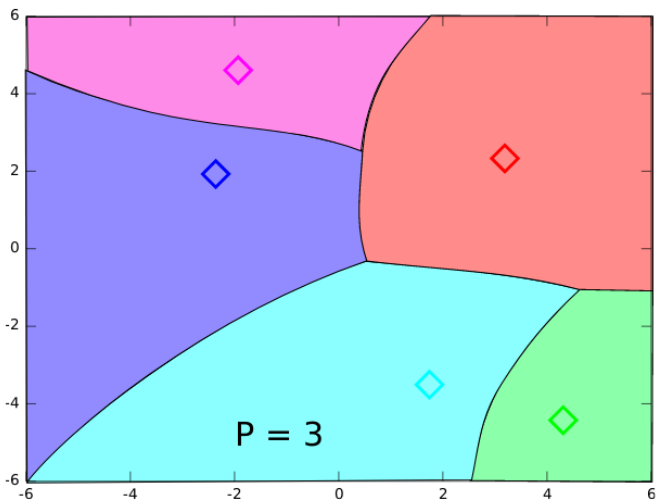
Neurony dzielą przestrzeń cech na rozdzielne obszary (obszary Voronoi), w danym obszarze tylko jeden neuron zwycięża konkurencję

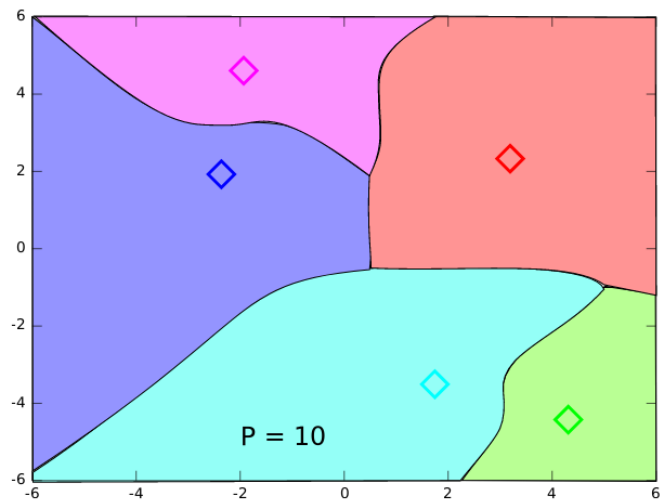
- **zbiór Voronoi** - zbiór wektorów wewnątrz obszaru Voronoi.
- łącząc węzły, których obszary Voronoia mają wspólną krawędź otrzymujemy **triangulację Delaunaya**

Strefy wpływów dla różnych metryk





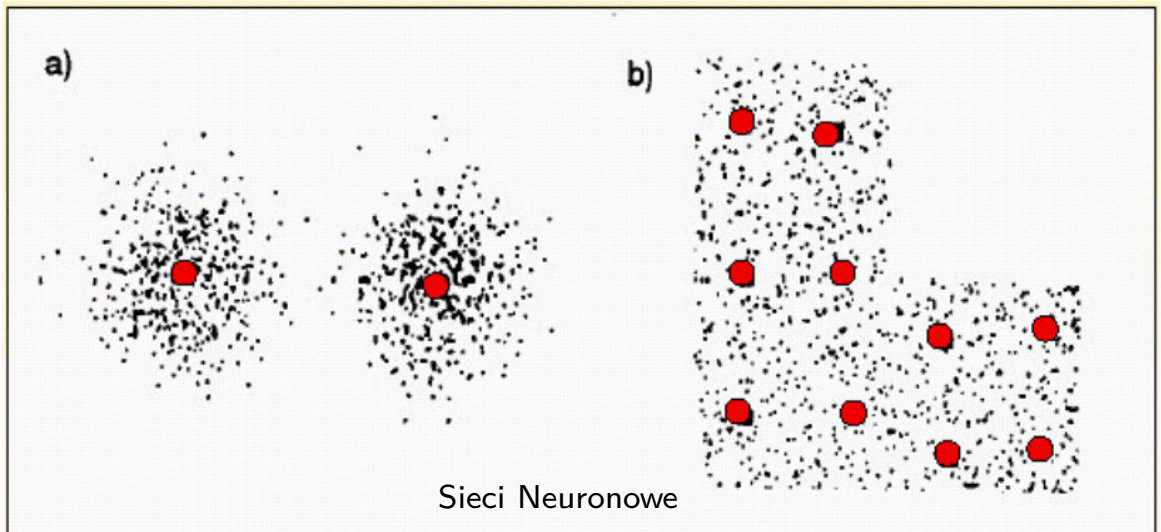




Rys: M. Czoków, J. Piersa, Wstęp do sieci neuronowych

Kwantowanie wektorowe

- **Kwantowanie wektorowe** - zakodowanie danych za pomocą wektorów prototypowych. Pojedynczy prototyp reprezentuje grupę wektorów wejściowych
- **Księga kodów** - zbiór prototypów kodujących (ich numery)
- Stratna kompresja danych - sygnał kodowany sekwencją numerów prototypów z księgi kodów, liczba prototypów k jest mniejsza od liczby wzorców n
- Do zakodowania wektora $\mathbf{x} \in \mathbb{R}^d$ wystarczy $\log_2 k$ bitów, gdzie k to rozmiar księgi kodów. Cały zbiór danych o rozmiarze $d \cdot n$ wymaga $n \cdot \log_2 k + d \cdot k$ bitów



Algorytm kwantowania wektorowego WTA

1. Zainicjuj wagi k neuronów
2. Dla każdego wzorca uczącego $\mathbf{x}$ znajdź neuron zwycięzcę o największej aktywacji
3. Przesuń wagi zwycięskiego neuronu w kierunku wektora $\mathbf{x}$ (**reguła Kohonena**)

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta(t)\left(\mathbf{x} - \mathbf{w}(t)\right)$$

4. Powtarzaj 2 i 3 aż do uzyskania zbieżności

Błąd kwantowania wektorowego dla k neuronów

$$E(\mathbf{x}; \mathbf{w}, t) = \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{x}_i - \mathbf{w}_{m(i)} \right\|^2$$

gdzie $\mathbf{w}_{m(i)}$ to wagi neuronu zwycięskiego dla sygnału $\mathbf{x}_i$

Przykład: kompresja obrazów

- obraz o wymiarach $N_x \times N_y$ dzielimy na równe ramki o wymiarach $n_x \times n_y$
- wektor $\mathbf{x}_i$ to wektor $n_x \times n_y$ pikseli (np. odcienie szarości) Zbiór wszystkich wektorów $\mathbf{x}_i$ tworzy zbiór treningowy
- uczenie samoorganizujące dowolną metodą w celu minimalizacji błędu kwantyzacji
- podobne ramki pobudzają ten sam neuron zwycięski
- kompresja: prezentacja wszystkich ramek $\mathbf{x}_i$ pozwala ustalić kolejność neuronów zwycięskich $1, 5, 1, 30, \ldots$, ich numery tworzą zakodowany strumień indeksów prototypów z księgi kodów
- współczynnik kompresji

$$k_r = \frac{N n_x n_y T}{N \log_2 k + k n_x n_y t}$$

gdzie N liczba ramek, k - liczba neuronów, T liczba bitów potrzebna do reprezentacji wartości $\mathbf{x}_i$, t liczba bitów potrzebna do reprezentacji wagi $\mathbf{w}_i$

oryginał



VQ 1024 codebook



kompresja: 2.6 bits/pixel

☞ Image Compression with Vector Quantization by Ivan-Assen Ivanov

Własności kwantyzacji wektorowej

- VQ nie musi odzwierciedlać gęstości rozkładu danych, przecenia rejony mało prawdopodobnych danych i nie docenia rejonów o dużym prawdopodobieństwie danych
- wynik zależy od punktu początkowego i kolejności prezentacji wzorców - każde uruchomienie może prowadzić do innej kwantyzacji
- zasada jednakowego zniekształcenia: każdy neuron powinien mieć podobny wkład do końcowego błędu kwantyzacji
- problem **martwych neuronów**: przy losowej inicjalizacji część neuronów może znaleźć się w obszarach gdzie nie będzie pobudzeń
- rozwiązanie: mechanizmy zmęczenia (np. neurony z sumieniem), odpowiadają chwilowemu zmęczeniu neuronu po wzbudzeniu w sieciach neuronowych biologicznych

Mechanizm sumienia (DeSieno)

Neurony z sumieniem (*conscience learning*) - mechanizm zmęczenia zapobiegający powstawaniu martwych neuronów

Miara częstości zwycięstw $p_i(t)$

$$p_i(t+1) = \begin{cases} p_i(t) + B \cdot (1 - p_i(t)) & \text{dla neuronu zwycięzcy} \\ p_i(t) - B \cdot p_i(t) & \text{dla pozostałych} \end{cases}$$

gdzie $0 < B \ll 1$ (np. $B = 0.0001$)

Zwycięzca spośród N neuronów

$$k = \arg\min_j \left(\|\mathbf{w}_j - \mathbf{x}\| - C \left(\frac{1}{N} - p_j(t) \right) \right) \quad j = 1, \cdots, N$$

- u często wygrywających neuronów rośnie „poczucie winy”
- dla $C = 0$ nie występuje zmęczenie (standardowy algorytm Kohonena)
- potencjał p_i ograniczony do 1

Mechanizm zmęczenia ze współczynnikiem czułości

Każdy prototyp $\mathbf{w}_i$ posiada współczynnik czułości $s_i \geq 0$ początkowo równy 0.
W kolejnych krokach treningu kwantyzacji wektorowej

$$s_i(t+1) = \begin{cases} 0 & \text{dla neuronu zwycięzcy} \\ s_i(t) + \gamma & \text{dla pozostałych} \end{cases}$$

gdzie $\gamma > 0$

Zwycięzca wybierany jest spośród N neuronów

$$k = \arg\min_i \left(\left\| \mathbf{w}_i - \mathbf{x} \right\| - s_i \right) \quad i = 1, \cdots, N$$

- lepsze odwzorowanie gęstości danych, nawet przy dużej zmienności gęstości w różnych obszarach

Mechanizm zmęczenia z modyfikacją odległości

Efektywna odległość zależna od częstości zwycięstw N_i

$$\hat{d}(\mathbf{x}, \mathbf{w}_i) = N_i d(\mathbf{x}, \mathbf{w}_i)$$

- kara dodawana przy określaniu zwycięzcy, częściej wygrywające neurony mają większą odległość efektywną
- w trakcie aktualizacji wag używa się oryginalnej miary odległości $d(\mathbf{x}, \mathbf{w})$
- po kilku początkowych cyklach, wszystkie neurony mają szansę wygrać przynajmniej raz

Gaz neuronowy (Schulten i Martinez 1991)

Wektorowa kwantyzacja WTM, w której wielkość kroku aktualizacji zależna jest od pozycji w rankingu zwycięzców

1. Losowa inicjalizacja N neuronów
2. Powtarzaj T razy kroki od 2 do 6
3. Wybierz losowy wektor $\mathbf{x}$
4. Zrób ranking neuronów posortowanych względem odległości od $\mathbf{x}$

   $$\|\mathbf{w}_i - \mathbf{x}\| \leq \|\mathbf{w}_j - \mathbf{x}\|, \qquad i < j$$

   Niech $m(i) = 0, 1, 2, \ldots$ określa miejsce neuronu w rankingu, gdzie dla zwycięzcy $m(i) = 0$
5. Reguła uczenia

   $$\Delta\mathbf{w}_i = \eta(t) h_i(t) (\mathbf{x} - \mathbf{w}_i)$$

   $$\text{gdzie} \quad h_i(t) = e^{-\frac{m(i)}{\lambda(t)}}$$

6. Zmniejsz eksponencjalnie obszar $\lambda(t)$ i stałą uczenia $\eta(t)$

Promień sąsiedztwa

Promień sąsiedztwa maleje w trakcie uczenia od λ_0 do pewnej wartości λ_{min}

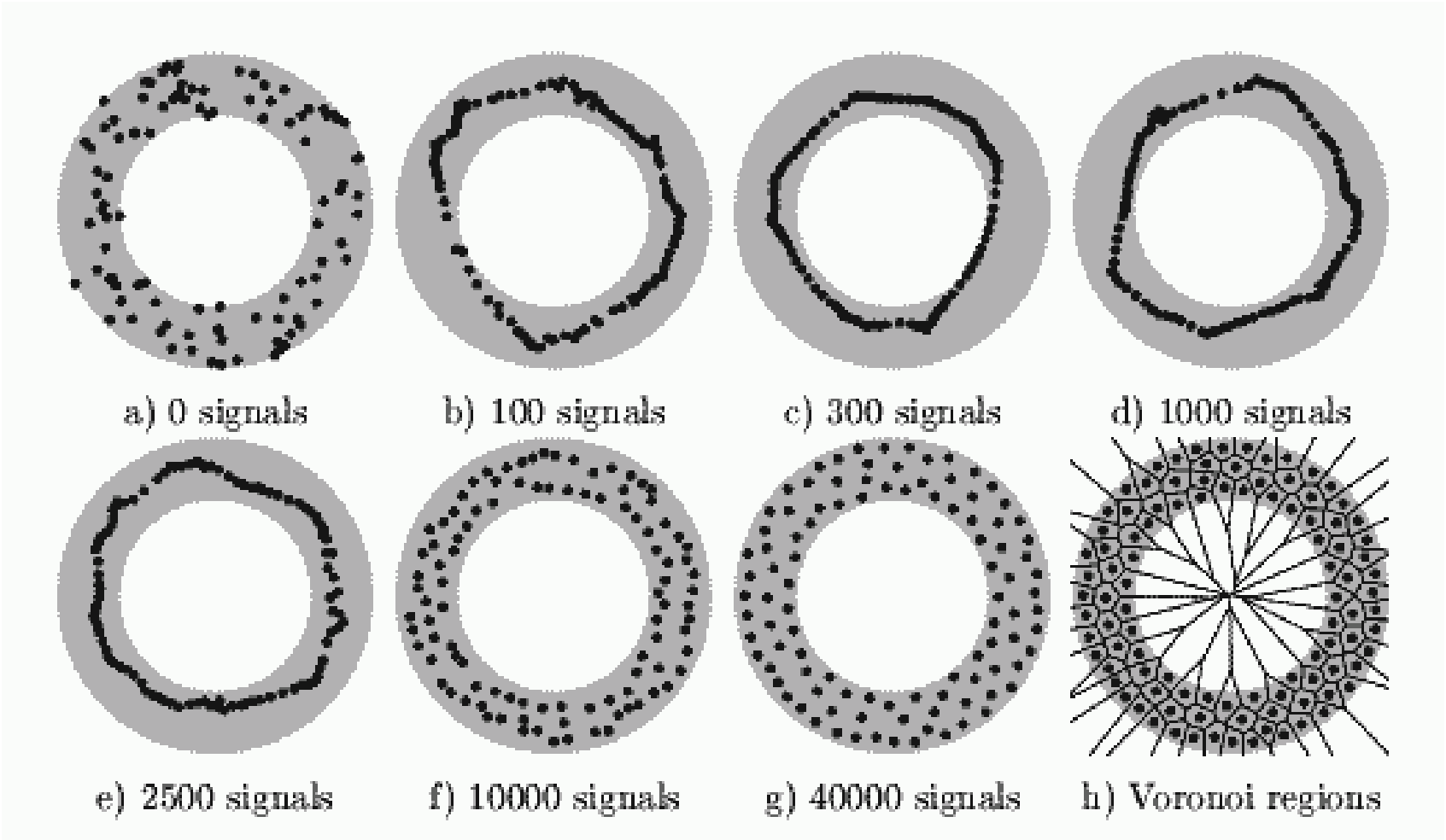
$$\lambda(t) = \lambda_0 \left(\frac{\lambda_{min}}{\lambda_0} \right)^{\frac{t}{T}}$$

- $\lambda > 0$ aktualizowane są wagi wszystkich neuronów z siłą zależna od pozycji w rankingu $h_i(t) = e^{-\frac{m(i)}{\lambda(t)}}$
- dla $\lambda \approx 0$ uczenie WTA, tylko zwycięzca jest aktualizowany

Współczynnik uczenia $\eta(t)$ zazwyczaj maleje wykładniczo

$$\eta(t) = \eta_0 \left(\frac{\eta_{min}}{\eta_0} \right)^{\frac{t}{T}}$$

Przykład kwantyzacji gazu neuronowego

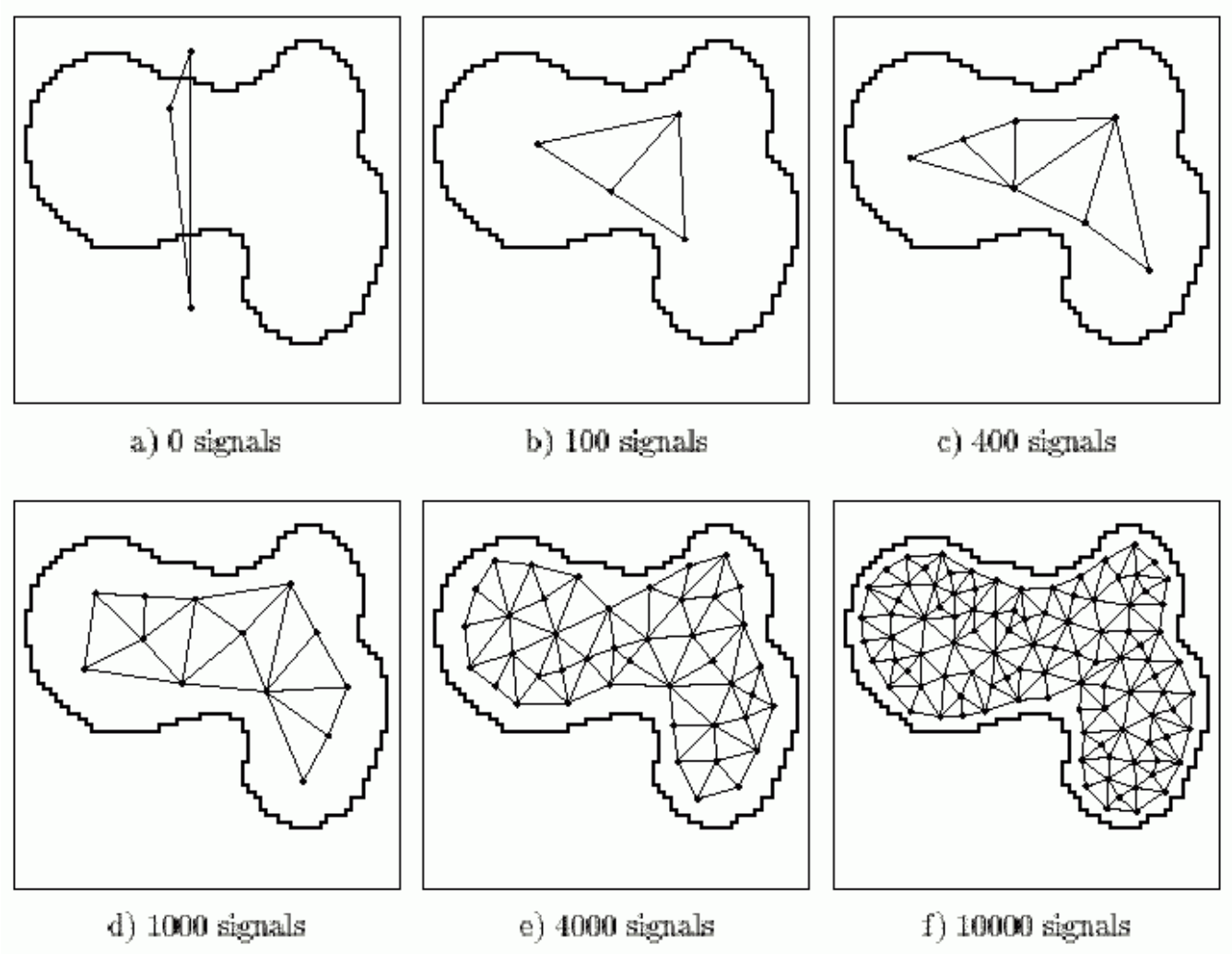


Growing Cell Structures

Rozrastająca się wektorowa kwantyzacja. Neurony połączone są krawędziami definiującymi ich sąsiedztwo.
Początkowa topologia: k-wymiarowy sympleks (np. k=1, 2, 3).

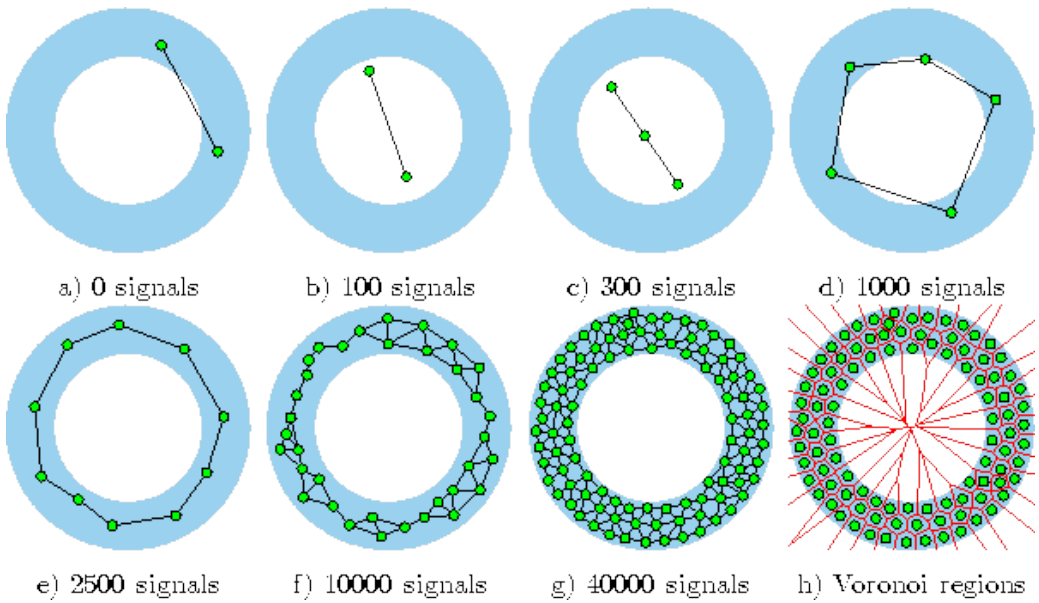
1. Dla losowego $\mathbf{x}$ znajdź zwycięzcę c
2. Aktualizuj wagi zwycięzcy oraz jego bezpośrednich sąsiadów: $\Delta \mathbf{w}_i = \eta_s(\mathbf{x} - \mathbf{w}_i)$
3. Dla zwycięzcy c zwiększ sumaryczny błąd $E_c = E_c + ||\mathbf{x} - \mathbf{w}_c||^2$
4. Po ustalonej liczbie epok T znajdź neuron o największym E_c i wstaw nowy neuron r w środku krawędzi łączącej go z najdalszym sąsiadem tworząc lokalny k-wymiarowy sympleks. Błąd sąsiadów r zmniejsz o wartość proporcjonalną do liczby tych sąsiadów. Błąd nowego neuronu ustaw na średni błąd jego sąsiadów.
5. Powtarzaj powyższe kroki aż do osiągnięcia kryterium stopu (np. określony rozmiar sieci)

Trening GCS

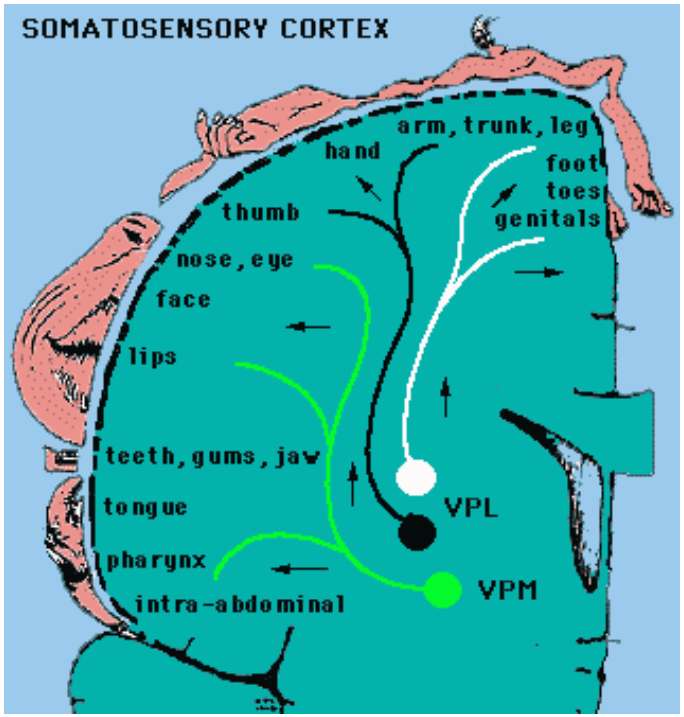

a) 0 signals
b) 100 signals
c) 400 signals
d) 1000 signals
e) 4000 signals
f) 10000 signals

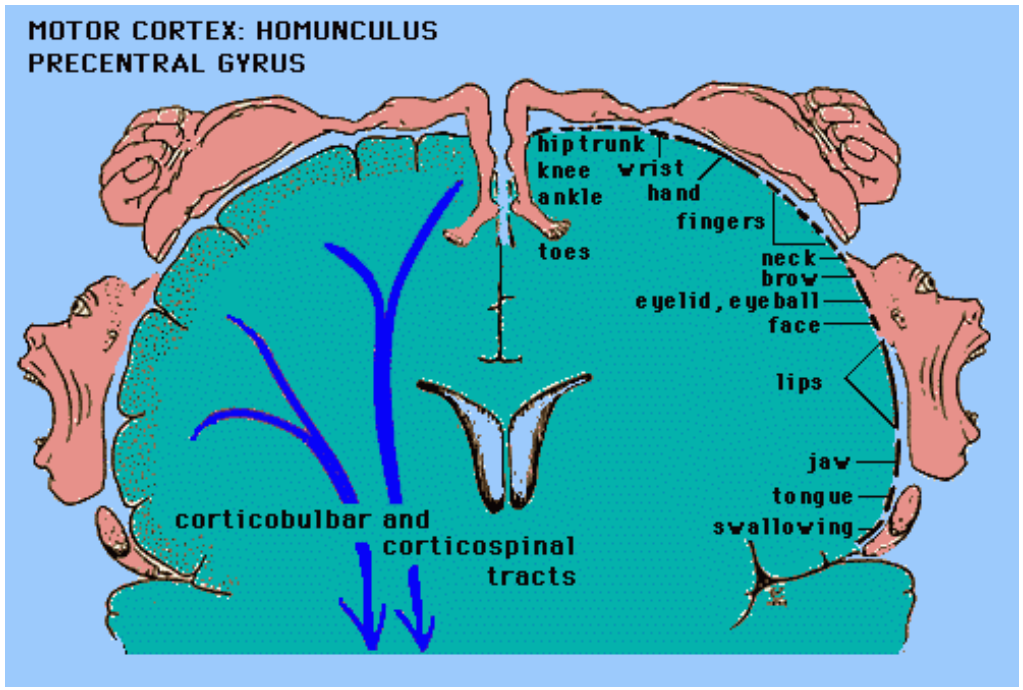
Growing Neural Gas (Fritzke, 1994)

- GNG, rozrastający się gaz neuronowy, algorytm zbliżony do GCS, startuje od pary neuronów i wprowadza mechanizmem starzenia się połączeń
- nowe połączenie powstaje między zwycięzcą i drugim w rankingu neuronem
- połączenia, które żyją za długo są usuwane



Mapy czuciowe i motoryczne





- mapa czuciowa - reprezentacja ciała w korze mózgowej, neurony reprezentujące sąsiednie części ciała są blisko siebie
- mapa motoryczna - reprezentacja mięśni i ruchów w korze mózgowej, neurony reprezentujące ruchy sąsiednich części ciała są blisko siebie

Inspiracje biologiczne SOM

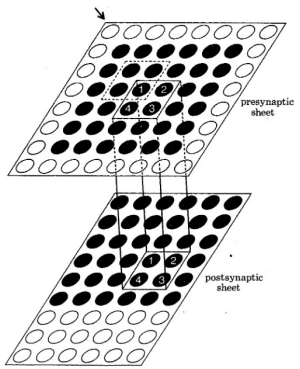
- przetwarzanie różnych danych zmysłowych i motorycznych (wzrok, słuch, dotyk, ...) odbywa się w różnych obszarach kory mózgowej
- istnieje topologiczna zależność w obliczeniach realizowanych przez mózg
- neurony o podobnych funkcjach są obok siebie
  $=>$ mapy topograficzne
- Przykłady
  - mapy somatosensoryczne układu czuciowego,
  - mapy motoryczne kory i móżdżku,
  - mapy tonotopiczne układu słuchowego,
  - mapy orientacji dwuocznej układu wzrokowego,
  - mapy wielomodalne układu orientacji (wzgórki czworacze górne)

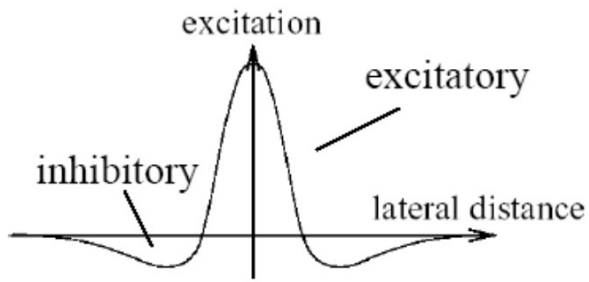
Modele samoorganizacji

- Samoorganizująca się mapa cech - SOM lub SOFM (Self-Organized Feature Mapping)
- Połączenia lokalne (sąsiednie neurony): neuron silnie pobudzany przez pobliskie, słabo przez odległe, hamowany przez neurony pośrednie
- von der Malsburg i Willshaw (1976) - model układu wzrokowego
- Amari (1980) - model ciągłej tkanki neuronowej uwzględniający pobudzenie i hamowanie zależne od odległości
- Kohonen (1981) uproszczony model, bez hamowania, dwie fazy uczenia: konkurencja i kooperacja.

Model Willshaw, van der Malsburg (1976)

Samoorganizacja w mapowaniu sygnału z siatkówki na korę wzrokową, transformacja 2D -> 2D , brak redukcji wymiaru, połączenia wewnątrzwarstwowe,

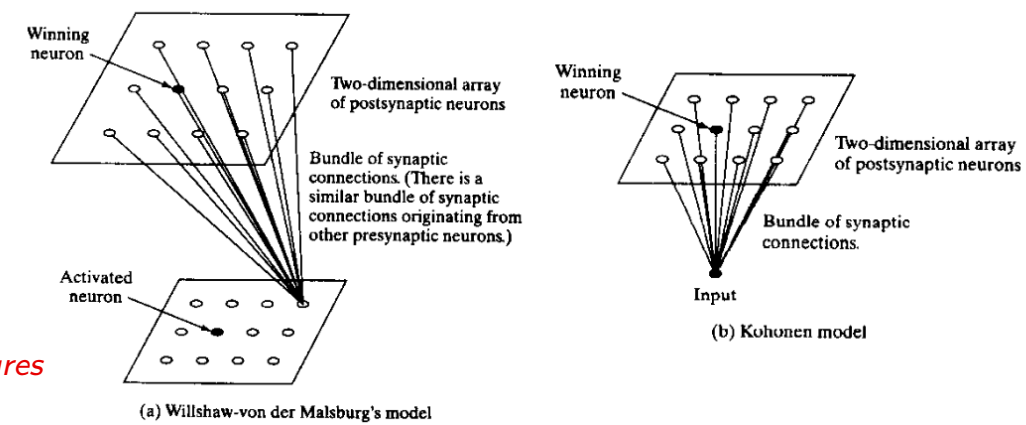




- aktywacja typu „Meksykańskiego kapelusza”
  - kooperacja - pobudzenia bliskiego zasięgu
  - konkurencja - hamowanie w dalszych regionach
- siatkówka (wejście) posiada narzuconą organizację topologiczną
- wybór zwycięskiego neuronu wynikiem dynamiki neuronalnej

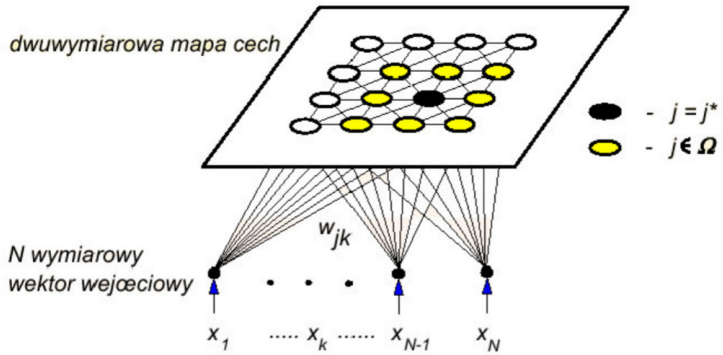
Sieć Kohonena (1981)

- **Sieć Kohonena** - najbardziej popularny model tworzenia map, stąd pojęcie SOM często utożsamiane z „siecią Kohonena”
- dwie fazy uczenia
  - konkurencja - najpierw wybierz globalnego zwycięzcę
  - kooperacja - uczenie neuronu zwycięzcy oraz neuronów w sąsiedztwie zwycięzcy
- sieci bez hamowania, brak połączeń wewnątrz warstwy
- wejście jest sygnałem ciągłym, nie posiada narzuconej struktury topologicznej, dowolna wymiarowość sygnału wejściowego, mapowanie nD do 1D, 2D, 3D



Rys: Marcus Kaiser, Neuroinformatics, lectures

Mapa cech Kohonena



- neurony zachowują porządek topologiczny (np. siatka 2D) tworząc **mapę cech**
- sąsiadujące na siatce neurony powinny reagować na podobne wzorce
- podobieństwo wzorców $\mathbf{x}$ i $\mathbf{w}_i$ określa odległość $\|\mathbf{x} - \mathbf{w}_i\|$ w przestrzeni wejściowej $\mathbf{x}, \mathbf{w}_i \in \mathbb{R}^N$
- odległość neuronów na siatce (mapie) $\|\mathbf{r}_i - \mathbf{r}_j\|$ gdzie $\mathbf{r}_i, \mathbf{r}_j \in \mathbb{R}^K$ (np. $K = 2$) określa zasięg sąsiedztwa

Rys: Małgorzata Krętowska, Sztuczne sieci neuronowe

Algorytm SOM

1. Zainicjuj wagi połączeń neuronów ukrytych
2. Dla danego sygnału $\mathbf{x}(\mathbf{t})$ w kroku $t = 1, 2, \ldots, T$ znajdź najsilniej reagujący neuron c (np. najbliższy)

$$\|\mathbf{x} - \mathbf{w}_j\| = \sqrt{\sum_i (x_i - w_{ij})^2}, \quad c = \arg\min_j \|\mathbf{x} - \mathbf{w}_j\|$$

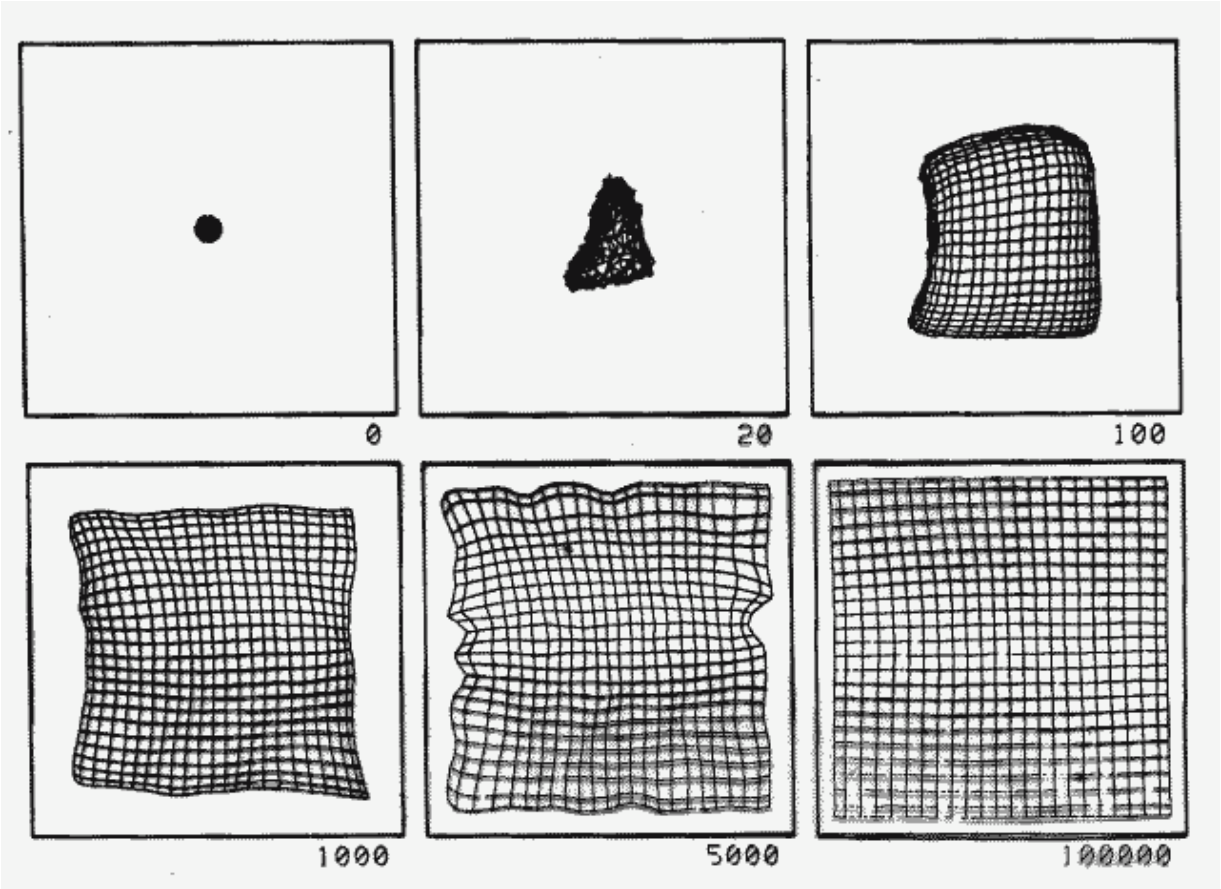
3. Zmień wagi neuronu c oraz neuronów w sąsiedztwie $O(c)$ przybliżając je w stronę wektora $\mathbf{x}$

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta(t) h\left(\mathbf{r}_i, \mathbf{r}_c, t\right) \left[\mathbf{x}(t) - \mathbf{w}_i(t)\right] \text{ dla } i \in O(c)$$

Funkcja $h(\mathbf{r}, \mathbf{r}_c, t)$ określa zasięg sąsiedztwa zwycięzcy

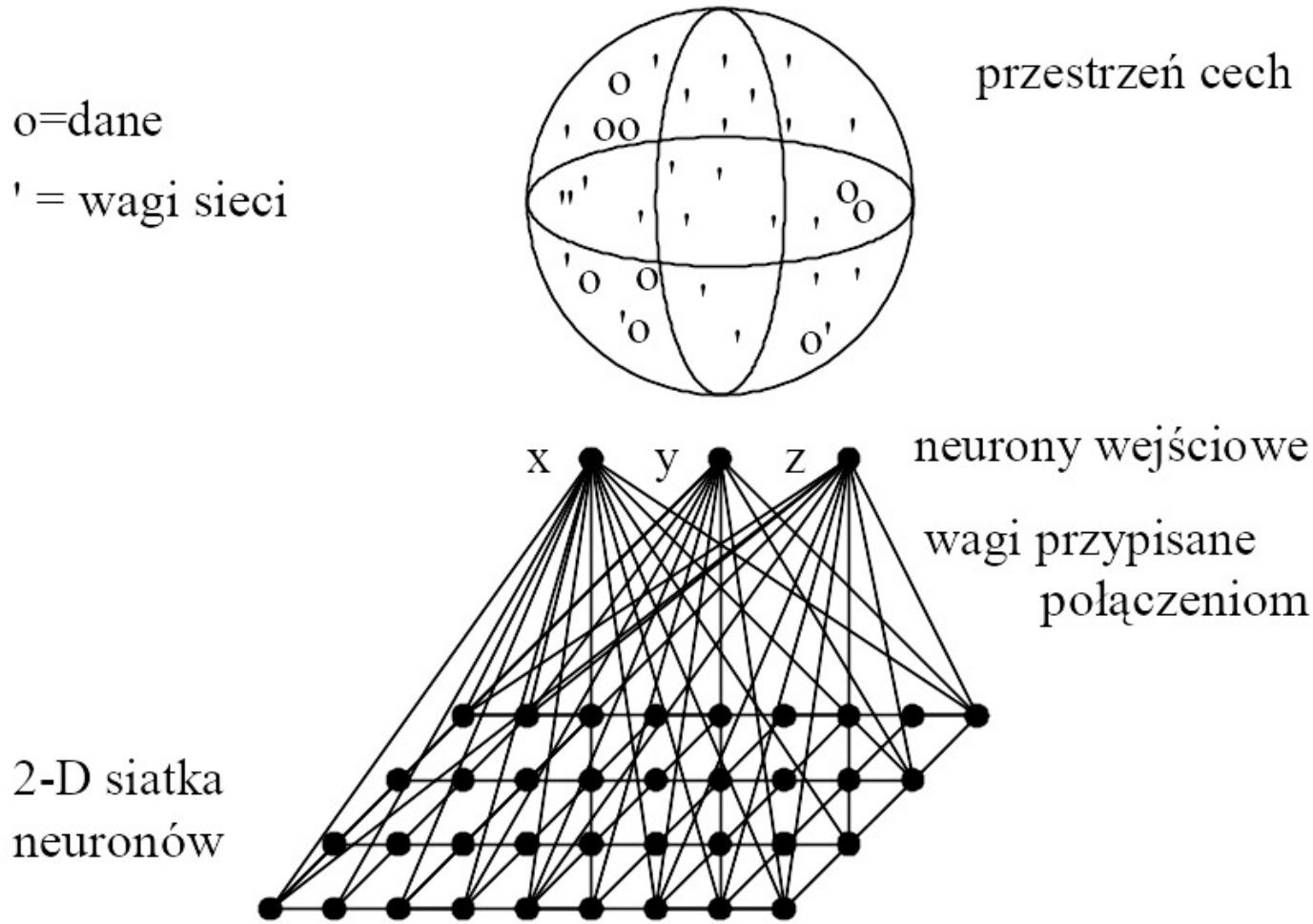
$$h\left(\mathbf{r}, \mathbf{r}_c, t\right) = h_0(t) \exp\left(-\frac{\|\mathbf{r} - \mathbf{r}_c\|^2}{2\sigma^2(t)}\right)$$

Przykład: mapowanie 2D -> 2D

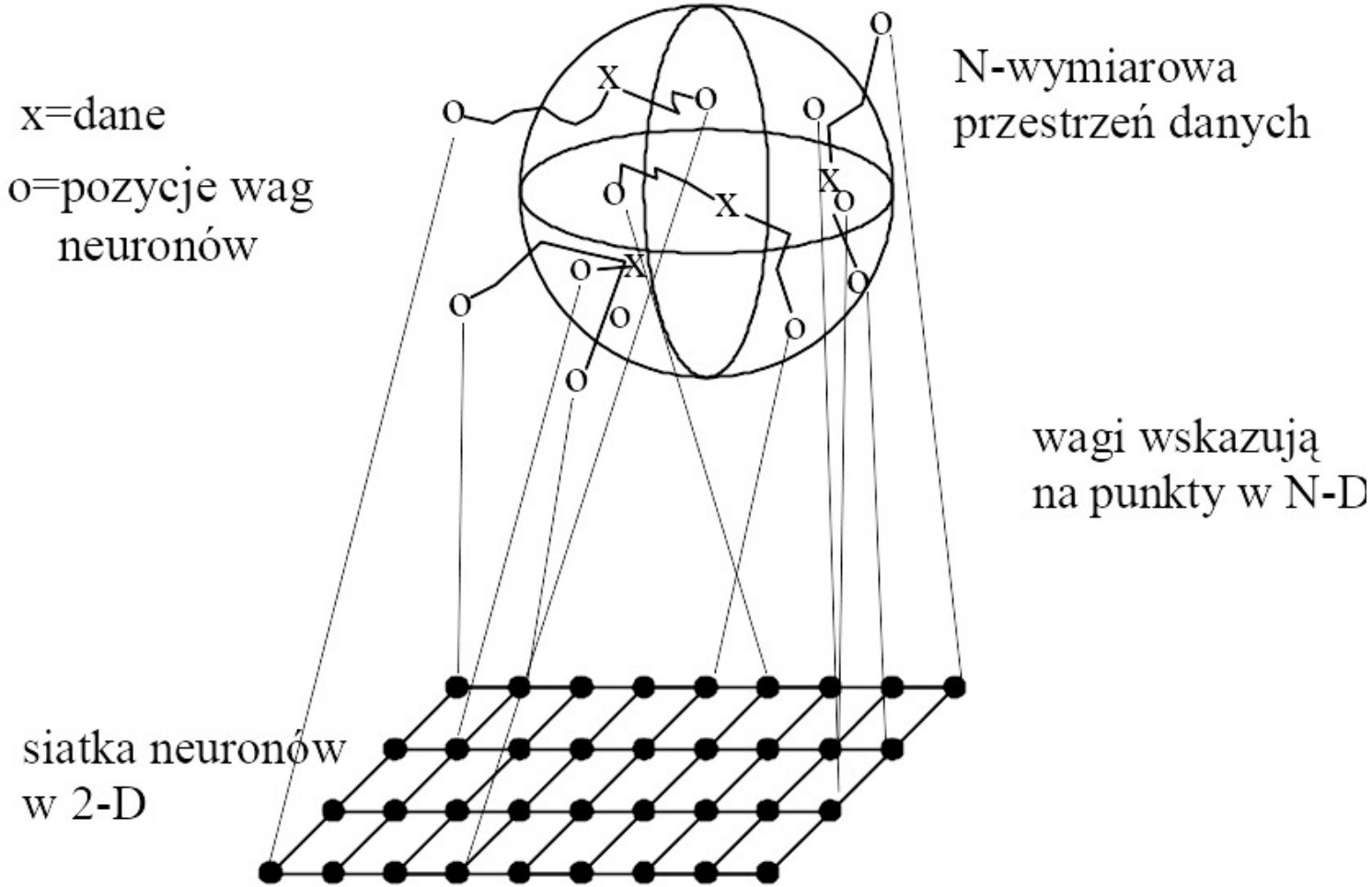


Rozkład jednostajny w kwadracie. SOM uczy się jednorodnego rozkładu. Początkowo wszystkie $\mathbf{w} \approx 0$

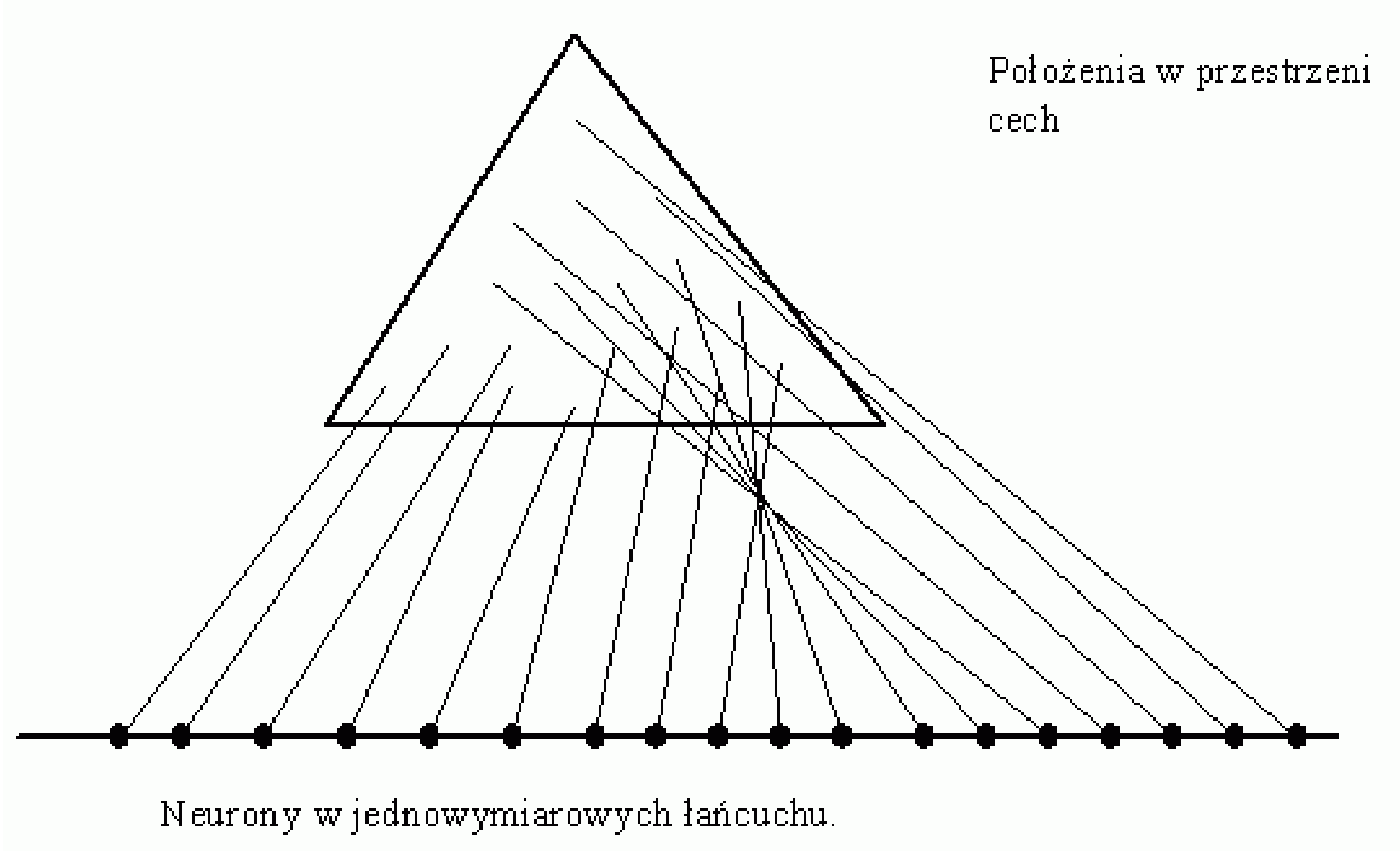
Sieć 2D, cechy 3D (mapowanie 3D->2D)



Uczenie sieci 2D

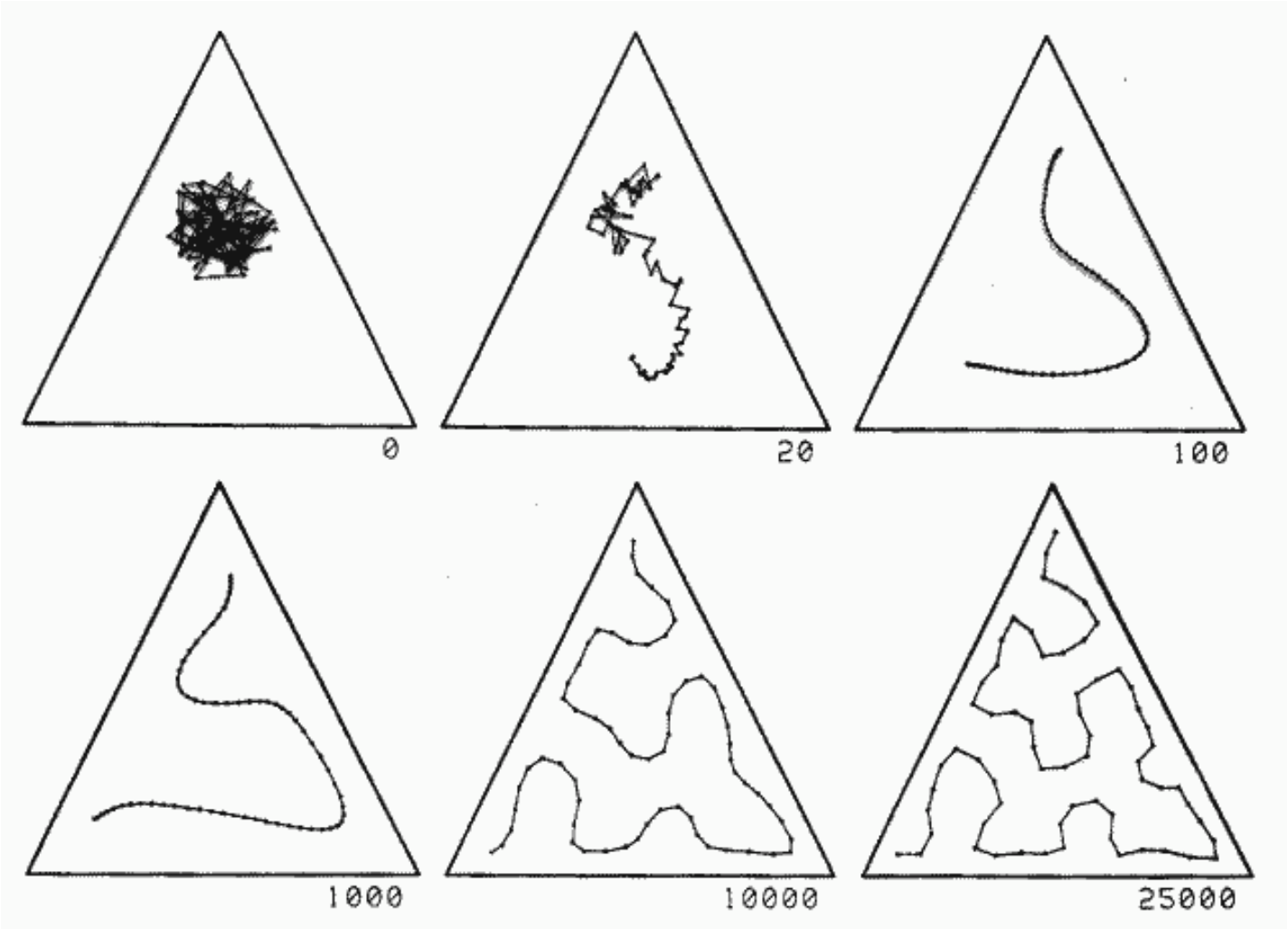


Sieć mapująca dane 2D do 1D (neurony ustawione w linii)

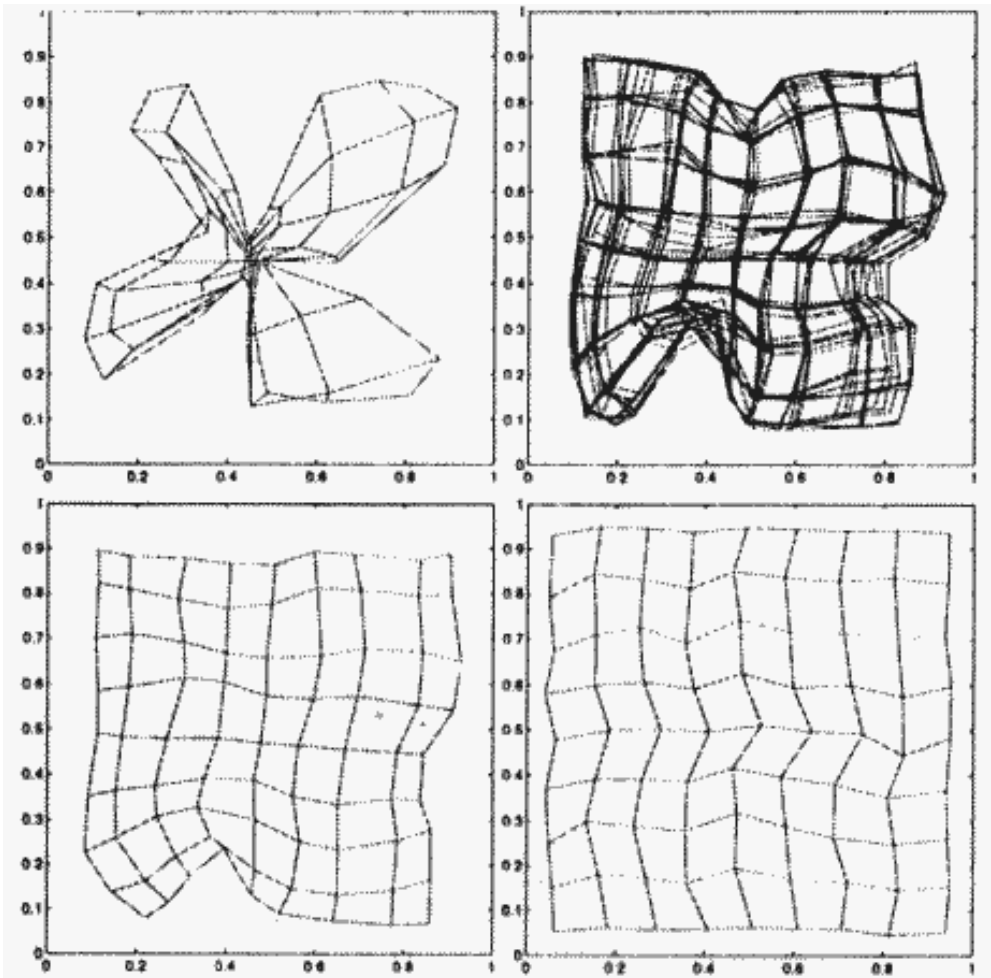


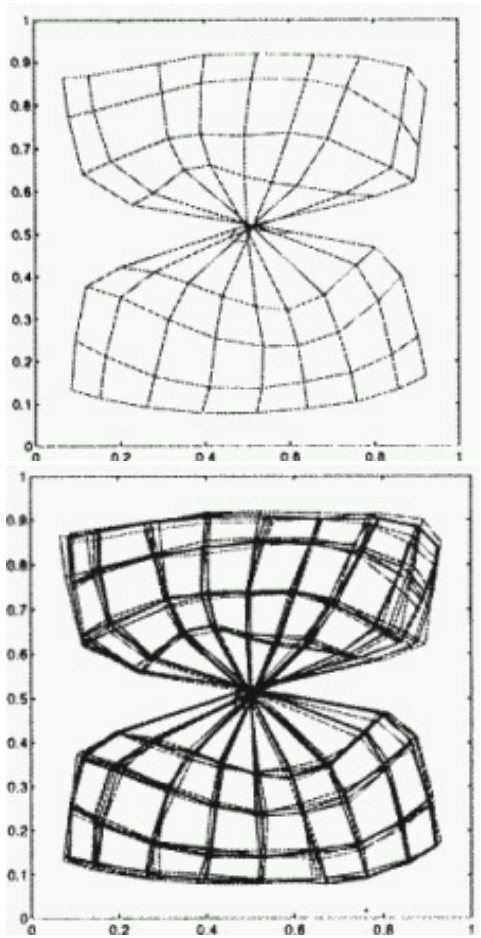
Przykład: fraktalne krzywe Peano

Krzywa Peano (lub Hilberta) - krzywa wypełniająca płaszczyznę



Zniekształcenia





Niektóre zniekształcenia nie mają szans na zniwelowanie

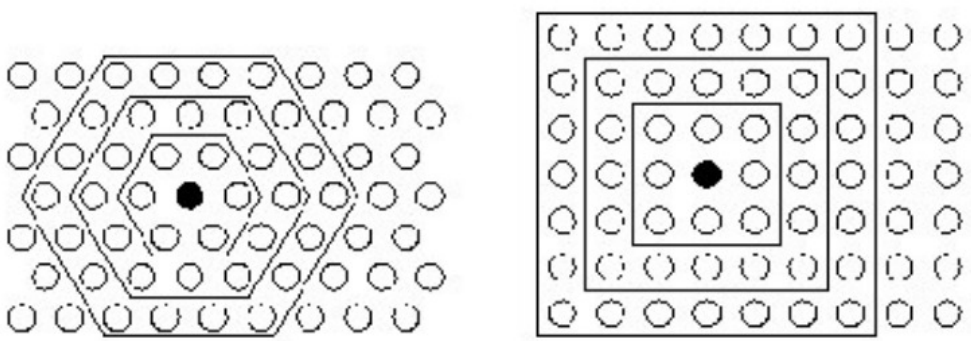
Rozmiar sąsiedztwa

Klasyczny algorytm Kohonena - sąsiedztwo prostokątne

$$h(\mathbf{r}_i, \mathbf{r}, t) = \begin{cases} 1 & \text{dla} \quad d(\mathbf{r}_i, \mathbf{r}) \leq \sigma(t) \\ 0 & \text{dla pozostałych} \end{cases}$$

Promień sąsiedztwa $\sigma(t)$ jest zmniejszany w czasie treningu, może być określony miarą odległości lub liczbą neuronów sąsiadujących

- w początkowej fazie uczenia uwzględnia duże sąsiedztwo
- w końcowej fazie tylko neuron zwycięski jest aktualizowany



(a) Hexagonal grid (b) Rectangular grid

Funkcja sąsiedztwa SOM

Dla gausowskiej funkcji sąsiedztwa

$$h\left(\mathbf{r}_i, \mathbf{r}_j, t\right) = \exp\left(-\frac{\left|\mathbf{r}_i - \mathbf{r}_j\right|^2}{2\sigma(t)^2}\right)$$

zasięg funkcji maleje w czasie treningu

$$\sigma(t) = \sigma_0 \mathrm{e}^{-t/t_{\max}}$$

Dla $\sigma \approx 0$ funkcja sąsiedztwa obejmuje tylko zwycięzcę i SOM sprowadza się do metody kwantyzacji wektorowej (np. on-line k-średnich)

Funkcja błędu SOM

Próba wprowadzenia funkcji błędu (Luttrell; Heskes i Kappen)

Błąd lokalny i-tego neuronu E_i (suma po wszystkich neuronach):

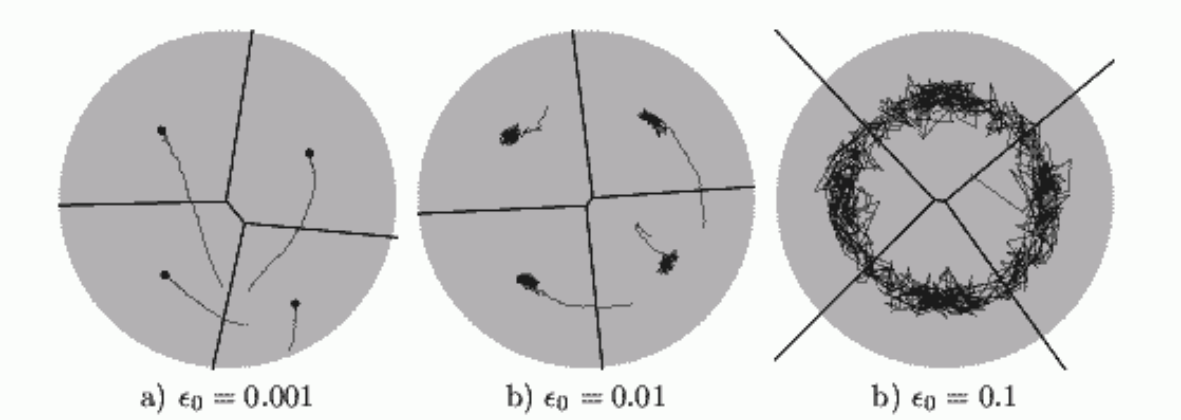
$$E_i(\mathbf{x}; t) = \frac{1}{2} \sum_j h\left(\mathbf{r}_i, \mathbf{r}_j, t\right) \left\|\mathbf{x} - \mathbf{w}_j(t)\right\|^2$$

Neuron-zwycięzca ma najmniejszy błąd lokalny:

$$c = \arg\min_i E_i(\mathbf{x}; t)$$

Stała uczenia

Duży współczynnik uczenia prowadzi do eksploracji znacznej części przestrzeni



Współczynnik uczenia maleje w kolejnych iteracjach

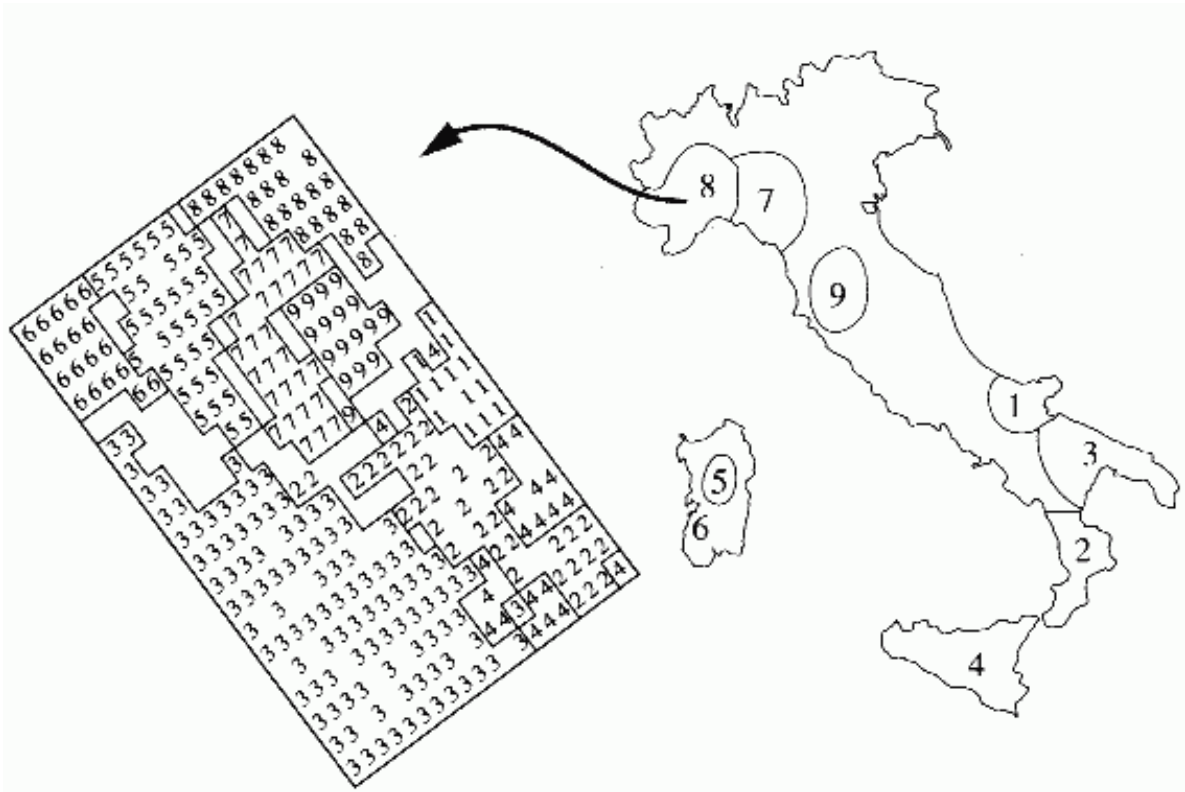
$$\eta(t) = \eta_0 e^{-\frac{t}{t_{max}}}$$

Własności SOM

- sąsiadujące neurony kodują sąsiednie obszary wejściowe, ale sąsiednie obszary mogą być kodowane przez odległe neurony, relacje odległości nie zawsze są zachowane
- powstają „skręcone" konfiguracje przy zbyt szybkim zmniejszaniu zasięgu funkcji sąsiedztwa
- powolna zbieżność algorytmu SOM, wymaga wielu iteracji
- problemy związane z ustaleniem zbieżności i punktów stacjonarnych, wyniki analityczne znane tylko w 1D dla ciągłego czasu (wartości wag porządkują się wzdłuż prostej)
- w oryginalnym SOM nie ma funkcji błędu, brakuje metody oceny stopnia zniekształceń, brak szukanego minimum i nie można wyznaczyć gradientu
- zwykle jakość klasyfikacji SOM będzie gorsza w porównaniu z dedykowanymi metodami, SOM służy głównie do wizualizacji ... ale nie ma gwarancji, że wizualizacja wiarygodnie oddaje naturę danych

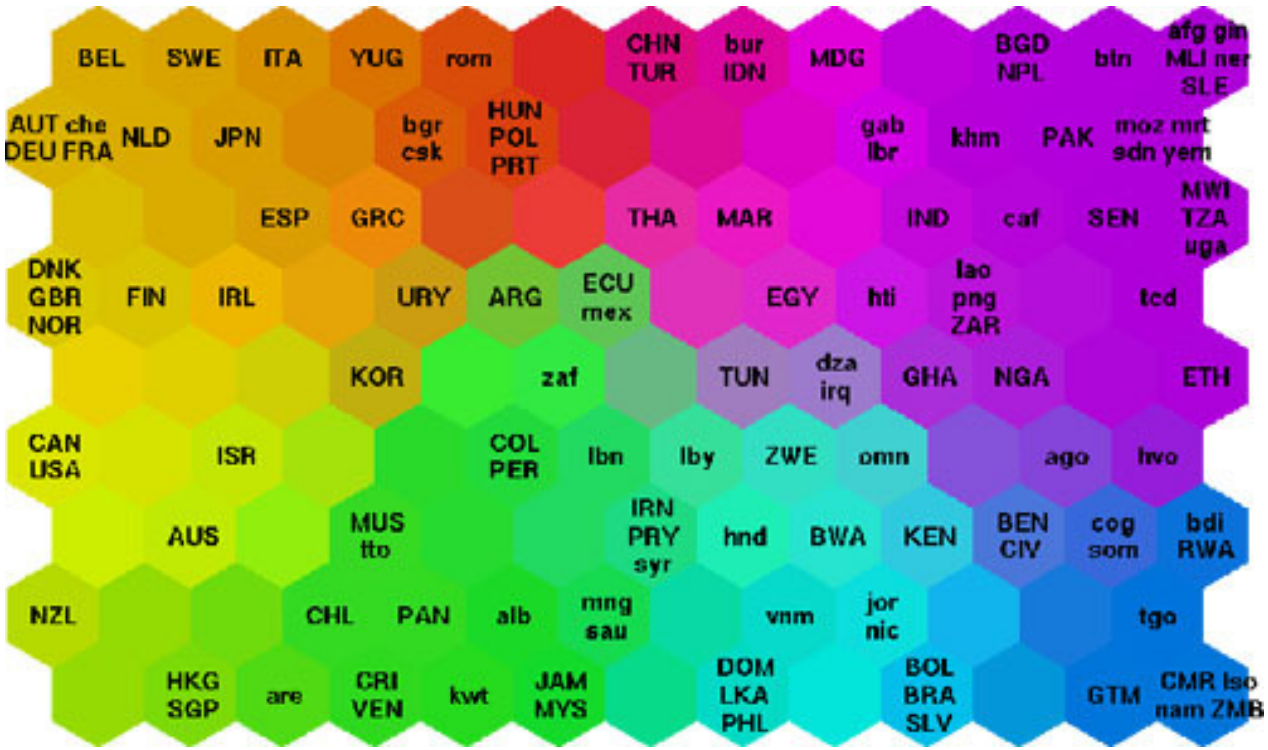
Przykład: Włoska oliwa

Dane: 572 próbki oliwy z 9 prowincji Włoch
Cechy: poziom 8 tłuszczy w każdej próbce.
Mapa SOM: 20 x 20, Redukcja 8D => 2D.
Dokładność klasyfikacji około 95-97%.
Topograficzne relacje zostały zachowane.



Przykład: World poverty map

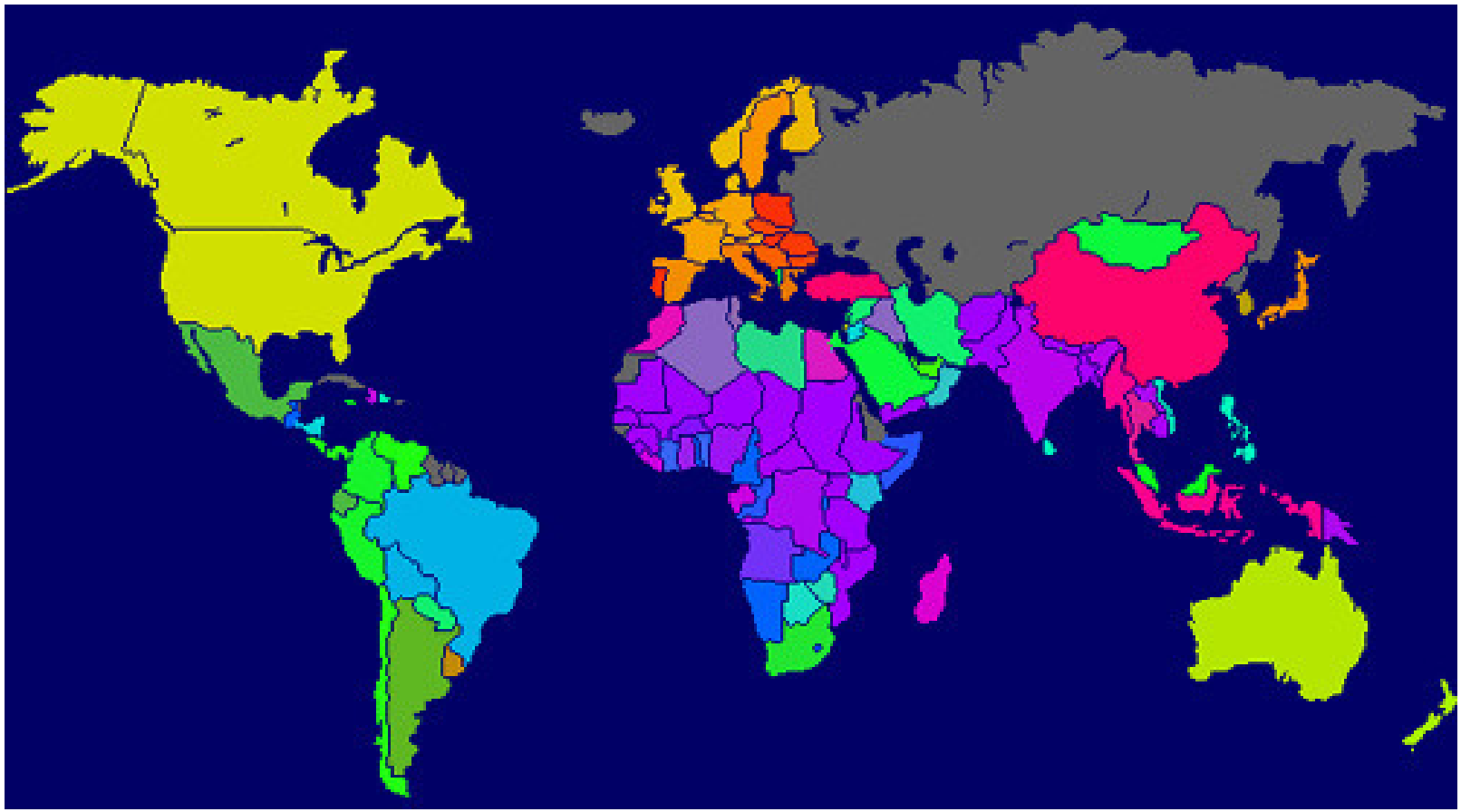
Dane: WorldBank data 1992 rok, 39 współczynników opisujących jakość życia w kraju (poziom służby zdrowia, edukacji, etc.)



Kraje o podobnej wartości poziomu życia pobudzają bliskie grupy neuronów, od krajów o najwyższym wsp. jakości życia (żółty) do najniższego (niebieski)

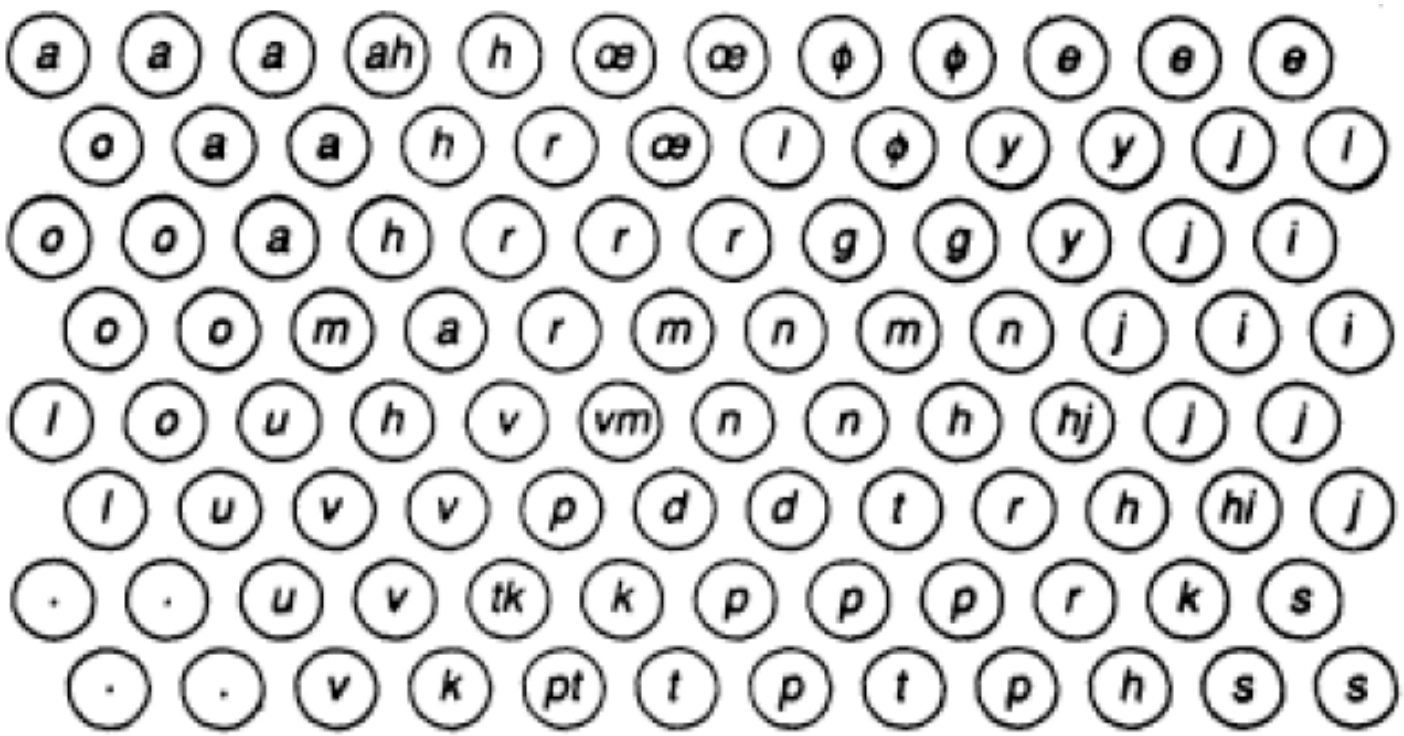
Przykład: World poverty map

Po naniesieniu kolorów na mapę świata



Klawiatura fonetyczna

Dane: 10 ms próbki FFT mowy



Mapa fonetyczna dla języka Fińskiego, Kohonen 1980

Demos

- Wizualizacje do wykładu M. Czoków, J. Piersa: ☞Sześcian, ☞Sześcian z funkcją gaussowską, ☞Kula, ☞Kula z funkcją gaussowską
- ☞SOM 2D dla pogrupowanych danych
- Self-organizing Maps: PyMVPA kolory na mapie 2D
- ☞Demo GNG
- ☞Klasyfikacja kolorów
- ☞GeoSOM Environmental modelling

Przykłady zastosowania SOM

Helsinki University of Technology web site
`http://www.cis.hut.fi/research/refs/` has a list (2010) of
$>$ 7700 papers on SOM and its applications !

- Badania mózgu: modelowanie topograficznych map w różnych obszarach kory (motoryczna, sensoryczna, wizyjna)
- Robotyka i AI: analiza danych z czujników, sterowanie ruchami robota, mapy orientacji przestrzennej
- Kategoryzacja dokumentów
- Analiza skupień genów, białek, cząsteczek chemicznych, fonemów, dźwięków zwierząt, obiektów astronomicznych, dane ekonomiczne i finansowe
- Kompresja danych (obraz, dźwięk), filtrowanie informacji
- Medical and technical diagnostics.

- Analiza języka naturalnego: linguistic analysis, parsing, learning languages, hyphenation patterns
- Problemy optymalizacyjne: configuration of telephone connections, VLSI design, time series prediction, scheduling algorithms.
- Przetwarzanie sygnałów: adaptive filters, real-time signal analysis, radar, sonar seismic, USG, EKG, EEG and other medical signals ...
- Rozpoznawanie i analiza obrazów: segmentation, object recognition, texture recognition ...
- Content-based retrieval: examples of WebSOM, PicSom – similarity based image retrieval, RGB and textures.
- Viscovery SOMine, komercyjny program do wizualizacji, eksploracji, klasyfikacji, prognozowania oparty na SOM.

Samoorganizacja

Uczenie hebbowskie

Samoorganizacja

- **uczenie nienadzorowane**: wykrywa samoistnie (bez nadzoru) istotne cechy powiązań między sygnałami
- globalne uporządkowanie sieci jest możliwe przez działania samoorganizujące prowadzone **lokalnie** w różnych punktach sieci, niezależnie od siebie
- pewne neurony lub grupy neuronów **współpracują** lub/i **konkurują** ze sobą specjalizując się w wykrywaniu określonych wzorców

Sieci samoorganizujące się

Uczenie konkurencyjne

- reguła Kohonena
- konkurencja WTA, WTM
- kwantyzacja wektorowa, SOM

Uczenie hebbowskie

- reguła Hebba, reguła Oja
- asocjacje i korelacje między sygnałami
- dekompozycja składowych głównych PCA (Principal Component Analysis)
- dekompozycja na składowe niezależne ICA (Independent Component Analysis), separacja źródeł BSS (Blind Source Separation), nielinowe PCA, ...

Reguła Hebba (1949)

„Neurons, that fire together, wire together."

Jeżeli neuron A jest wielokrotnie pobudzany przez neuron B, to staje się on jeszcze bardziej czuły na pobudzenie tego neuronu

- synchroniczne pobudzanie neuronów A i B wzmacnia połączenie synaptyczne między nimi
- brak współpobudzenia nie prowadzi do wzmacniania połączenia

Zmiana wagi połączenia pomiędzy neuronami A i B

$$w_{AB}(k+1) = w_{AB}(k) + \eta y_A(k) y_B(k)$$

Reguła Hebba

Dla neuronu liniowego

$$y = \mathbf{w}^T \mathbf{x}$$

reguła Hebba przybiera postać

$$\Delta \mathbf{w} = \eta y \mathbf{x} = \eta \mathbf{w}^T \mathbf{x} \mathbf{x} = \eta \mathbf{x} \mathbf{x}^T \mathbf{w}$$

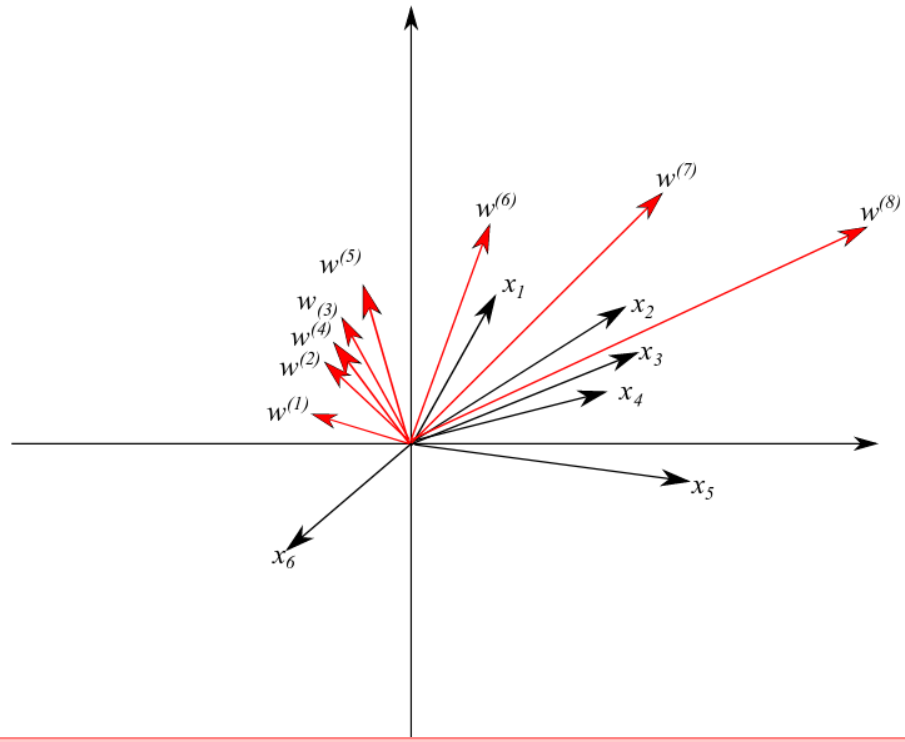
Decydujący wpływ na wartość końcową $\mathbf{w}$ będą miały sygnały zgrupowane (rozciągające się) wzdłuż znalezionego kierunku - maksymalizacja wariancji $\mathbf{w}^T \mathbf{x}$.

Przy prezentacji wielu próbek uczących wektor wag kumuluje wartości macierzy autokorelacji $\mathbf{R}$ wektorów $\mathbf{x}$

$$\Delta \mathbf{w} = \eta \mathbf{R} \mathbf{w} \quad \text{gdzie} \quad \mathbf{R} = E[\mathbf{x} \mathbf{x}^T]$$

Interpretacja geometryczna reguły Hebba

Wektor wag przemieszcza się w kierunku odpowiadającym największemu rozrzutowi danych (maleje kąt względem kierunku największej wariancji danych)



Trening prowadzi do nieograniczonego wzrostu wag (norma $\|\mathbf{w}\|$ rośnie w każdym kroku)

Modyfikacje reguły Hebba

Zapobieganie nieograniczonemu wzrostowi wag:

- sztywne granice

$$w_{min} < w_i < w_{max}$$

- miękkie granice

$$\Delta w_i = \eta x_i y (w_i - w_{min})(w_{max} - w_i)$$

- normalizacja wag po każdym kroku uczenia
- dodanie czynnika normalizującego wagi, np. reguła Oji (1982)

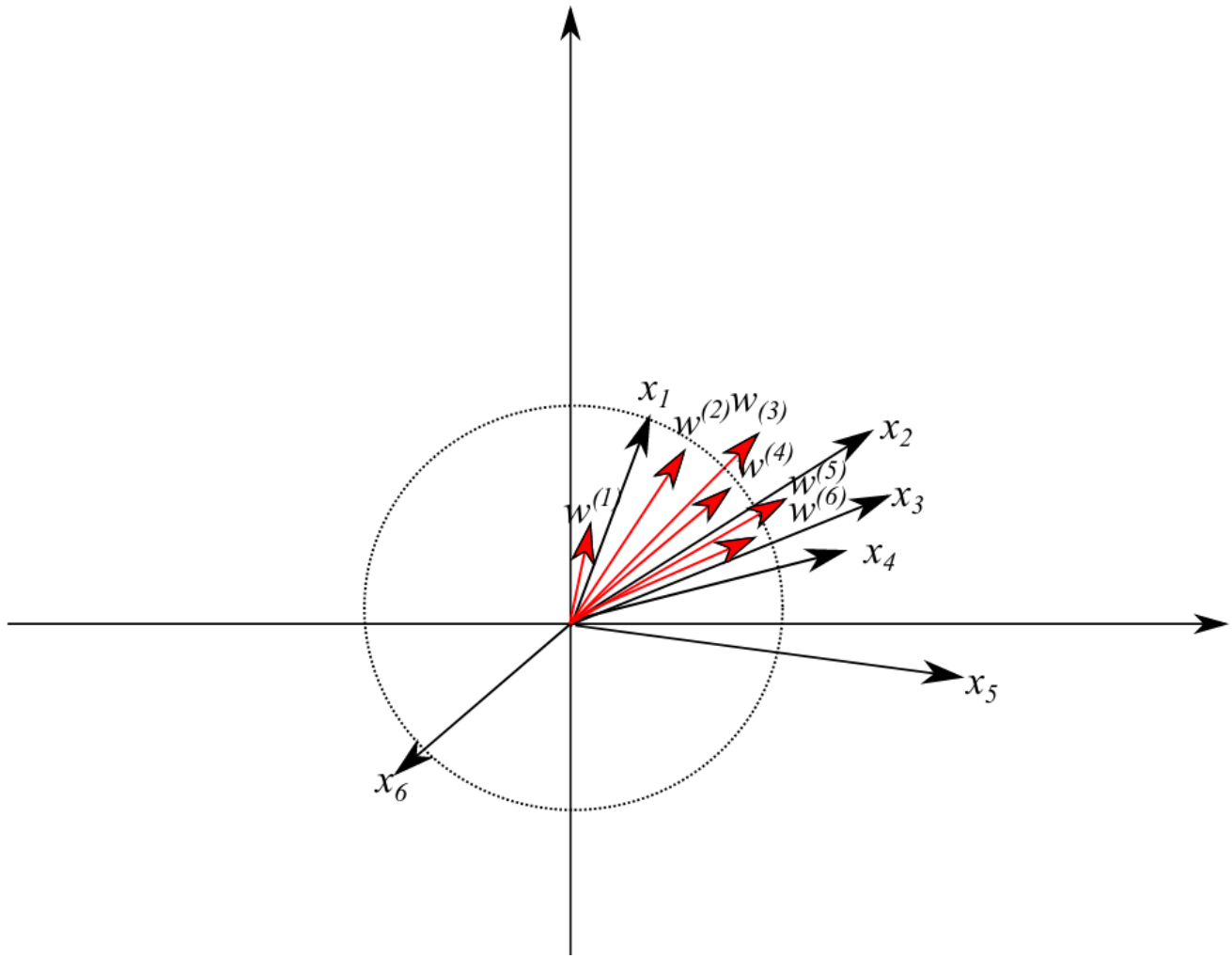
$$\Delta w_i = \eta y \left(x_i - y w_i\right)$$

Reguła Oji

$$\Delta \mathbf{w} = \eta \left(y\mathbf{x} - \alpha y^2 \mathbf{w} \right)$$

- dodanie wyrazu zanikania wagi proporcjonalnego do y^2
- adaptacja prowadzi do unormowania wag $\|\mathbf{w}\| = \frac{1}{\sqrt{\alpha}}$
- wektor wag zbliża się do wektora własnego macierzy kowariancji $E[\mathbf{X}\mathbf{X}^T]$ o największej wartości własnej
- wektor wag leży wzdłuż kierunku maksymalizującego wartość oczekiwaną y^2 (wariancja wartości wyjściowej neuronu)

Interpretacja geometryczna reguły Oji



Własności reguły Oji

Dla wyuczonego wektora wag wartość oczekiwana zmian dla wszystkich wektorów uczących

$$E[\Delta \mathbf{w}] = 0$$

stąd

$$E[\Delta \mathbf{w}] = \eta E[y(\mathbf{x} - y\mathbf{w})] = \eta E[(\mathbf{w}^T\mathbf{x})\mathbf{x} - y^2\mathbf{w}]$$
$$= \eta \left(E[\mathbf{x}\mathbf{x}^T]\mathbf{w} - E[y^2]\mathbf{w}\right) = 0$$

Macierz kowariancji $E[\mathbf{x}\mathbf{x}^T] = \mathbf{R}_x$.
Oznaczmy $\lambda = E[y^2]$

Reguła Oji prowadzi do wyznaczenia wektora własnego $\mathbf{w}$ macierzy kowariancji $\mathbf{R}_x$

$$\mathbf{R}_x\mathbf{w} = \lambda\mathbf{w}$$

Normalizacja wag

$$\lambda = E[y^2] = E[(\mathbf{w}^T\mathbf{x})(\mathbf{w}^T\mathbf{x})] = \mathbf{w}^T E[\mathbf{x}\mathbf{x}^T]\mathbf{w} = \mathbf{w}^T\mathbf{R}_x\mathbf{w}$$

ponieważ

$$\mathbf{R}_x\mathbf{w} = \lambda\mathbf{w}$$

stąd

$$\lambda = \mathbf{w}^T\lambda\mathbf{w} \quad \Rightarrow \quad \|\mathbf{w}\| = 1$$

Dodatkowy czynnik ogranicza wartość wag.

Analiza składowych głównych PCA

Transformacja liniowa

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

gdzie

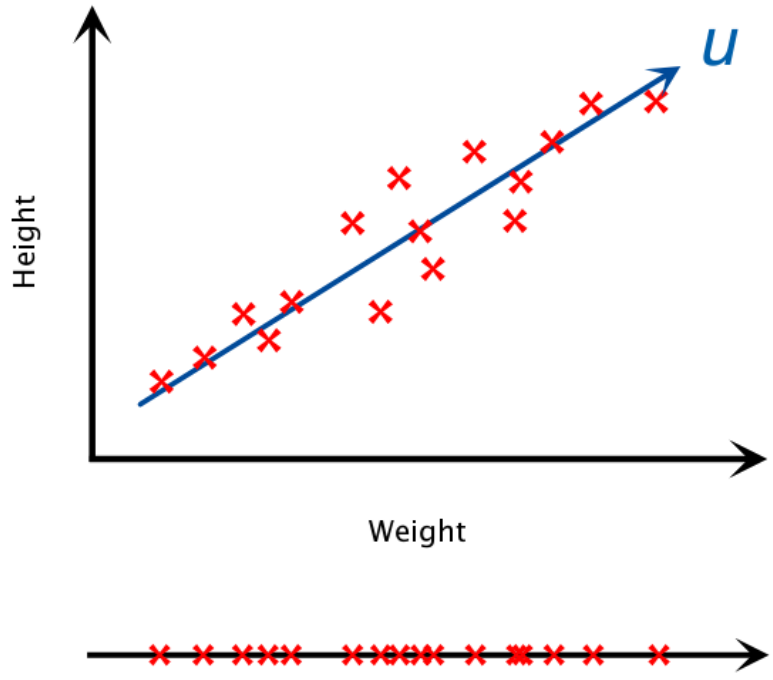
$$\mathbf{x} \in \mathbb{R}^N \qquad \mathbf{y} \in \mathbb{R}^K \quad \mathbf{W} \in \mathbb{R}^{K \times N} \qquad K < N$$

Redukcja wymiarowości z zachowaniem maksymalnej informacji zawartych w danych.

W PCA kierunek zawierający najwięcej informacji to kierunek o największej wariancji

Projekcja danych na wektor wag

Projekcja danych 2D na kierunek o największej wariancji



Rys: Boedecker, Machine Learning Summer 2015

PCA

Niech $\mathbf{X} \in \mathbb{R}^N$ wektor losowy o zerowej wartości oczekiwanej, wówczas wartość oczekiwana macierzy kowariancji

$$R_x = E[\mathbf{X}\mathbf{X}^T]$$

dla skończonej próbki o liczebności p

$$R_x \approx \frac{1}{p} \sum_{i=1}^{p} \mathbf{x}_i \mathbf{x}_i^T$$

PCA

Wartości własne λ_i oraz wektory własne $\mathbf{w}_i$ macierzy kowariancji

$$\mathbf{R}_x \mathbf{w}_i = \lambda_i \mathbf{w}_i \qquad i = 1, \ldots, N$$

R jest symetryczna i dodatnio półokreślona $\Rightarrow$ wartości własne są rzeczywiste i nieujemne
Niech

$$\lambda_1 > \lambda_2 > \ldots > \lambda_N \geq 0$$

Ograniczając do K pierwszych składowych

$$\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_k]^T$$

otrzymujemy transformację PCA

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

PCA

Wariancja wzdłuż i-tego kierunku

$$Var(\mathbf{w}_i^T \mathbf{x}) = E[(\mathbf{w}_i^T \mathbf{x} - E[\mathbf{w}_i^T \mathbf{x}])^2] = E[(\mathbf{w}_i^T \mathbf{R}_x \mathbf{w}_i)] = \lambda_i$$

Celem PCA jest znalezienie wektorów $\mathbf{w}_1, \ldots, \mathbf{w}_K$, które maksymalizują wariancję przy zachowaniu ortogonalności

$$\mathbf{w}_i^T \mathbf{w}_j = 0 \quad \text{dla} \quad j \neq i$$

$$\mathbf{w}_i^T \mathbf{w}_i = 1$$

Algorytm PCA

1. Wyśrodkowanie danych dla każdej cechy j

$$x_{ij} \leftarrow x_{ij} - \frac{1}{n}\sum_{j=1}^{n} x_{ij}$$

2. Wyznaczenie macierzy kowariancji

$$\mathbf{R}_x = \frac{1}{n}\mathbf{X}\mathbf{X}^T$$

3. Diagonalizacja za pomocą rozkładu SVD

$$\mathbf{S} = \mathbf{W}^{-1}\mathbf{R}_x\mathbf{W}$$

   $\mathbf{W}$ zawiera wektory własne, $\mathbf{S}$ wartości własne na diagonali

4. Projekcja danych na kierunki wektorów własnych odpowiadającym K największym wartościom własnym

$$\mathbf{y}_i = \mathbf{W}_K\mathbf{x}_i$$

Rekonstrukcja

Rekonstrukcja obrazu wejściowego

$$\hat{\mathbf{x}} = \mathbf{W}_k^T \mathbf{y}$$

PCA minimalizuje błąd rekonstrukcji

$$E_r = E[\|\mathbf{x} - \hat{\mathbf{x}}\|^2]$$

dla k składowych PCA

$$E_r = \sum_{i=k+1}^{N} \lambda_i$$

Minimalizacja E_r odpowiada maksymalizacji $\sum_{i=1}^{K} \lambda_i$

PCA

- Gdy zmienne są skorelowane to znajomość tylko części zmiennych pozwala określić pozostałe
- Względny wkład poszczególnych składowych

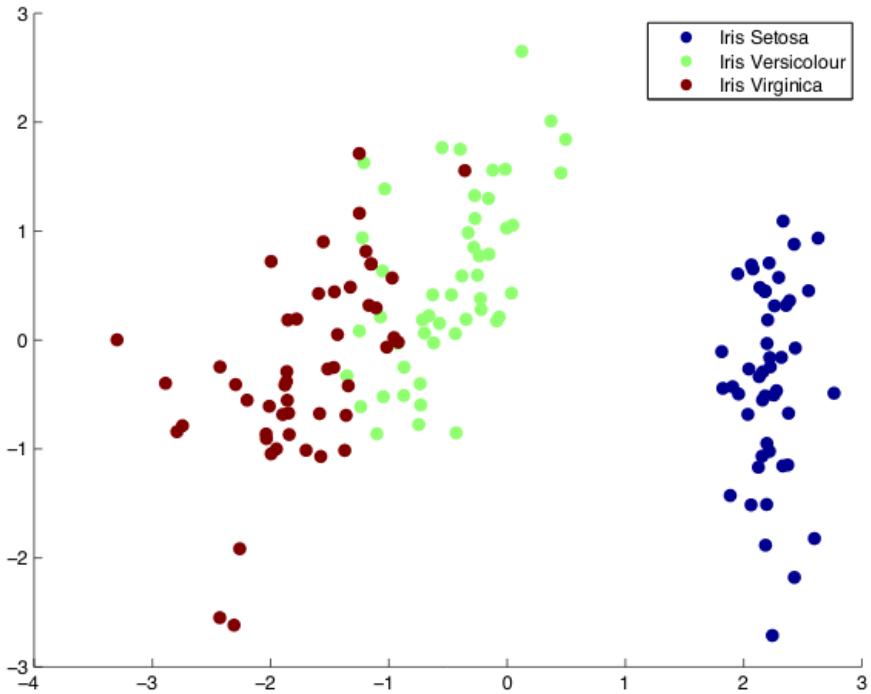
$$m_i = \frac{\lambda_i}{\sum \lambda_j}$$

- Standardowe metody wyznaczania wektorów własnych macierzy $\mathbf{R}_x$ dla wysokowymiarowych danych są złożone

Metody adaptacyjne wyznaczania składowych głównych (np. reguła Oja)

- nie wymagają obliczania macierzy kowariancji $\mathbf{R}_x$
- mogą być stosowane w metodach on-line, gdy nie mamy dostępu do wszystkich danych jednocześnie

Przykład: PCA na danych Iris

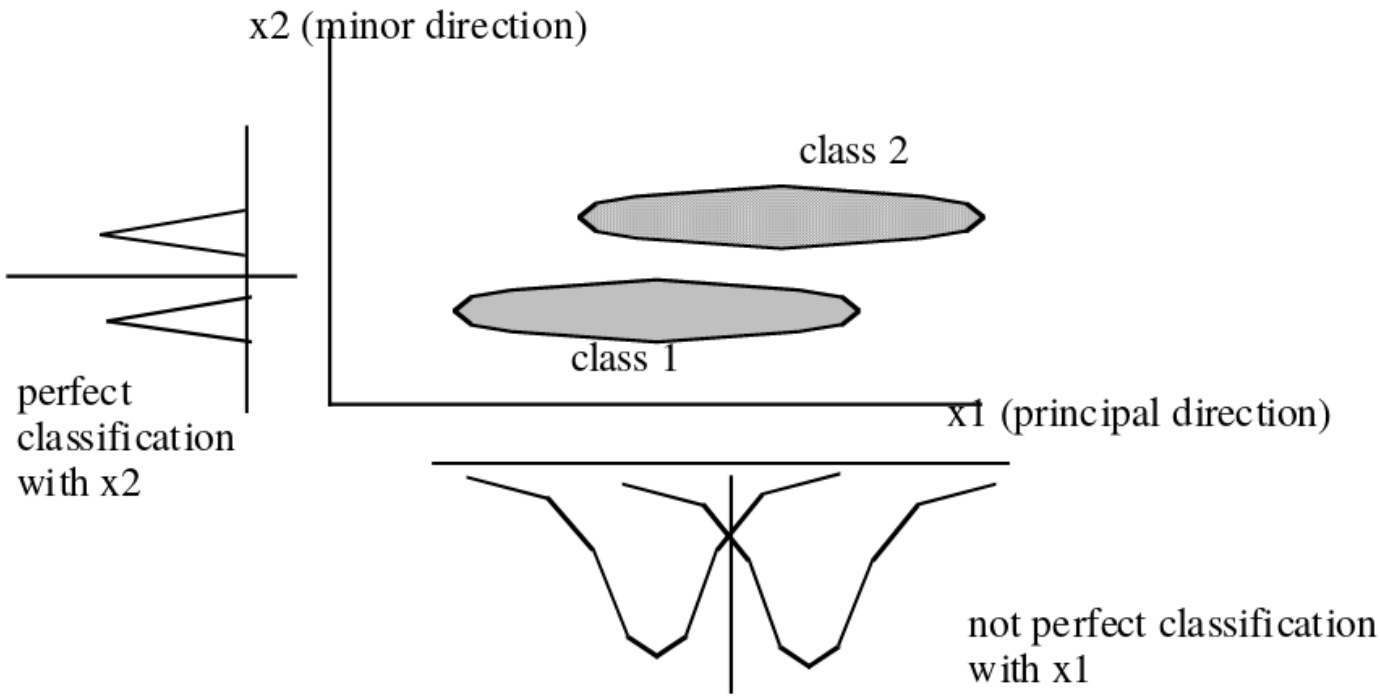


$$\lambda_1 = 2.8914 \quad \lambda_2 = 0.9151 \quad \lambda_3 = 0.1464 \quad \lambda_4 = 0.0205$$

PCA często używane do przetworzenia danych przed treningiem algorytmów uczenia maszynowego

Rys: Boedecker, Machine Learning Summer 2015

PCA i klasyfikacja



Projekcja PCA nie gwarantuje uzyskania najlepszej dyskryminacji

Rys: Principe, Neural and Adaptive Systems

Trening pierwszego kierunku

Estymator pierwszego kierunku (neuron liniowy)

$$y_1 = \mathbf{w}_1^T \mathbf{x} = \sum_{j=0}^{N} w_{1j} x_j$$

uczony regułą Oja

$$\mathbf{w}_1(k+1) = \mathbf{w}_1(k) + \eta(k) y_1(k) \left(\mathbf{x}(k) - \mathbf{w}_1(k) y_1(k)\right)$$

Dobór $\eta(k)$ jest istotny - powinien maleć z czasem, np.:

$$\eta(k) = \frac{\eta(0)}{k^\gamma} \qquad \text{gdzie} \qquad 0.5 \leq \gamma \leq 1$$

Sieci neuronowe uczone regułą Hebba

W warstwie składającej się z neuronów liniowych uczonych regułą Hebba wszystkie neurony będą zbiegały do tego samego kierunku maksymalizującego wariancję.

Mechanizmy konkurencji wymuszające różnorodność rozwiązań:

- WTA, zwycięzca bierze wszystko, tylko neuron z najsilniejszą aktywacją podlega uczeniu
- WTM, grupa k najbardziej wzbudzonych neuronów podlega adaptacji
- wymuszenie ortogonalności między wektorami wag, uogólniona reguła Hebba (GHA)
- dodanie połączeń ze sprzężeniem zwrotnym wpływającymi na pobudzenia sąsiednich neuronów, np. APEX

Uogólniony algorytm hebbowski (GHA)

Jednowarstwowa sieć liniowa (Sanger, Oja, 1989)

$$y_i = \mathbf{w}_i^T \mathbf{x}$$

Reguła uczenia Sangera z czynnikiem ortogonalizującym Grama-Schmidta

$$\Delta \mathbf{w}_i = \eta y_i \left(\mathbf{x} - \sum_{k=1}^{i} y_k \mathbf{w}_k \right)$$

oznaczając

$$\hat{\mathbf{x}}_i(k) = \mathbf{x} - \sum_{k=1}^{i-1} y_k \mathbf{w}_k$$

otrzymujemy lokalną regułę Oji dla przeskalowanych wejść $\hat{\mathbf{x}}_i$

$$\Delta \mathbf{w}_i = \eta y_i \left(\hat{\mathbf{x}} - \mathbf{w}_i y_i \right)$$

- Wektory wag $\mathbf{w}_i$ kolejnych neuronów zbiegają do wartości własnych macierzy kowariancji $\mathbf{R}$ (kolejne składowe główne).
- GHA jest algorytmem on-line, nie wymaga obliczania macierzy kowariancji $\mathbf{R}$

Uczenie anty-hebbowskie

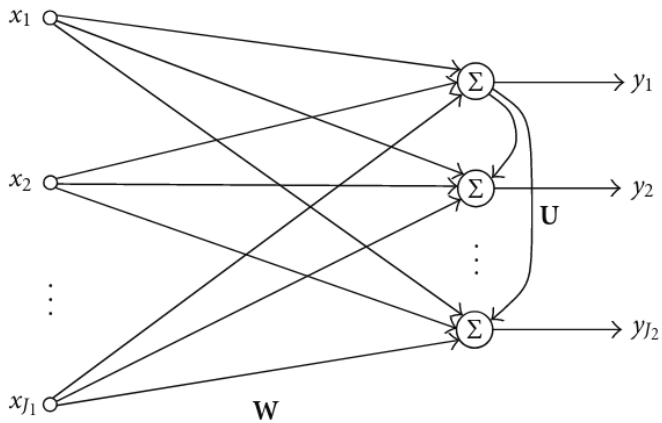
Reguła anty-Hebbowska

$$\Delta w_{ij} = -\eta x_i y_j$$

- znak minus sprawia, że trening prowadzi do znalezienia kierunku **minimalizującego** wariancję wyjścia
- uczenie jednostki liniowej prowadzi do zerowania wyjścia - trening dąży do znalezienia projekcji danych do punktu
- jeżeli nie istnieje komponent (kierunek) o zerowej wariancji to wagi zbiegają do 0

APEX

Adaptive Principal Component Extraction (Diamantaras 1990)
Dla określonych $i-1$ wektorów $\mathbf{w_1}, \ldots, \mathbf{w}_{i-1}$ realizujących kierunki PCA sieć iteracyjnie wyznacza i-tą składową główną



$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

$$y_i = \mathbf{w}_i^T \mathbf{x} + \mathbf{u}_i^T \mathbf{y}$$

$\mathbf{y} = (y_1, \ldots, y_{i-1})^T$ wyjścia pierwszych $i-1$ neuronów

$\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_{i-1}]^T$ macierz wag pierwszych $i-1$ neuronów

$\mathbf{u}_i = [u_{1i}, u_{2i}, \ldots, u_{(i-1)i}]^T$ wagi połączeń wewnątrz warstwy między i-tym neuronem a pozostałymi neuronami

Rys: Qiu, et.al., Neural Network Implementations for PCA and Its Extensions

Aktualizacja wag APEX

Dla ustalonych $i-1$ składowych uczeniu podlegają wagi i-tego neuronu $\mathbf{w}_i$ i $\mathbf{u}_i$

- aktualizacja wag $\mathbf{w}_i$ za pomocą reguły Oji

$$\Delta \mathbf{w}_i = \eta(n) y_i(n) \left[\mathbf{x}(n) - y_i(n)\mathbf{w}_i\right]$$

- połączenia między-warstwowe u_{ij} uczone regułą anty-hebbowską (odwrotna reguła Oji)

$$\Delta \mathbf{u}_i = -\eta(n) y_i(n) \left[\mathbf{y}(n) + y_i(n)\mathbf{u}_i\right]$$

- wagi $\mathbf{u}$ odpowiadają za wydzielenie pierwszych $i-1$ składowych z wag i-tego neuronu
- uczenie anty-hebbowskie wymusza ortogonalizację $\mathbf{w}_i$ względem poprzednich składowych

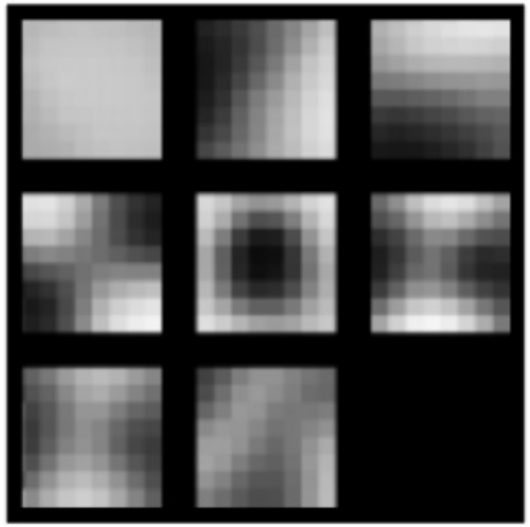
Zastosowanie PCA

- Redukcja wymiarowości
- Wizualizacja danych nD $\rightarrow$ 2D, 3D
- Usuwanie szumu
- Usuwanie korelacji z danych
- Kompresja sygnału, np. kompresja obrazów
- Dane po transformacji PCA często używane do treningu algorytmów uczenia maszynowego

Kompresja danych za pomocą PCA

Wektory treningowe to ramki obrazu (8x8 = 64 współrzędne)
Przykład: obraz 512x512 z 8 bitami na piksel





ordering of the neurons:

neuron 0	neuron 1	neuron 2
neuron 3	neuron 4	neuron 5
neuron 6	neuron 7	(unused)

Neurony w kolejności zawierającej najwięcej informacji potrzebnej do rekonstrukcji

- neuron 0: średni poziom szarości
- neuron 1 i 2: gradient obrazu
- neurony 3-5: pochodna drugiego rzędu obrazu

Rys: Krose, Smagta, An Introduction to Neural Networks, 1996

Przykład: Kompresja obrazu za pomocą APEX

a) oryginał
b) 4 PC
c) 8 PC
d) 12 PC



Rys: Hu Y. H., Hwang J., Handbook of Neural Network Signal Processing

Sieci Heraulta-Juttena

- liniowe sieci samoorganizujące oparte o regułę uczenia Hebba uogólnioną do funkcji nieliniowych
- pierwotnie stosowane do ślepej separacji sygnałów, używane również do analizy składników niezależnych (ICA), składników głównych (PCA), dekonwolucji
- struktura sieci rekurencyjna lub jednokierunkowa
- przetwarzanie sygnałów w czasie rzeczywistym (realizacje sprzętowe)
- funkcje nieliniowe odgrywają istotną rolę w algorytmie uczenia ale nie wpływają na strukturę sieci

Separacja sygnałów

Dla n niezależnych sygnałów źródłowych $s_i(t)$ dostępne są tylko sygnały $x_i(t)$ będące liniową superpozycją sygnałów oryginalnych $s_i(t)$

$$x_i(t) = \sum_{j=1}^{n} a_{ij} s_j(t)$$

Macierz mieszająca

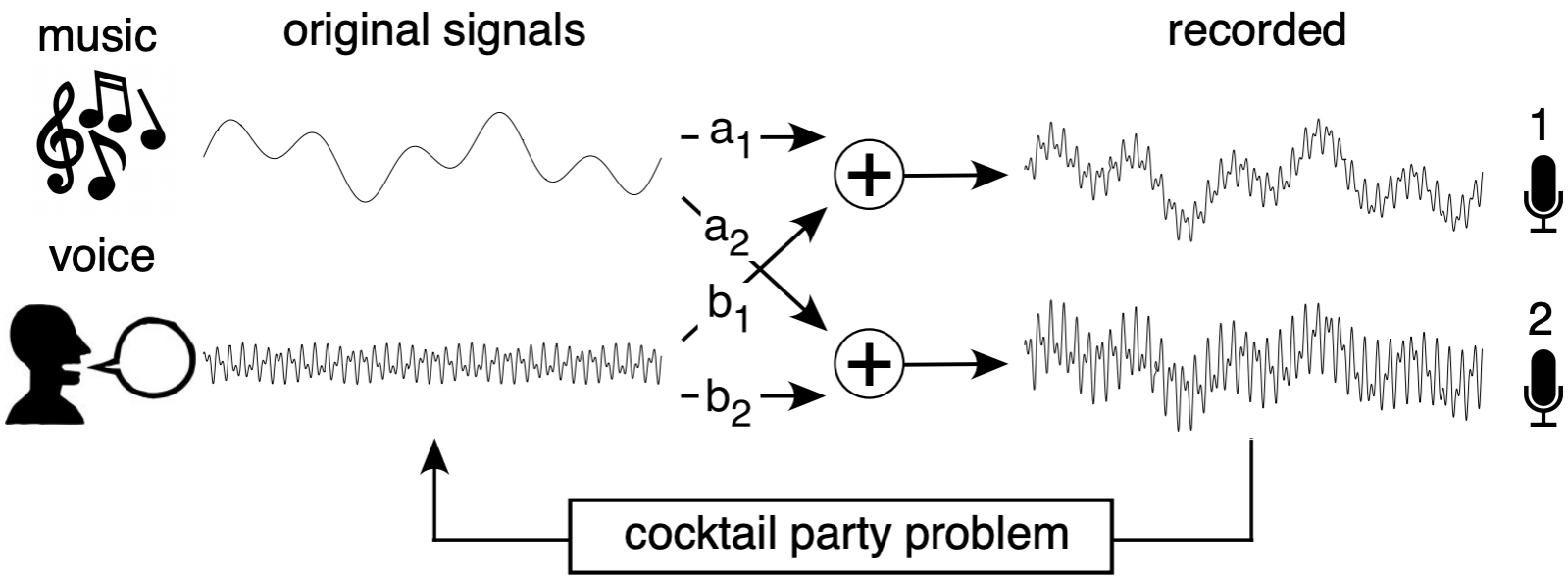
$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

W notacji macierzowej

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t)$$

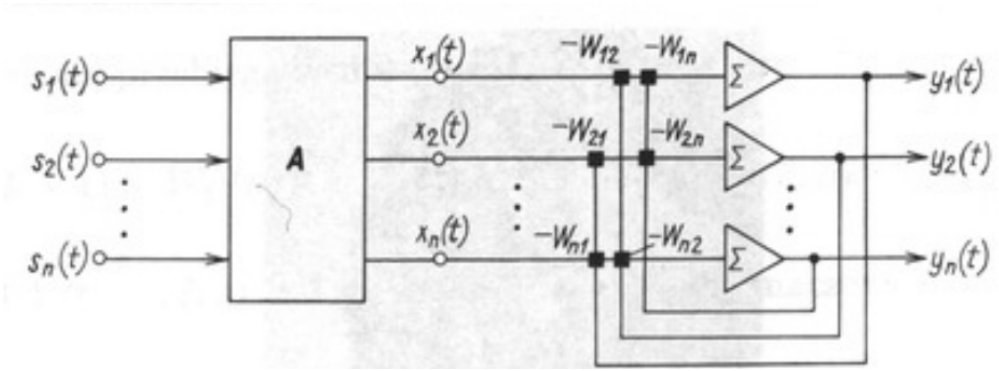
Ślepa separacja źródeł (BSS): zadanie odzyskania oryginalnych komponentów sygnału $\mathbf{s}(t)$ przy nieznanych $\mathbf{A}$ jak i $\mathbf{s}(t)$.
Przykład: *cocktail party problem*

Separacja sygnałów



☞ *J. Shlens, A Tutorial on Independent Component Analysis*

Sieć rekurencyjna Heraulta-Juttena



Działanie realizowane przez sieć rekurencyjną

$$\boldsymbol{x}(t) = \boldsymbol{A}\boldsymbol{s}(t)$$
$$\boldsymbol{y}(t) = \boldsymbol{x}(t) - \mathbf{W}\boldsymbol{y}(t)$$

gdzie macierz wag (w oryginalnej wersji sieci)

$$\boldsymbol{W} = \begin{bmatrix} 0 & w_{12} & \cdots & w_{1n} \\ w_{21} & 0 & \cdots & w_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ w_{n1} & w_{n2} & \cdots & 0 \end{bmatrix}$$

Szukamy rozwiązania

$$\mathbf{y}(t) = (\mathbf{1} + \mathbf{W})^{-1}\mathbf{x}(t)$$

które odtwarza sygnał pierwotny $\mathbf{s}(t)$.
Warunki:

- odtwarzanie z dokładnością do skali

  $$\boldsymbol{y}(t) = \boldsymbol{D}\boldsymbol{s}(t)$$

  gdzie $\mathbf{D}$ jest macierzą diagonalną
- odtwarzanie z dokładnością do kolejności sygnałów

  $$\mathbf{y}(t) = \mathbf{P}\mathbf{s}(t)$$

  gdzie $\mathbf{P}$ jest macierzą permutacji

Dla małej liczby kanałów ($n = 2$) istnieje rozwiązanie analityczne.
Dla większej liczby kanałów stosuje się metodę adaptacyjną.

Reguła Heraulta-Juttena

Przy założeniu statystycznej niezależności sygnałów można wyprowadzić regułę uczenia tożsamą z regułą Hebba

$$\frac{dW_{ij}}{dt} = \eta y_i(t) y_j(t)$$

Zbieżność procesu uczącego wymaga aby w stanie ustalonym $E[\frac{dW_{ij}}{dt}] = 0$ stąd sygnały $y_i(t)$ i $y_j(t)$ nie mogą być skorelowane $E[(y_i - E[y_i])(y_j - E[y_j])] = 0$

Reguła Heraulta-Juttena

$$\frac{dW_{ij}}{dt} = \eta(t) f\left(y_i(t)\right) g\left(y_j(t)\right)$$

$f()$ i $g()$ to funkcje nieparzyste, które przy założeniu statystycznej niezależności sygnałów zapewniają $E[f\left(y_i(t)\right) g\left(y_j(t)\right)] = 0$, co jest warunkiem zbieżności algorytmu uczenia.

Przykłady: $f(x) = x^3, x^5$, $g(x) = \tanh(x), \arctan(x), x, \mathrm{sgn}(x)$

Zmodyfikowany algorytm Cichockiego

Reguła Heraulta-Juttena skuteczna tylko dla sygnałów $s_i(t)$ o niewielkim zróżnicowaniu amplitudy (dla względnej różnicy amplitud $< 10^2$). Sygnały o małej amplitudzie nie są separowane.

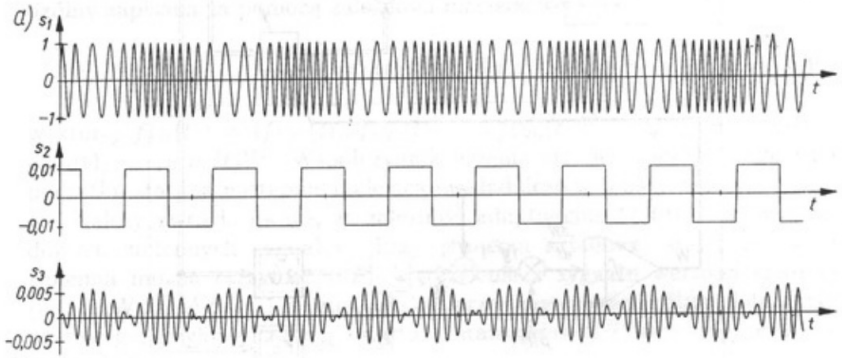
Modyfikacja Cichockiego

$$\frac{d\mathbf{W}}{dt} = \eta(t)\left[\boldsymbol{f}(\boldsymbol{y}(t))[\boldsymbol{g}(\boldsymbol{y}(t))]^T - \mathbf{1}\right]$$

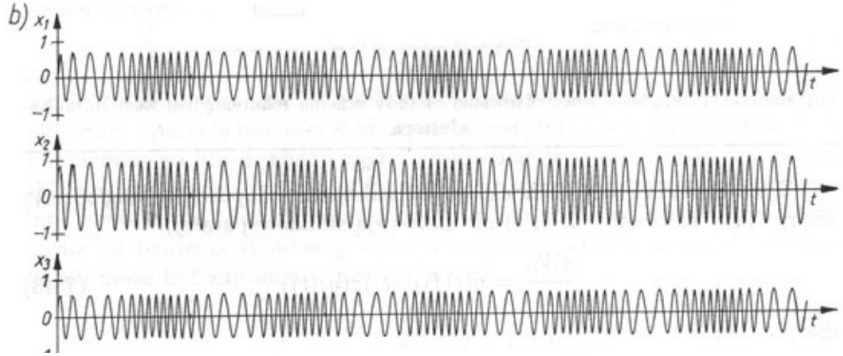
gdzie $\boldsymbol{y}(t) = (\mathbf{1} + \boldsymbol{W})^{-1}\boldsymbol{x}(t)$

- wprowadza sprzężenia zwrotne własne neuronu $w_{ii} \neq 0$ powodujące samonormalizację sygnałów wyjściowych $y_i(t)$
- w stanie ustalonym $E[\frac{d\mathbf{W}}{dt}] = 0$ stąd $E[f(y_i)g(y_j)] = \delta_{ij}$, co jest warunkiem zbieżności algorytmu uczenia
- separacja możliwa nawet dla sygnałów o różnicy amplitud rzędu 10^{10}

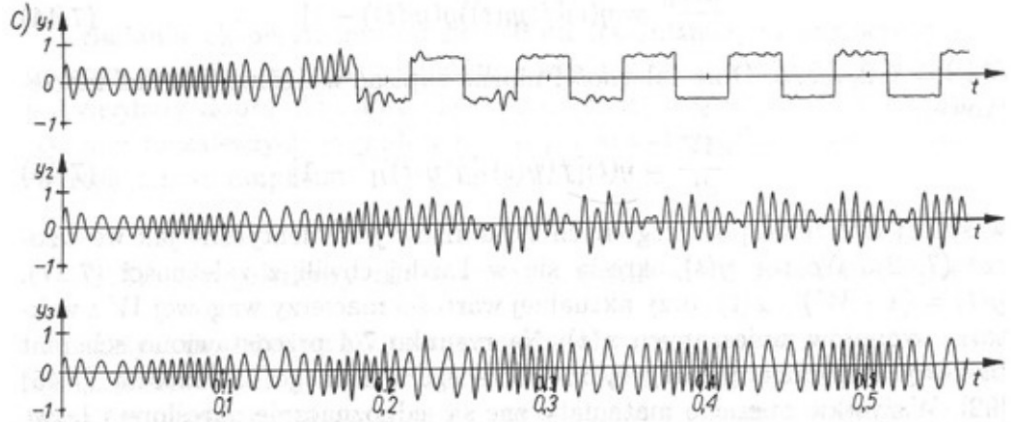
Sygnały oryginalne $s_i(t)$



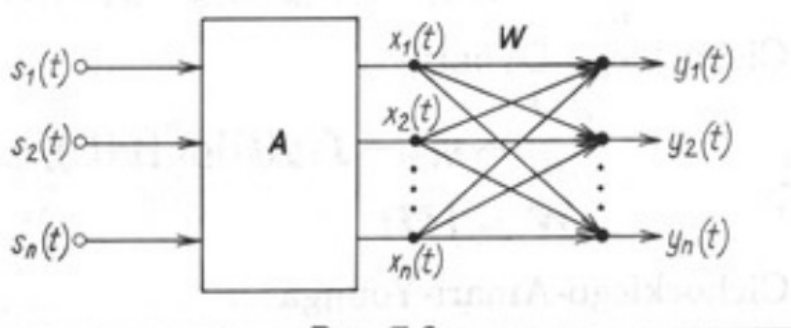
Sygnały zmieszane $x_i(t)$



Proces wydzielania sygnałów $y_i(t)$ odtwarzających sygnały źródłowe $s_i(t)$



Sieć jednokierunkowa



$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

macierz $\mathbf{W} \in \mathbb{R}^{n \times n}$ jest macierzą pełną.

Sieć jednokierunkowa jest równoważna sieci ze sprzężeniami zwrotnymi gdy $\mathbf{W} = (\hat{\mathbf{W}} + \mathbf{1})^{-1}$ w której $\hat{\mathbf{W}}$ to macierz wag sieci rekurencyjnej

Zmodyfikowana reguła adaptacyjna Cichockiego

$$\frac{d\mathbf{W}}{dt} = \eta(t)\mathbf{W}\left[\mathbf{1} - \boldsymbol{f}(\mathbf{y}(t))[\boldsymbol{g}(\mathbf{y}(t))]^T\right]\boldsymbol{W}$$

Sieci dynamiczne

Pamięć asocjacyjna

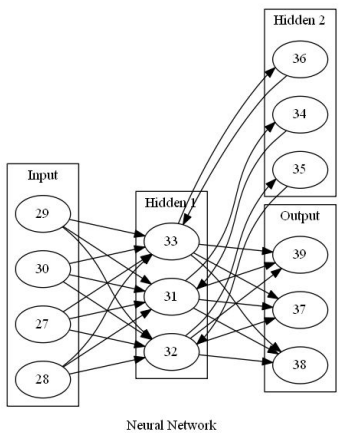
Plan

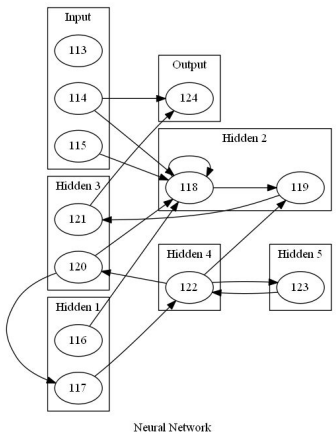
- **Sieci dynamiczne:** sieci ze sprzężeniami zwrotnymi (rekurencyjne), stan sieci w chwili t zależy od stanu w chwili poprzedniej $t-1$
- Modele pamięci asocjacyjnej (autoasocjacyjnej i heteroasocjacyjnej)
- Model Hopfielda, sieć Hamminga, BAM

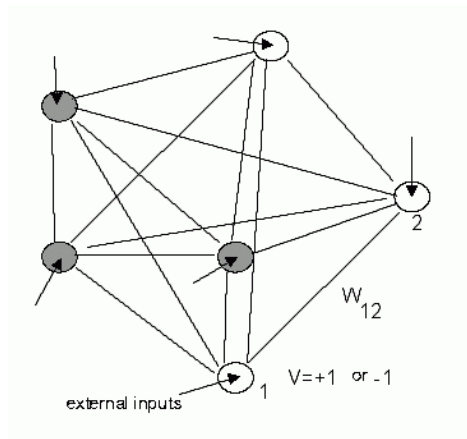
Sieci ze sprzężeniami zwrotnymi

Sieci rekurencyjne

- połączenia tworzą cykle w grafie połączeń
- skomplikowana dynamika - trudna do analizy
- zbliżone do sieci biologicznych, w układach biologicznych neurony mają silne sprzężenia zwrotne







Dynamika sieci: zmiana stanu jednego neuronu przenosi się za pomocą sprzężeń zwrotnych na całą sieć, tworząc nowe stany pośrednie

Sieci ze sprzężeniami zwrotnymi

Najprostsze modele sieci z rekurencją:

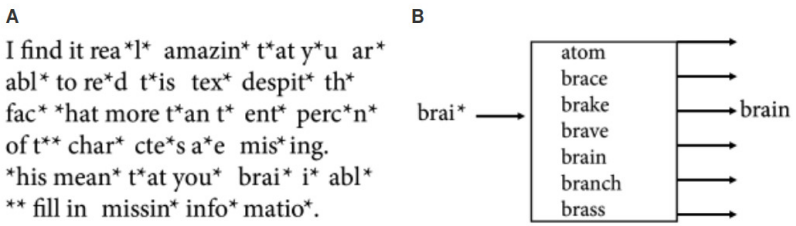
- sieci Hopfielda,
- sieć Hamminga.
- BAM
- Restricted Boltzman Machines (RBM)

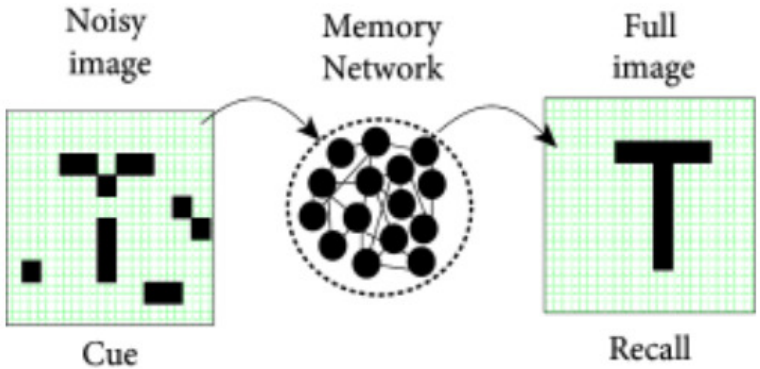
Modele złożone:

- sieć Elmana i inne o uproszczonej strukturze rekurencji
- RTRN - Real Time Recurrent Network, przetwarzająca sygnały w czasie rzeczywistym
- Głębokie sieci rekurencyjne (RNN), LSTM, Long Short Term Memory

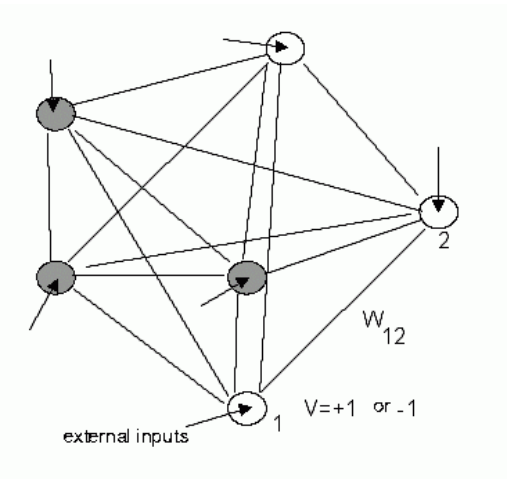
Model pamięci autoasocjacyjnej

Pamięć skojarzeniowa (asocjacyjna) - odtwarza zapamiętane wzorce zbliżone do sygnału wejściowego. Pamięć adresowana kontekstowo (CAM)





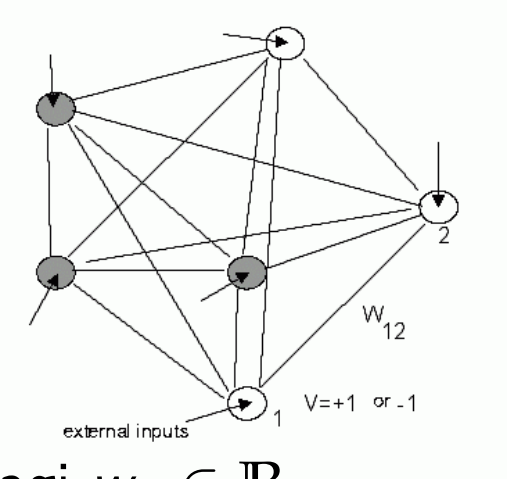
Model pamięci autoasocjacyjnej Hopfielda:



- realizuje pamięć skojarzeniową z binarnymi neuronami progowymi (dyskretne stany neuronów $y_i = \pm 1$)
- wszystkie neurony są ze sobą wzajemnie połączone, brak warstw, każdy neuron jest wejściem i wyjściem
- model stosowany do zrozumienia działania ludzkiej pamięci

Model Hopfielda (1982, 1984)

$$y_i(t+1) = \text{sgn}\left(\sum_j w_{ij} y_j(t)\right)$$



- dyskretne stany neuronów $y \in \{-1, +1\}$, wagi $w_{ij} \in \mathbb{R}$
- zmiany stanów następują w dyskretnych jednostkach czasu
- macierz wag połączeń jest symetryczna, $w_{ij} = w_{ji}$
- brak połączeń zwrotnych do tego samego neuronu $w_{ii} = 0$
- dwa tryby pracy:
  - uczenie - zapamiętywanie wzorców
  - odtwarzanie wzorców przy ustalonych wagach

Symetria wag upraszcza analizę sieci i pozwala wprowadzić funkcję energii, jednak jest nierealistyczna z biologicznego punktu widzenia. Bliżej bioligii są realizacje sieci o ciągłych funkcjach aktywacji, w których zmiany aktywacji następują w sposób ciągły.

Dynamika odtwarzania wzorców

- dla ustalonych wag $\mathbf{W}$ w chwili $t = 0$ doprowadzamy sygnał wejściowy $\mathbf{y}(0) = \mathbf{x}$
- w kolejnym kroku t aktualizowane są stany neuronów

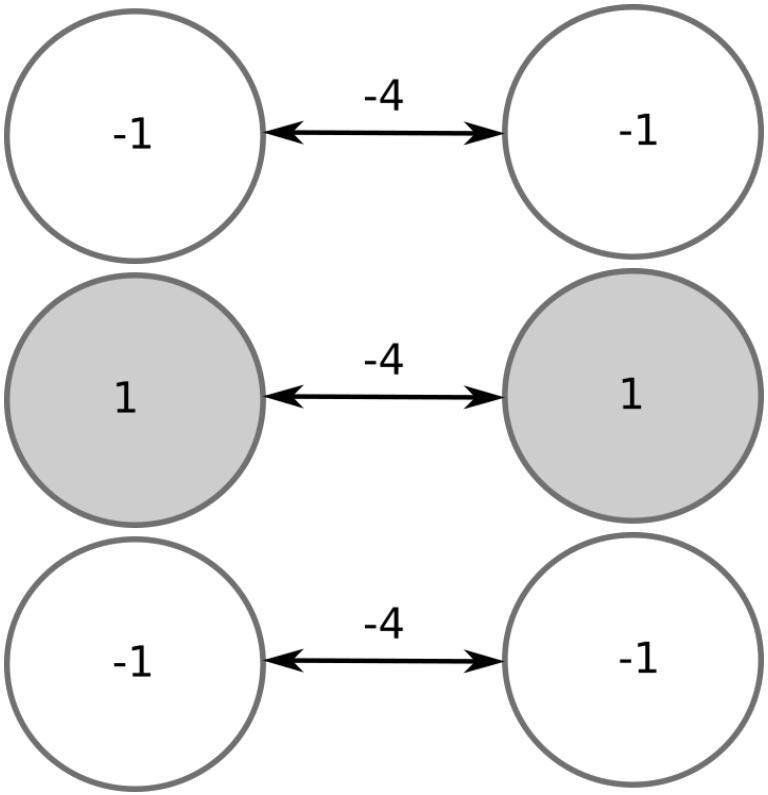
$$y_i(t+1) = \text{sgn}\left(\sum_j w_{ij} y_j(t)\right) \qquad i = 1, \ldots, n$$

- **proces asynchroniczny** - w chwili t aktualizowane jest wyjście pojedynczego (losowo wybranego) neuronu, jego wyjście zależy od stanu pozostałych neuronów z poprzedniego kroku
- **stan stacjonarny** - stan, w którym proces odtwarzania zatrzymuje się, brak zmian wyjść przy pełnym cyklu

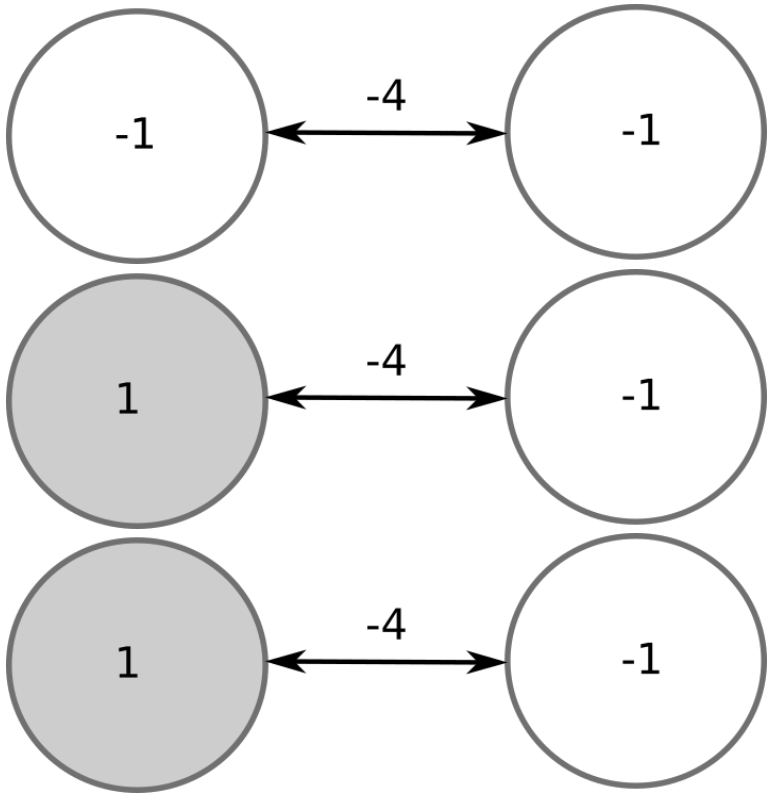
$$\mathbf{y}(t+1) = \mathbf{y}(t)$$

- asynchroniczność zapobiega występowaniu cykli i sieć zbiega do atraktora punktowego
- autoasocjacja - wyjście odtwarza pewien wzorzec wejściowy $\mathbf{x}$

Zmiany asynchroniczne i synchroniczne



Zmiana synchroniczna,
oba neurony zmieniają stan,
nieskończone oscylacje



Zmiana asynchroniczna,
aktualizacja pojedynczego
losowego neuronu,
sieć zbiega do stabilnego wzorca

Źródło:
https://julien-vitay.net/lecturenotes-neurocomputing/4-neurocomputing/2-Hopfield.html

Minimalizacja energii

Dla sieci o symetrycznych wagach taka dynamika prowadzi do minimalizacji **funkcji energii**

$$E = -\sum_{i,j:i<j} w_{ij} y_i y_j - \sum_i b_i y_i$$

gdzie b_i wyraz wolny ($b_i = w_{0i}$ można włączyć do wektora wag)

Zmiana energii w czasie odtwarzania jest ujemna (energia nie rośnie)

$$\Delta E = -\Delta y_i \sum_j w_{ij} y_j = -\Delta y_i I_i$$

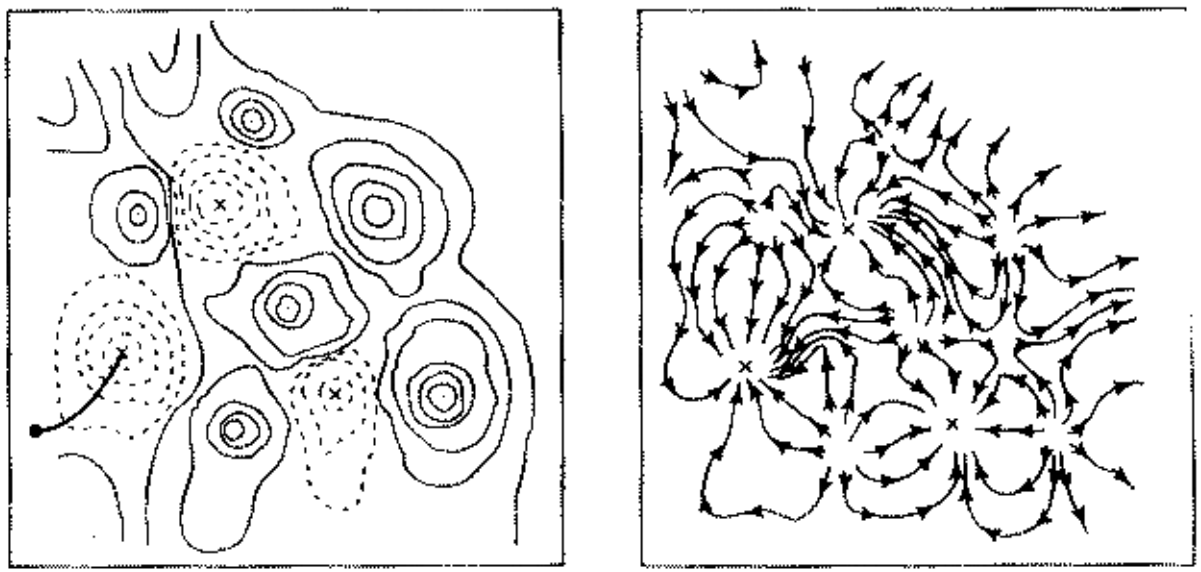
- jeżeli $I_i \geq 0$ to y_i osiąga stan $+1$ i $\Delta y_i \geq 0$
- jeśli $I_i < 0$ to y_i osiąga stan -1 i $\Delta y_i \leq 0$

Każda zmiana stanu neuronu może tylko obniżyć energię sieci

Atraktory

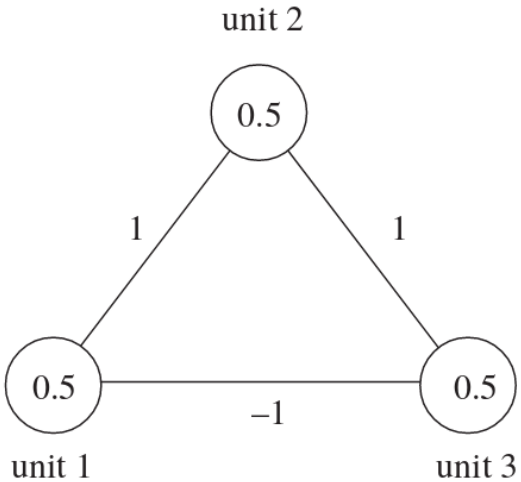
Dynamika: ruch po hiperpowierzchni energii, zależnej od potencjałów neuronów, aż do osiągnięcia lokalnego minimum energii

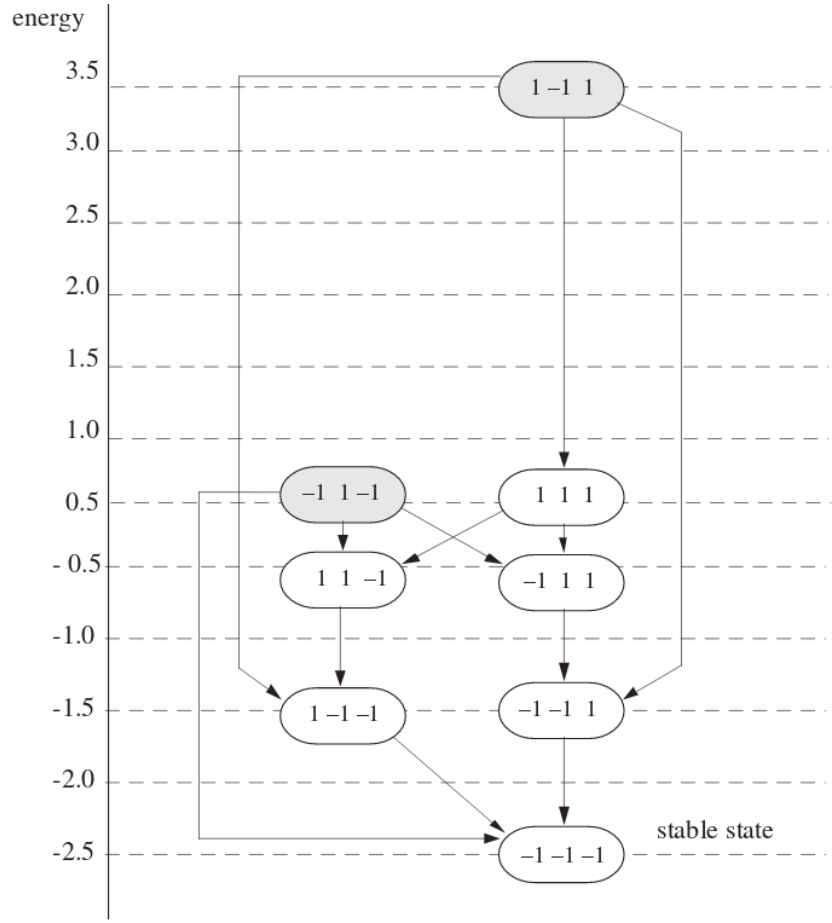
Portret fazowy układu dynamicznego sieci rekurencyjnej o połączeniach symetrycznych: linie ekwipotencjalne i kierunki zmian potencjału w trakcie uczenia



Jeśli y_i dyskretne $\{-1, +1\}$ to ruch odbywa się po wierzchotkach hiperkostki

Przykład: 3 neurony

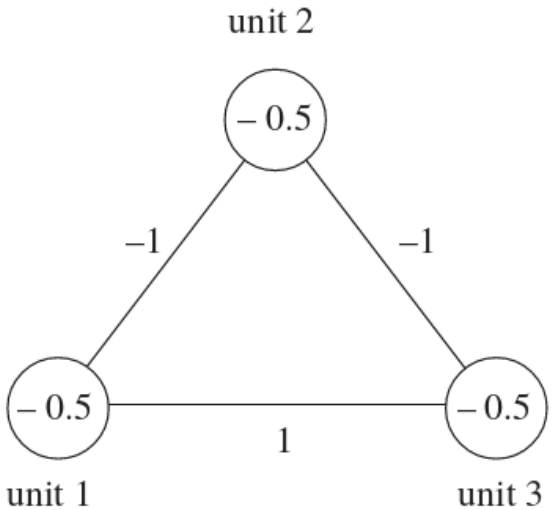


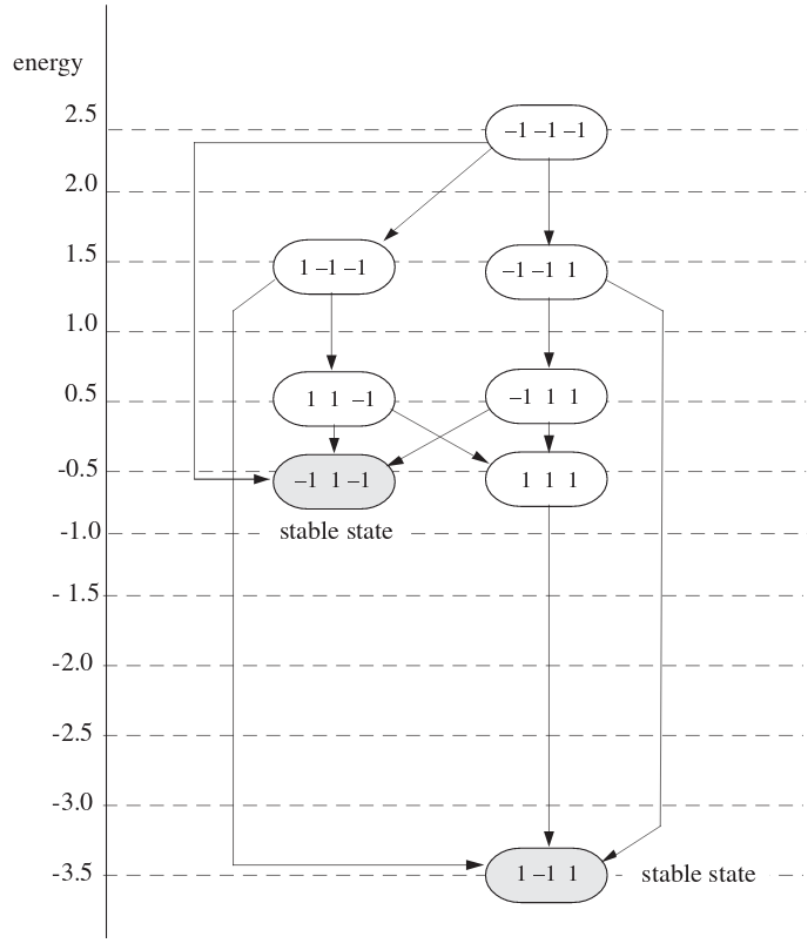


$$E = -y_1y_2 - y_2y_3 + y_1y_3 - 0.5y_1 - 0.5y_2 - 0.5y_3$$

Rys: Rojas, Neuroal Netowrks

Przykład: 3 neurony





$$E = y_1y_2 + y_2y_3 - y_1y_3 + 0.5y_1 + 0.5y_2 - 0.5y_3$$

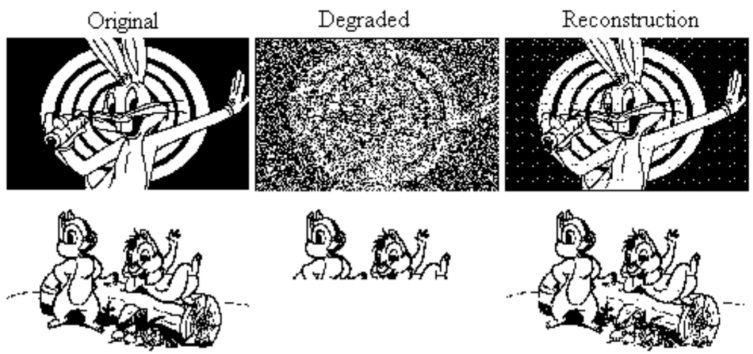
Rys: Rojas, Neuroal Netowrks

Zastosowanie praktyczne

Pamięć asocjacyjna

- **odtwarzanie wzorców**, przy prezentacji wektora wejściowego $\mathbf{x}_i$ odpowiedzią sieci będzie jeden z zapamiętanych wzorców $\mathbf{x}_1, \ldots, \mathbf{x}_p$
- **korekcja**, odtwarzany jest wzorzec najbardziej „podobny” do sygnału wejściowego.
- **uzupełnianie wzorca**: prezentowany jest fragment wzorca (ciąg pierwszych bitów) a siec odtwarza pozostałe (stany początkowych bitów się nie zmieniają)

Odtwarzanie i rekonstrukcja obrazów



Źródło:
https://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2015/slides/lec14.hopfield.pdf

Zastosowanie praktyczne

Rozwiązywanie problemów optymalizacyjnych - odpowiednia konstrukcja sieci i funkcji energii pozwala rozwiązywać problemy optymalizacyjne, także NP-trudne

- problem komiwojażera,
- gospodarowanie zasobami,
- optymalizacja połączeń w centralach telefonicznych,
- rozmieszczanie układów scalonych,
- tworzenie rozkładów jazdy,
- rozwiązywanie problemów programowania liniowego i nieliniowego

W fizyce: modelowanie szkła spinowego

W neuronaukach: modele hipokampa, pamięć epizodyczna

Pamięć asocjacyjna

- pamięć **autoasocjacyjna** - skojarzenie tego samego obiektu (model Hopfielda)
- pamięć **heteroasoscjacyjna** - skojarzone są dwa różne obiekty (dla **a** sieć odpowiada **b**), np. BAM, sieć Hamminga

Pamięć asocjacyjna vs. RAM

- **adresowanie kontekstowe** - odtworzony wzorzec uzyskiwany jest na podstawie odległości (podobieństwa) od zapamiętanego wzorca
- **pamięć rozproszona** - wszystkie wagi kodują wzorce, brak wyraźnie określonego miejsca przechowywania wzorca
- **odporność na uszkodzenia** - lokalne uszkodzenie wzorca lub pamięci nie blokuje możliwości odczytu

Odległość Hamminga

Pamięć asocjacyjna odtwarza zapamiętane wzorce najbliższe względem wektora wejściowego (w zasięgu basenu atrakcji).
Dla stanów binarnych miarą odległości jest odległość Hamminga

Odległość Hamminga - liczba różniących się bitów

- dla wielkości binarnych $x_i, y_i = \{0, 1\}$

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \left[x_i \left(1 - y_i\right) + \left(1 - x_i\right) y_i \right]$$

- dla wartości bipolarnych $x_i, y_i = \{+1, -1\}$

$$(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \left(n - \sum_{j=1}^{n} x_i y_j \right)$$

Uczenie pamięci asocjacyjnej

Dla pojedynczego wzorca **x** sieć powinna odtwarzać sygnał wejściowy

$$x_i = \text{sgn}\left(\sum_{j=1}^{n} w_{ij} x_j\right)$$

wystarczy zażądać by:

$$w_{ij} \sim x_i x_j; \qquad \text{np. } w_{ij} = \frac{1}{n} x_i x_j \qquad \text{reguła Hebba}$$

$$\frac{1}{n}\left(\sum_{j=1}^{n} x_i x_j x_j\right) = \frac{1}{n}\left(\sum_{j=1}^{n} x_j^2\right) x_i = x_i$$

Uczenie pamięci asocjacyjnej regułą Hebba

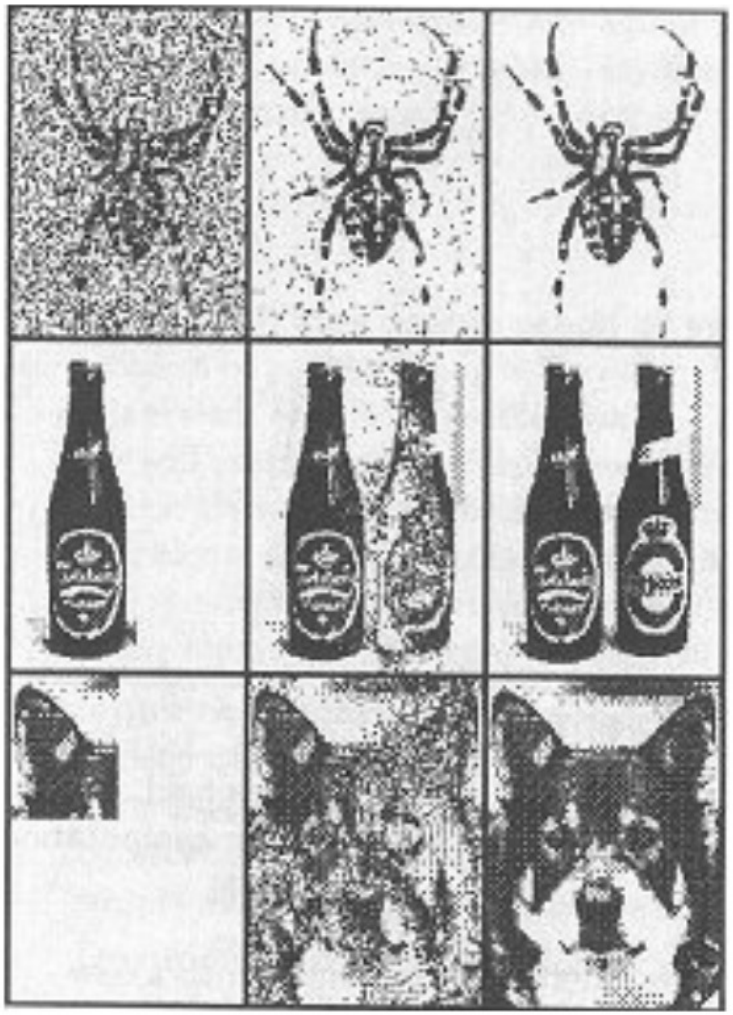
Dla wielu wzorców $\mathbf{x}^{(i)}, i = 1, \ldots, p$ korzystamy z reguły Hebba uśredniając:

$$w_{ij} = \frac{1}{n} \sum_{k=1}^{p} x_i^{(k)} x_j^{(k)}$$

W praktyce stosuje się często metodę iteracyjną, gdzie adaptacja przy prezentacji k-tego wektora $\mathbf{x}^{(k)}$ przybiera postać

$$\begin{aligned} \Delta w_{ij} &= x_i^{(k)} x_j^{(k)} \quad \text{dla} \quad i, j = 1, \ldots, n \\ \Delta b_i &= x_i^{(k)} \quad \text{dla} \quad i = 1, \ldots, n \end{aligned}$$

Przykłady działania



odtwarzanie niekompletnych lub zaszumionych danych

Stabilność pamięci

Stabilność działania wymaga

$$\operatorname{sgn}\left(\sum_{j=0}^{N} w_{ij} x_j^{(l)}\right) = \operatorname{sgn}\left(\frac{1}{N}\sum_{j=0}^{N}\sum_{k=1}^{p} x_i^{(k)} x_j^{(k)} x_j^{(l)}\right) = x_i^{(l)}$$

co można zapisać

$$\operatorname{sgn}\left(x_i^{(l)} + \frac{1}{N}\sum_{j=0}^{N}\sum_{k\neq l} x_i^{(k)} x_j^{(k)} x_j^{(l)}\right) = \operatorname{sgn}\left(x_i^{(l)} + C\right)$$

gdzie C nazywamy **przesłuchem**

Stabilność osiągamy gdy składnik przesłuchu C jest na tyle mały aby nie zmienić znaku $x_i^{(l)}$, tzn. $|C| < |x_i^{(l)}|$

Pojemność modelu

- Pojemność p_{max} określa maksymalną liczbę odtwarzanych wzorców przy określonym poziomie błędu
- 2^N możliwych stanów sieci binarnej złożonej z N neuronów
- przy zapamiętywaniu regułą Hebba liczba poprawnie pamiętanych losowych wzorców dla sieci z n neuronami przy prawdopodobieństwie błędu 0.37% wynosi

$$p_{max} = 0.138 \cdot n$$

- na ogół (przy założeniu $n/2p > \log n$) pojemność spełnia

$$p_{max} = \frac{n}{2 \log n}$$

- mała efektywność, potrzeba n^2 wag o wartościach rzeczywistych

Stany fałszywe

W sieci mogą pojawiać się stany fałszywe lub przekłamania pamięci

- funkcja energii jest symetryczna względem polaryzacji (moga występować negatywy stanów o tej samej energii)
- występuje mieszanie różnych składowych zapamiętanych wzorców i tworzenie stabilnego zmieszanego stanu
- przy przepełnieniu pamięci powstają pośrednie minima lokalne nie odpowiadające żadnemu zapamiętanemu wzorcowi (fałszywa pamięć)
- zbyt wiele wzorców $\Rightarrow$ chaos, zapominanie, zwiększając liczbę wzorców rośnie prawdopodobieństwo przekłamań

Uczenie pamięci metodą rzutowania

Zakładamy, że każdy wzorzec $\mathbf{x}$ podany na wejście generuje natychmiastowo na wyjściu sieci stan ustalony $\mathbf{x}$

$$\mathbf{W} \cdot \mathbf{X} = \mathbf{X} \quad \Rightarrow \quad \mathbf{W} = \mathbf{X} \cdot \mathbf{X}^{+}$$

gdzie $\mathbf{X} = [\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(p)}]$
Dla $\mathbf{x}^{(i)}$ liniowo niezależnych możemy zastąpić psudoinwersję macierzy $n \times p$ inwersją macierzy $p \times p$

$$\mathbf{W} = \mathbf{X} \cdot \mathbf{X}^{+} = \mathbf{X} \left(\mathbf{X}^{T} \cdot \mathbf{X}\right)^{-1} \mathbf{X}^{T}$$

Możliwe też wyznaczenie $\mathbf{W}$ w sposób iteracyjny podczas jednokrotnej prezentacji wzorców uczących $i = 1, \ldots, p$ bez konieczności wyznaczania macierzy odwrotnej

$$\mathbf{W}^{(i)} = \mathbf{W}^{(i-1)} + \frac{\left(\mathbf{W}^{(i-1)}\mathbf{x}^{(i)} - \mathbf{x}^{(i)}\right)\left(\mathbf{W}^{(i-1)}\mathbf{x}^{(i)} - \mathbf{x}^{(i)}\right)^{T}}{\left(\mathbf{x}^{(i)}\right)^{T}\mathbf{x}^{(i)} - \left(\mathbf{x}^{(i)}\right)^{T}\mathbf{W}^{(i-1)}\mathbf{x}^{(i)}}$$

gdzie stan początkowy $\mathbf{W}^{(0)} = \mathbf{0}$

Metoda rzutowania Δ

Gradientowa odmiana metody rzutowania

$$\mathbf{W} \leftarrow \mathbf{W} + \frac{\eta}{n}\left[\mathbf{x}^{(i)} - \mathbf{W}\mathbf{x}^{(i)}\right]\left(\mathbf{x}^{(i)}\right)^T$$

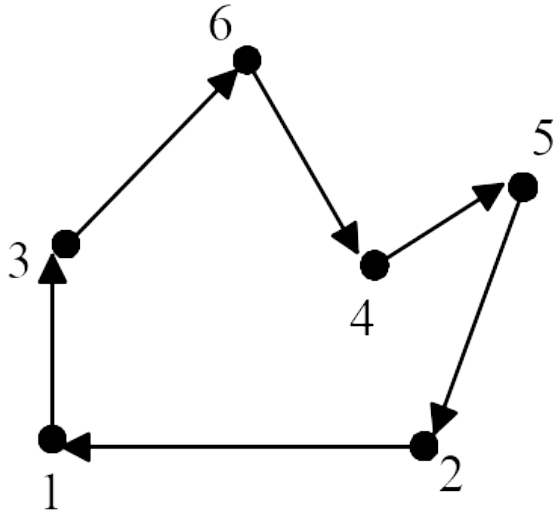
- η stała uczenia zazwyczaj z przedziału $[0.7 - 0.9]$
- wymaga wielokrotnej prezentacji wzorców aż do uzyskania zbieżności, np. gdy zmiana wartości wag nie przekracza pewnego progu tolerancji ϵ
- metoda rzutowania i rzutowania Δ zapewniają większą pojemność w stosunku do uczenia reguła Hebba, maksymalna liczba zapamiętanych wzorców $n - 1$ (dla wzorców niezależnych liniowo)

Zastosowanie w problemach optymaliacyjnych

- jeżeli optymalizowana funkcja kosztu (z warunkami) może zostać zapisana w postaci funkcji energii Hopfielda to istnieje sieć Hopfielda, której stany stabilne odpowiadają rozwiązaniu problemu optymalizacji
- dynamika odtwarzania minimalizuje energię powodując zmniejszenie wartości funkcji kosztu przy zachowaniu warunków optymalizacji
- warunki optymalizacji są „zaszyte" w wagach sieci
- problemy NP-trudne - np. problem komiwojażera

Optymalizacja - problem komiwojażera

Problem komiwojażera: znaleźć najkrótszą droga pomiędzy N miastami.



		Kolejność 1 2 3 4 5 6
M	1	1 0 0 0 0 0
i	2	0 0 0 0 0 1
a	3	0 1 0 0 0 0
s	4	0 0 0 1 0 0
t	5	0 0 0 0 1 0
o	6	0 0 1 0 0 0

Problem NP-trudny, $\frac{N!}{2N}$ wszystkich możliwości

- sieć zawierająca N^2 neuronów unipolarnych ciągłych
- stan neuronu $y_{i\alpha} = 1$ oznacza, że miasto i zostało odwiedzone w kolejności α
- numer miasta $i, j = 1, 2, \ldots, N$,
  kolejność odwiedzin $\alpha, \beta = 1, \ldots, N$

Przykład: problem komiwojażera

Ogólna postać funkcji energii

$$E = -\frac{1}{2} \sum_{i \neq k} \sum_{\alpha \neq \beta} w_{i\alpha,k\beta} y_{i\alpha} y_{k\beta}$$

Jak dobrać W?

Dobór wag

Kara za powtórne odwiedzenie miasta

$$E_1 = \frac{A}{2} \sum_i \sum_{\alpha \neq \beta} y_{i\alpha} y_{i\beta}$$

Kara za rozdwojenie komiwojażera (odwiedzenie dwóch miejsc w tym samym czasie)

$$E_2 = \frac{B}{2} \sum_{i \neq k} \sum_{\alpha} y_{i\alpha} y_{k\alpha}$$

Kara za wzbudzenie większej liczby neuronów niż N

$$E_3 = \frac{C}{2} \left(\sum_{i,\alpha} y_{i\alpha} - N \right)^2$$

Czynnik promujący najkrótszą trasę

$$E_4 = \frac{D}{2} \sum_{i \neq k} \sum_{\alpha} d_{ik} y_{i\alpha} \left(y_{k\alpha-1} + y_{k\alpha+1} \right)$$

Dobór wag

Całkowita energia

$$E = E_1 + E_2 + E_3 + E_4$$

stąd wagi sieci

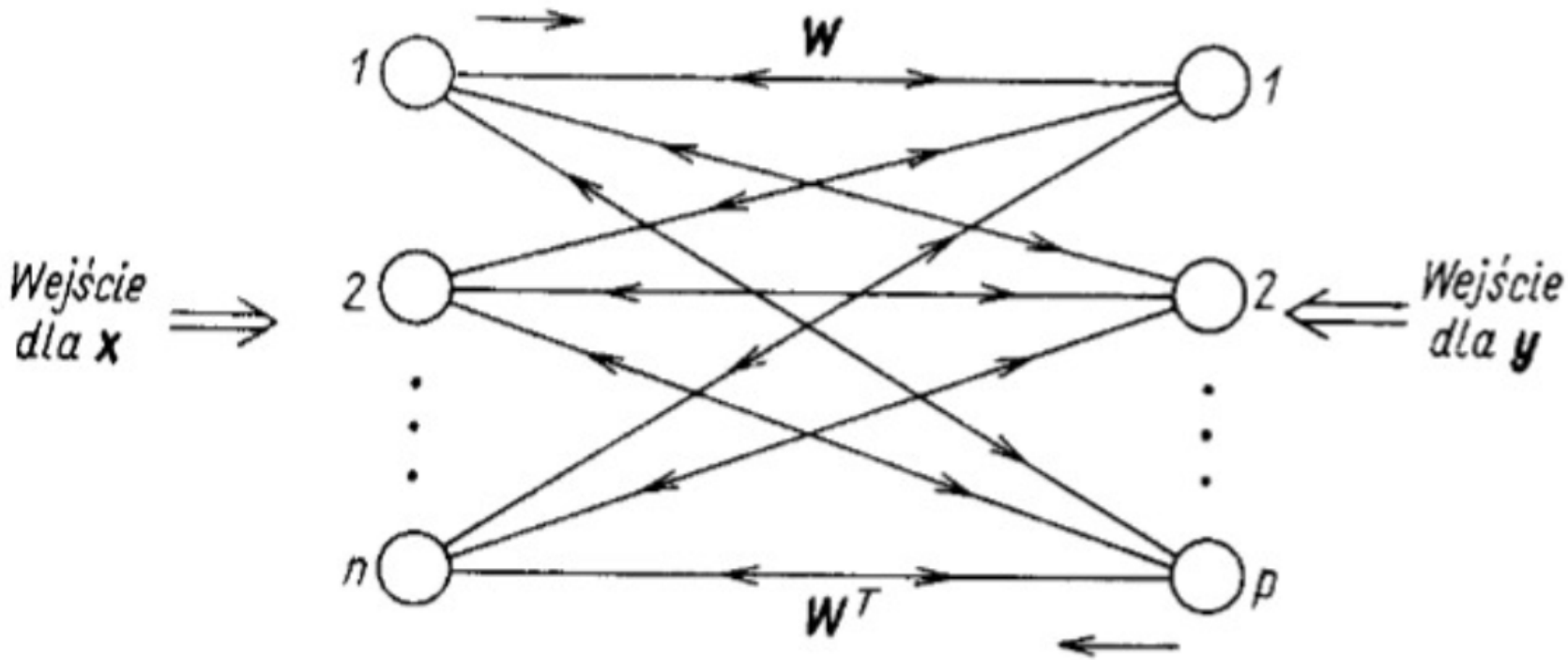
$$\begin{aligned} w_{i\alpha,k\beta} = \quad & -A\left(1-\delta_{\alpha\beta}\right)\delta_{ik} - B\left(1-\delta_{ik}\right)\delta_{\alpha\beta} - C \\ & -Dd_{ik}\left(1-\delta_{ik}\right)\left(\delta_{\alpha-1,\beta}+\delta_{\alpha+1,\beta}\right) \end{aligned}$$

Delta Kroneckera $\delta_{ij} = 1$ dla $i = j$ oraz $\delta_{ij} = 0$ dla $i \neq j$

Wartości A, B, C, D dobierane heurystycznie, np.
$A = B = C = 500$ i $C = 200$

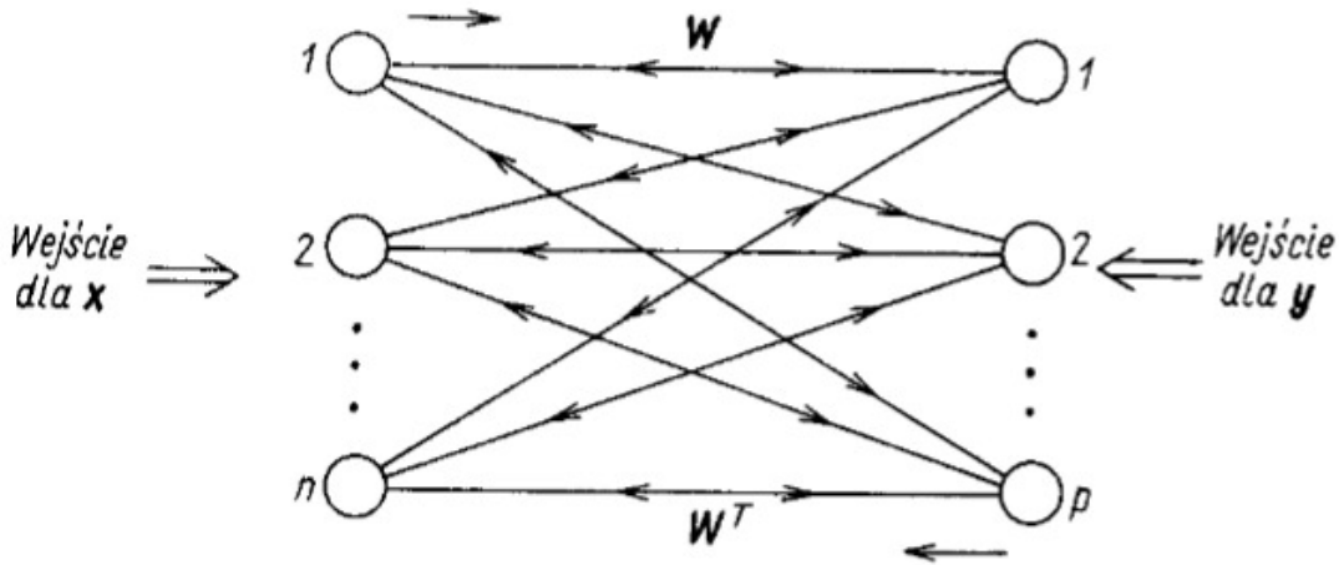
Sieć BAM

Bidirectional Associative Memory (BAM) - uogólnienie sieci Hopfielda na dwuwarstwową sieć rekurencyjną (Kasko)



BAM

- przepływ sygnału odbywa się w dwóch kierunkach
- praca synchroniczna lub asynchroniczna
- progowa funkcja aktywacji $f(x)$ (binarna lub bipolarna)
- $\mathbf{W}$ o wartościach rzeczywistych, niesymetryczna
- zapamiętuje pary wzorców $\mathbf{x}_i, \mathbf{y}_i$
- realizuje **pamięć heteroasocjacyjną**
- pojemność $p < \min(n, m)$, dla n neuronów w warstwie wejściowej i m neuronów w warstwie wyjściowej



Działanie BAM

Trening: ustalenie wag wartościami macierzy korelacji

$$\mathbf{W} = \sum_{i=1}^{p} \mathbf{x}_i^T \mathbf{y}_i$$

Dla wejść bipolarnych $x_i, y_j \in \{-1, +1\}$ otrzymujemy uczenie hebbowskie - sygnały o tym samym znaku mają wagę wzmacniającą, o przeciwnym - hamującą

Odtwarzanie sygnału

$$\begin{aligned} f(\mathbf{x}_0\mathbf{W}) = \mathbf{y}_1 \quad &\rightarrow \quad f(\mathbf{y}_1\mathbf{W}^T) = \mathbf{x}_1 \quad \rightarrow \quad \dots \\ \dots \quad &\rightarrow \quad f(\mathbf{x}_k\mathbf{W}) = \mathbf{y}_k \quad \rightarrow \quad f(\mathbf{y}_k\mathbf{W}^T) = \mathbf{x}_k \end{aligned}$$

stan stacjonarny generuje dwa stabilne wzorce $\mathbf{x}_k$ i $\mathbf{y}_k$

Funkcja energii

$$E_k = -\mathbf{x}_k\mathbf{W}\mathbf{y}_k^T$$

Przykład

asynchroniczny BAM
wzorce zapamiętane:
(S,E) (M,V) (G,N)
wejście $10 \times 14 = 140$
wyjście $9 \times 12 = 108$
wzorzec początkowy (S,E)
z 40% szumem (99
odwrócone bity)

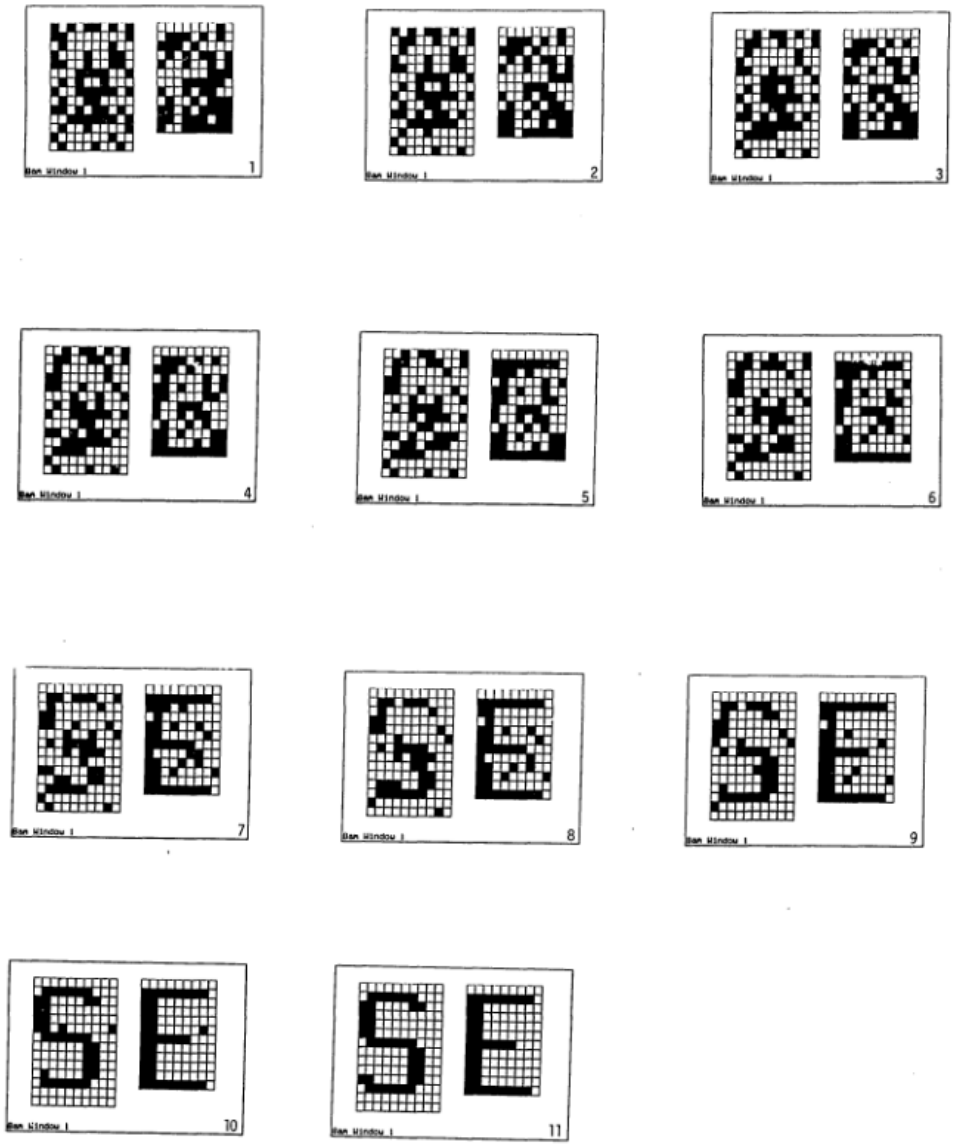


Fig. 1. Asynchronous BAM recall. Approximately six neurons update per snapshot. The associated spatial patterns (S,E), (M,V), and (G,N) are stored. Field F_A contains 140 neurons; F_B,108. Perfect recall of (S,E) is achieved when recall is initiated with a 40% noise-corrupted version of (S,E).

Kosko (1987) Ądaptive bidirectional associative memories"

Głębokie sieci neuronowe

Głębokie sieci neuronowe - Plan

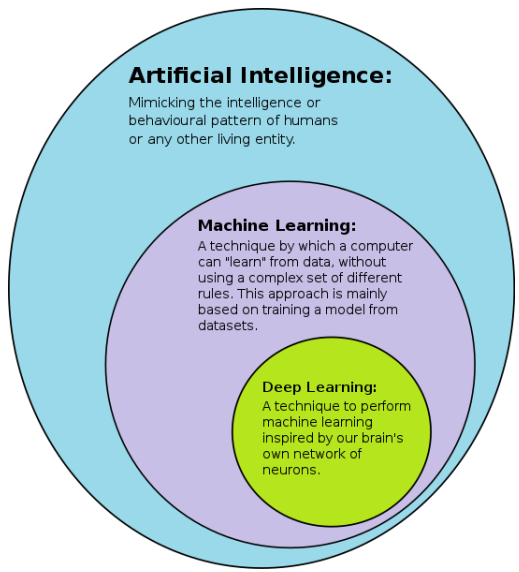
1. Głębokie MLP (DNN)
2. Sieci splotowe (CNN)
3. Sieci rekurencyjne (RNN) i uczenie sekwencji
4. Autoenkodery (AE), Deep belief network (DBN)
5. Generative Adversarial Networks (GAN)
6. Modele z uwagą, transformery i LLMy

Literatura

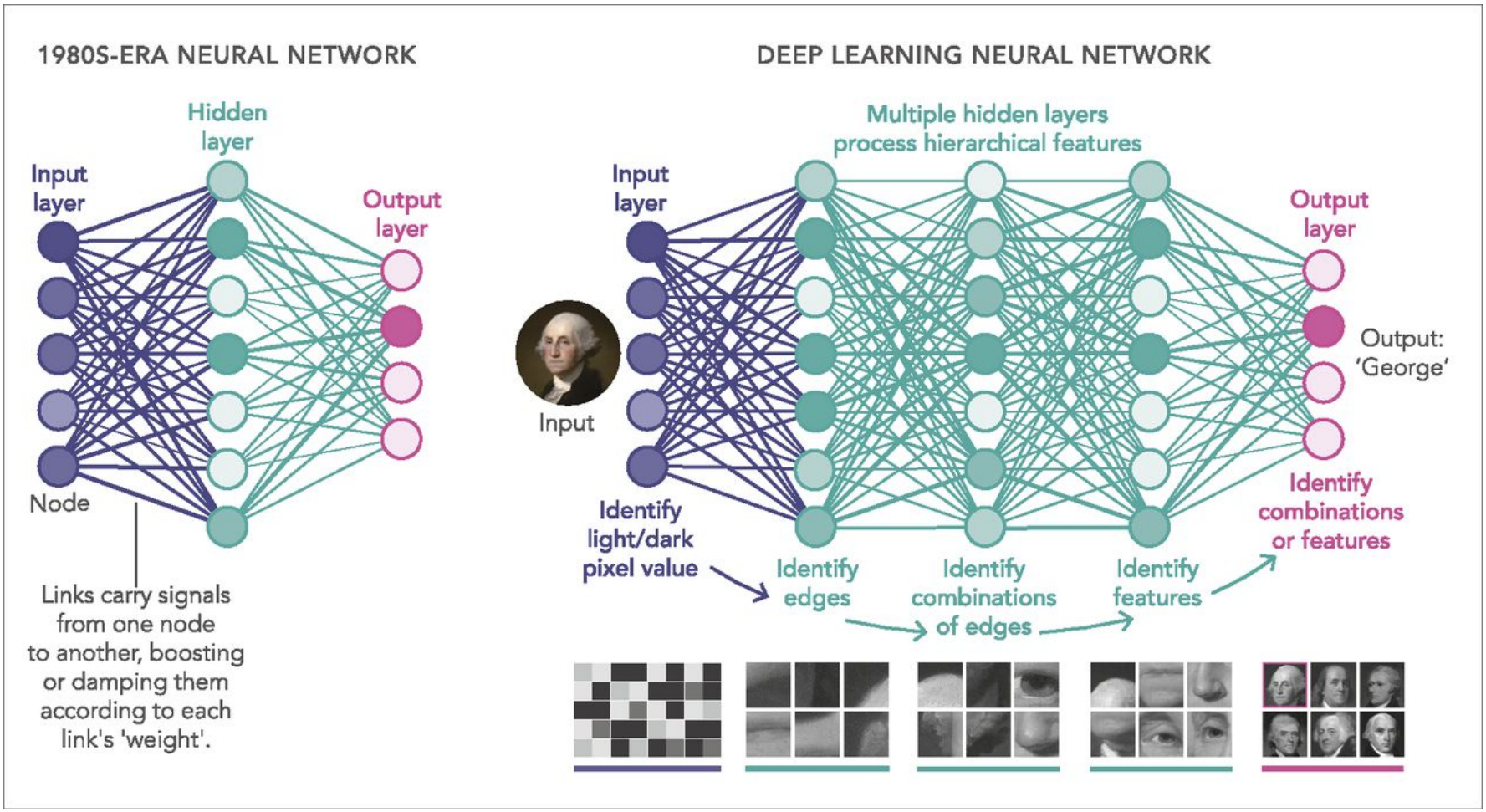
- Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`
- Michael Nielsen, *Neural Networks and Deep Learning*, `http://neuralnetworksanddeeplearning.com/`
- Aston Zhang, Zachary C. Lipton, Mu Li, Alexander J. Smola, *Dive into Deep Learning*, `https://d2l.ai/`
- Julien Vitay, *Neurocomputing*, `https://julien-vitay.net/lecturenotes-neurocomputing/`
- Denny Britz, *Deep Learning Glossary*, `http://www.wildml.com/deep-learning-glossary/`

Głębokie uczenie to zbiór metod uczenia maszynowego

- zbudowane z **wielu warstw realizujących nieliniowe transformacje**
- tworzą **model hierarchiczny**
  warstwy odwzorowują kolejne poziomy abstrakcji, najniższe warstwy tworzą reprezentację najprostszych cech (*surowego*) sygnału wejściowego, wyższe warstwy generują bardziej ogólne koncepty, bazując na relacjach z poprzednich warstw
- automatycznie **uczą się reprezentacji** (wykrywają cechy)
- **Transfer learning** - wykorzystanie wiedzy zdobytej przy rozwiązywaniu innego (podobnego) problemu, np. „przeszczep" wag wybranych warstw sieci



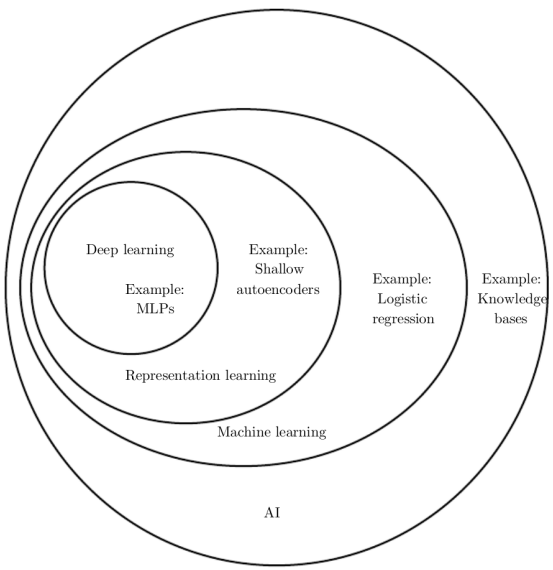
Hierarchia reprezentacji



Rys: ☞ What are the limits of deep learning? by Mitchell Waldrop

Representation learning

Metody uczące się reprezentacji
metody uczenia maszynowego posiadające mechanizmy automatycznego tworzenia takiej reprezentacji danych (ekstrakcji cech), która istotnie poprawia zdolność algorytmów w rozwiązywaniu poszczególnych zadań.



Goodfellow, 2016 [1]

Głębokie vs. płytkie uczenie

Krótka (nieformalna) definicja: głębokie uczenie, gdy model ma więcej niż 2 nieliniowości w najkrótszej ścieżce prowadzącej od wejścia do wyjścia

Płytkie modele

- do 2 warstw (nieliniowych transformacji), np. regresja logistyczna, SVM, MLP z jedną warstwą ukrytą
- wymagają odpowiednio przygotowanych cech

Głębokie modele

- więcej niż 2 warstwy przetwarzania, np. wielowarstwowe MLP, sieci rekurencyjne RNN, sieci splotowe CNN
- model hierarchiczny, wiele transformacji od wejścia do wyjścia
- każda „warstwa" tworzy reprezentację danych na innym poziomie abstrakcji
- możliwość pracy na „surowych" danych $\rightarrow$ **modele end-to-end**

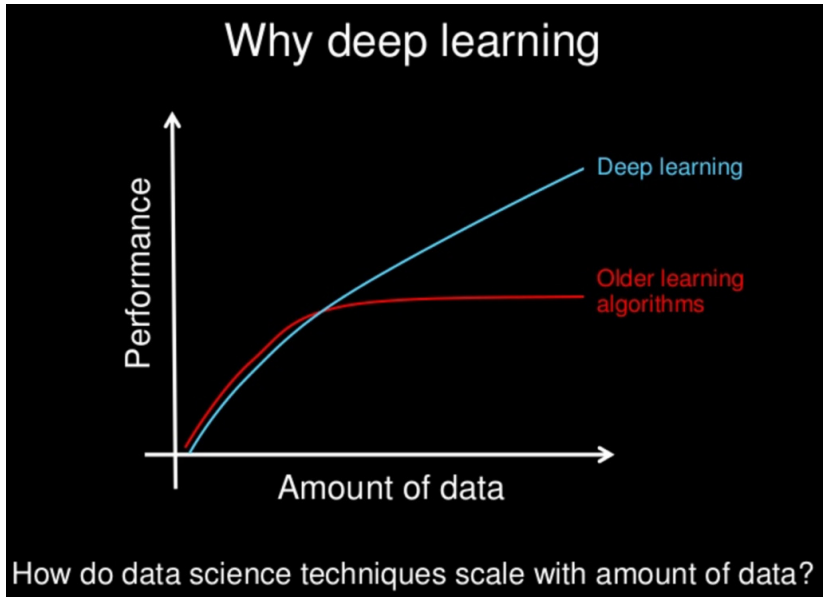
Po co nurkować w głąb?

- **Uniwersalny aproksymator:** sieć MLP z jedną warstwą ukrytą jest w stanie aproksymować dowolną funkcję z dowolną dokładnością
- Głębokie uczenie jest trudniejsze w realizacji od uczenia płytkich modeli; problemy dotyczą nie tylko treningu, ale też interpretacji działania modeli

Po co więc tworzyć głębsze sieci?

- „Płytkie" modele wymagają o wiele (wykładniczo) więcej neuronów do osiągnięcia tej samej dokładności co głębokie modele
- „Płytkie", ale szerokie modele łatwiej się przetrenowują; znalezienie rozwiązania złożonego jest trudniejsze niż w modelach głębszych
- Złożone problemy wymagają złożonych modeli

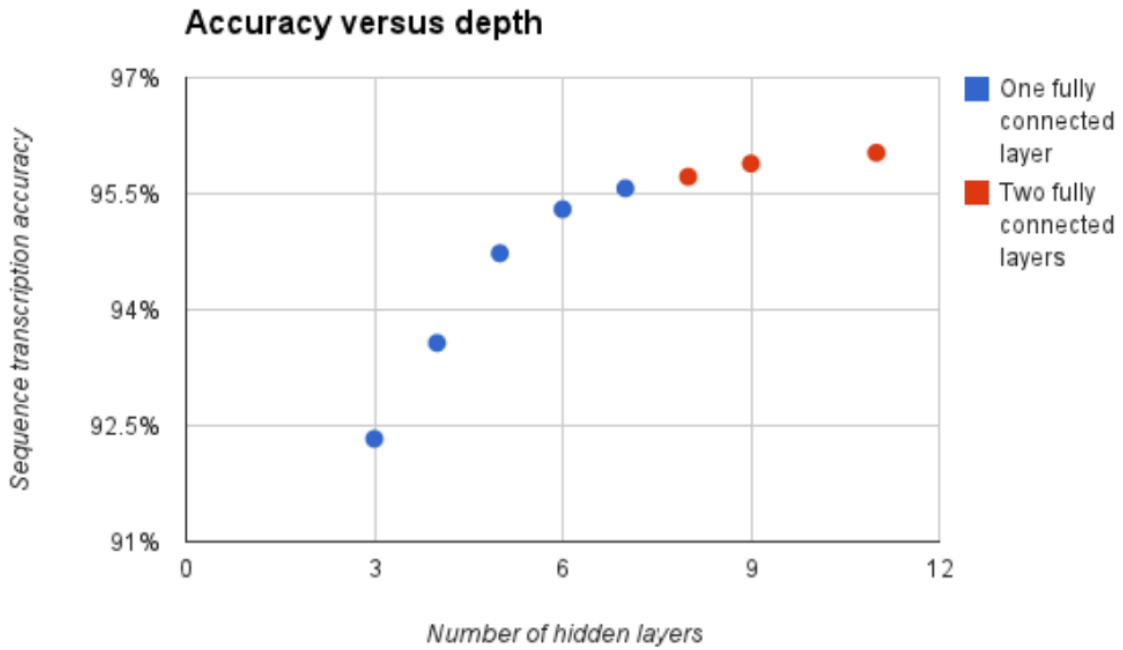
Skalowalność wyników



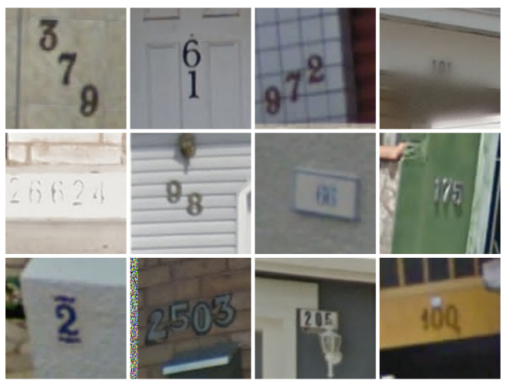
- duża ilość dostępnych danych (z etykietami) - Big Data, sieci głębokie wymagają dużej liczby danych
- duża dostępna moc obliczeniowa (GPU, TPU), głębokie uczenie wiąże się z wykonywaniem wielu operacji mnożenia macierzy
- nowe metody pozwalające skutecznie uczyć głębokie modele

Rys: ☞ Why should you care about deep learning? Andrew Ng

Głębokość a poprawność klasyfikacji



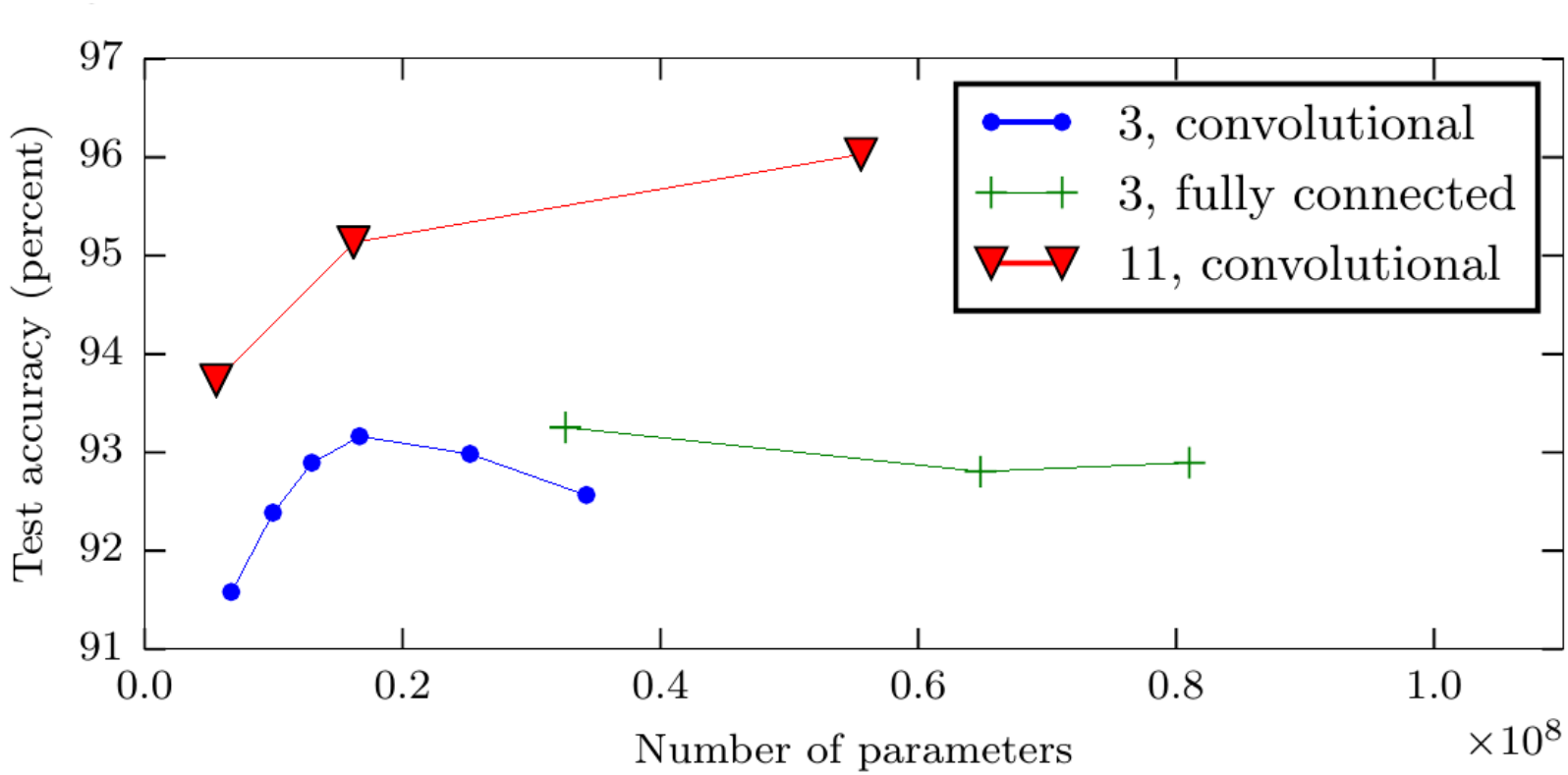
Street View House Numbers dataset (SVHN)
200k numerów domów



Głębokość jest niezbędna do zbudowania odpowiedniej reprezentacji i osiągnięcia lepszej poprawności

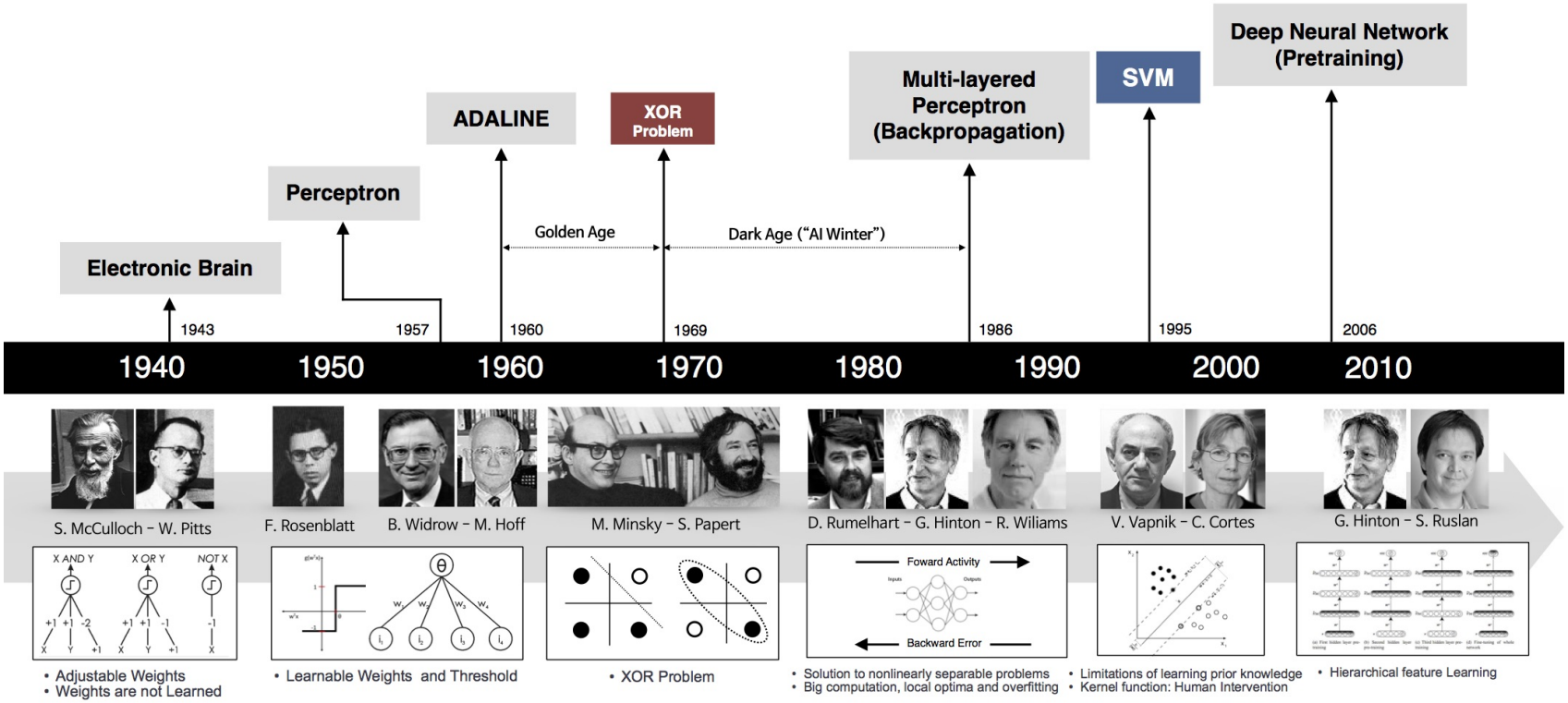
Goodfellow, et al. (2014) Multi-digit Number Recognition from Street ViewImagery using Deep Convolutional Neural Networks

Głębokość ma znaczenie



Zwiększanie liczby parametrów płytkich modeli nie wystarczy do osiągnięcia poprawności modeli głębokich
Głębokość może mieć większe znaczenie od liczby parametrów modelu

Goodfellow, et al. (2014) Multi-digit Number Recognition from Street ViewImagery using Deep Convolutional Neural Networks

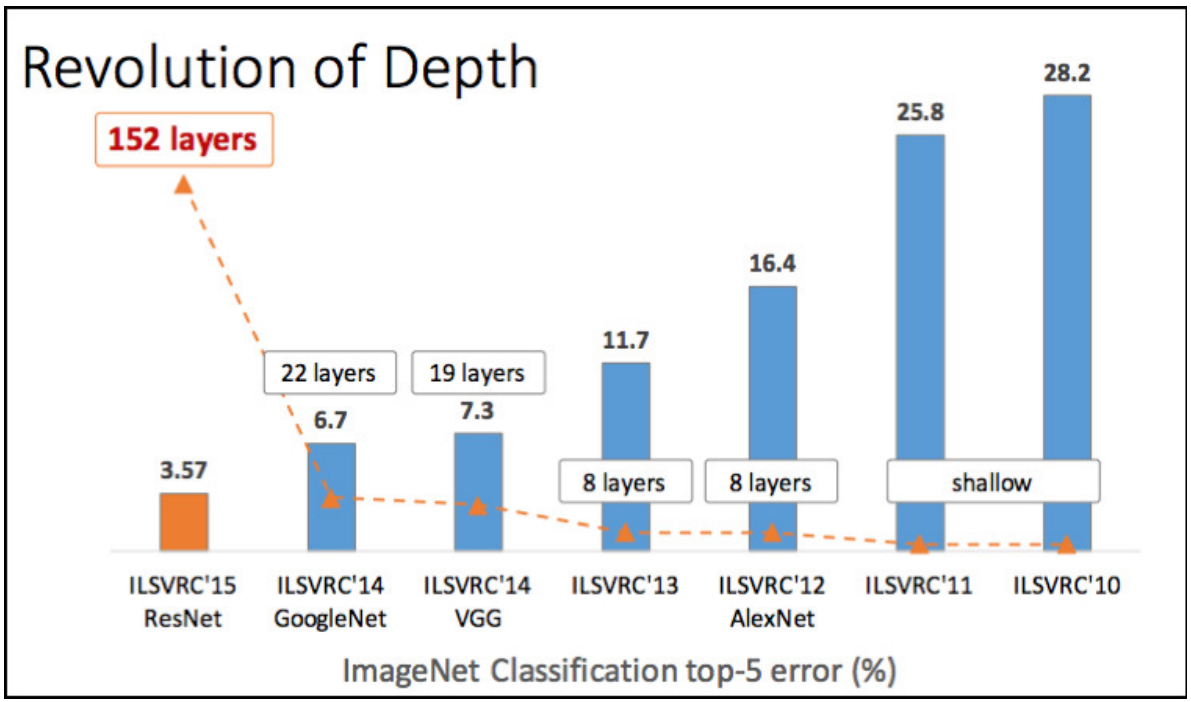


Geoffrey E. Hinton et al., „A fast learning algorithm for deep belief nets", 2006
R. Salakhutdinov, G. Hinton, „Deep Boltzmann Machines", 2009

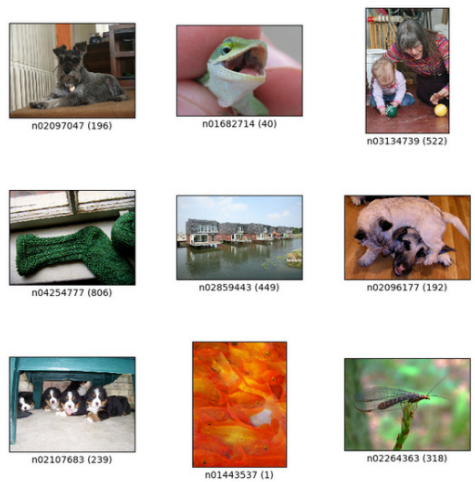
https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

State-of-the-art results

☞ ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



Dane ImageNet 2012: klasyfikacja obrazów, trening 1.26 M zdjęć, 1000 kategorii, poziom człowieka $\approx$ 5% błędów



Rankingi modeli:

- klasyfikacja obrazów ☞ Are we there yet?
- ☞ Discover SOTA papers with code
- ☞ The open, callable registry of SOTA

Rys: ☞ The revolution of deepth, Armando Vieira

Wszędzie AI

- rozpoznawanie obiektów (twarzy, emocji), etykietowanie i lokalizacja obiektów, segmentacja obrazów
- automatyczne tłumaczenie, analiza tekstu, mowy,
- modele języka (GPT, BERT, ...), modele wielo-modalne
- przewidywanie wyników wyborów, trzęsień ziemi, sekwencjonowanie białek (AlphaFold)
- sterowanie robotami, samochodami, gry komputerowe (AlphaGo, AlphaStar)
- kompresja sygnałów, rekonstrukcja obrazów, odszumianie sygnałów, kolorowanie czarno-białych filmów, zwiększanie rozdzielczości ...
- generowanie i synteza sygnałów: muzyki, obrazów (Midjourney, DALL-E 2), tekstu, pisma ręcznego, mowy, kodu komputerowego, ...
- AI tworzące AI (Google AutoML), AI rozmawiające z AI (Google Duplex)

DNN

Głębokie MLP

MLP przypomnienie

Typy neuronów

- liniowe, sigmoidalne, tanh
- funkcje sigmoidalne nasycają się, co utrudnia optymalizację
- ReLU, Maxout - szybsza zbieżność, nie nasycają się

Optymalizacja metodami spadku gradientu lub innymi:

- wsteczna propagacja
- SGD, trening z momentem, RPROP
- metody 2 rzędu: Newtona, Levenberg-Marquarda, gradientów sprzężonych, kosztowne - wymagają obliczenia odwrotności Hessianu
- Adam, AdaGrad, moment Nesterova - nowe algorytmy uczenia

Metody regularyzacji:

- L_1, L_2, wczesne zatrzymanie
- dropout, batch normalization, gradient clipping, data augmentation, ...

Funkcje kosztu - przypomnienie

Błąd średniokwadratowy (MSE)

$$E = \frac{1}{N} \sum (f(\mathbf{x}) - y)^2$$

- problemy aproksymacji (wyjścia ciągłe)
- liniowe neurony wyjściowe

Entropia krzyżowa, Cross-Entropy (CE)

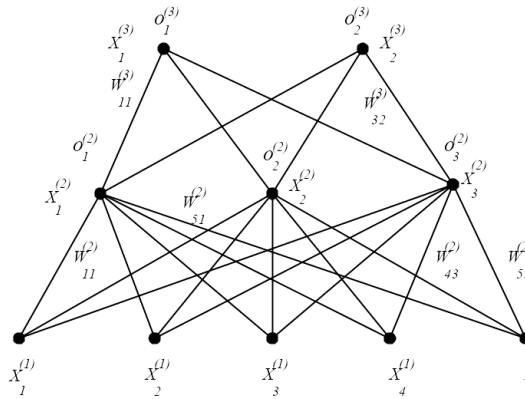
$$E = -\sum y \log f(\mathbf{x})$$

- klasyfikacja (wyjścia binarne)
- softmax lub sigmoidalne neurony wyjściowe

Wsteczna propagacja błędu - przypomnienie

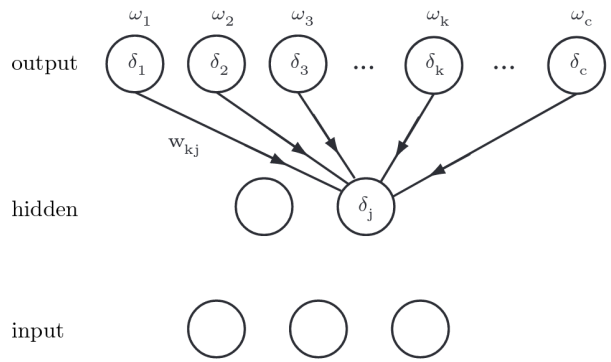
1. Zainicjuj wagi $w_{ij}^{(k)}$ małymi losowymi wartościami
2. Dopóki nie zostanie spełniony warunek stopu, wykonuj
   - propagacja sygnału od wejścia do wyjścia

$$o_i^{(K)} = \sigma\left(\sum_j w_{ji}^{(K)} o_j^{(K-1)}\right)$$



   - propagacja sygnału błędu od warstwy wyjściowej do wejściowej

$$\delta_i^{(K)} = \sigma'\left(z_i^{(K)}\right) \sum_j \delta_j^{(K+1)} w_{ij}^{(K+1)}$$



   - aktualizuj wagi

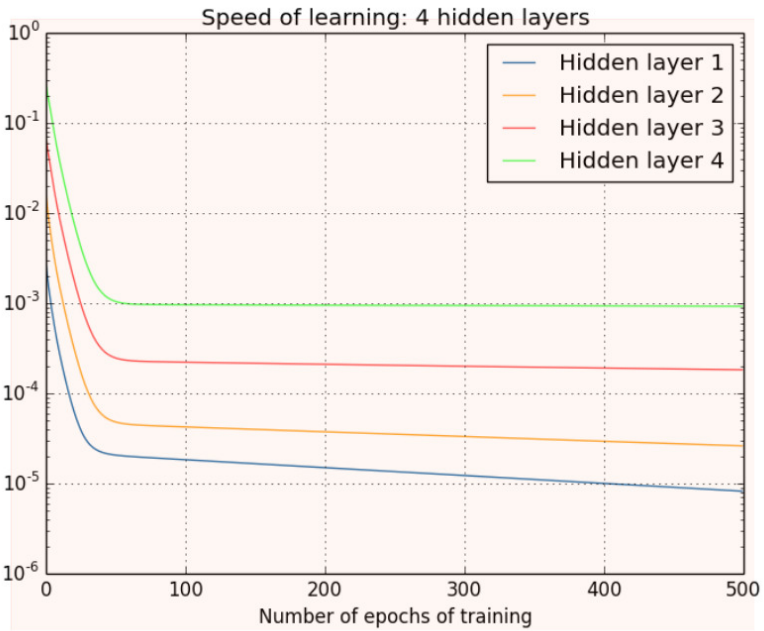
$$\Delta w_{ij}^{(K)} = \eta \delta_j^{(K)} o_i^{(K-1)}$$

Problemy związane z uczeniem (głębokich) sieci

- **zanikający gradient** - błąd propagowany od warstwy wyjściowej zanika przy niższych warstwach powodując zanik tempa uczenia, szczególnie w przypadku funkcji sigmoidalnych
- funkcje ograniczone (sigmoidalna, tangens hiperboliczny) „nasycają się" i posiadają niezerowy gradient tylko w wąskim przedziale aktywacji bliskim 0
- **przeuczenie** - większa liczba parametrów sprzyja przetrenowaniu, potrzebne skuteczne techniki regularyzacyjne
- większa ilość parametrów + duże dane $\rightarrow$ długi czas uczenia i duże wymagania dotyczące pamięci

Zanikający gradient

Przykład:
MNIST, MLP 784x30x30x30x30x10, regularyzacja $\|\mathbf{w}\|^2$

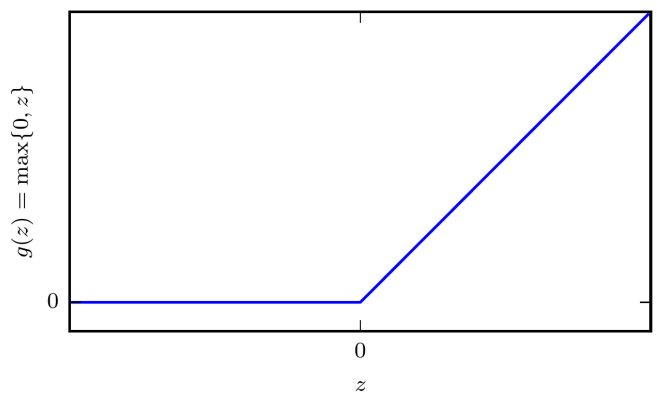


Szybkość uczenia $||\delta||$ drastycznie maleje w głębszych warstwach

Nielsen, 2016 [2]

Rectified linear unit (ReLU)

$$g(z) = \max(0, z), \qquad g'(z) = \begin{cases} 1 & \text{dla} \quad z > 0 \\ 0 & \text{dla} \quad z < 0 \end{cases}$$



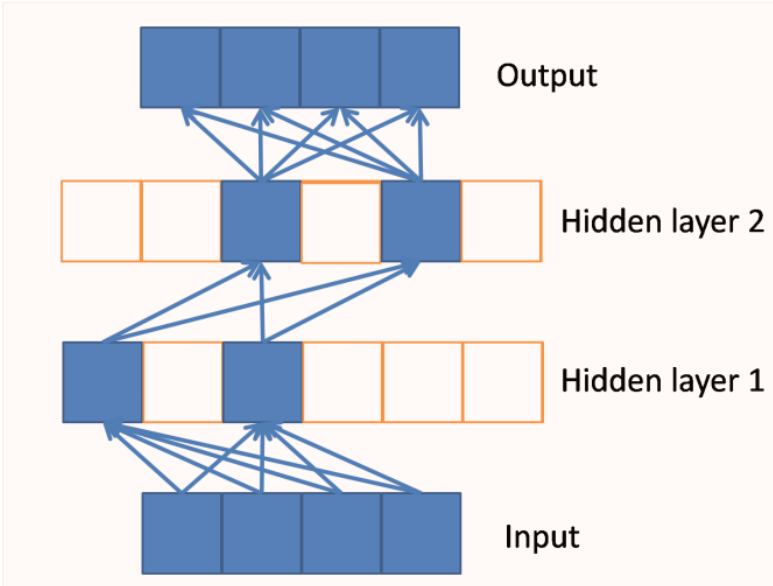
X. Glorot, A. Bordes and Y. Bengio (2011). Deep sparse rectifier neural networks

Korzyści ReLU

- gradient nie znika, gdy jednostka jest aktywna pochodna wynosi 1, to ogranicza problem znikającego gradientu
- fragmentami liniowa, skuteczniejsza optymalizacja metodami gradientowymi
- efektywna obliczeniowo: mnożenie, dodawanie, warunek
- w głębokich sieciach ReLU można pominąć pre-trening (Glorot, 2011)
- **rzadka reprezentacja** (*sparse representation*), tylko część jednostek jest aktywna w czasie przetwarzania sygnału, reszta ma aktywację równą 0

X. Glorot, A. Bordes and Y. Bengio (2011). Deep sparse rectifier neural networks

Rzadka reprezentacja



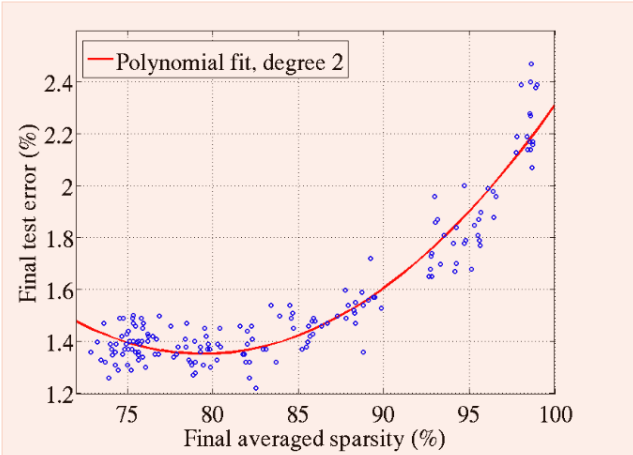


Figure 3: **Influence of final sparsity on accuracy.** 200 randomly initialized deep rectifier networks were trained on MNIST with various L_1 penalties (from 0 to 0.01) to obtain different sparsity levels. Results show that enforcing sparsity of the activation does not hurt final performance until around 85% of true zeros.

Rzadka reprezentacja dodaje nadmiarowość do modelu, zmiana wywołana przez sygnał wejściowy wprowadza zmiany tylko w podzbiorze neuronów. To działa jak automatyczny wybór cech: różne podzbiory neuronów „odpalają się” dla różnych wzorców danych.

ReLU networks results

Sieci z jednostkami ReLU uzyskują dobre wyniki bez konieczności stosowania pre-treningu, czyli metody ustalenia początkowych wartości wag w procesie wstępnego treningu (zazwyczaj nienadzorowanego)

Neuron	MNIST	CIFAR10	NISTP	NORB
***With* unsupervised pre-training**				
Rectifier	**1.20%**	**49.96%**	**32.86%**	**16.46%**
Tanh	**1.16%**	**50.79%**	35.89%	17.66%
Softplus	**1.17%**	**49.52%**	**33.27%**	19.19%
***Without* unsupervised pre-training**				
Rectifier	**1.43%**	**50.86%**	**32.64%**	**16.40%**
Tanh	1.57%	52.62%	36.46%	19.29%
Softplus	1.77%	53.20%	35.48%	17.68%

Glorot, Deep Sparse Rectifier Neural Networks, 2011

Problemy ReLU

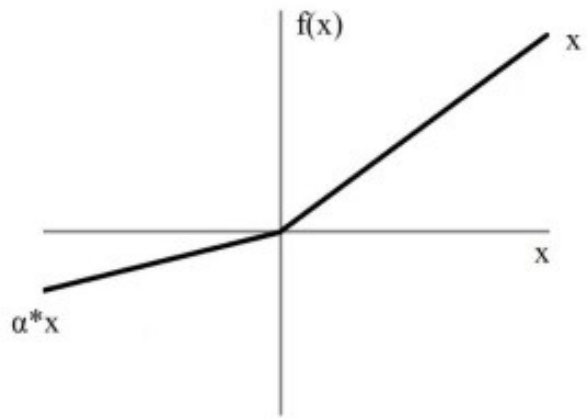
- nie jest różniczkowalne dla $z = 0$
- nie jest symetryczna i wycentrowana w 0
- nieograniczona, może powodować nieograniczony wzrost wartości aktywacji
- **problem umierających ReLU** - gdy neurony osiągają stan permanentnego braku aktywacji $g(z) = 0$ i ich wagi nie są modyfikowane w trakcie uczenia
- druga pochodna równa 0, nie można stosować metod 2 rzędu

Uogólnienia ReLU

- **Absolute ReLU** (Jarrett, 2009), $g(z) = |z|$ skuteczna w szczególnych zastosowaniach, np. dla obrazów z symetrią
- **Leaky ReLU** (Maas, 2013), wprowadza małą (np. $\alpha = 0.01$) wartość aktywacji dla $z < 0$

$$g(z) = \begin{cases} z & \text{dla} \quad z > 0 \\ \alpha z & \text{dla} \quad z < 0 \end{cases}$$

- **Parametric ReLU** odmiana Leaky ReLU, gdzie α podlega optymalizacji w trakcie uczenia



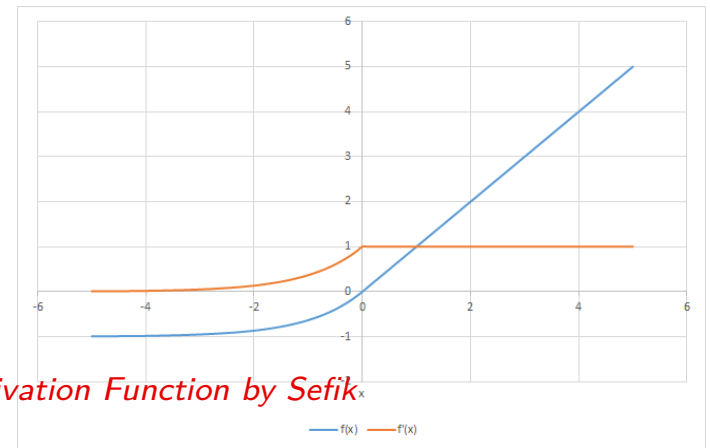
Modyfikacje ReLU

- **Noisy ReLU** dodaje niewielki szum z rozkładu normalnego

$$g(z) = \max(0, z + \epsilon), \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

- **Exponential linear unit (ELU)** (Clevert, 2015), aktywacje w warstwie mają średnio wartość bliższą 0, co zwykle poprawia warunki optymalizacji i prowadzi do szybszej zbieżności w stosunku do ReLU i leaky ReLU

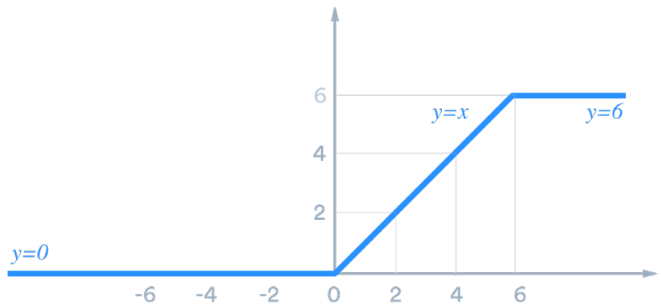
$$g(z) = \begin{cases} z & \text{dla} \quad z > 0 \\ \alpha(e^z - 1) & \text{dla} \quad z < 0 \end{cases} \qquad g'(z) = \begin{cases} 1 & z > 0 \\ \alpha e^z, & z < 0 \end{cases}$$



Rys: ☞ ELU as a Neural Networks Activation Function by Sefik

ReLU6

$$g(z) = \min(\max(0, z), 6)$$

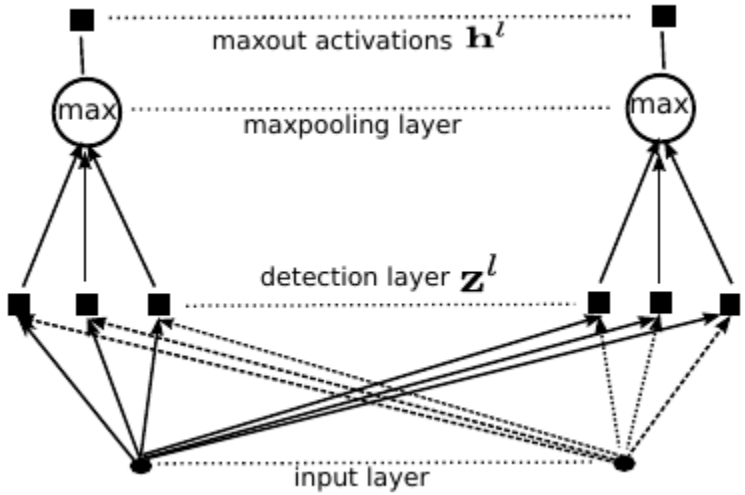


- ReLU6 jest ograniczona $[0, 6]$, co może pomóc w stabilizacji procesu uczenia i zapobiegać eksplodującym gradientom i nieskończonym wartościom aktywacji
- Górne ograniczenie przyspiesza tworzenie rzadkiej reprezentacji, dobre wyniki na CIFAR-10 (Krizhevsky)
- Stosowana w modelach mobilnych (MobileNets) ze względu na efektywność obliczeniową i możliwość kwantyzacji do 8-bitowych liczb całkowitych, co jest korzystne dla urządzeń o ograniczonych zasobach

Rys: ☞ A Practical Guide to ReLU by Danqing Liu

Maxout

$$g_i(\mathbf{z}) = \max_{j \in G_i} z_j$$



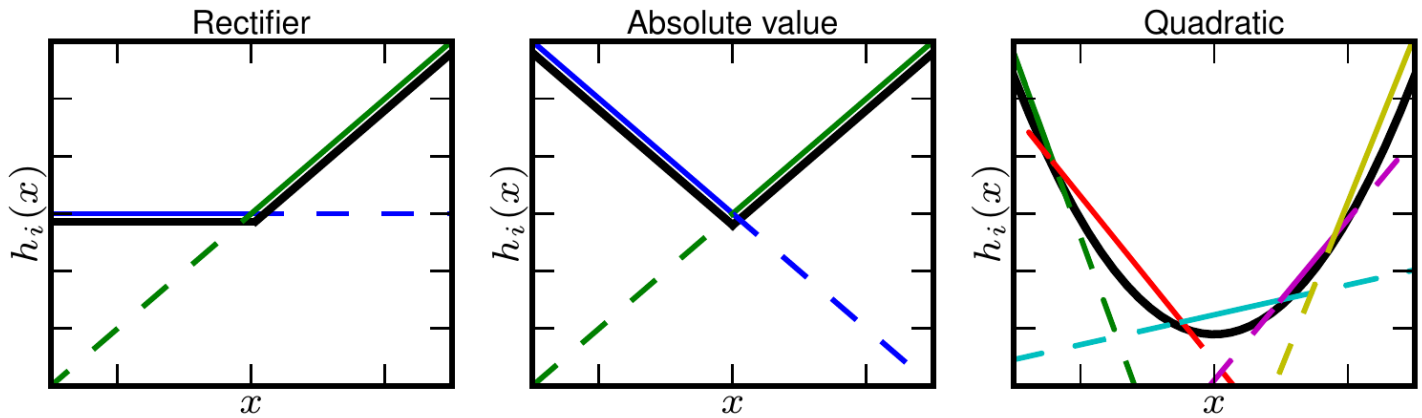
Zwraca wyjście najaktywniejszego neuronu z grupy k neuronów w warstwie, gdzie G_i to zbiór neuronów należących do i-tej grupy

$$g'(z_j) = \begin{cases} 1 & \text{dla} \quad j = \arg\max_i z_i \\ 0 & \text{dla} \quad j \neq \arg\max_i z_i \end{cases}$$

Rys:Pawel Swietojanski, Investigation of maxout networks for speech recognition.
Goodfellow, et al, (2013). "Maxout Networks". JMLR WCP. 28 (3): 1319–1327.

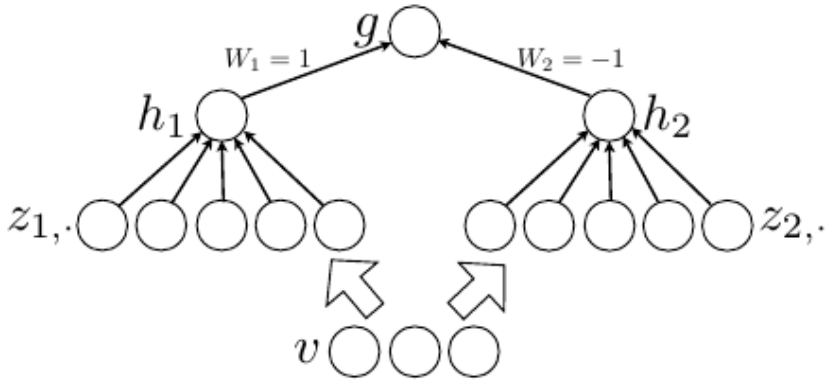
Maxout

- realizuje funkcję wypukłą, odcinkami liniową, złożoną z k fragmentów; w efekcie uczony jest kształt funkcji aktywacji



- ReLU i PReLU to szczególne przypadki Maxout
- odcinkami liniowa, więc korzystna dla optymalizacji metodami gradientowymi
- rzadka reprezentacja: większość neuronów ma niezerowe wyjścia, ale tylko sygnał zwycięzcy z grupy jest propagowany; sygnał błędu również jest propagowany tylko przez jednostkę zwycięską

Sieć Maxout



- złożenie funkcji wypukłych pozwala modelować dowolną funkcję ciągłą
- sieć z neuronami maxout jest uniwersalnym aproksymatorem
- aktywacja zależy od grupy wag, więc wprowadza do modelu redundancję, reprezentacja informacji jest rozproszona, co pozwala niwelować fenomen **katastrofalnego zapominania**, gdy sieć zapomina wyuczone wcześniej wzorce podczas uczenia nowych danych

Rys: Goodfellow, et al, (2013). "Maxout Networks". JMLR WCP. 28 (3): 1319–1327.

Maxout network on MNIST

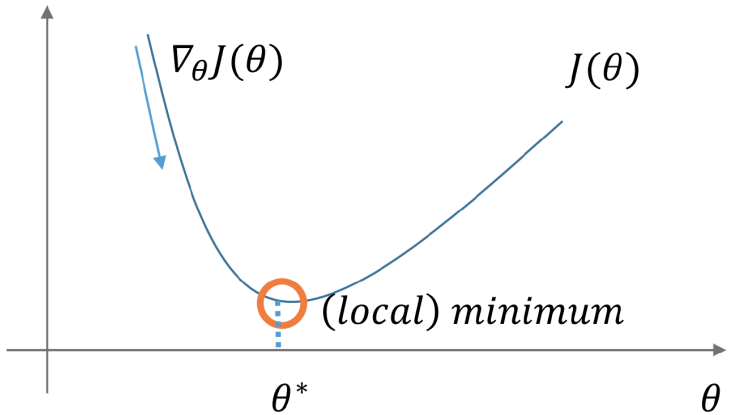
Method	Test error
Rectifier MLP + dropout (Srivastava, 2013)	1.05%
DBM (Salakhutdinov & Hinton, 2009)	0.95%
Maxout MLP + dropout	**0.94%**
MP-DBM (Goodfellow et al., 2013)	0.91%
Deep Convex Network (Yu & Deng, 2011)	0.83%
Manifold Tangent Classifier (Rifai et al., 2011)	0.81%
DBM + dropout (Hinton et al., 2012)	0.79%

Goodfellow, Maxout Networks, et al., 2013

Optymalizacja spadkiem gradientu

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E$$

$$E = \frac{1}{N} \sum_{\mathbf{x}} E(f(\mathbf{x}), y)$$



- **Batch gradient descent** - dla całego zbioru, niepraktyczne dla dużych danych
- **SGD** *on-line* - stochastic gradient descent dla $N = 1$, duża wariancja
- **SGD** *mini-batch* estymacja błędu na podstawie paczki zawierającej n losowych przypadków

Jak dobrać stałą uczenia η?

Rys: S. Ruder, „Optimising for Deep Learning"

SGD z momentem (Polyak, 1964)

$$w(t+1) = w(t) - \eta \nabla E(t) + \gamma \Delta w(t)$$

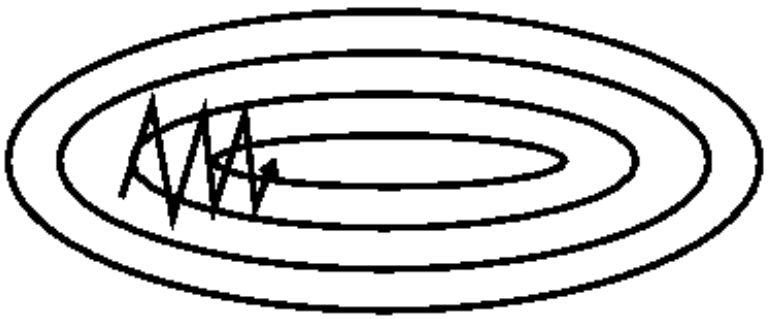
- $v(t) = \Delta w(t)$ prędkość uczenia akumuluje historyczne wartości gradientów (średnia krocząca)

$$v(t) = \gamma v(t-1) - \eta \nabla E(t)$$

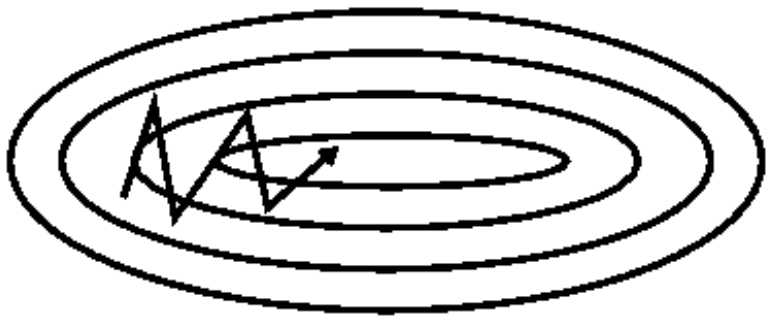
$$w(t+1) = w(t) + v(t)$$

- pęd zwiększa krok, gdy gradient nie zmienia kierunku
- redukuje oscylacje (spowalnia uczenie), gdy gradient zmienia kierunek
- gdy $\nabla E = 0$ wagi są nadal modyfikowane (bezwładność)
- typowo współczynnik $\gamma = 0.9$

Przyspieszenie zbieżności SGD za pomocą momentu



SGD bez momentu



SGD z momentem

Nesterov's accelerated gradient (NAG)

- przyspieszenie uczenia metodą momentu Nesterova (1968)

$$v(t) = \gamma v(t-1) - \eta \nabla E\big(w(t) + \gamma v(t-1)\big)$$
$$w(t+1) = w(t) + v(t)$$

- gradient $\nabla E(\hat{w}(t))$ jest liczony dla przybliżonej przyszłej pozycji

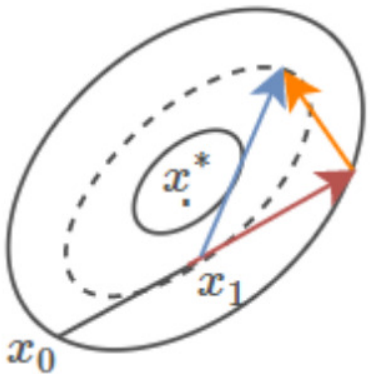
$$\hat{w}(t) = w(t) + \gamma v(t-1)$$

- trening szybciej jest w stanie zareagować na zmiany gradientu

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). Doklady ANSSSR (translated as Soviet.Math.Docl.), vol. 269, pp. 543– 547.

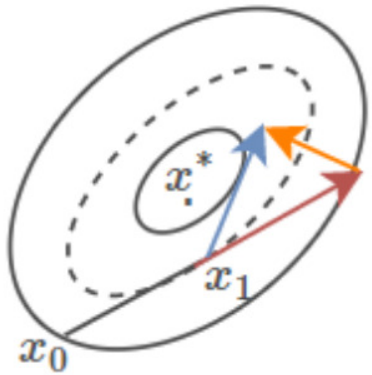
Polyak vs Nesterov

Polyak's Momentum



$$x_{t+1} = x_t - \alpha \nabla f(x_t) + \mu(x_t - x_{t-1})$$

Nesterov Momentum



$$x_{t+1} = x_t + \mu(x_t - x_{t-1}) - \gamma \nabla f(x_t + \mu(x_t - x_{t-1}))$$

Dla małych wartości η metody stają się równoważne

Rys: Ioannis Mitliagkas, Theoretical principles for deep learning, 2018

AdaGrad (Duchi, 2011)

$$w(t+1) = w(t) - \frac{\eta}{\sqrt{v(t)}} \nabla E(t)$$

gdzie $v(t)$ akumuluje historyczne wartości gradientów

$$v(t) = v(t-1) + [\nabla E(t)]^2 = \sum_{t' < t} [\nabla E(t')]^2$$

Efektywny współczynnik uczenia $\frac{\eta}{\sqrt{v(t)}}$ jest ustalany dla każdego optymalizowanego parametru niezależnie

- strome spadki funkcji błędu $\Rightarrow$ duży gradient $\Rightarrow$ mniejszy krok uczenia, szybka zbieżność dla funkcji wypukłych
- małe gradienty $\Rightarrow$ większy krok uczenia przeciwdziała efektowi znikającego gradientu
- domyślna wartość $\eta = 0.01$, zazwyczaj nie wymaga zmiany
- wada: mianownik może szybko przybrać duże wartości, co powoduje przedwczesny zanik uczenia (problem ten niwelują algorytmy RMSProp, Adadelta, Adam)

RMSProp (Hinton, 2012)

$$w(t+1) = w(t) - \frac{\eta}{\sqrt{v(t)}} \nabla E(t)$$

gdzie $v(t)$ jest wykładniczo ważoną średnią kroczącą kwadratów gradientów (EMA, *exponential moving average*)

$$v(t) = \beta v(t-1) + (1-\beta) \left[\nabla E(t)\right]^2$$

- $RMS(\nabla E(t)) = \sqrt{v(t)}$ estymacja RMS (*root mean square*)
- modyfikacja AdaGrad zapewniająca lepszą zbieżność dla nie-wypukłych funkcji
- największy wpływ na wielkość $v(t)$ mają gradienty z ostatnich iteracji
- β kontroluje skalę długości ruchomej średniej, typowo $\beta = 0.9$ (średnia z 10 kroków)
- dobre wyniki w połączeniu z momentem Nesterova

Adadelta (Zeiler, 2012)

$$w(t+1) = w(t) - \frac{\sqrt{m(t)}}{\sqrt{v(t)}} \nabla E(t)$$

gdzie $\sqrt{v(t)} = RMS(\nabla E(t))$ wyznaczany średnią kroczącą

$$v(t) = \beta v(t-1) + (1-\beta) \left[\nabla E(t)\right]^2$$

$\sqrt{m(t)} = RMS(\Delta w(t-1))$ wyznaczany średnią kroczącą kwadratów zmian parametrów Δw przybliżoną wartością z poprzedniej iteracji

$$m(t) = \beta m(t-1) + (1-\beta) \left[\Delta w(t-1)\right]^2$$

- brak globalnej stałej uczenia η ale należy dobrać współczynnik β (typowo $\beta = 0.9$)
- metoda jest heurystycznym przybliżeniem metod kwadratowych (bez wyznaczania Hessianu)
- krok Δw wykonywany jest w tych samych hipotetycznych jednostkach (gdzie inne metody 1 rzędu $\nabla E \approx \frac{1}{w}$ nie zachowują jednostki)

Adaptive Moment Estimation (Adam)

$$w(t) = w(t-1) - \frac{\eta}{\sqrt{\hat{v}(t)}}\hat{m}(t)$$

Estymaty pierwszego (średnia) i drugiego (wariancja) momentu gradientów

$$m(t) = \beta_1 m(t-1) + (1-\beta_1)\nabla E(t)$$

$$v(t) = \beta_2 v(t-1) + (1-\beta_2)[\nabla E(t)]^2$$

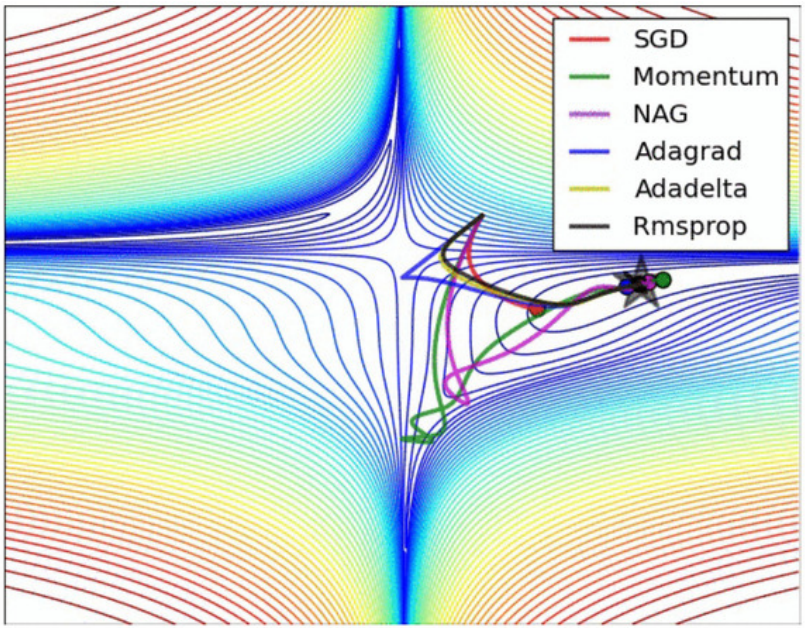
Poprawka na odchylenie w stronę wartości 0 (z powodu zerowych początkowych wartości estymaty)

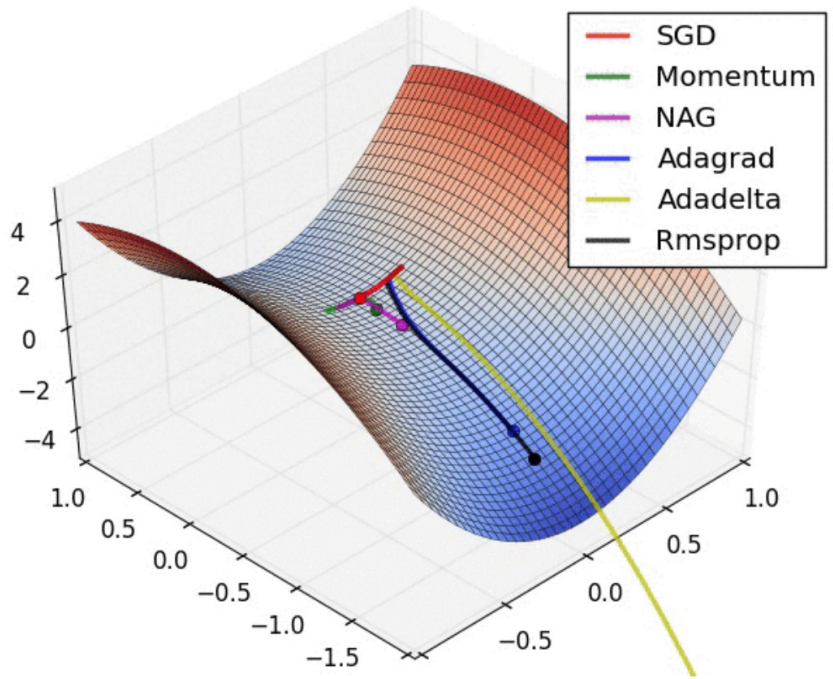
$$\hat{m}(t) = \frac{m(t)}{1-\beta_1}, \qquad \hat{v}(t) = \frac{v(t)}{1-\beta_2}$$

współczynniki zanikania $\beta_1 = 0.9$, $\beta_2 = 0.999$
stała uczenia $\eta = 0.001$

Kingma, D. P., Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13

Porównanie





☞ An overview of gradient descent optimization algorithms

Podsumowanie

- DNN są dobre do analizy danych bez wyraźnej struktury, np. przestrzennej lub czasowej (sekwencje)
- w większości zastosowań aktywacja ReLU sprawdza się bardzo dobrze, podobnie jak algorytm Adam z domyślnymi parametrami η, β_1, β_2
- SGD z wykładniczo malejącą wartością stałej uczenia może dorównać algorytmom takim jak Adam

CNN
Convolutional Neural Networks
Sieci splotowe

CNN

Convolutional Neural Networks (CNN, ConvNet) wariant MLP inspirowany biologicznie, gdzie mnożenie macierzy wag i sygnału wejściowego zastąpione jest operacją splotu

Zdolne do generalizacji sygnału posiadającego relacje przestrzenne

- 1D sygnały czasowe
- 2D obrazy
- 3D fMRI, video, obrazy RGB
- sygnały wielokanałowe

Potrafią być odporne na przestrzenne transformacje sygnału (skalowanie, obrót, przesunięcie)

Splot 1D

- **Splot** (*convolution*) funkcji $x(t)$ oraz $w(t)$

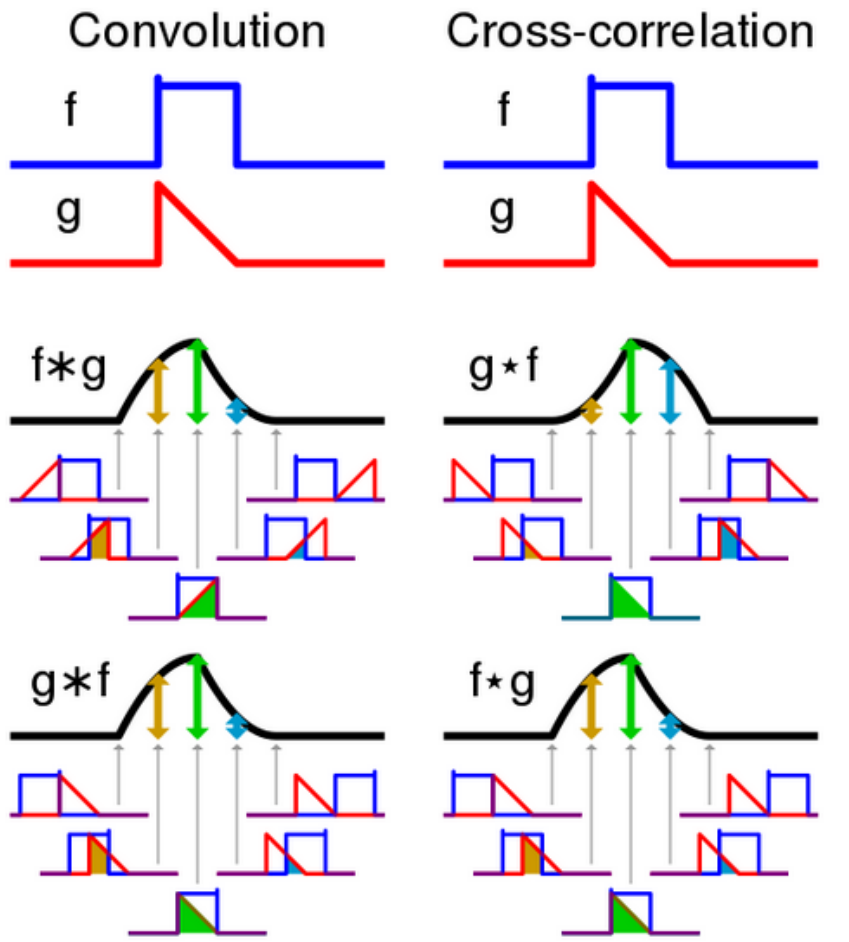
$$(x \star w)(t) = \int x(\tau) w(t - \tau) d\tau$$

- **Korelacja wzajemna** (*cross correlation*)

$$(x \star w)(t) = \int x(\tau + t) w(\tau) d\tau$$

- wynikiem jest funkcja, miara podobieństwa dwóch sygnałów jako funkcja przesunięcia jednego sygnału względem drugiego

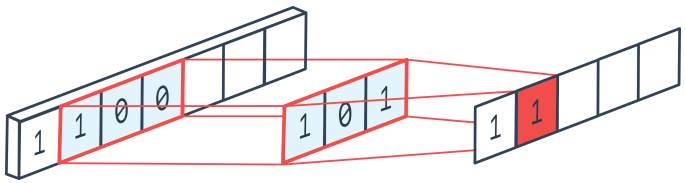
Splot i korelacja wzajemna



https://en.wikipedia.org/wiki/Cross-correlation

Splot 1D w sieciach splotowych

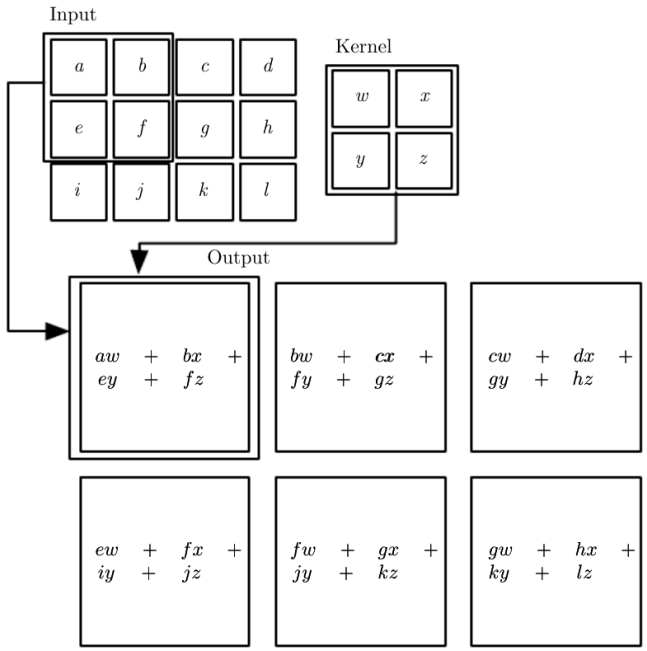
$$s(i) = \sum_m x(m+i)w(m)$$



- w sieciach neuronowych splotowych CNN mamy splot dyskretny
  x - sygnał wejściowy
  w - kernel (filtr), wagi połączeń neuronu, niezerowe tylko w ograniczonym obszarze (pole recepcyjne)
  $s(i)$ - sygnał wyjściowy, aktywacja neuronu i
- wartości wyjściowe $s(i)$ tworzą **mapę cech** (*feature map*)

Splot 2D

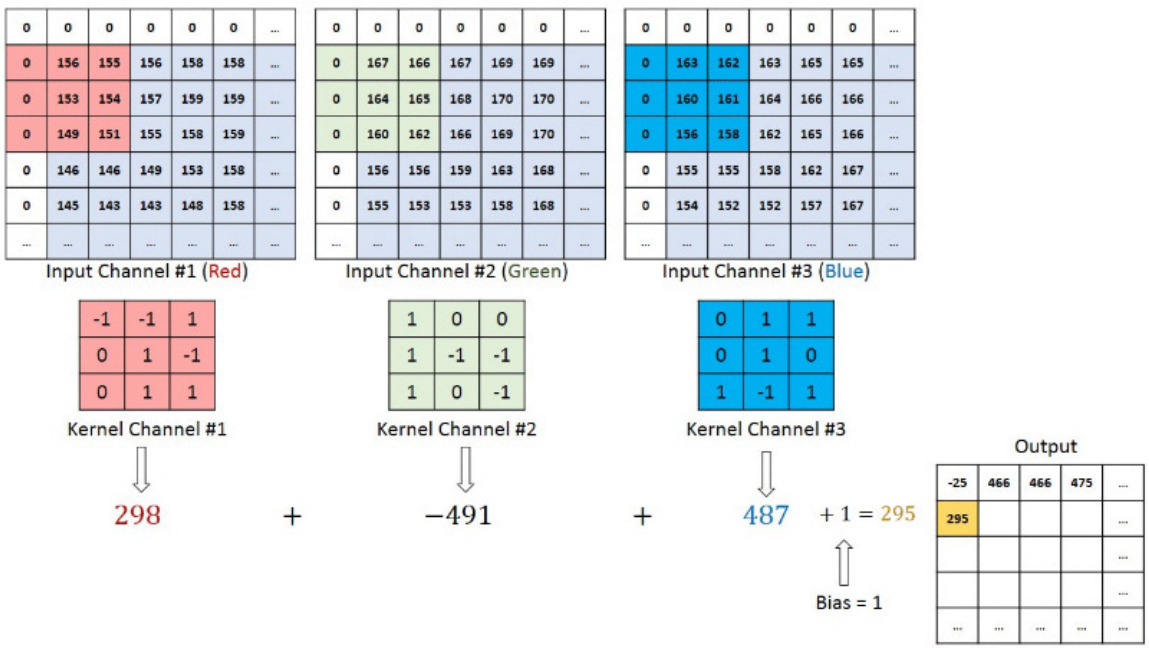
- wejście 2D, np. obraz w skali szarości WxH (jeden kanał)
- filtr 2D (kernel) o mniejszych wymiarach niż wejście (2x2) + ewentualny wyraz wolny (bias)
- wynikiem jest mapa cech 2D, gdzie każdy punkt jest sumą ważoną wartości z obszaru wejściowego (pole recepcyjne) i wag filtra



$$s(i,j) = \sum_m \sum_n w(m,n) \cdot x(i+m, j+n)$$

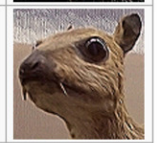
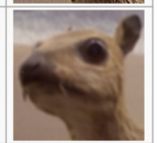
Splot 2D obrazu RGB

- wejście 3D, np. obraz RGB o wymiarach $W \times H \times 3$
- filtr 3D o wymiarach $k \times k \times 3$, gdzie $k \times k$ to rozmiar pola recepcyjnego, a 3 to liczba kanałów (RGB) + ewentualny wyraz wolny (bias)
- wyjście: mapa cech o wymiarach $(W - k + 1) \times (H - k + 1)$



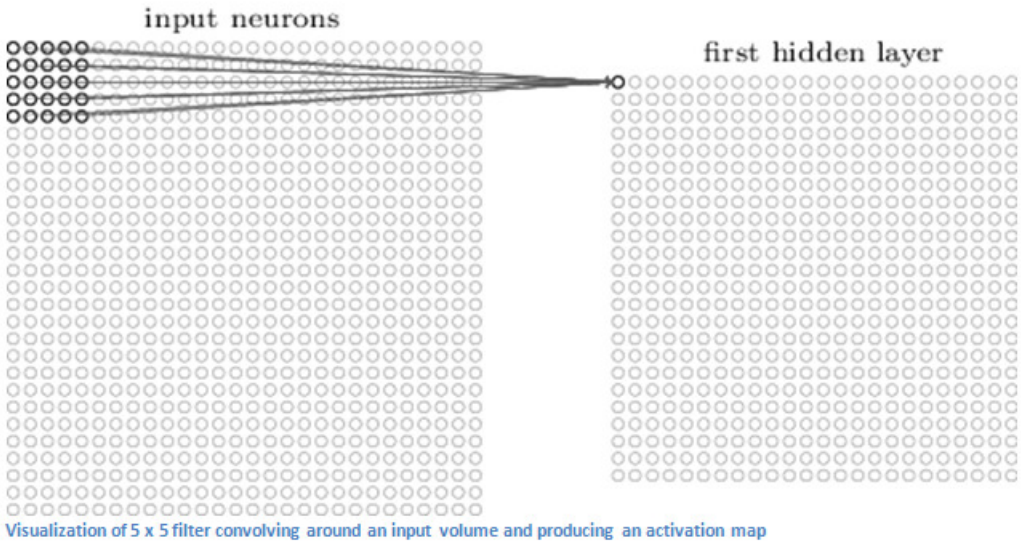
$$s(i,j) = \sum_m \sum_n \sum_c w(m,n,c) \cdot x(i+m, j+n, c) + bias$$

Filtry graficzne 2D - przykłady

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Pola recepcyjne i mapy cech



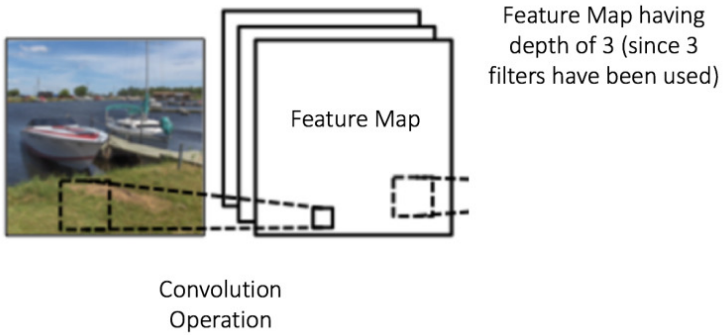
Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

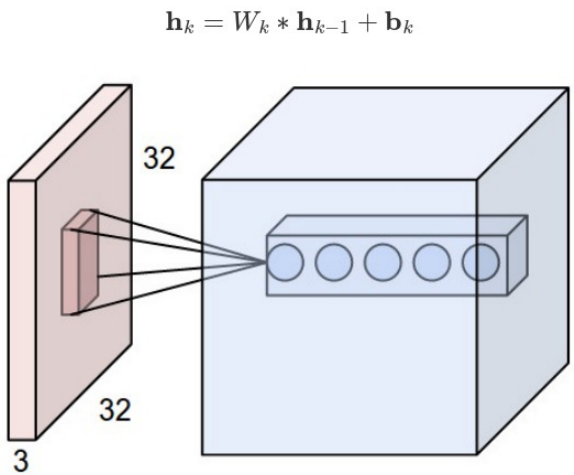
- **Pole recepcyjne** - obszar „widziany” przez neuron (rozmiar filtra)
- wyjście neuronu: splot sygnału i liniowego filtra, ewentualny wyraz wolny (bias) i nieliniowa funkcja wyjściowa (ReLU, tanh), generują **mapę cech** (*feature map)*

$$s(i,j) = \sigma\left(b + \sum_m \sum_n \sum_k w(m,n,k) \cdot x(i+m, j+n, k)\right)$$

Pola recepcyjne i mapy cech

- N neuronów (filtrów) tworzy na wyjściu N map cech, tworząc tensor o wymiarach $M_x \times M_y \times N$, gdzie M_x, M_y - rozmiary mapy wyjściowej, N - liczba filtrów (kanałów wyjściowych)





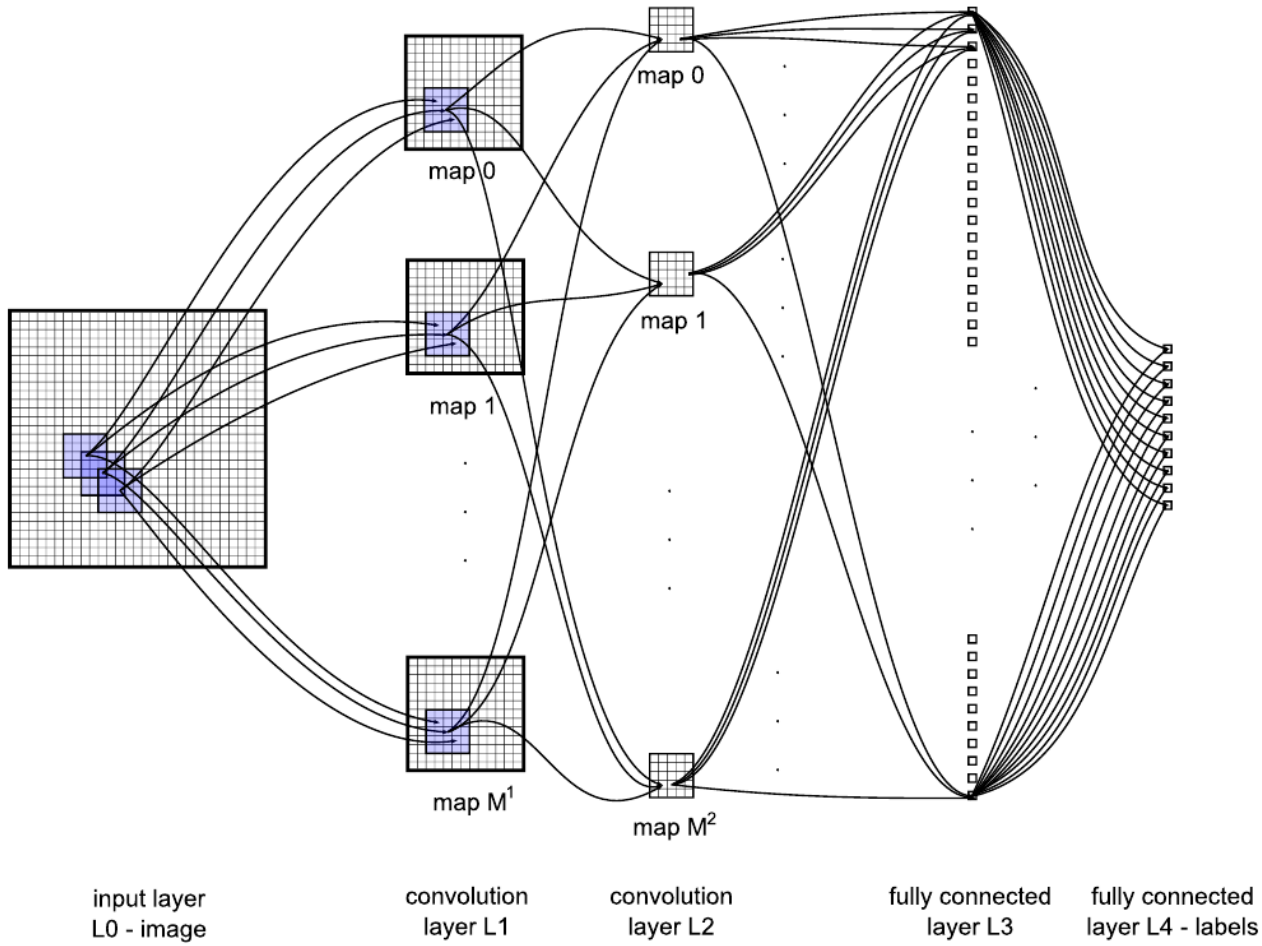
- Przy przetwarzaniu obrazów sygnał wejściowy x to tensor 3D: szerokość, wysokość, kanał (RGB).

$$s(i,j,l) = \sum_{mnk} x(i+m, j+n, k) w(m,n,k,l)$$

gdzie l to indeks filtra (kanału wyjściowego)

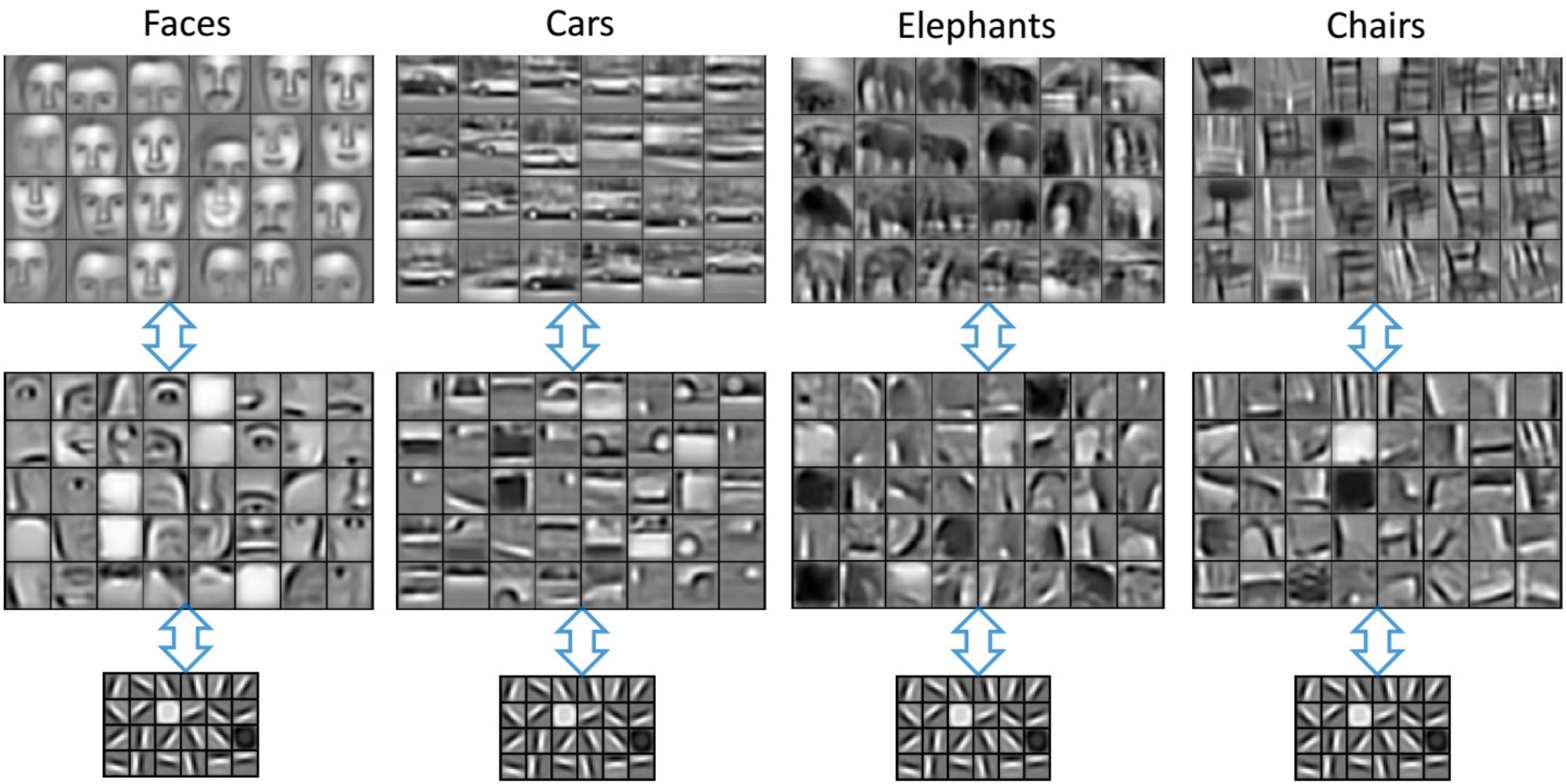
- w praktyce x rozszerzony do 4D o wymiar związany z rozmiarem mini-batcha

Typowa architektura CNN



☞ An Interactive Node-Link Visualization of Convolutional Neural Networks by Adam W. Harley

Przykłady filtrów



Goodfellow, 2016 [1]

Rozmiar mapy cech w splocie 2D

Rozmiar mapy wyjściowej (M_x^n, M_y^n) w warstwie n po wykonaniu splotu 2D z przesunięciem (S_x^n, S_y^n) oraz uzupełnieniem brzegów sygnału wejściowego o P_x^n, P_y^n (zero padding) wynosi

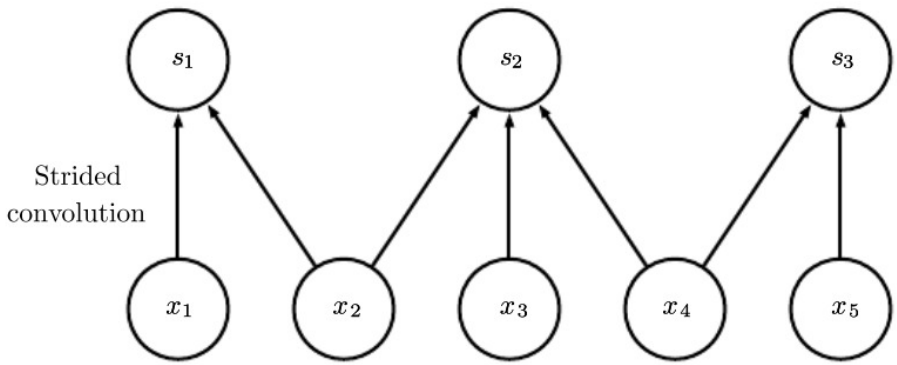
$$M_x^n = \frac{M_x^{n-1} - K_x^n + 2P_x^n}{S_x^n} + 1 \qquad M_y^n = \frac{M_y^{n-1} - K_y^n + 2P_y^n}{S_y^n} + 1$$

gdzie
M_x^{n-1}, M_y^{n-1} to rozmiar mapy cech warstwy $n-1$ (sygnał z warstwy poprzedniej),
K_x, K_y to rozmiar filtra (szerokość i wysokość)
S_x, S_y to przesunięcie (*stride*) wzdłuż osi x i y,
P_x, P_y wielkość rozszerzenia brzegu sygnału wejściowego (*zero padding*)
Splot 2D z przesunięciem

$$s(i,j,l) = \sum_{mnk} x(i \cdot S_x + m, j \cdot S_y + n, k) w(m,n,k,l)$$

Stride - krok przesunięcia

Przykład w 1D: S=2, K=3, P=1

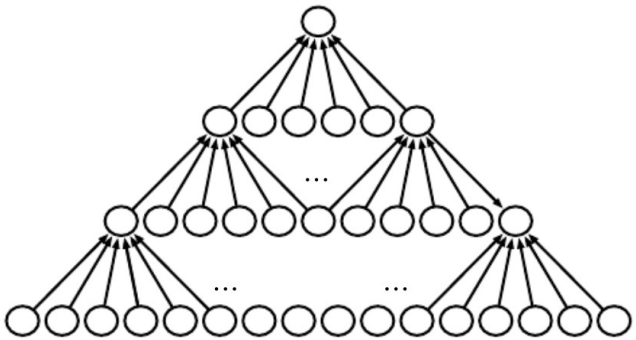


$$M^n = \frac{M^{n-1} - 3 + 2}{2} + 1$$

- redukcja wymiaru - zmniejszenie wymogów obliczeniowych i pamięciowych
- zmniejszenie rozdzielczości sygnału
- kosztem mniej dokładnej reprezentacji

Zero padding

valid zero padding: brak rozszerzenia na brzegach ($P = 0$)



$$M^n = \frac{M^{n-1} - K^n}{S^n} + 1$$

- rozmiar map maleje w kolejnych warstwach
- ogranicza to możliwość budowania głębokich sieci i wymusza stosowanie małych filtrów
- sygnał wejściowy na brzegach mapy jest słabiej reprezentowany w sygnale wyjściowym (mniej aktywacji neuronów w pobliżu brzegów)

Zero padding

same zero padding: brzegi wypełnione dodatkowymi wartościami (zerami) aby zapewniać odtworzenie wymiaru sygnału wejściowego $P = \lfloor \frac{K}{2} \rfloor, S = 1$



- pozwala budować bardzo głębokie sieci o dowolnych wielkościach filtrów $M_i^n = \lceil \frac{M_i^{n-1}}{S_i} \rceil$

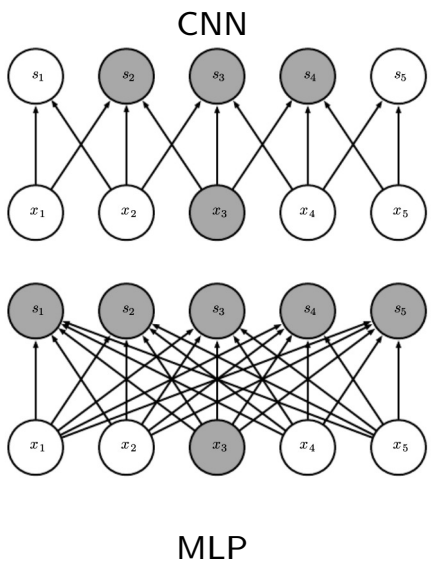
Właściwości CNN

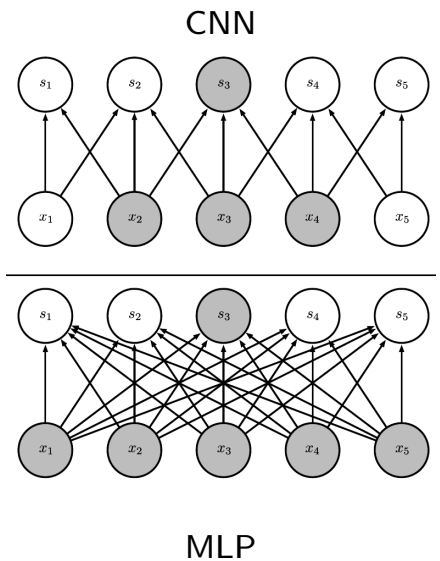
- **rzadka reprezentacja** - rozmiar filtra jest dużo mniejszy od rozmiaru sygnału wejściowego, pojedyncze wejście oddziałuje tylko na grupę neuronów w zasięgu pola recepcyjnego
- **współdzielenie parametrów** - warstwa splotowa może być widziana jako w pełni połączona warstwa ze współdzielonymi wagami (dużo mniejsza liczba parametrów w stosunku do MLP)
- **równoważność względem przesunięcia** sygnału - ta sama cecha znajdująca się w różnych miejscach obrazu będzie aktywowała ten sam filtr
- możliwość zastosowania sygnału wejściowego o nieustalonym rozmiarze - większy obraz wejściowy wygeneruje większe mapy. W przypadku MLP zwiększenie rozmiaru wektora wejściowego wymaga rozbudowy architektury.

Rzadka reprezentacja

pojedyncze wejście aktywuje tylko grupę neuronów w małym obszarze wyjściowej mapy cech, podczas gdy w MLP każdy neuron jest aktywowany przez całe wejście

wyjście neuronu zależy od małego obszaru sygnału wejściowego (pole recepcyjne), podczas gdy w MLP każdy neuron widzi całe wejście





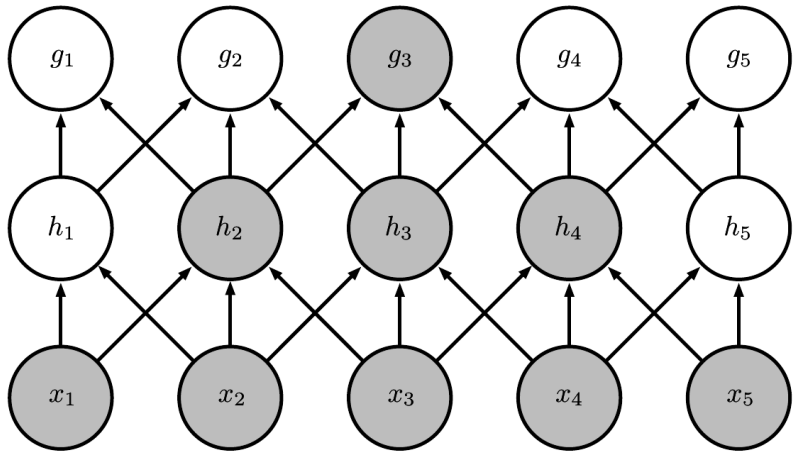
MLP: mnożenie macierzy $O(m \times n)$ połączeń

CNN: warstwa splotowa $O(k \times n)$ połączeń, gdzie rozmiar filtra $k \ll m$

Źródło grafiki: Goodfellow, 2016 [1]

Efektywne pole recepcyjne

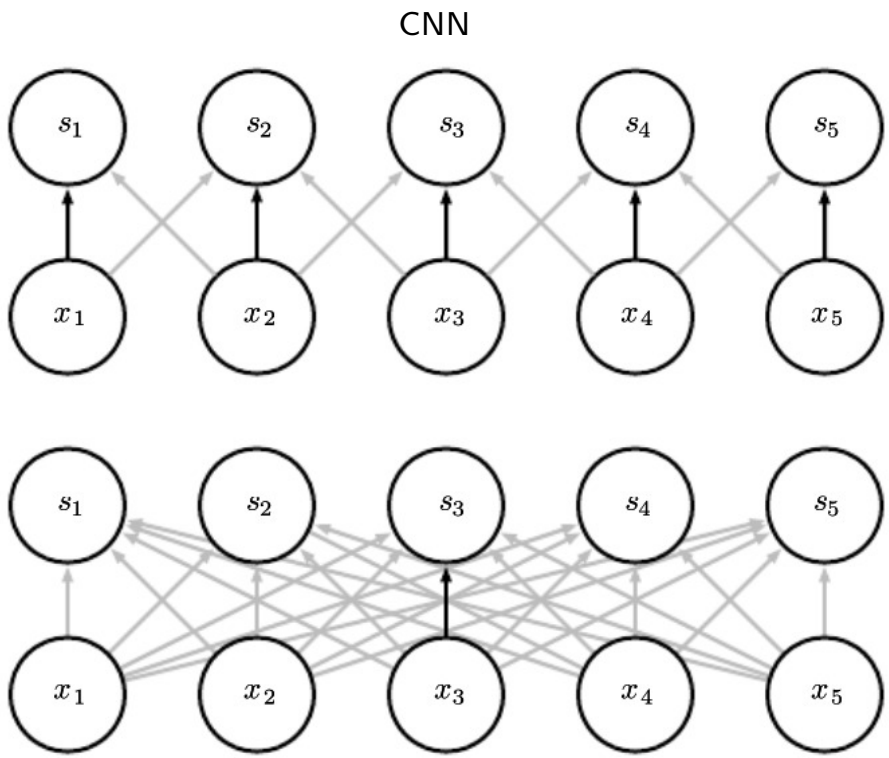
- **Efektywne pole recepcyjne** - obszar sygnału wejściowego rejestrowany przez neurony w wyższych warstwach rośnie z głębokością



- stride, pooling i dilation (rozrzedzony splot) dodatkowo zwiększają efektywne pole recepcyjne
- pomimo rzadkich połączeń i ograniczonego zasięgu pól recepcyjnych głęboka sieć jest w stanie modelować złożone (odległe) zależności

Współdzielenie wag

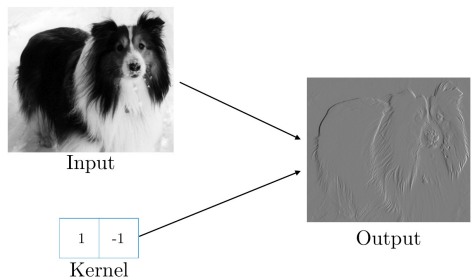
- ta sama waga jest używana przy przetwarzaniu każdego punktu wejściowego, gdzie w MLP każda waga jest unikalna dla każdego połączenia
- pojedynczy filtr pozwala wykryć tę samą cechę w różnych położeniach obrazu wejściowego (*equivariance to translation*)



Źródło grafiki: Goodfellow, 2016 [1]

Rzadkie połączenia i współdzielenie wag

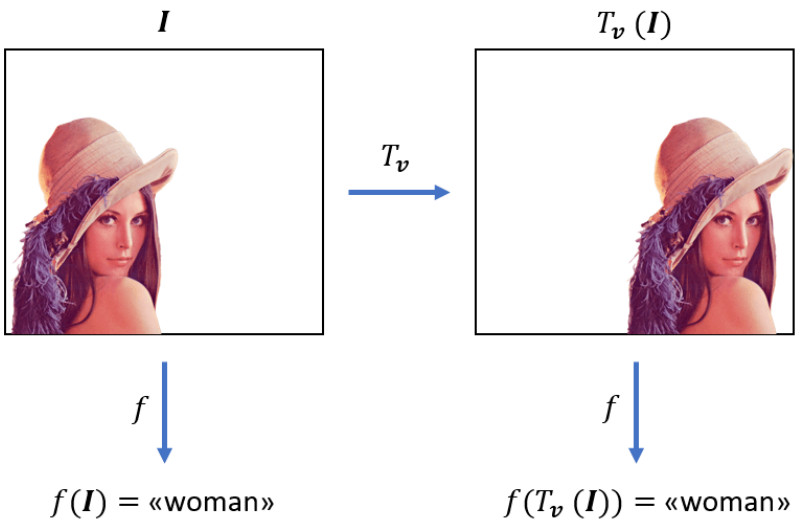
Przykład: wykrywanie pionowych krawędzi obrazu $n \times n$



- MLP: mnożenie pełnej macierzy $n^2 \times n^2 = n^4$ wag, $n^2 \times (n^2 + 1)$ operacji
- CNN: splot filtrem 1x2 (2 wagi) wymaga $n^2 \times 3$ operacji (2 mnożenia i dodawanie)
- warstwa splotowa jest istotnie wydajniejsza w mapowaniu powtarzających się relacji w małych, lokalnych rejonach sygnału wejściowego

Źródło grafiki: Goodfellow, 2016 [1]

Równoważność przy przesunięciu



Przesunięcie sygnału wejściowego $\mathbf{x}$ o wektor $\mathbf{v}$

$$T_{\mathbf{v}}(\mathbf{x})(\mathbf{u}) = \mathbf{x}(\mathbf{u} - \mathbf{v})$$

powoduje identyczne przesunięcie sygnału na mapie cech

$$f(T_{\mathbf{v}}(\mathbf{x})) = T_{\mathbf{v}}(f(\mathbf{x}))$$

Aktywowane są te same filtry wykrywające obiekt.
Równoważność translacji jest pożądaną własnością w rozpoznawaniu obrazów, gdzie lokalne cechy (np. krawędzie) mogą wystąpić w wielu miejscach obrazu, a ich położenie nie jest istotne dla klasyfikacji.
W przypadku sieci MLP przesunięcie sygnału wejściowego powoduje zmianę aktywacji wszystkich neuronów w sieci

Rys. https://www.baeldung.com/cs/translation-invariance-equivariance

Równoważność przy przesunięciu

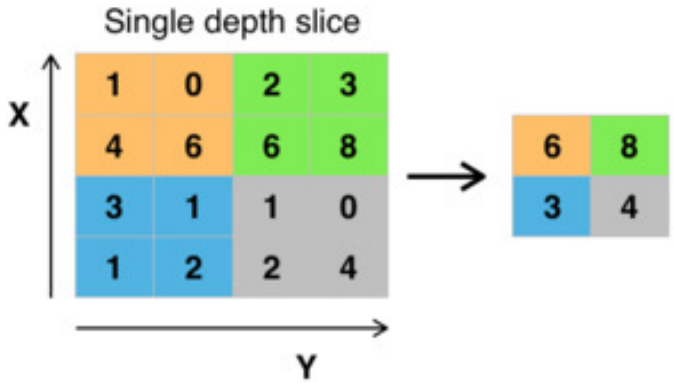
- współdzielenie wag dla całego wejścia nie zawsze jest pożądane, można zastosować różne filtry w różnych obszarach obrazu, np. rozpoznawanie twarzy ze zdjęć paszportowych, które posiadają charakterystyczne cechy występujące wyłącznie w określonych rejonach sygnału wejściowego
- Sieci CNN (operacje splotu) **nie są niezmiennicze** względem zmiany **skali, obrotu oraz przekształceń afinicznych obrazu**. Taką własność osiąga się poprzez rozszerzenie zbioru treningowego o przypadki zdeformowane (szum, skalowanie, obrót, zmiana kontrastu itd.)

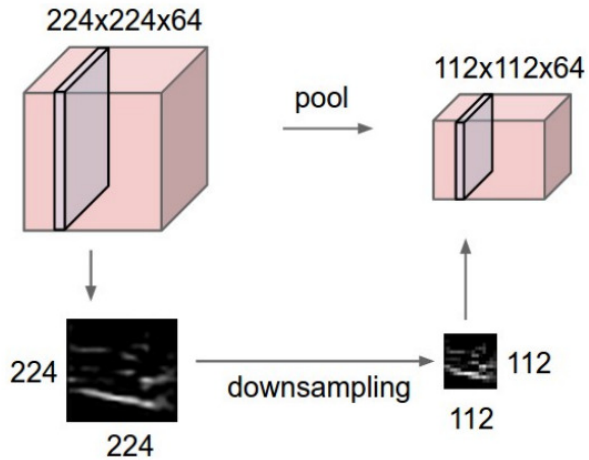
Pooling

W skład typowego bloku realizującego warstwę splotową wchodzi:

- liniowa aktywacja (splot)
- detekcja (nieliniowość np. ReLU)
- redukcja (pooling) - „uogólnienie” wartości sąsiadujących wyjść

Max pooling - maksimum z pewnego podobszaru



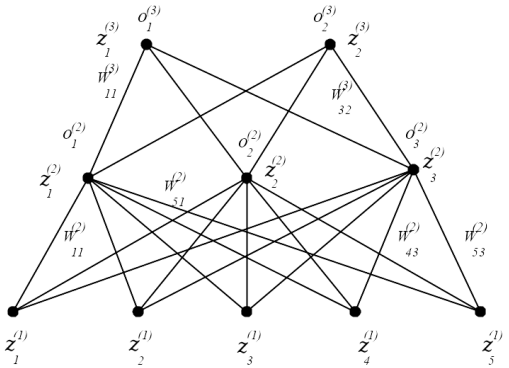


- redukcja wymiarowości - przesunięcie filtra S typowo równe jego wielkości K (maksimum z rozłącznych obszarów)
- inne podejścia: *avg. pooling* (średnia z sąsiedztwa), *norm pooling* (norma z sąsiadujących wyjść), ważona średnia od centrum, niektóre architektury rezygnują z tej warstwy

Wsteczna propagacja w MLP (przypomnienie)

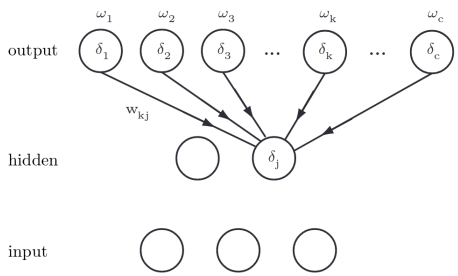
Sygnał propagowany od wejścia do wyjścia

$$o_i^h = f(z_i^h) \qquad z_i = \sum_j w_{ji}^h o_j^{h-1} + b_i^h$$



Sygnał błędu neuronów δ_i^k propagowany od warstwy wyjściowej do wejścia

$$\delta_i^h = \frac{\partial E}{\partial z_i^h} = f'(z_i^h) \sum_j \delta_j^{h+1} w_{ij}^{h+1}$$



Aktualizacja wag w kierunku spadku gradientu

$$\Delta w_{ij}^h = -\eta \frac{\partial E}{\partial w_{ij}^h} = \eta \delta_j^h o_i^{h-1}$$

Wsteczna propagacja błędu w sieciach CNN

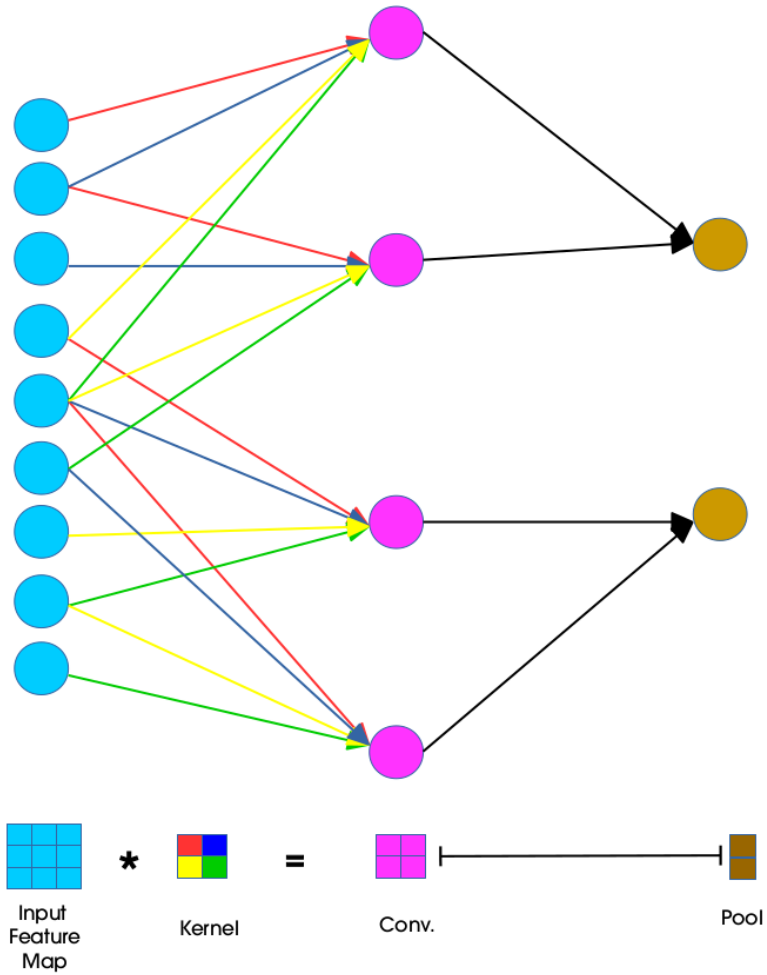
Sygnał propagowany od wejścia do wyjścia (dla jednego kanału $H \times W$ i filtra $k_1 \times k_2$)

$$o_{ij}^h = f(z_{ij}^h)$$

$$z_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} w_{mn}^h o_{i+m,j+n}^{h-1} + b^h$$

oznaczmy $*$ operację splotu

$$o_{ij}^h = f\left((\mathbf{w}^h * \mathbf{o}^{h-1})_{ij} + b^h \right)$$



Źródło
https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

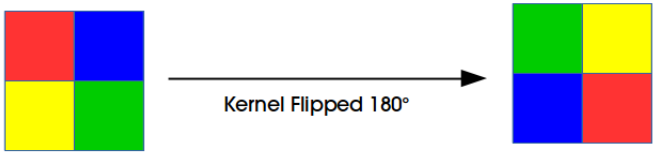
Wsteczna propagacja błędu w sieciach CNN

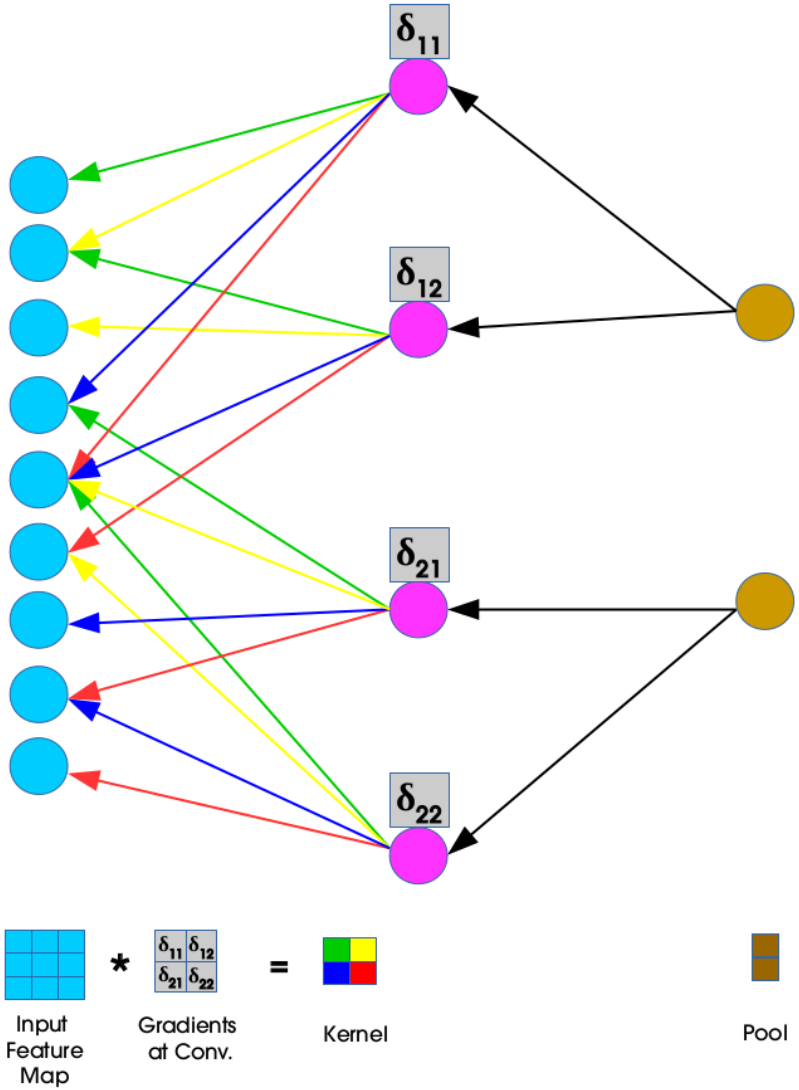
Sygnał błędu neuronów δ_i^k od warstwy wyjściowej do wejścia

$$\delta_{ij}^h = \frac{\partial E}{\partial z_{ij}^h} =$$

$$= f'(z_{ij}^h) \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i-m,j-n}^{h+1} w_{mn}^{h+1}$$

$$= f'(z_{ij}^h) \left(\delta^{h+1} * rot_{180^\circ}(\mathbf{w}^{h+1}) \right)_{ij}$$





Źródło
https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Wsteczna propagacja błędu w sieciach CNN

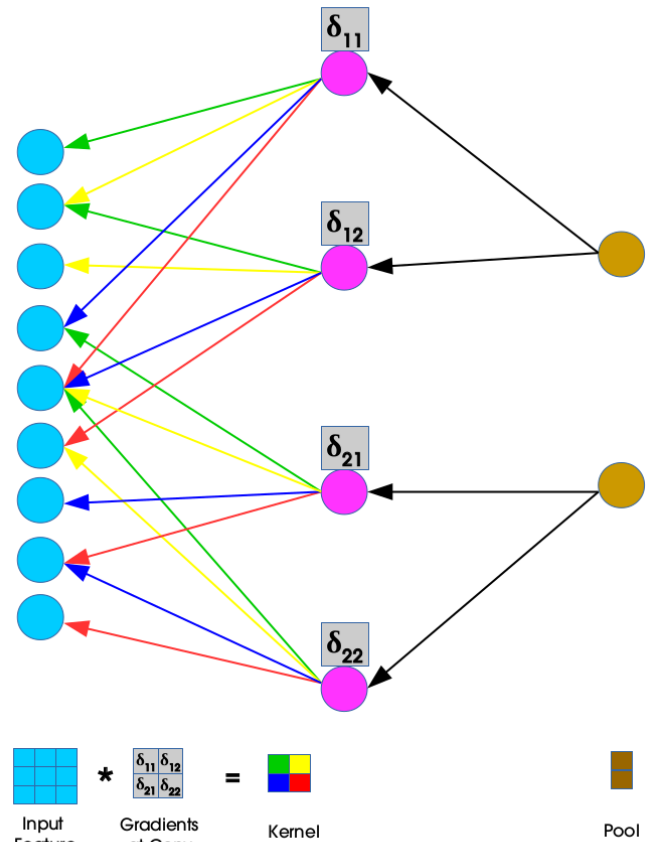
Splot mapy wejściowej o wymiarze $H \times W$ i filtra o wymiarze $k_1 \times k_2$ tworzy mapę wyjściową o wymiarze $(H - k_1 + 1) \times (W - k_2 + 1)$.

Aktualizacja wag w kierunku spadku gradientu:

$$\Delta w_{ij}^h = -\eta \frac{\partial E}{\partial w_{ij}^h} =$$

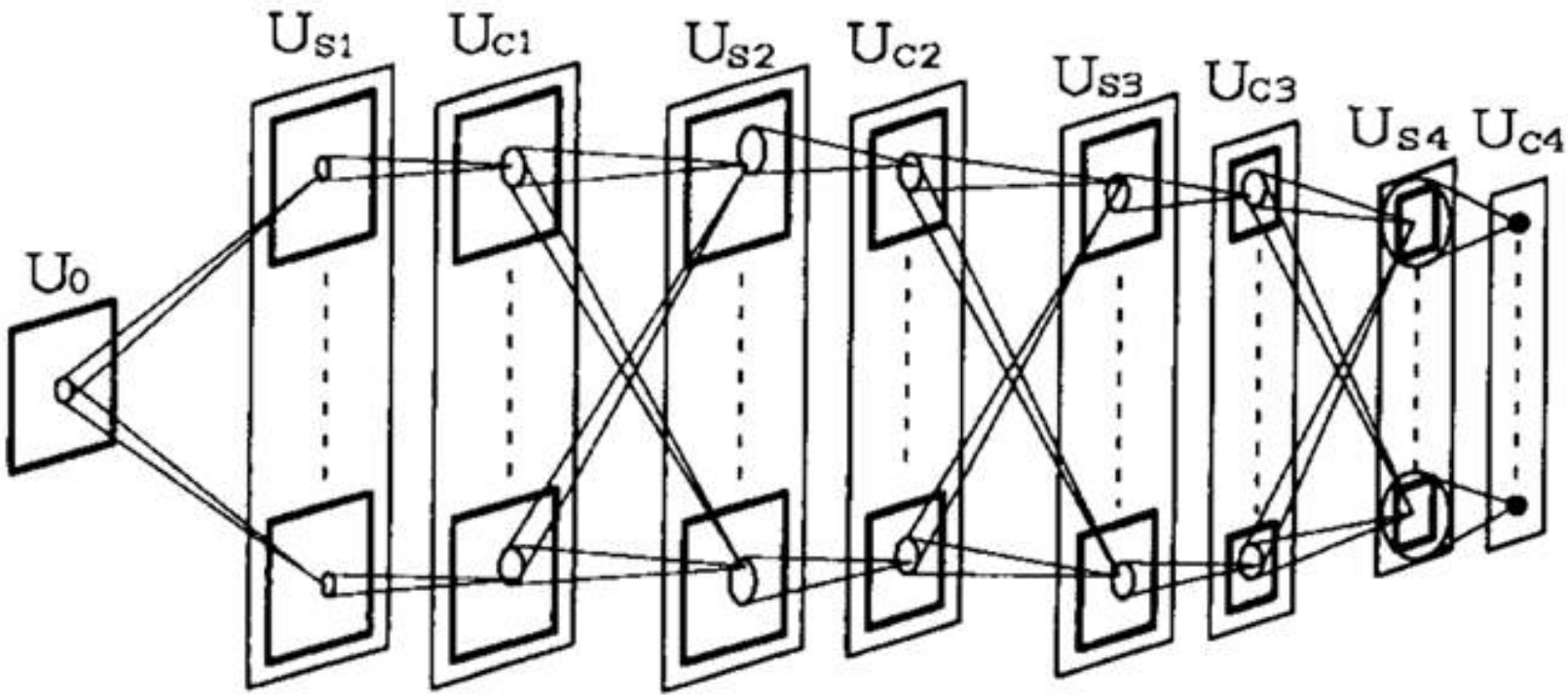
$$= \eta \sum_{m=0}^{H-k_1} \sum_{n=0}^{W-k_2} \delta_{mn}^h o_{m+i,n+j}^{h-1} =$$

$$= \eta (\delta^h * \mathbf{o}^{h-1})_{ij}$$



Źródło
https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Neocognitron (Fukushima, 1980)



- pierwsza sieć CNN do rozpoznawania ręcznie pisanych cyfr
- architektura inspirowana biologicznie ludzkim układem przetwarzania obrazu w mózgu
- uczenie nienadzorowane (bez wstecznej propagacji błędu)

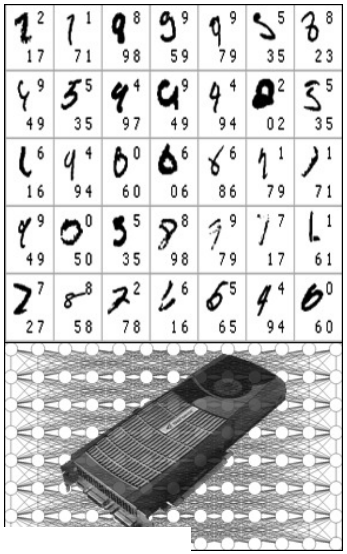
LeNet5 (LeCun 1998)

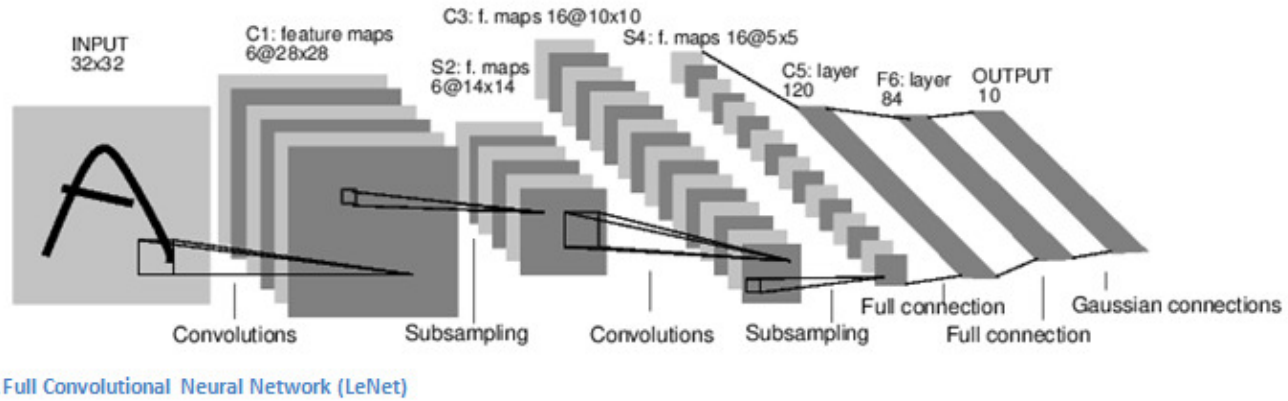
klasyfikacja cyfr pisanych ręcznie
(odczytywanie czeków)

☞ MNIST

trening: 60k cyfr, 250 osób,

test: 10k cyfr



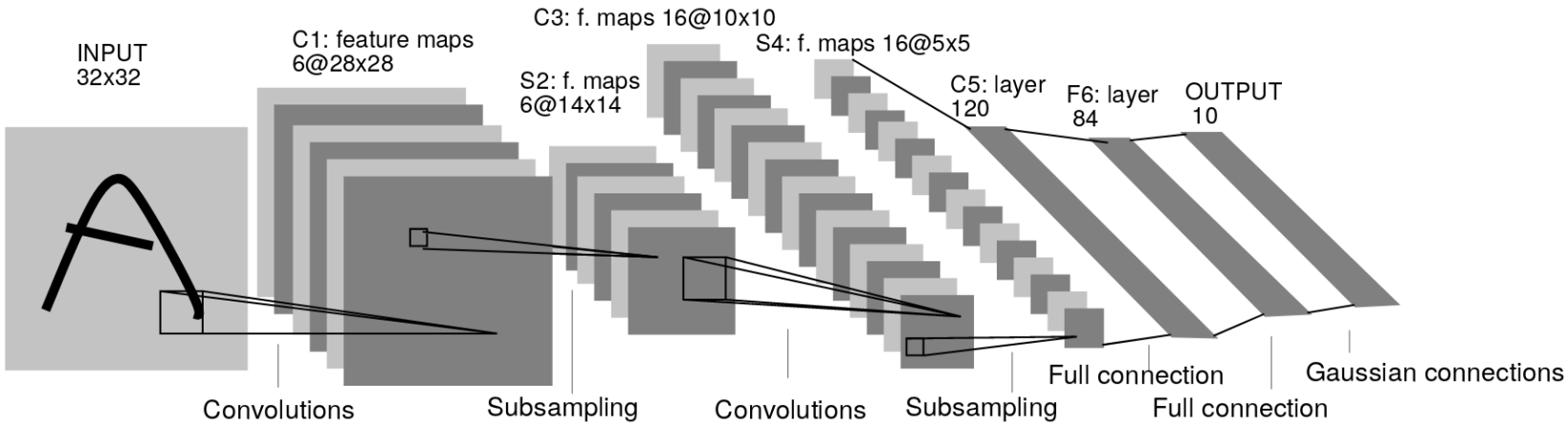


A Full Convolutional Neural Network (LeNet)

☞ LeNet-5, convolutional neural networks

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998

LeNet-5



Layer	kernels	size	output	connections	parameters
Input			1x32x32		
C1	6	5x5	6x28x28	122304	156
S2		2x2	6x14x14	5880	12
C3	16	5x5	16x10x10	151600	1516
S4		2x2	16x5x5	2000	32
C5	120	5x5	120x1x1	48120	48120
F6	84		84x1x1	10164	10164
Output	10		10x1x1		

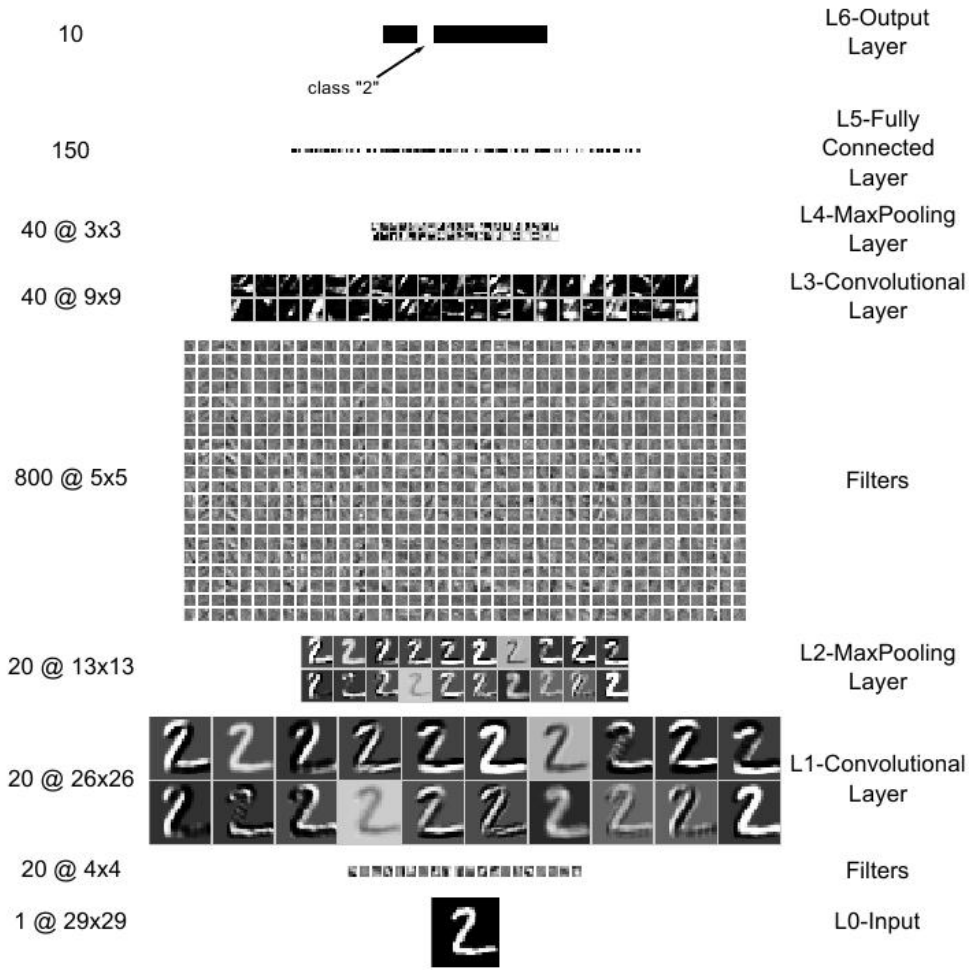
LeNet-5 połączenia S2-C3

Mapy C3 połączone wyłącznie z wybranymi mapami z S2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

- redukcja liczby połączeń
- przełamanie symetrii sieci - różne sygnały wejściowe pozwalają uzyskać różnorodne (być może komplementarne) zestawy detektorów cech

Przykład wizualizacji aktywacji sieci

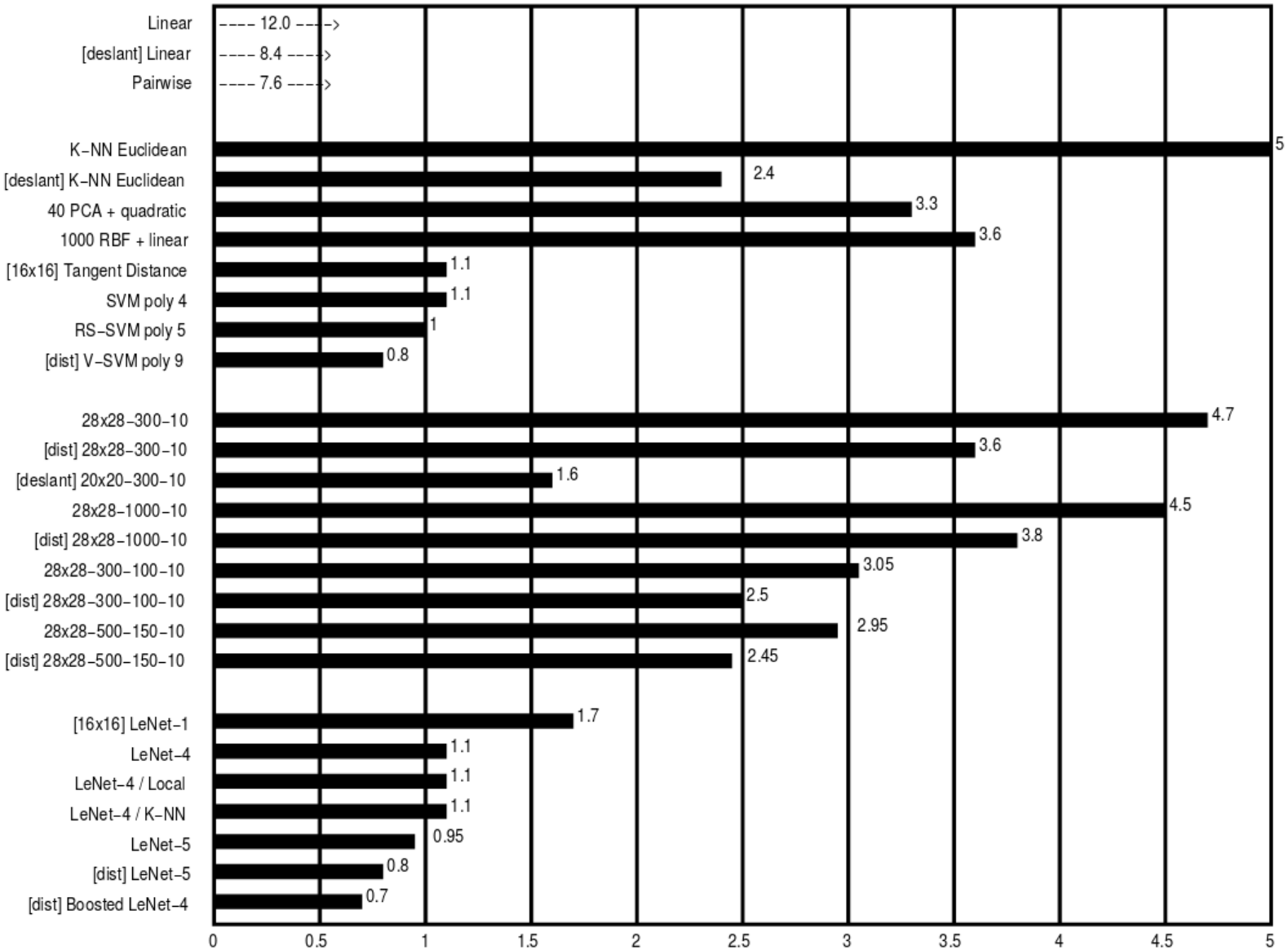


☞ Wizualizacja sieci splotowej

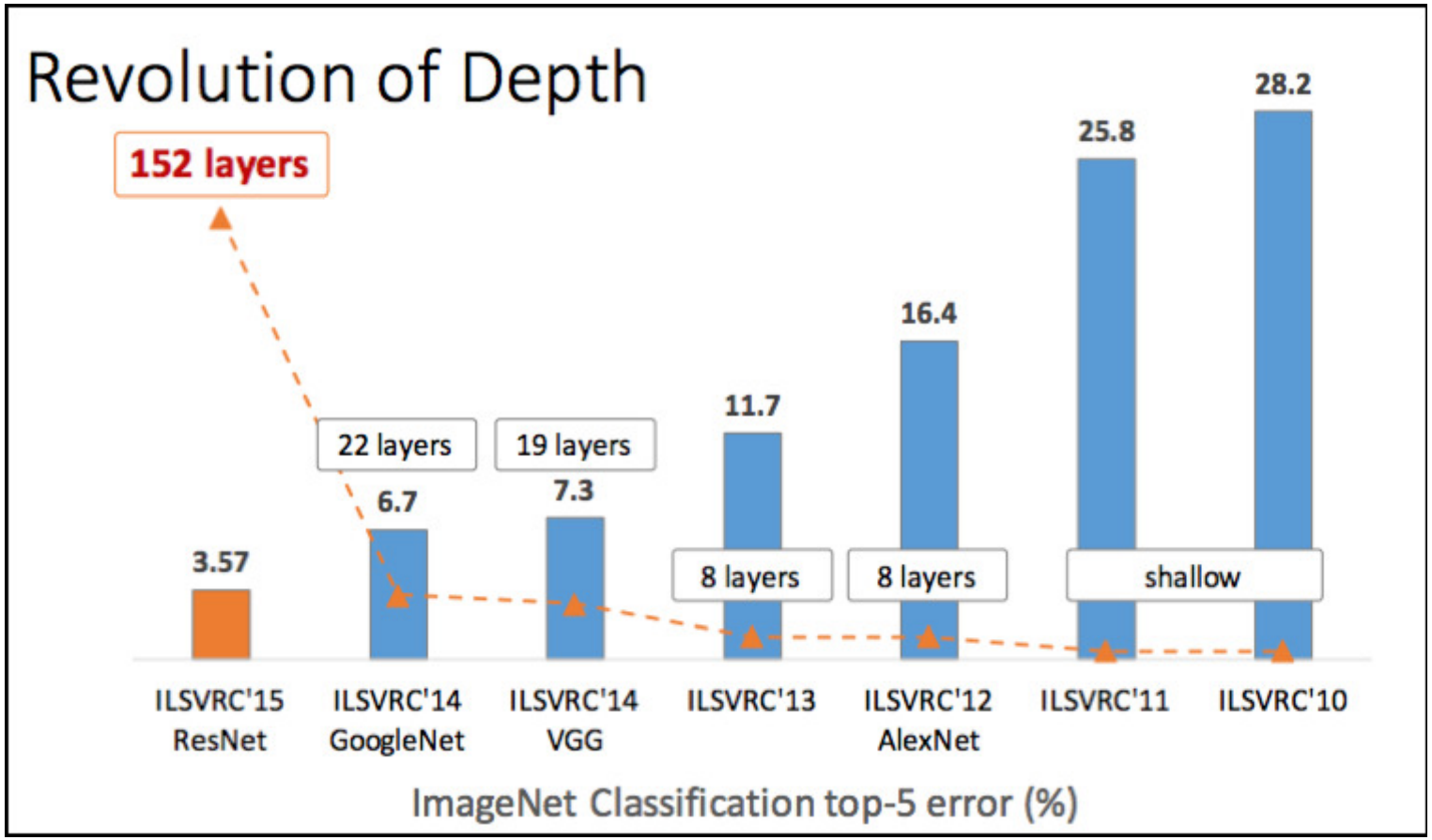
LeNet-5 błędy klasyfikacji



LeNet-5 porównanie z innymi metodami



ImageNet

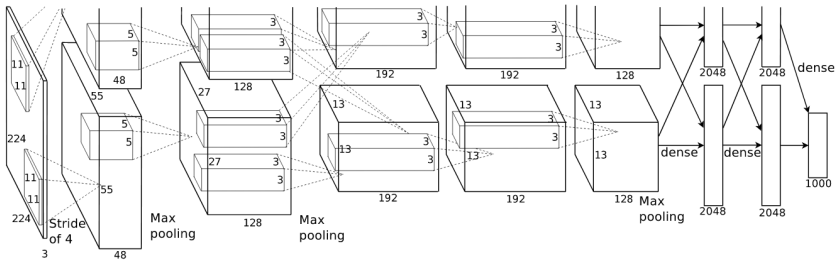


- ☞ CNNs Architectures used on ImageNet

AlexNet (A. Krizhevsky, 2012)

- Zwycięzca ☞ImageNet w 2012, błąd klasyfikacji 15.3% (poprawa z 26%)
- 1.2M obrazów, 1000 klas



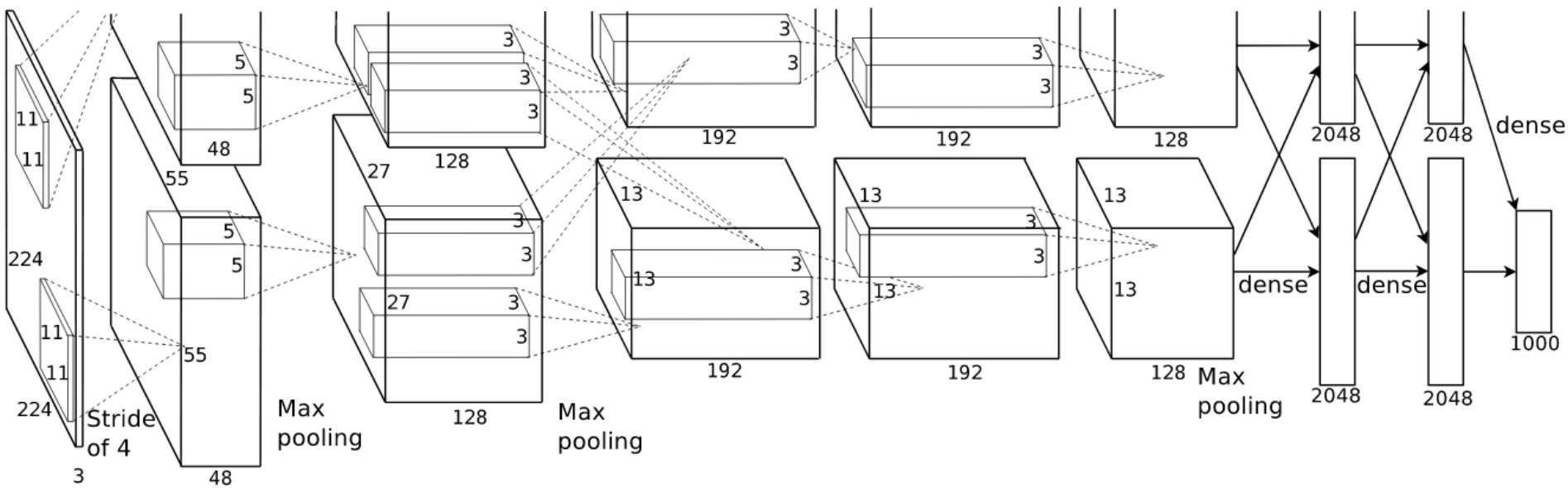


- architektura wzorowana LeNet5 ale głębsza (8 warstw) i więcej filtrów na warstwę, filtry 11x11, 5x5, 3x3
- sekwencje warstw splotowych z jednostkami ReLU
- regularyzacja: dropout, L2, rozszerzanie danych, normalizacja wyjść wybranych warstw
- SGD z momentem, 20 epok, 6 dni treningu (2x NVIDIA GTX 580 3GB GPUs), 60M parametrów

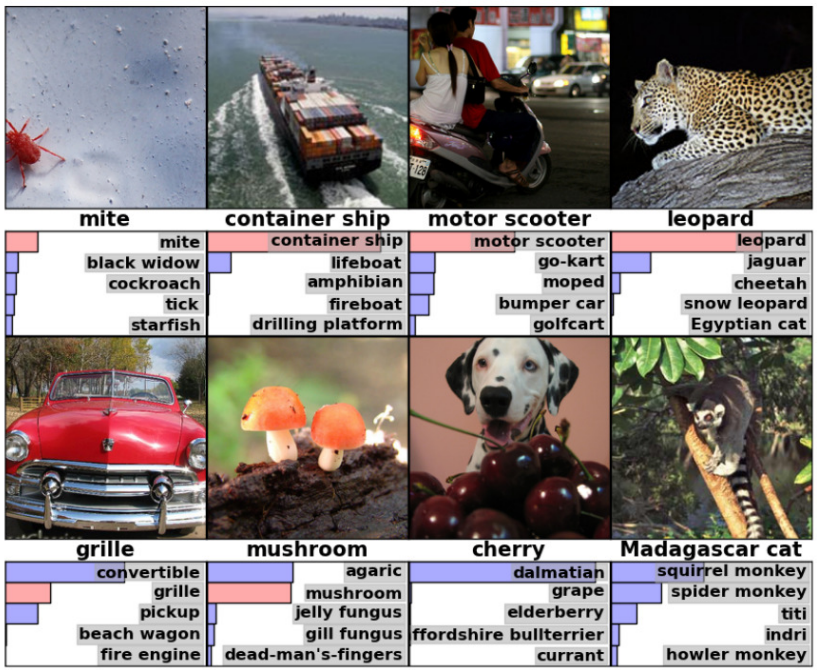
A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2012

AlexNet

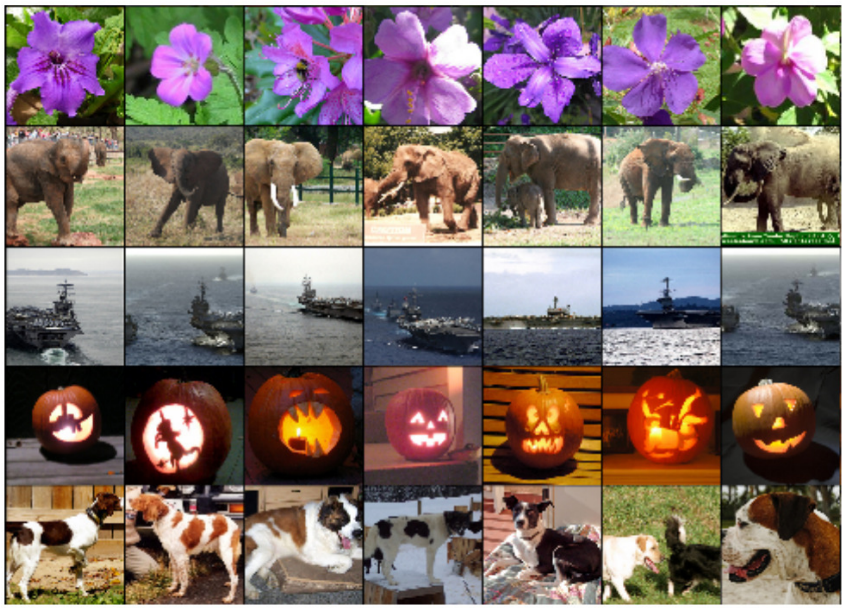
Layer	kernels	size	stride	output
Input				224x224x3
C1 + LRN	96	11x11	4x4	
MaxPooling		3x3	2x2	55x55x96
C2 + LRN	256	5x5		
MaxPooling		3x3	2x2	27x27x256
C3	384	3x3		13x13x384
C4	384	3x3		13x13x384
C5	256	3x3		
MaxPooling		3x3	2x2	13x13x256
F6 + dropout($p = 0.5$)	4096			
F7 + dropout($p = 0.5$)	4096			
Output softmax	1000			



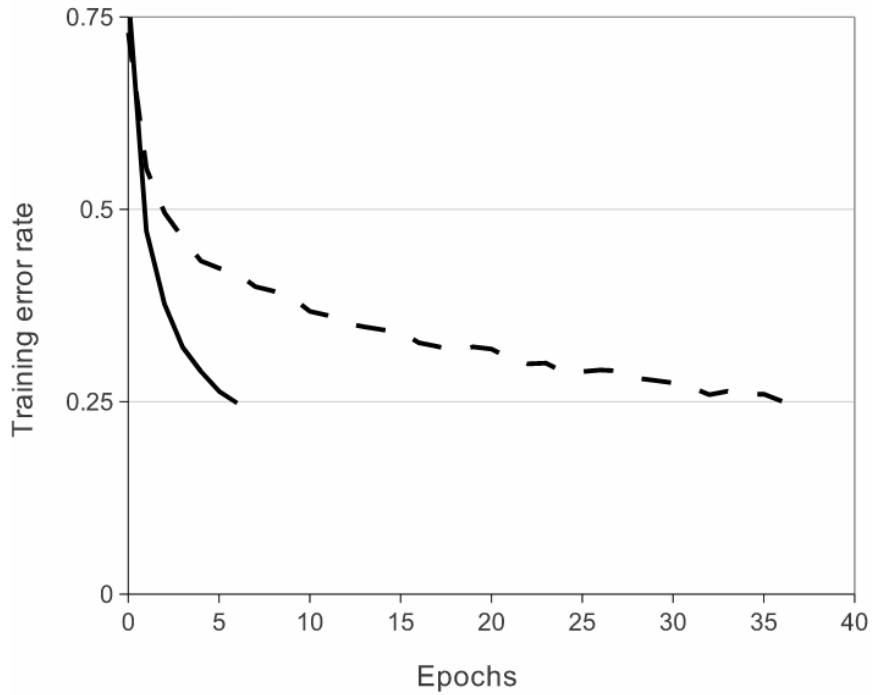
AlexNet



5 najsilniejszych odpowiedzi



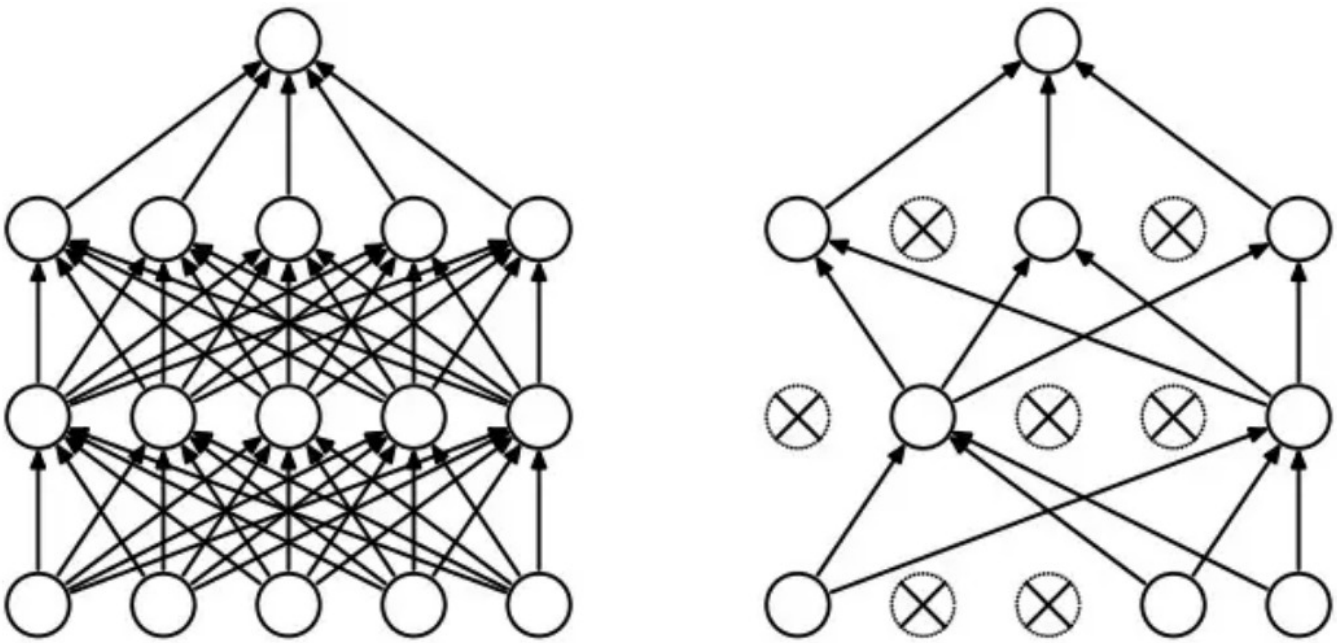
obraz wejściowy i 6 najbliższych wzorców względem odległości Euklidesowej wektora pobudzeń ostatniej warstwy ukrytej



- Porównanie szybkości zbieżności 4 warstwowej sieci splotowej z jednostkami ReLU i tanh na danych ImageNet
- Zastosowanie ReLU do kilkukrotnego przyspieszenia zbieżności
- Inicjalizacja: wagi z rozkładu Gaussa $N(0, 0.01)$, obciążenia (bias) $b = 1$, stąd większość ReLU aktywnych na początku treningu

Dropout

Dropout (Srivastava et al., 2014) dla każdego wzorca treningowego z prawdopodobieństwem p „wyrzuca” jednostki (zeruje ich aktywacje), odrzucone jednostki nie biorą udziału w treningu



(a) Standard Neural Net

(b) After applying dropout.

Rys: ☞ ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks by Koustubh Sinhal

Dropout

- Silnie zapobiega przeuczeniu w warstwach w pełni połączonych
- Używany wyłącznie w czasie treningu. Podczas predykcji (*inference*) dropout jest wyłączany, czyli $p = 0$
- Każdy krok uczenia odbywa się ze zmienioną losowo architekturą sieci, uczenie przebiega innymi ścieżkami grafu połączeń
- Dla n jednostek mamy 2^n możliwych architektur, trening w mini-batchu uśrednia gradient względem różnych, wylosowanych sieci
- Przeciwdziała powstawaniu złożonych relacji pomiędzy wieloma neuronami, stąd pojedynczy neuron jest zmuszony wykrywać bardziej wartościowe cechy, niezależne od wpływu innych neuronów
- Wolniejsza zbieżność uczenia (AlexNet 2x wolniej z dropoutem)

Local response normalization

Normalizacja wartości $a^i_{x,y}$ pikseli w pozycji x, y i kanału (mapy cech) i wejściowych do warstwy splotowej

$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} \left(a^j_{x,y}\right)^2 \right)^{\beta}$$

- $k = 2, n = 5, \alpha = 10^{-4}, \beta = 0.75$ dobrane heurystycznie ze zbioru walidacyjnego
- realizuje normalizację jasności dla n sąsiadujących kolejnych map cech
- w AlexNet poprawa o 1.4% (top 1) oraz 1.2% (top 5)

Dodatkowa regularyzacja

Sztuczne rozszerzenie zbioru danych (*Data augmentation*)

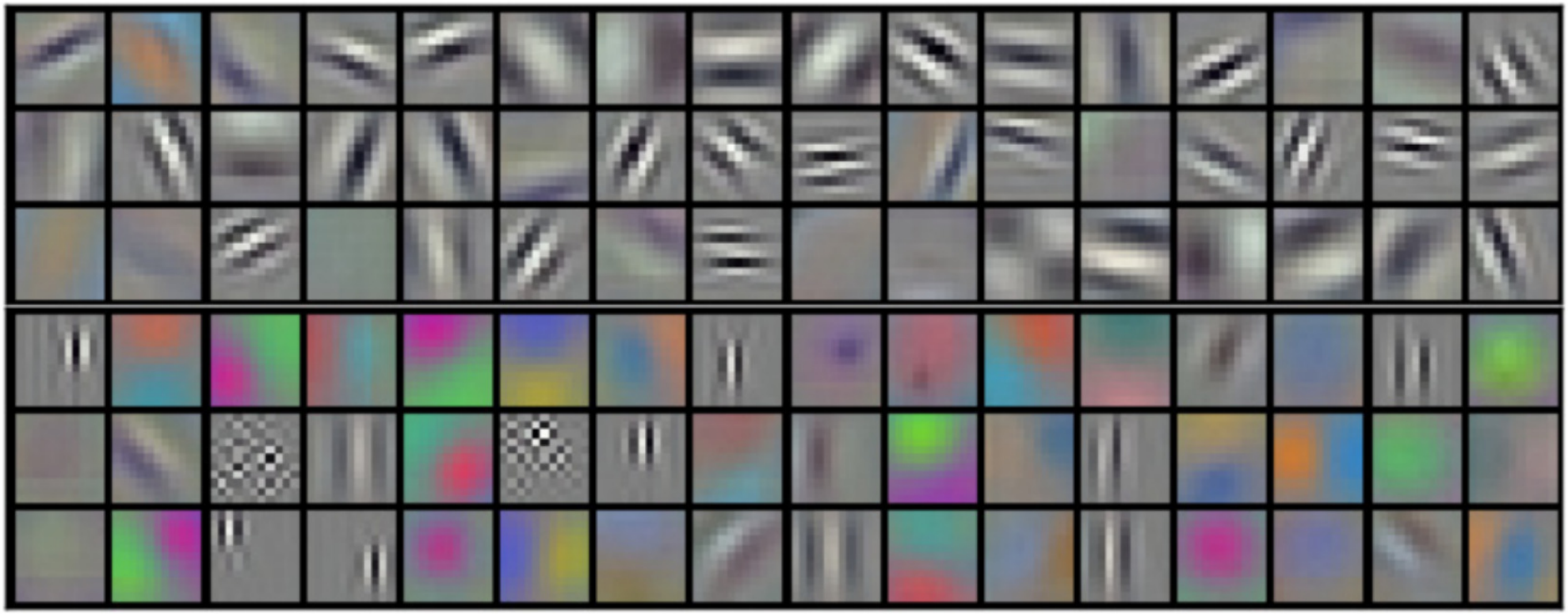
- przesunięcie i odbicia: losowanie obrazków 224x224 z 256x256, zwiększenie zbioru 2048 razy niezbędne do uniknięcia przeuczenia przy tak dużej sieci
- losowa modyfikacja intensywności kanałów RGB
- w AlexNet poprawa ponad 1% (top 1, top 5)

MaxPooling z nakrywaniem

- kernel 3x3, stride 2x2
- poprawa 0.4% (top 1) i 0.3% (top 5) względem architektury bez nakrywania (kernel 2x2)
- obserwacja: AlexNet rzadziej ulegał przeuczeniu

Zastosowanie 2 jednostek GPU pozwoliło na szkolenie sieci o większej liczbie filtrów co zaowocowało poprawą 1.7% (top-1) oraz 1.2% (top-5) w porównaniu z mniejszą siecią uczoną na pojedynczym GPU

Filtry w pierwszej warstwie

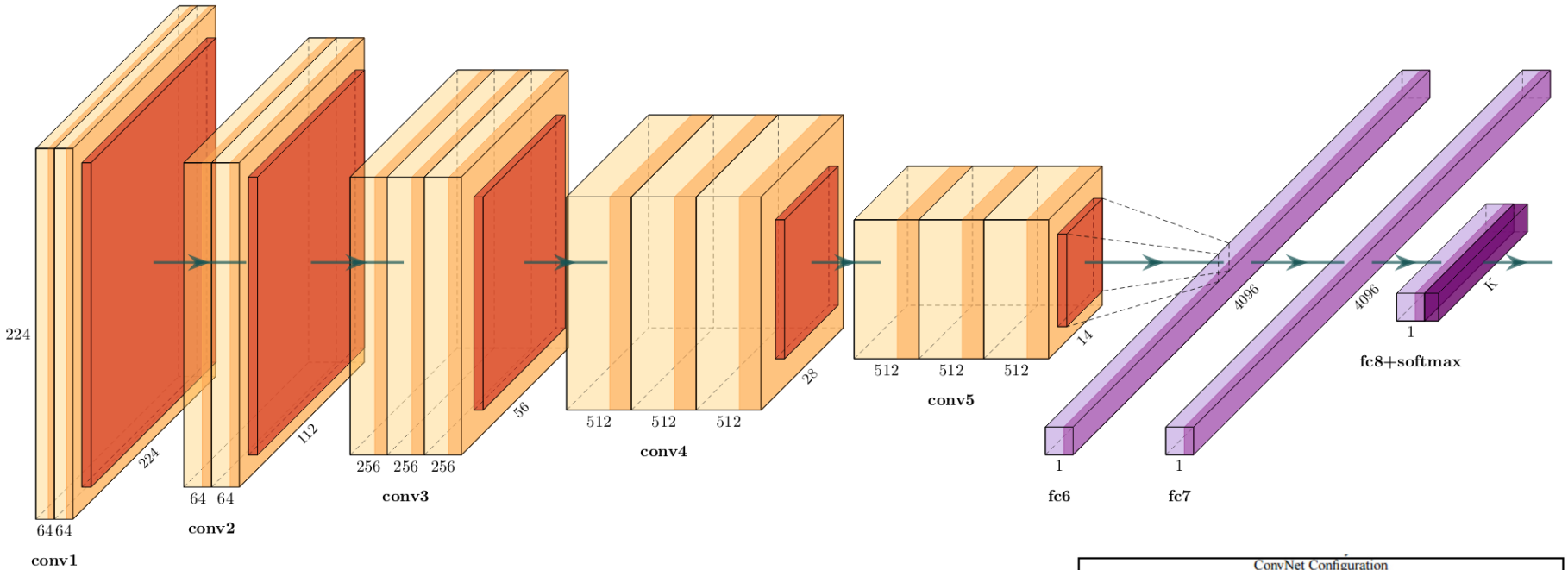


96 filtrów pierwszej warstwy splotowej, osobne kolumny warstw splotowych uczą się dwóch rodzin cech niskiego poziomu (obserwowane przy każdym treningu)

- filtry kolorów
- filtry krawędzi

VGGNet (Simonyan and Zisserman, 2014)

Visual Geometry Group, University of Oxford



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- bloki VGG: sekwencja warstw splotowych z *padding same* + max pooling
- wielkość map zmniejszana o połowę po każdym bloku
- liczba filtrów zwiększana dwukrotnie po każdym bloku
- małe filtry (3×3), gęsty splot (stride=1)

VGGNet

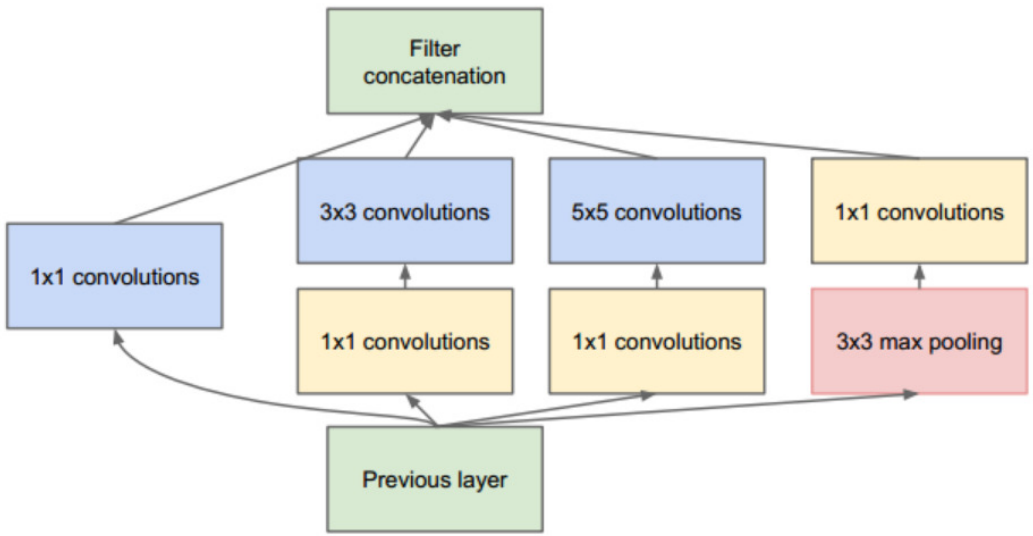
- zamiast dużych filtrów (w AlexNet 11x11, 7x7, 5x5) VGG stosuje we wszystkich warstwach filtry 3x3 zwiększając ich efektywne pola recepcyjne sekwencjami warstw splotowych
- małe filtry 3x3 zdolne do wykrycia bardziej subtelnych relacji w obrazach
- zwiększenie głębokości pozwala trenować bardziej złożone cechy
- stride=1, brak utraty informacji, gęsta konwolucja
- kilka architektur o różnej głębokości od 11 do 19 warstw
- sieć VGG16 138M parametrów, VGG19 143M parametrów
- trening 2-3 tygodnie (4x GPU), duże wymagania pamięciowe i obliczeniowe

K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv technical report, 2014

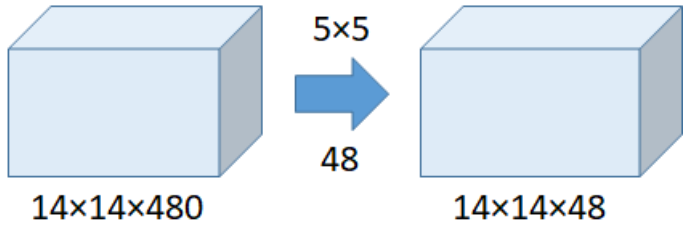
GoogLeNet/Inception

Sieć złożona z wielu bloków incepcyjnych (*inception module*) skonstruowanych według schematu:

- rozmiar filtru odwrotnie proporcjonalny do liczby filtrów, np. 128 filtrów 3x3 i 32 filtry 5x5
- równoległe przetwarzanie w kilku warstwach splotowych o zróżnicowanej wielkości filtrów 1x1, 3x3, 5x5 - detektory cech w różnej skali
- duże filtry poprzedzone splotem 1x1 w celu redukcji liczby parametrów. Splot 1x1 redukuje liczbę kanałów do zadanej liczby f

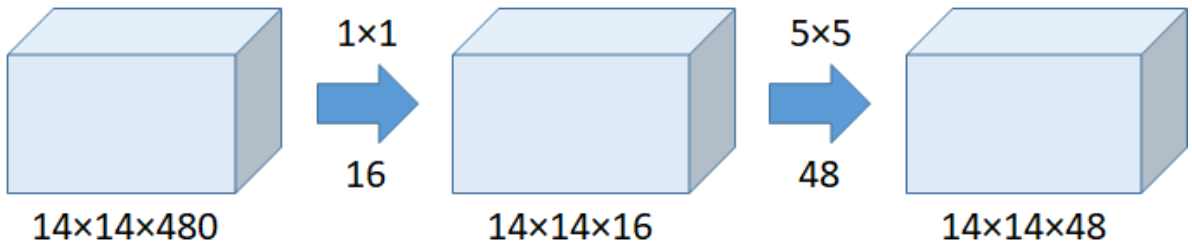


Redukcja złożoności splotem 1x1



ilość parametrów $5 \times 5 \times 48 \times 480 = 576K$

ilość operacji $(14 \times 14 \times 48) \times (5 \times 5 \times 480) = 112.9\text{M}$



ilość parametrów $1 \times 1 \times 16 \times 480 + 5 \times 5 \times 48 \times 16 = 26.88K$

ilość operacji

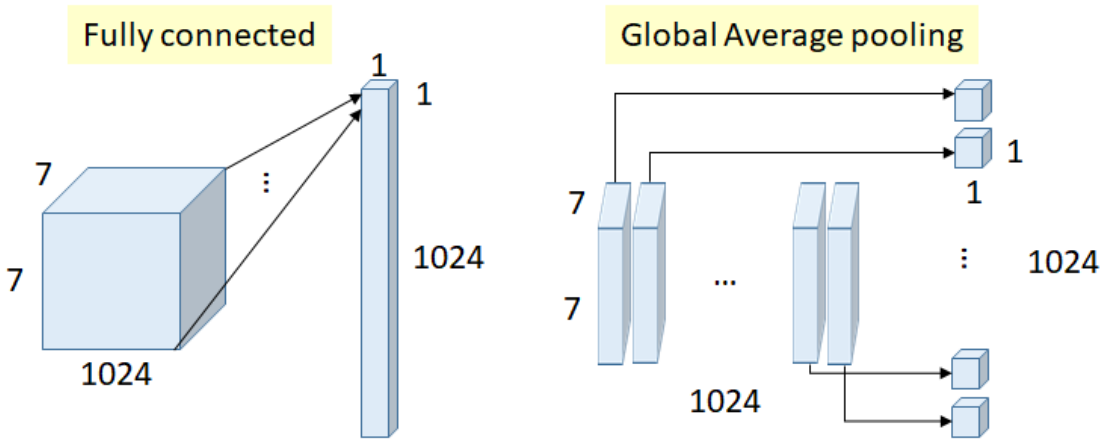
$(14 \times 14 \times 16) \times (1 \times 1 \times 480) + (14 \times 14 \times 48) \times (5 \times 5 \times 16) =$

$1.5\text{M} + 3.8\text{M} = 5.3\text{M}$

Rys: ☞ Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification) by Sik-Ho Tsang

Global average pooling

Globalne uśrednienie (*global average pooling*) uśrednia wartości przestrzenne dla poszczególnych kanałów, np. dla mapy cech 7x7x1024 tworzy wektor 1024 elementowy, który jest bezpośrednio podawany do warstwy softmax

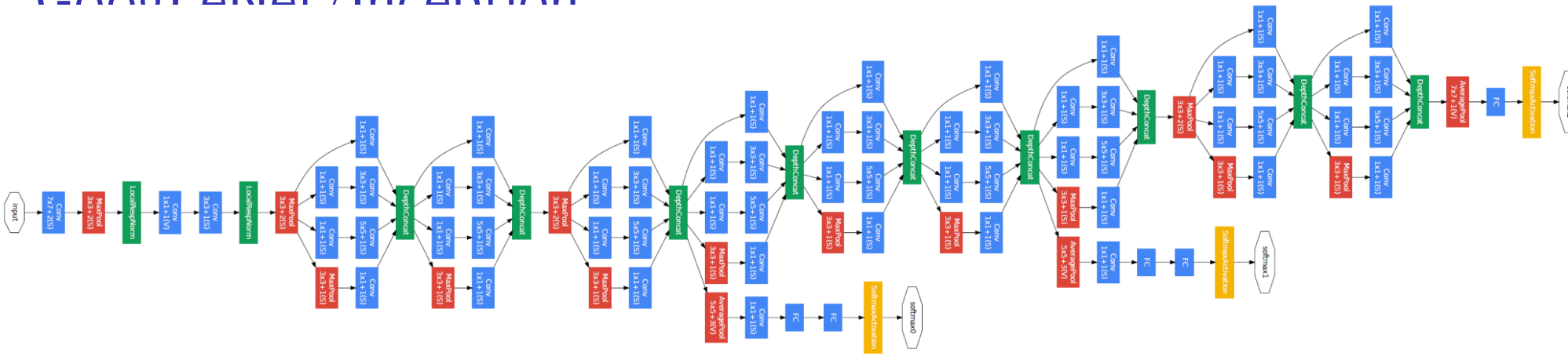


$7 \times 7 \times 1024 \times 1024 = 51.3\text{M}$

- Warstwy w pełni połączone zawierają najwięcej parametrów (w AlexNet 90% parametrów) w sieci
- global average pooling zastępuje warstwy w pełni połączone, redukując liczbę parametrów i zapobiegając przeuczeniu
- w GoogleNet poprawiło to wynik top-1 o 0.6%

Rys: ☞ Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification) by Sik-Ho Tsang

GoogLeNet /Inception



- 22 warstwy
- warstwy przewężające (bottleneck) dodatkowo redukują złożoność sieci
- kilka wyjść softmax na różnych poziomach sieci, stabilizują trening i przyspieszają zbieżność, zapobiegają zanikaniu gradientu. Całkowita funkcja kosztu:

$$J(\theta) = \mathbb{E}_D[-\mathbf{t} \log \mathbf{y}_1 - \mathbf{t} \log \mathbf{y}_2 - \mathbf{t} \log \mathbf{y}_3]$$

- poprawność 93.3% top-5 na ImageNet, dużo szybszy w treningu od VGG
- kolejne iteracje: Inception v2, v3, InceptionResNet, Xception

ResNet (Residual Neural Network

Bloki ze skrótami (residual module)

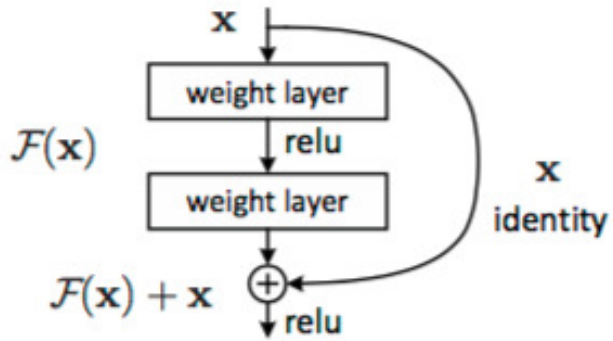
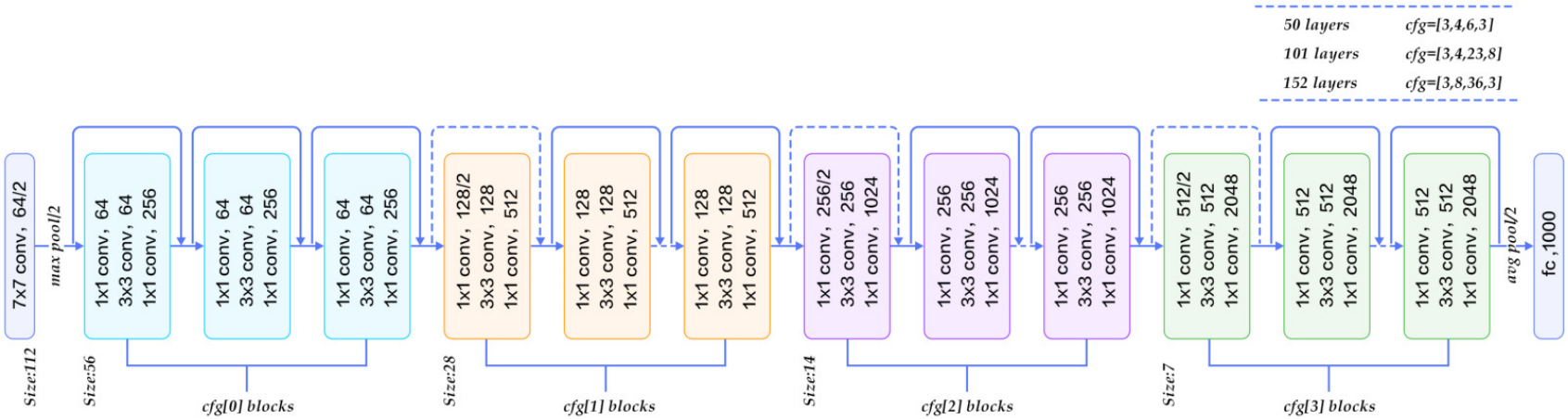


Figure 2. Residual learning: a building block.

- „skróty" łączą wejścia bloku z wyjściem poprzez odwzorowanie jednostkowe
- przejścia „skrótowe" pomagają uczyć bardzo głębokie sieci (152 warstwy)
- zapobiegają zanikaniu gradientu (sygnał może być propagowany skrótami)

ResNet



- wiele bloków zawierających sploty 1x1, 3x3, 1x1
- regularyzacja: batch normalization

Batch normalization

Batch normalization (Ioffe, 2015) normalizuje wejścia $x_i^{(k)}$ warstwy względem wartości średniej $\mu^{(k)}$ i wariancji $\sigma^{(k)}$ wzdłuż wymiaru k dla paczki danych w danej iteracji

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}$$

gdzie γ i β podlegają adaptacji w czasie treningu (uczenie normalizacji).
Czynnik normalizujący każde wejście

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}}$$

średnia i odchylenie dla minipakietu o rozmiarze m

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i, \qquad \sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2$$

Ioffe, Sergey; Szegedy, Christian (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift".

Batch normalization

- **przyspieszenie zbieżności**, poprawia stabilność treningu, zmniejsza liczbę epok potrzebnych do osiągnięcia dobrych wyników
- **redukuje problem wewnętrznego przesunięcia kowariancji** - gdy zmienia się wyjście warstwy poprzedniej wówczas warstwa następna musi dopasować się do nowego rozkładu danych
- można stosować **wyższe współczynniki uczenia** bez obawy znikającego lub eksplodującego gradientu
- **zmniejsza zależność od inicjalizacji wag**, zwiększa odporność na złe ustawienia początkowe
- **poprawa generalizacji**: modele często lepiej generalizują na danych testowych
- wygładza powierzchnię błędu (Santurkar, 2018)
- **uniwersalność**: można stosować w różnych typach sieci

Batch normalization w ResNet

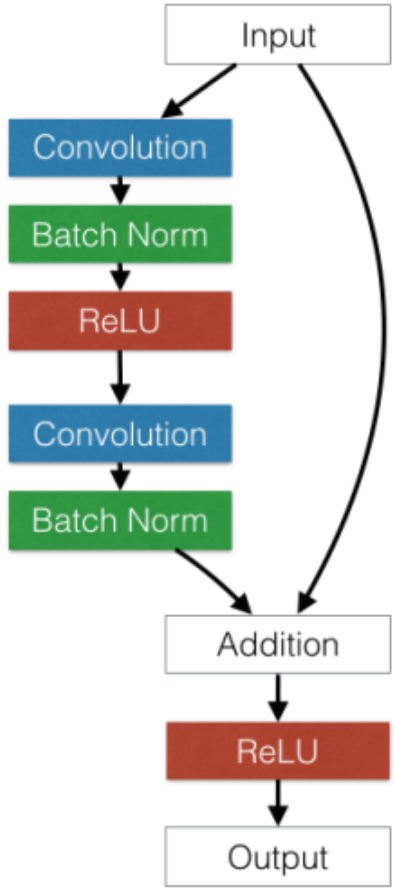
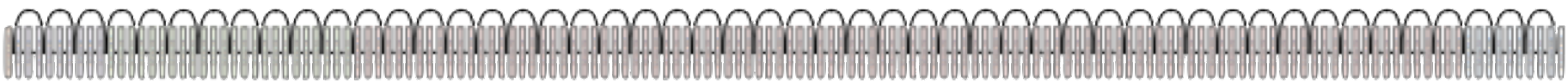


Figure 1. A RestNet basic block

HighNets: Highway networks

Sieci rezydualne, które starają się balansować wpływ głównej ścieżki (primary pathway) oraz ścieżki skrótowej (skip pathway)

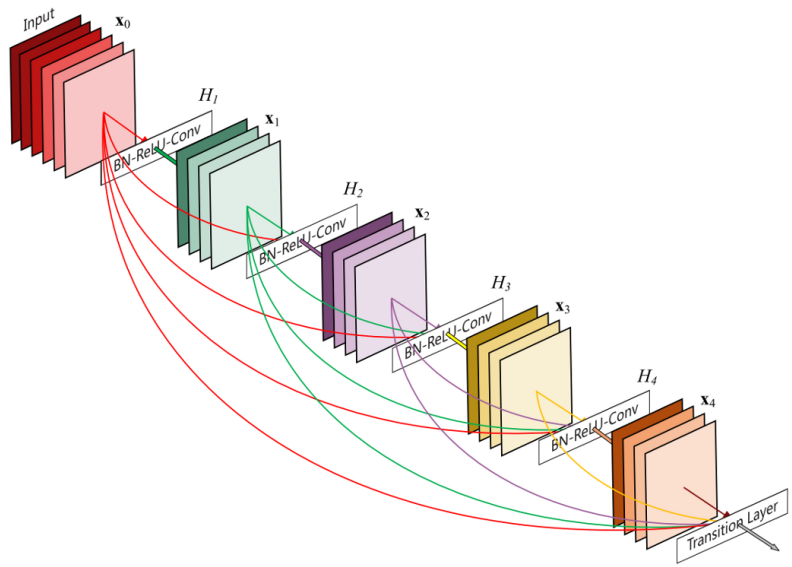
$$\mathbf{h}_n = T_{W'} f_W(h_{n-1}) + (\mathbf{1} - T_{W'}) h_{n-1}$$



- nawet do 1000 warstw
- wyniki state-of-the-art na MNIST i CIFAR 2015

DenseNets: Dense networks

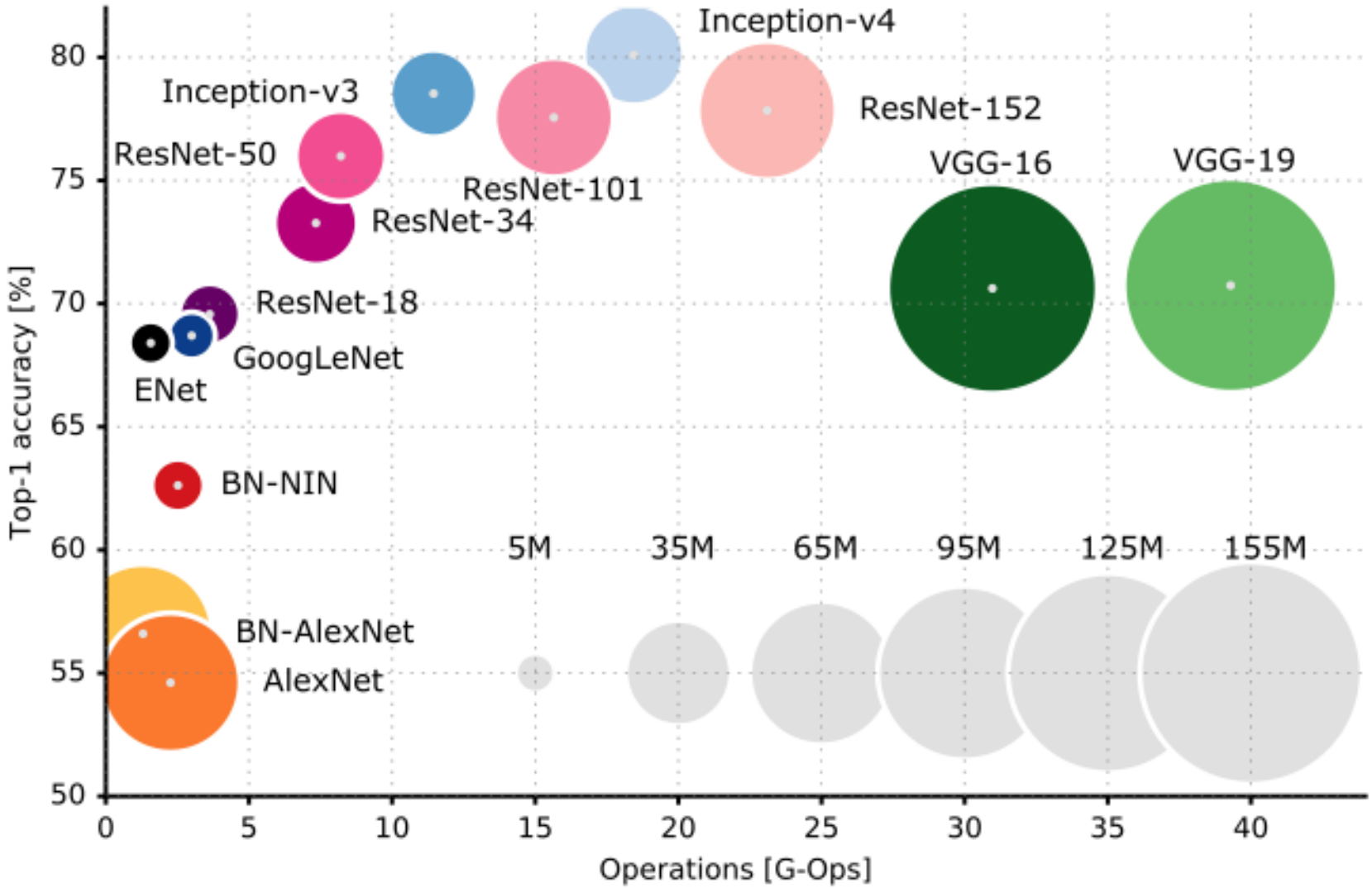
Sieci rezydualne, które posiadają połączenia skrótowe między wszystkimi blokami warstw sieci (do 5 bloków)



- do 100 warstw
- wyniki state-of-the-art na wielu danych benchmarkowych 2018

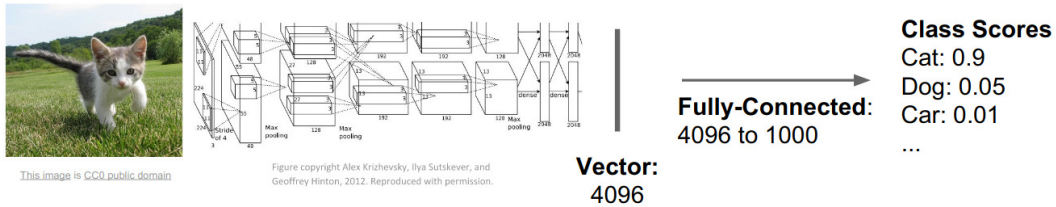
Porównanie popularnych sieci CNN

Poprawność klasyfikacji (top-1) zbioru ImageNet, liczba parametrów (zużycie RAM), szybkość działania (flops)

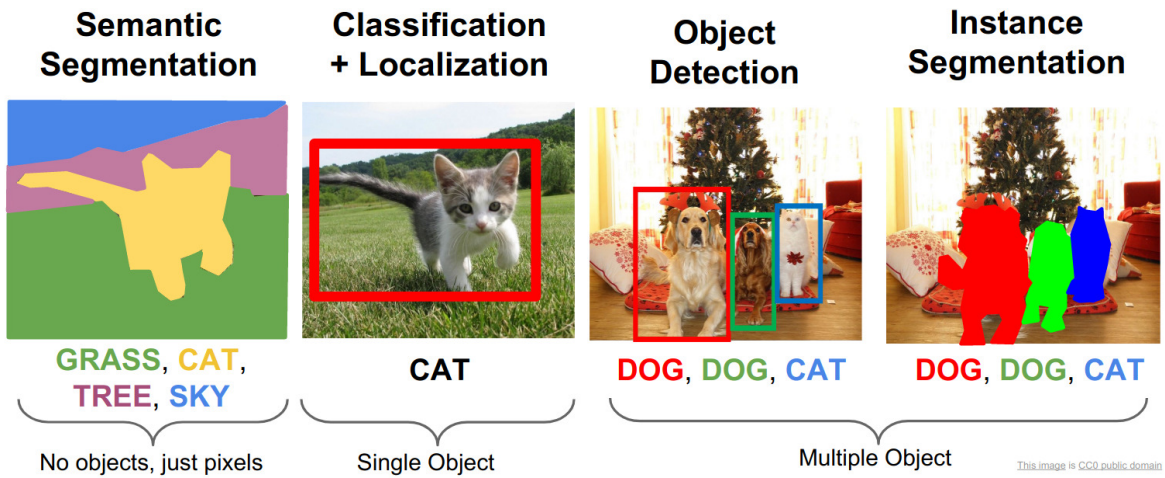


Analiza obrazów

- rozpoznawanie obiektów (klasyfikacja obrazów)



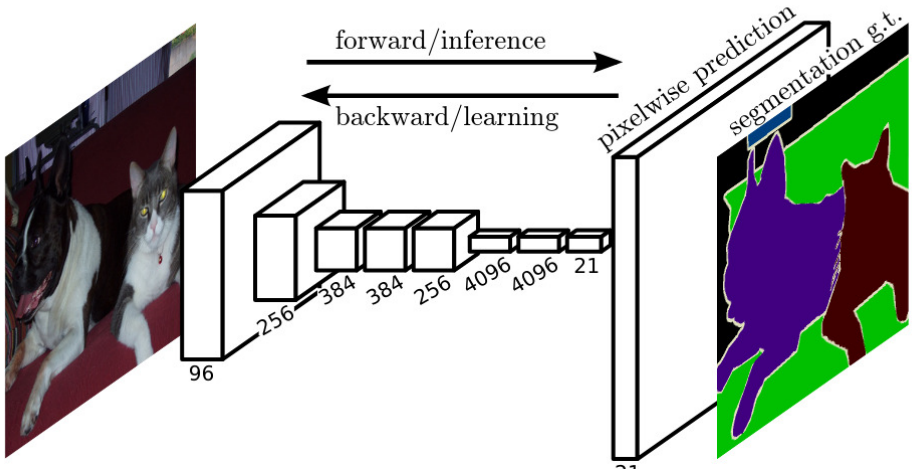
- detekcja i lokalizacja obiektów - określenie położenia obiektów na obrazie
- segmentacja semantyczna (klasyfikacja pikseli) i segmentacja obiektów



Rys: Fei-Fei Li, et al., "Detection and Segmentation", lecture

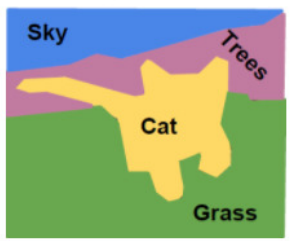
Fully Convolutional Network (FCN)

Segmentacja semantyczna - predykcja obiektów na poziomie pikseli. Wyjście sieci jest rozmiaru WxHxK, gdzie WxH - rozmiar obrazu wejściowego, K - ilość obiektów do segmentacji (tło, kot, pies, itd...)





Input Image

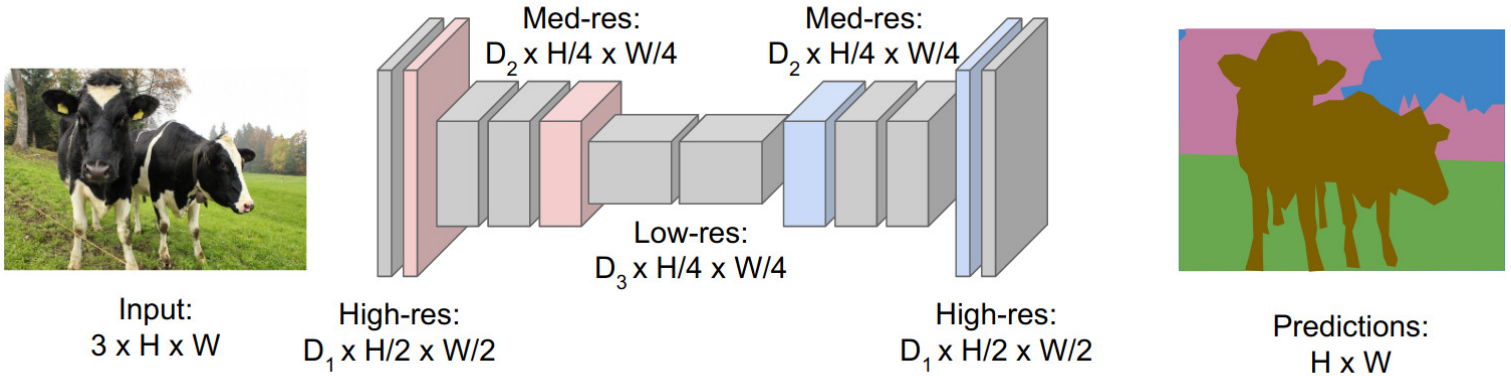


Expected Output (Source – Ref. 3)

Backbone network (kręgosłup) - wykorzystanie sieci do klasyfikacji obrazów (AlexNet, Googlenet, ResNet, VGG) jako część składowa FCN. Kręgosłup może być wstępnie wytrenowany na innym problemie analizy obrazów (np. ImageNet).

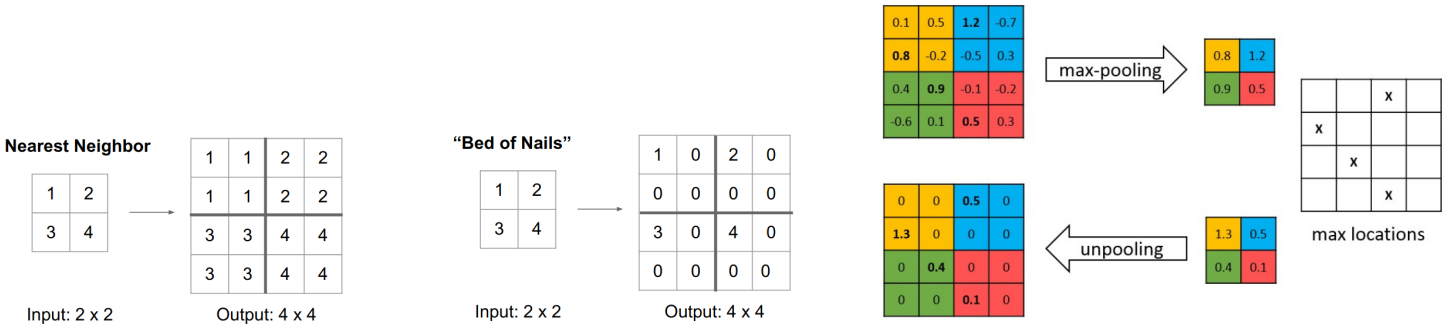
FCN: Downsampling and upsampling

- głębokie przetwarzanie obrazu o dużej rozdzielczości bardzo kosztowne
- FCN używa upsamplingu do odtworzenia oryginalnego rozmiaru obrazu

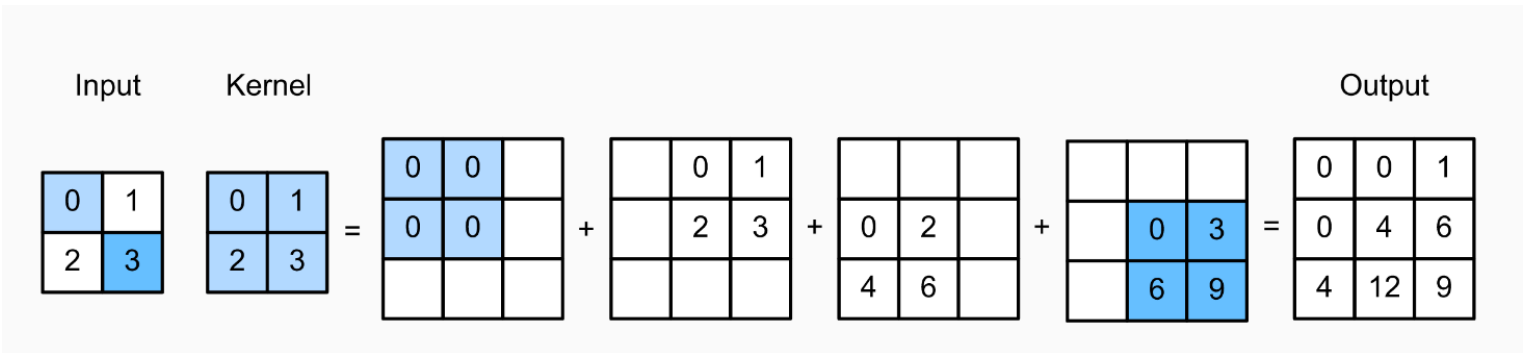


Unpooling i dekonwolucja

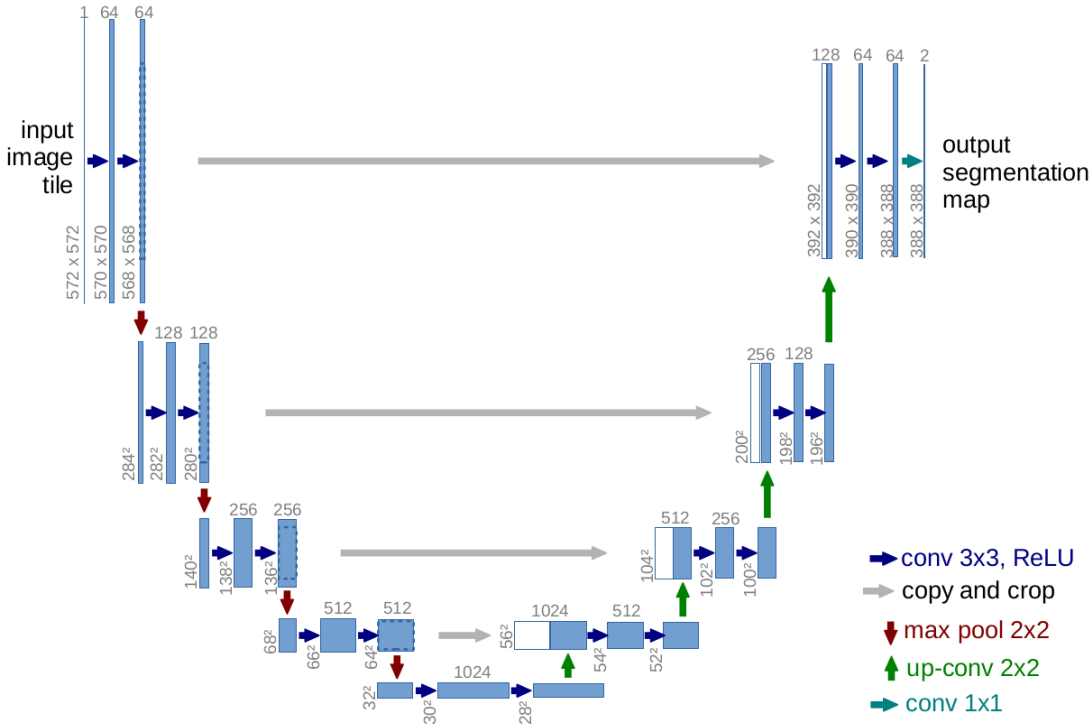
- unpooling



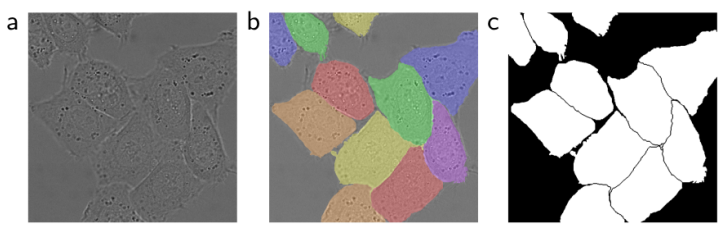
- deconvolution (transpozycja splotu)



U-Net (2015)



Przykład: segmentacja obrazów medycznych



☞ *O. Ronneberger, P. Fischer, . Brox U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015*

RNN

Sieci rekurencyjne i modelowanie sekwencji

Modelowanie sekwencji

Sieć jednokierunkowa realizuje odwzorowanie jeden do jednego

$$F(\mathbf{x}_i; \theta) = \mathbf{y}_i$$

Przy prezentacji sekwencji $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$ na wejście MLP każda ramka $\mathbf{x}_i$ jest niezależna od innych ramek, więc sieć nie może modelować zależności czasowych.

$$\begin{aligned} F(\mathbf{x}_0; \theta) &= \mathbf{y}_0 \\ F(\mathbf{x}_1; \theta) &= \mathbf{y}_1 \\ &\ldots \\ F(\mathbf{x}_t; \theta) &= \mathbf{y}_t \end{aligned}$$

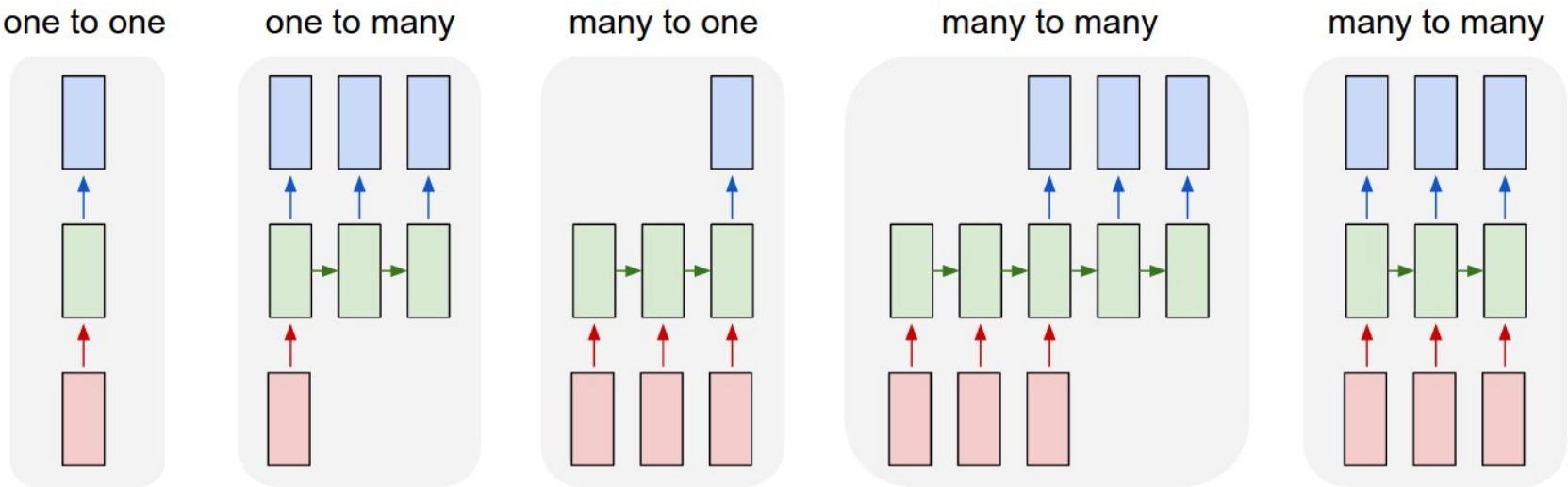
Modelując szeregi czasowe chcemy uzyskać odpowiedź na podstawie sekwencji historycznych wartości

$$F(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t; \theta) = \mathbf{y}$$

Problemy modelowania sekwencji

- Dane sekwencyjne lub zależne od czasu: sygnał audio, tekst, ruch obiektów, EEG, ...
- Modelowanie sekwencji: przewidywanie kolejnych wartości szeregu (np. przewidywanie pogody, notowań giełdowych), generowanie tekstu (chatboty), generowanie muzyki, sterowanie ruchem robota, ...
- Przetwarzanie sekwencji: maszynowe tłumaczenie, rozpoznawanie mowy, rozpoznawanie pisma, przetwarzanie strumieni wideo, ...
- Klasyfikacja sygnałów czasowych, etykietowanie obrazów, wykrywanie wzorców w sekwencjach

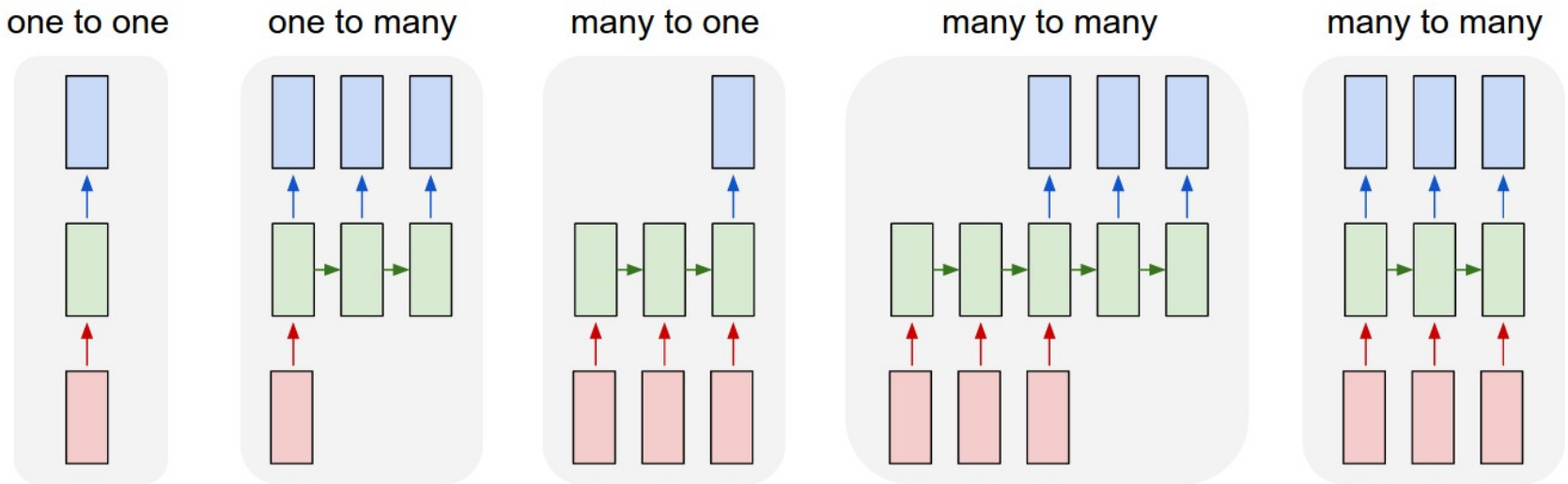
Modelowanie sekwencji



- *one-to-one*: pojedynczy sygnał wejściowy produkuje pojedynczy sygnał wyjściowy, brak modelowania relacji zależnych od czasu, np. klasyfikacja obrazów
- *one-to-many*: pojedyncze wejście generuje sekwencję na wyjściu, np. tworzenie opisów obrazów, generowanie muzyki, generowanie tekstu
- *many-to-one*: sekwencja wejściowa produkuje pojedynczy sygnał wyjściowy, np. klasyfikacja dokumentów, analiza sentymentu

Rys: ☞ Andrej Karpathy, Hacker's guide to Neural Networks

Modelowanie sekwencji



many-to-many: wejście i wyjście sekwencyjne

- bez dopasowania ramek, długość sekwencji wejściowej jest różna od długości sekwencji wyjściowej, np. tłumaczenie maszynowe, transkrypcja mowy
- z dopasowaniem ramek (*alignment*), np: etykietowanie ramek wideo, klasyfikacja fonemów w sygnale mowy (model akustyczny)

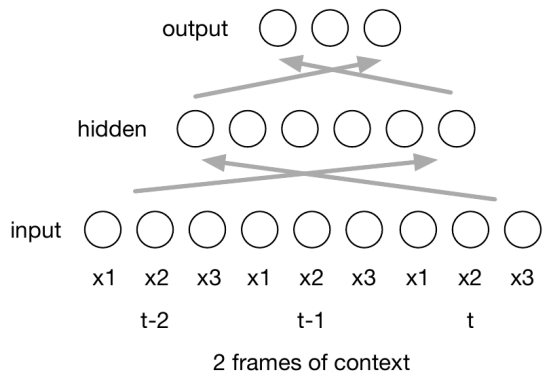
Rys: ☞ Andrej Karpathy, Hacker's guide to Neural Networks

Modelowanie sekwencji sieciami jednokierunkowymi

Naiwne podejście stosowane w MLP: dodanie **stałego w czasie kontekstu** do aktualnego wejścia $\mathbf{x}_i$

Przykład: po 2 ramki kontekstu z lewej i prawej (poprzednie i przyszłe wektory wejściowe)

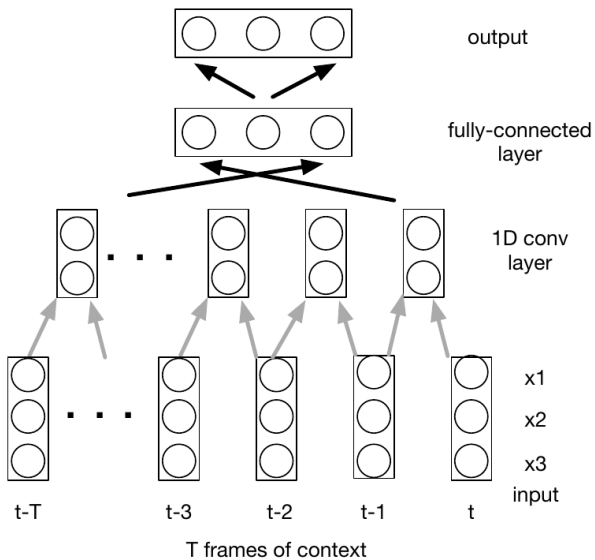
$$\hat{\mathbf{x}}^t \leftarrow [\mathbf{x}^{t-2}, \mathbf{x}^{t-1}, \mathbf{x}^t, \mathbf{x}^{t+1}, \mathbf{x}^{t+2}]$$

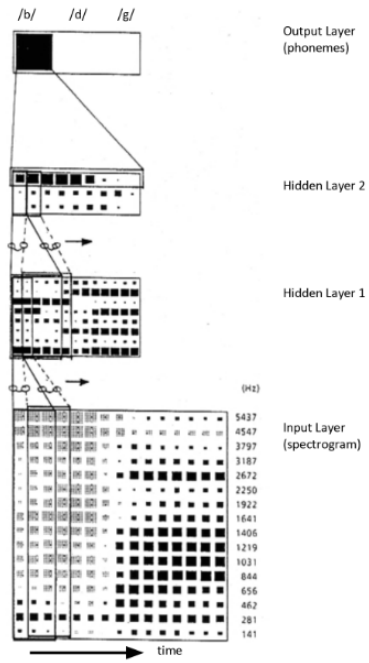


- zwiększenie wymiaru wejściowego istotnie zwiększa złożoność modelu, więc potrzebna jest większa liczba próbek uczących
- modelowanie tylko zależności krótkiego zasięgu (*short-term memory*), ograniczonych wielkością okna kontekstu

Uczenie sekwencji z MLP i CNN

Sieci CNN: splot względem wymiaru czasu (splot 1D), stały kontekst odpowiada wielkości efektywnego pola recepcyjnego w wymiarze czasu.



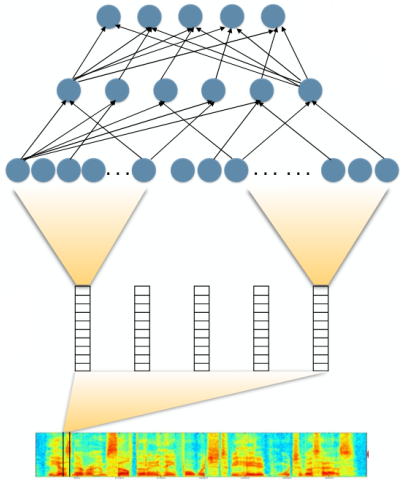


TDNN (Waibel, 1987), *time-delay neural network*, model ze splotem 1D w domenie czasu.

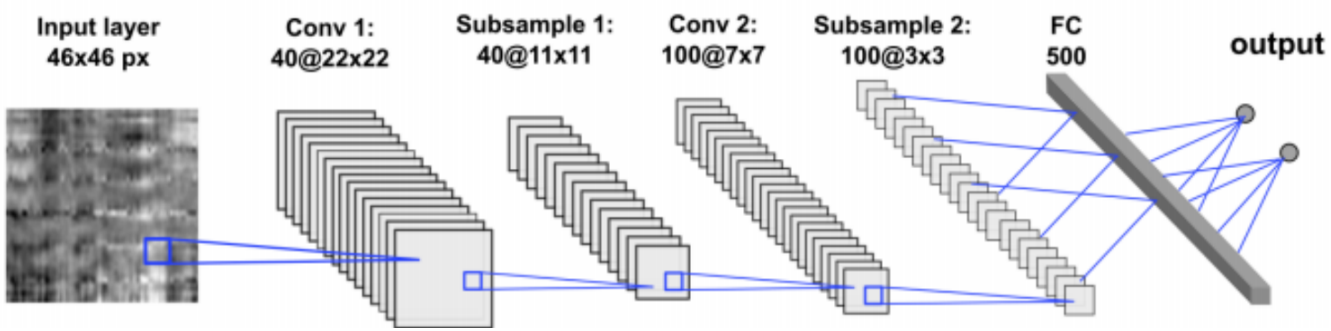
Źródło grafik: Steve Renals, Machine Learning Practical — MLP Lecture 9
Alexander Waibel, Phoneme Recognition Using Time-Delay Neural Networks, 1987.

Przykłady

Model akustyczny: rozpoznawanie fonemów z nagrań audio



Analiza sygnałów EEG: sygnał w postaci obrazów (elektroda $\times$ czas) lub widma mocy, klasyfikacja obrazów siecią CNN



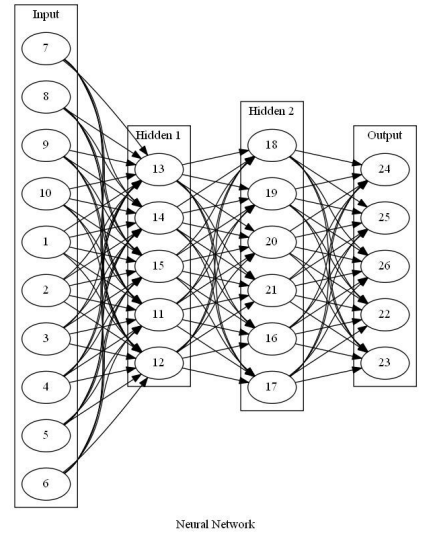
Nurse, E., Mashford, B. S., Yepes, A. J., Kiral-Kornek, I., Harrer, S., & Freestone, D. R. (2016). Decoding EEG and LFP Signals using Deep Learning: Heading TrueNorth

RNN - Sieci rekurencyjne

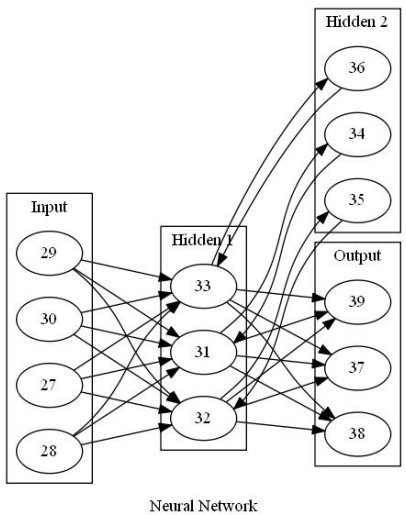
- **Sieci jednokierunkowe** używają jedynie kontekstu o skończonej długości, nie są w stanie wykryć zależności w dłuższych okresach czasu (*long-term*)

- **Sieci rekurencyjne (RNN)** posiadają połączenia do wcześniejszych warstw, wyjście w chwili t zależy od stanu z czasu $t-1$ oraz z kroków poprzednich dzięki rekurencji

Sieć jednokierunkowa

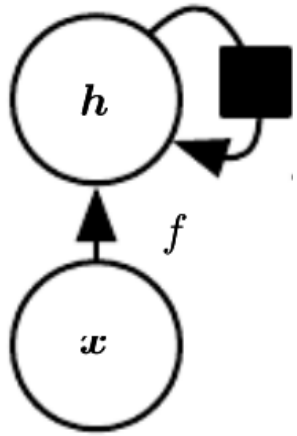


Sieć rekurencyjna Elmana



Stan ukryty jednostek RNN

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta})$$

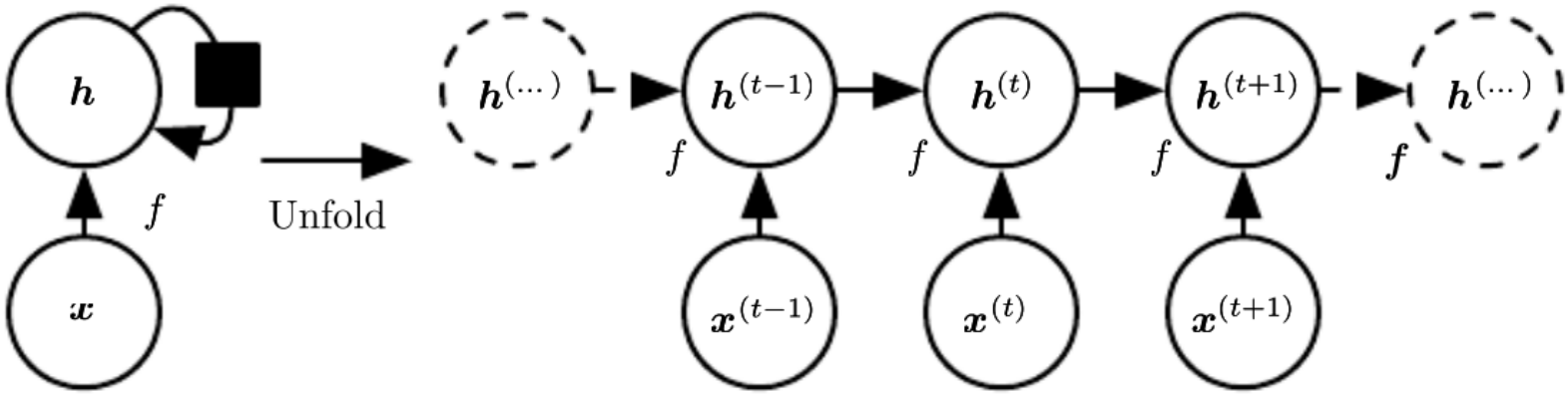


- neurony z połączeniami rekurencyjnymi działają jak pamięć, potencjalnie są w stanie modelować relacje występujące z dowolnie długim odstępem czasowym (nieskończony kontekst)

$$\mathbf{h}^{(t)} = g(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \ldots, \mathbf{x}^{(1)})$$

- wektor $\mathbf{h}^t$ (aktywacje neuronów warstwy rekurencyjnej) tworzy skompresowaną reprezentację sekwencji
- stan początkowy $\mathbf{h}^0$ może być interpretowany jako dodatkowy sygnał wejściowy, typowo inicjowany wartością zero

Rozwinięcie rekurencji w czasie

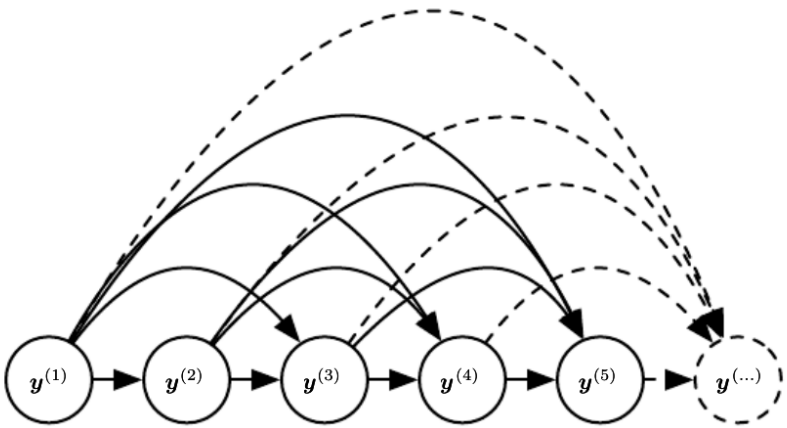


Rozwinięcie sieci RNN do postaci sieci jednokierunkowej, gdzie wagi połączeń są współdzielone.

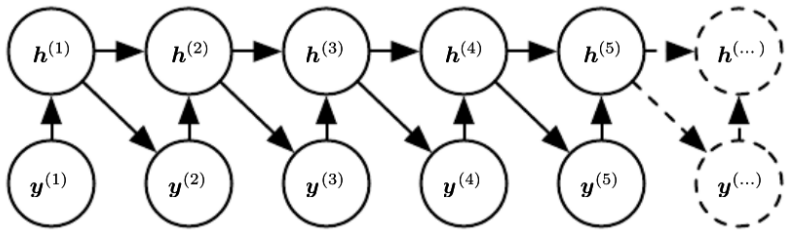
Głębokie uczenie w sieciach RNN: od pierwszego sygnału $\mathbf{x}^1$ do wyjścia $\mathbf{h}^t$ jest wiele warstw

Efektywność kodowania

- Sieć RNN z rekurencją w warstwach ukrytych spełnia wymagania **uniwersalnej maszyny Turinga**
- Ilość parametrów opisujących stan **h** ma wpływ na wielkość pamięci, ale nie musi zależeć od długości sekwencji

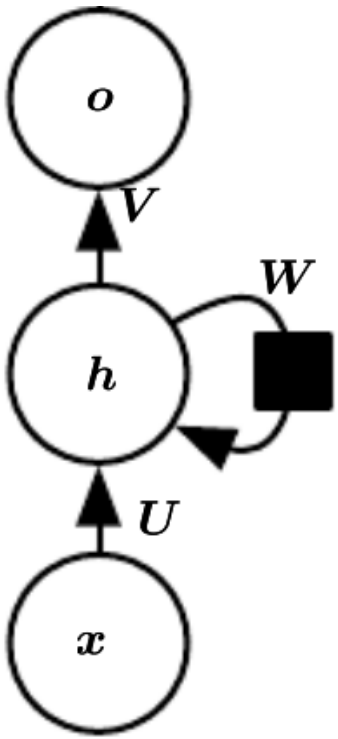


pełny model sekwencji, ilość parametrów $O(k^{\tau})$



RNN, stan obecny zależy pośrednio od stanów z przeszłości, ilość parametrów stała względem długości sekwencji $O(1)$

Sieć z jedną warstwą rekurencyjną



$$o_i^{(t)} = softmax_i \left(\sum_{r=1}^{k} v_{ir} h_r^{(t)} + b_i \right)$$

$$h_i^{(t)} = \sigma \left(\sum_{j=1}^{d} u_{ij} x_j^{(t)} + \sum_{r=1}^{k} w_{ir} h_r^{(t-1)} + c_i \right)$$

To samo zwięźlej w zapisie macierzowym

$$\mathbf{o}^{(t)} = softmax \left(\mathbf{V}\mathbf{h}^{(t)} + \mathbf{b} \right)$$

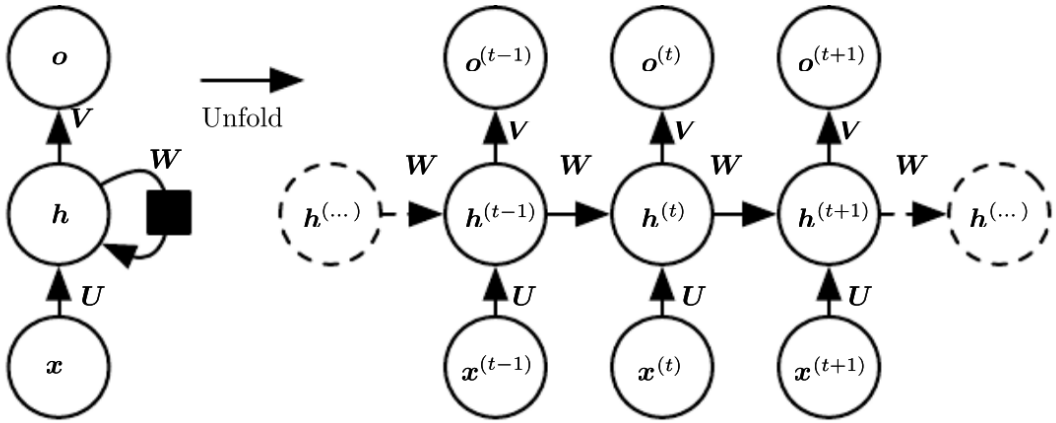
$$\mathbf{h}^{(t)} = \sigma \left(\mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{c} \right)$$

$\mathbf{x}^{(t)} \in \mathbb{R}^d$ sygnał wejściowy w chwili t
$\mathbf{U} \in \mathbb{R}^{d \times k}$, $\mathbf{W} \in \mathbb{R}^{k \times k}$
d ilość wejść sieci
k ilość neuronów w warstwie ukrytej

Źródło grafiki: Goodfellow, Deep Learning, 2016

Wsteczna propagacja w czasie

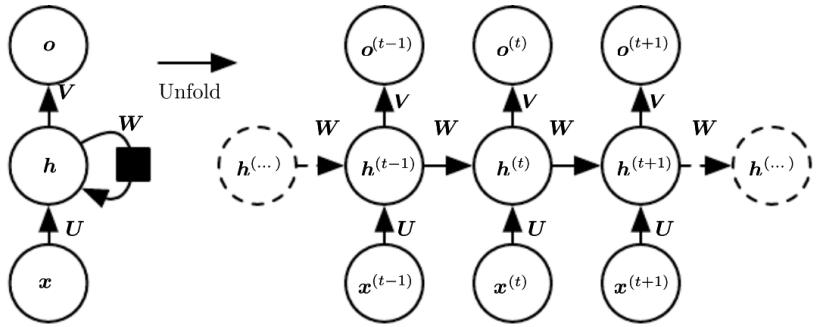
Rozwinięta sieć odpowiada głębokiej sieci jednokierunkowej ze współdzielonymi wagami $\mathbf{W}, \mathbf{V}, \mathbf{U}$



BPTT *backpropagation through time* - trening za pomocą wstecznej propagacji sieci „rozwiniętej w czasie", odbywa się analogicznie jak w jednokierunkowych sieciach.
Aktualizacja wagi w chwili t wymaga:

- całej historii sygnałów przeszłych $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}$
- historii stanów neuronów $\mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(t)}$ i ich pochodnych $\mathbf{h'}^{(1)}, \ldots, \mathbf{h'}^{(t)}$

Wsteczna propagacja w czasie



Funkcja kosztu dla całej sekwencji (model *many-to-many*):

$$L\left(\{\mathbf{x}^{(1)},\mathbf{x}^{(2)},\ldots,\mathbf{x}^{(\tau)}\},\{\mathbf{y}^{(1)},\mathbf{y}^{(2)},\ldots,\mathbf{y}^{(\tau)}\}\right)=\frac{1}{\tau}\sum_{t=1}^{\tau}L^{(t)}$$

$L^{(t)}$ - koszt w kroku t (np. cross-entropy) zależny od $\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(t)}$

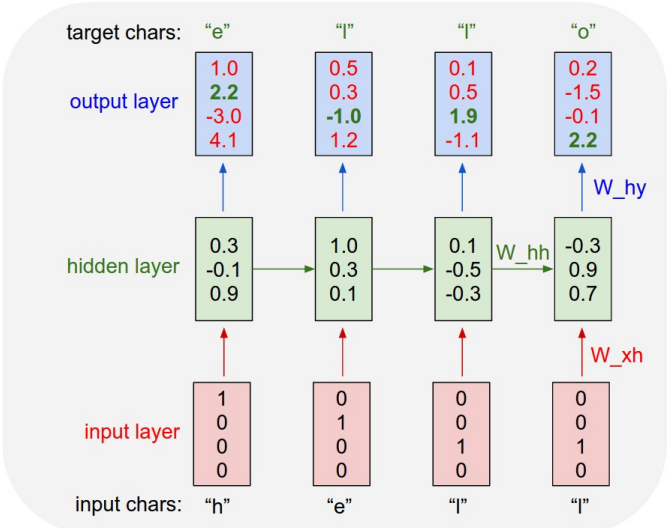
$$\partial_w L=\sum_{t=1}^{\tau}\partial_w L\left(y^{(t)},o^{(t)}\right)$$

Aktualizacje wag są akumulowane dla sygnału błędu propagowanego w kolejnych krokach czasu. Gradient w BPTT zawiera człony łańcuchowe przez stany ukryte i współdzielenie wag w czasie.

Źródło grafiki: Goodfellow, Deep Learning, 2016

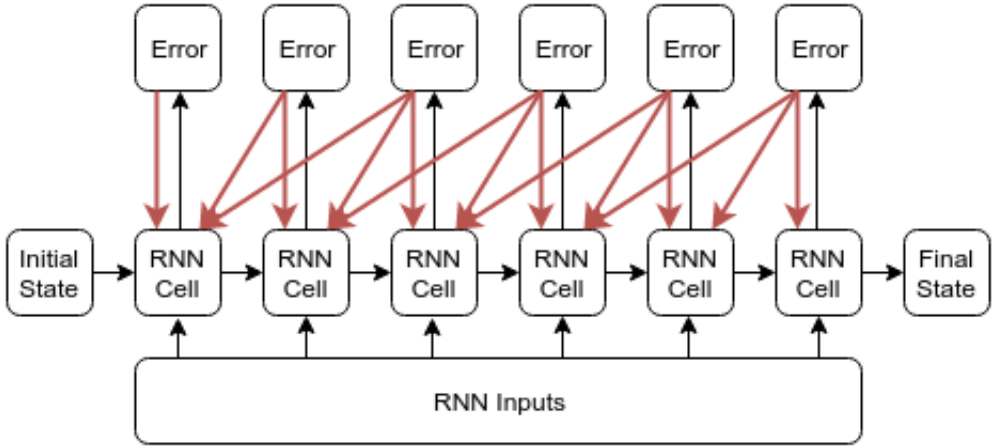
Przykład: CharRNN

- model języka oparty na znakach, wejście i wyjście stanowi sekwencja znaków
- znaki kodowane z pomocą wektorów binarnych o długości równej liczbie wszystkich znaków w alfabecie (kodowanie *one-hot*)
- problem klasyfikacji przewidywania kolejnego znaku w sekwencji $f\left(x^{(t)}\right) \longrightarrow x^{(t+1)}$
- wyjście softmax modeluje rozkład prawdopodobieństwa $P\left(x^{(t+1)} | x^{(1)}, x^{(2)}, \ldots, x^{(t)}\right)$



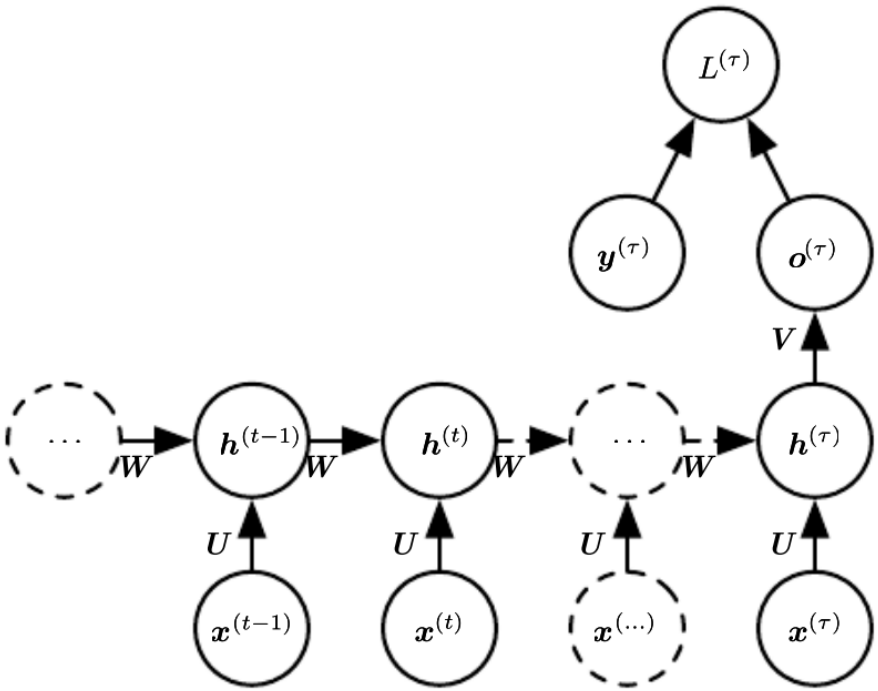
☞ The Unreasonable Effectiveness of Recurrent Neural Networks

Wsteczna propagacja w czasie



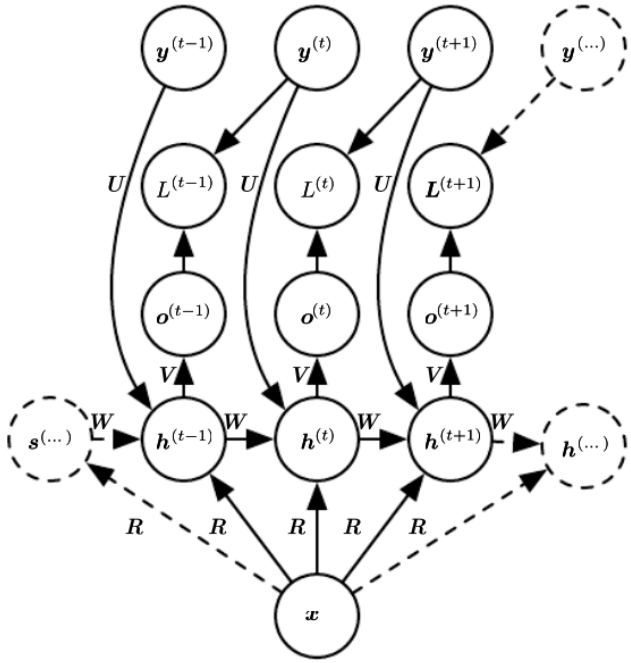
- Rozwinięta w czasie sieć tworzy głęboką sieć neuronową ze wszystkimi tego konsekwencjami (zanikający gradient dla relacji odległych w czasie, wybuchający gradient)
- Obliczenia dla długich sekwencji są kosztowne
- Nie zawsze potrzebujemy informacji z odległej przeszłości
- **Truncated BPTT** - ograniczenie długości sekwencji wstecznej propagacji do stałej wartości T (np. $T = 20$). Gradient dla zależności z dalszej przeszłości $t > T$ jest obcinany do 0
- problem z wykrywaniem odległych relacji (*long-term*) przez obcięcie T oraz zanikający gradient

Pojedyncze wyjście podsumowujące sekwencję



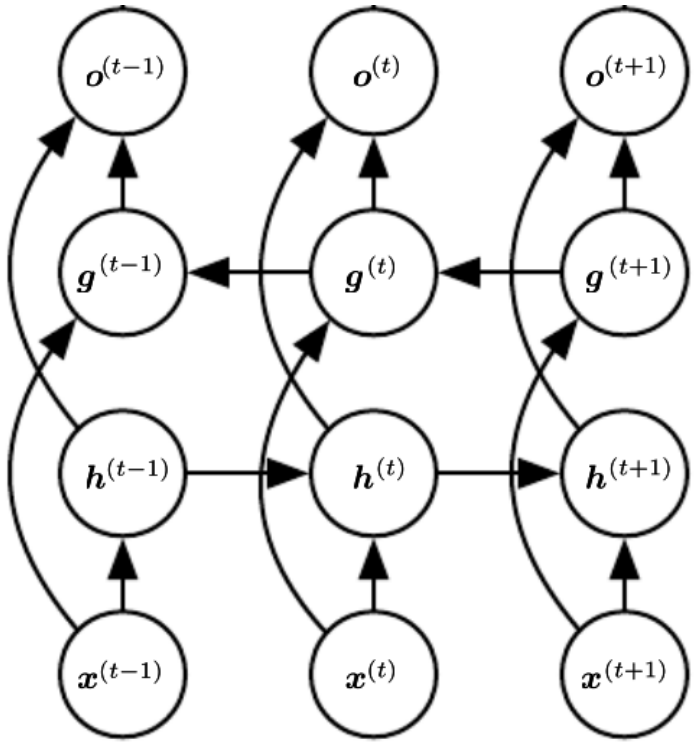
Sieć produkuje wyjście po obejrzeniu całej sekwencji, np. klasyfikacja wpisów w portalach społecznościowych (tzw. analiza sentymentów)

Pojedyncze wejście generujące sekwencję



- Sieć produkuje sekwencję wyjść przy prezentacji stałego wzorca (np. etykietowanie zdjęć, opis sceny)
- Sygnał wejściowy **x** pełni rolę kontekstu
- Modelowanie sekwencji $\mathbf{y}^{(t)} = f(\mathbf{y}^{(t-1)}; \mathbf{x})$

Dwukierunkowa sieć RNN



Sieci dwukierunkowe RNN łączą warstwy z rekurencją zależną od sygnału przeszłego **h** z rekurencją zależną od przyszłego sygnału **g**

Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks.Śignal Processing, 1997

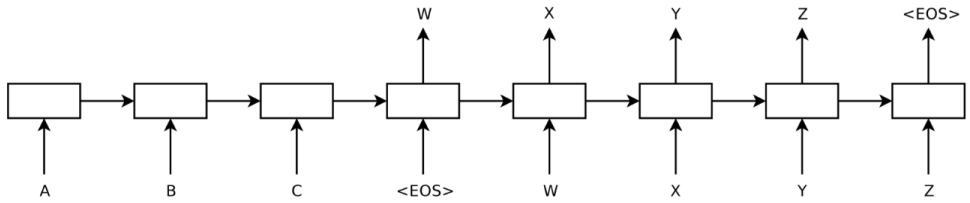
Dwukierunkowa sieć RNN

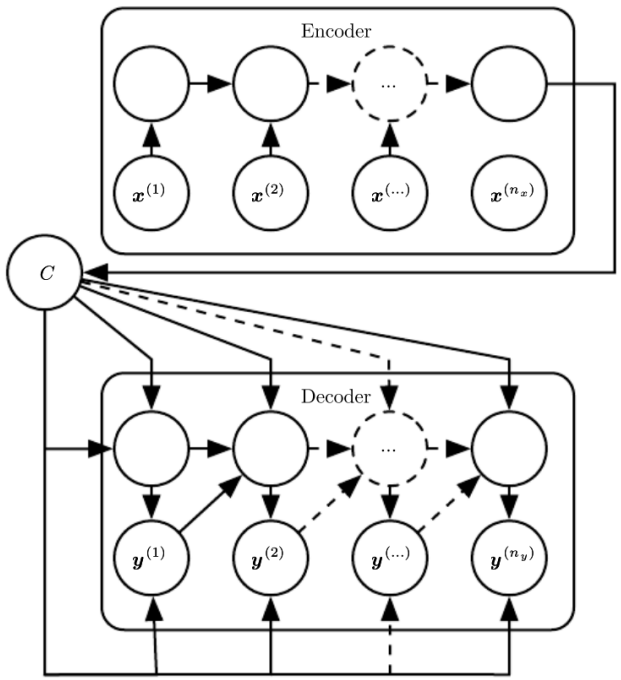
- informacja z przyszłości może zawierać istotną informację, np. dźwięki kolejnych fonemów (koartykulacja), znaczenie słowa w zdaniu może zależeć od słów występujących później
- wyjścia sieci w dowolnym momencie zależą od całej sekwencji wejściowej, możliwe tylko uczenie *offline*, gdy cała sekwencja jest znana
- zastosowanie: rozpoznawanie mowy (Graves, 2008, 2009), bioinformatyka (Baldi, 1999), rozpoznawanie pisma (Liwicki, Marcus, 2007)
- dla obrazów 2D możliwe zastosowanie sieci 4 kierunkowych (lewo, prawo, góra, dół), co pozwala modelować odległe relacje, jednak nie są one tak wydajne w tych zastosowaniach jak CNN

Model Encoder-Decoder

Transformacja sekwencji w inną sekwencję

$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n_x)} \Rightarrow \mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(n_y)}$





Sutskever et al (2014). "Sequence to Sequence Learning with Neural Networks", NIPS.
K Cho et al (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", EMNLP.

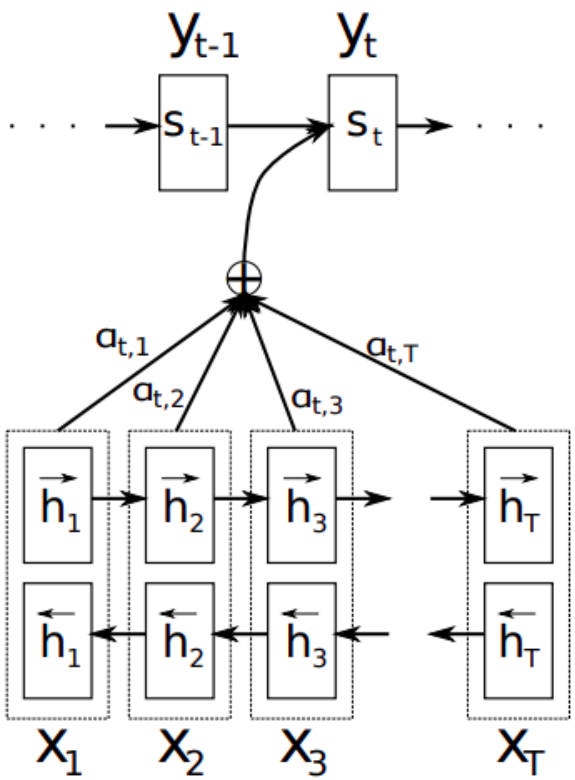
Model Encoder-Decoder

seq2seq (Sutskever, 2014) to dwie sieci RNN uczone jednocześnie:

- **(en)koder** (reader, input) przetwarza sekwencję wejściową generując kontekst **c** o stałej wielkości
- **dekoder** (writer, output) zależnie od kontekstu **c** generuje sekwencję wyjściową
- wielkość kontekstu **c** ogranicza zdolność sieci do reprezentacji dalekich w czasie relacji

Model seq2seq z systemem uwagi (Bahdanau, 2015)

Model RNN enkoder-dekoder z **mechanizmem uwagi** (*attention*) umożliwia dekoderowi „skupić uwagę” na wybranych elementach sekwencji wejściowej w różnych momentach i decydować o ich wpływie na generowane wyjście



Sieć modeluje prawdopodobieństwo

$$p(\mathbf{y}_i|\mathbf{y}_1,\ldots,\mathbf{y}_{i-1},\mathbf{X}) = g(\mathbf{y}_{i-1},\mathbf{s}_i,\mathbf{c}_i)$$

$\mathbf{s}_i$ to stany ukryte dekodera RNN

$$\mathbf{s}_i = f(\mathbf{y}_{i-1},\mathbf{s}_{i-1},\mathbf{c}_i)$$

$\mathbf{c}_i$ to wektor kontekstu, który jest sumą ważoną stanów enkodera $\mathbf{h}_j$

$$\mathbf{c}_i = \sum_{j=1}^{T} a_{ij}\mathbf{h}_j$$

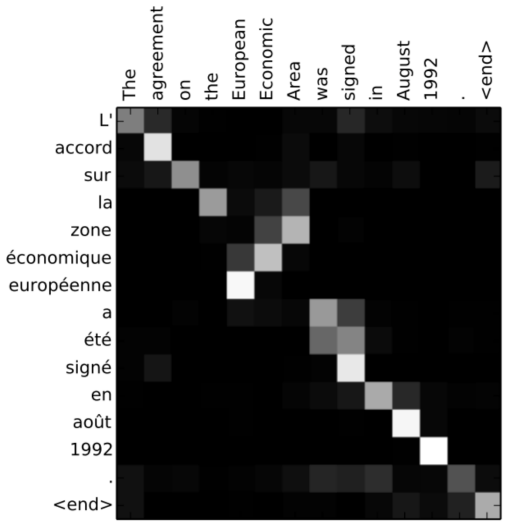
D. Bahdanau, et al (2016). Ńeural Machine Translation by Jointly Learning to Align and Translate”.

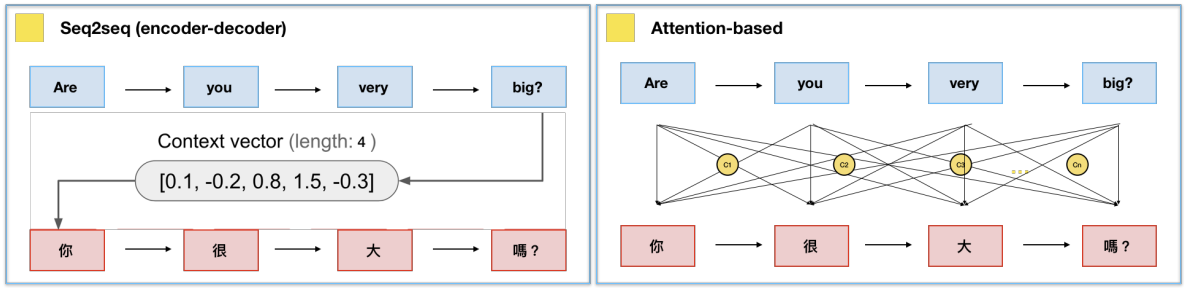
Model seq2seq z systemem uwagi (Bahdanau, 2015)

- **współczynniki uwagi** *attention score* a_{ij} są unormowane operacją softmax
$$a_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{T} \exp e_{ik}}$$
- **model dopasowania** (alignment model) e_{ij} jest funkcją, która określa jak dobrze stan dekodera $\mathbf{s}_{i-1}$ pasuje do stanu enkodera $\mathbf{h}_j$
$$e_{ij} = f(\mathbf{s}_{i-1}, \mathbf{h}_j)$$
- model dopasowania może być prostą funkcją liniową, ale może też być złożoną siecią neuronową, np. MLP, trenowaną razem z siecią enkoder-dekoder, np.:
$$e_{ij} = \mathbf{w}_{ij}^T [\mathbf{s}_{i-1}; \mathbf{h}_j]$$

D. Bahdanau, et al (2016). Ńeural Machine Translation by Jointly Learning to Align and Translate".

Model seq2seq z systemem uwagi (Bahdanau, 2015)

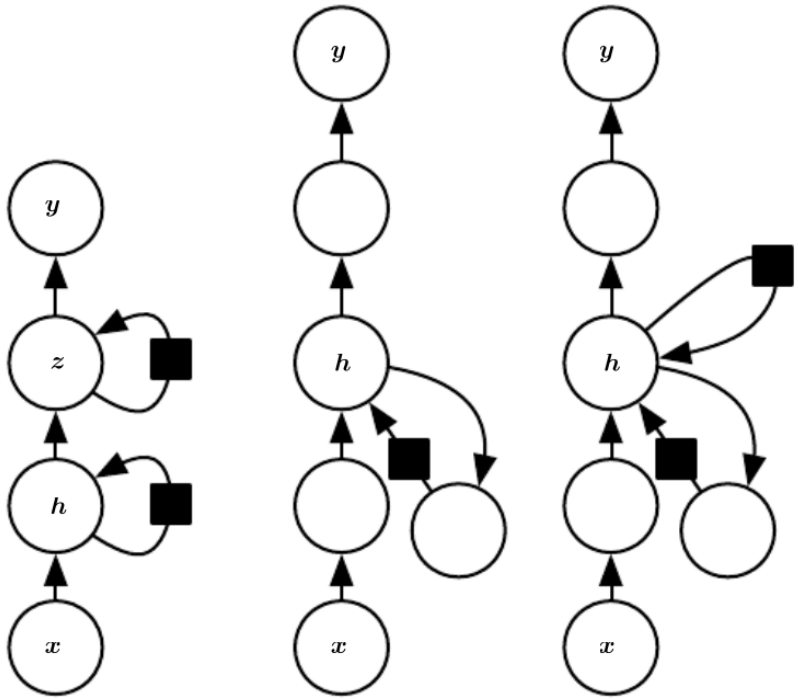




- współczynniki dopasowania a_{ij} są interpretowane jako prawdopodobieństwo, że dekoder w chwili i powinien skupić uwagę na stanie enkodera $\mathbf{h}_j$
- w trakcie generowania sekwencji dekoder może dynamicznie zmieniać uwagę na różne stany enkodera
- w przypadku tłumaczenia maszynowego dekoder skupia uwagę na różnych słowach w zdaniu źródłowym

D. Bahdanau, et al (2016). Ńeural Machine Translation by Jointly Learning to Align and Translate".

Deep RNN

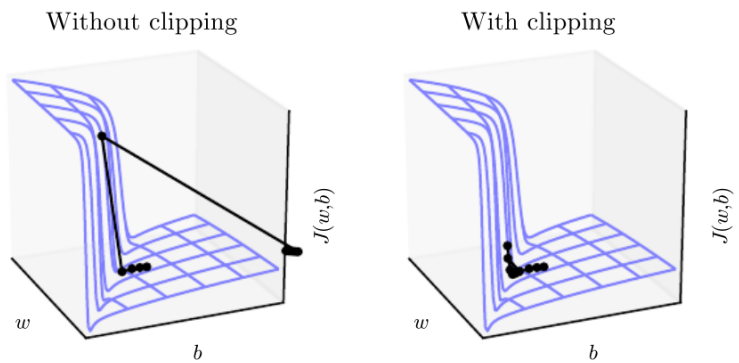


Wielowarstwowe RNN - kolejne warstwy tworzą hierarchię reprezentacji (Graves, 2013)

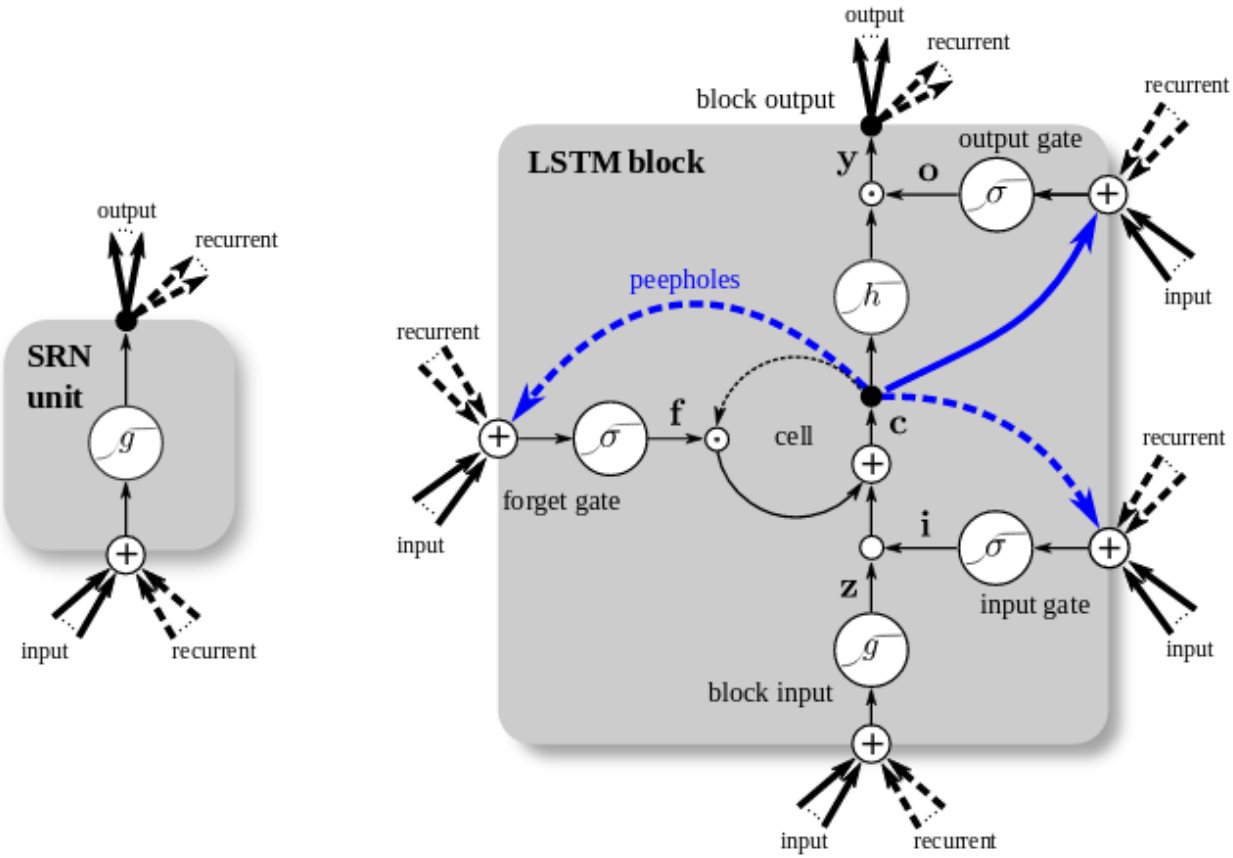
Goodfellow. Deep Learning, 2016

Problemy uczenia głębokich sieci RNN

- znikający gradient dla odległych w czasie relacji (zanika wykładniczo z czasem), rozwiązaniem LSTM
- niestabilność uczenia - eksplodujący gradient, rozwiązanie: przycinanie gradientu i regularyzacja
- **gradient clipping** - ograniczenie wartości gradientu dla wszystkich wymiarów w mini-batchu, może zmienić kierunek spadku gradientu
- **norm clipping**: jeśli $||g|| > v$ to $g \leftarrow \frac{gv}{||g||}$



Long Short-Term Memory



Sepp Hochreiter, Jürgen Schmidhuber, „Long short-term memory", 1997

LSTM

kandydat nowego stanu

$$\mathbf{z}^t = g\left(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z\right)$$

bramka wejściowa

$$\mathbf{i}^t = \sigma\left(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{b}_i\right)$$

bramka zapominania

$$\mathbf{f}^t = \sigma\left(\mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{b}_f\right)$$

stan wewnętrzny

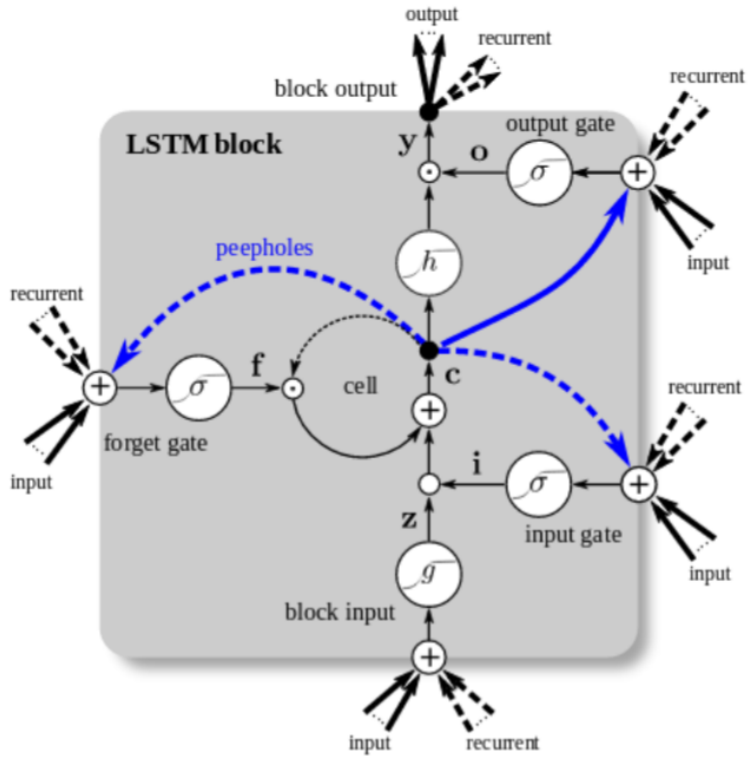
$$\mathbf{c}^t = \mathbf{i}^t \odot \mathbf{z}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1}$$

bramka wyjściowa

$$\mathbf{o}^t = \sigma\left(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{b}_o\right)$$

sygnał wyjściowy

$$\mathbf{y}^t = \mathbf{o}^t \odot h\left(\mathbf{c}^t\right)$$



Sepp Hochreiter, Jürgen Schmidhuber, „Long short-term memory", 1997

LSTM

- LSTM niweluje wpływ zanikania gradientu, dzięki czemu pozwala mapować odległe w czasie relacje
- wprowadza dodatkową wewnętrzną jednostkę stanu $\mathbf{c}$
- stan $\mathbf{c}^t$ zależy w sposób liniowy od stanu poprzedniego $\mathbf{c}^{t-1}$ (liniowa rekurencja), więc nawet przy bardzo długich rekurencjach gradient ma szansę nie zaniknąć (*long-term memory*)
- iloczyn Hadamanda $\mathbf{x} \odot \sigma(\mathbf{y})$ (*po współrzędnych*), realizacja bramki wyłączającej/włączającej sygnał $\mathbf{x}$
- bramki wejściowa, wyjściowa i bramka zapominania sterują zapamiętywanymi informacjami, zależą od sygnału wejściowego i stanu poprzedniego
- funkcja różniczkowalna - można uczyć metodami spadku gradientu i wsteczną propagacją

LSTM

- bramka wejściowa ma wpływ na to, jaki sygnał zostanie zapisany w jednostce stanu $\mathbf{c}$
- bramka zapominania steruje wpływem poprzedniego stanu $\mathbf{c}^{t-1}$ na obecny stan $\mathbf{c}^{t}$
- bramka wejściowa i zapominania wpływają na to, co jest umieszczane i co jest usuwane ze stanu
- bramka wyjściowa steruje ilością informacji przepływającą ze stanu wewnętrznego do wyjścia jednostki, pozwala np. zachować na później informacje, które w tym momencie nie są istotne

Peephole LSTM

$$\mathbf{z}^t = g\left(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z\right)$$

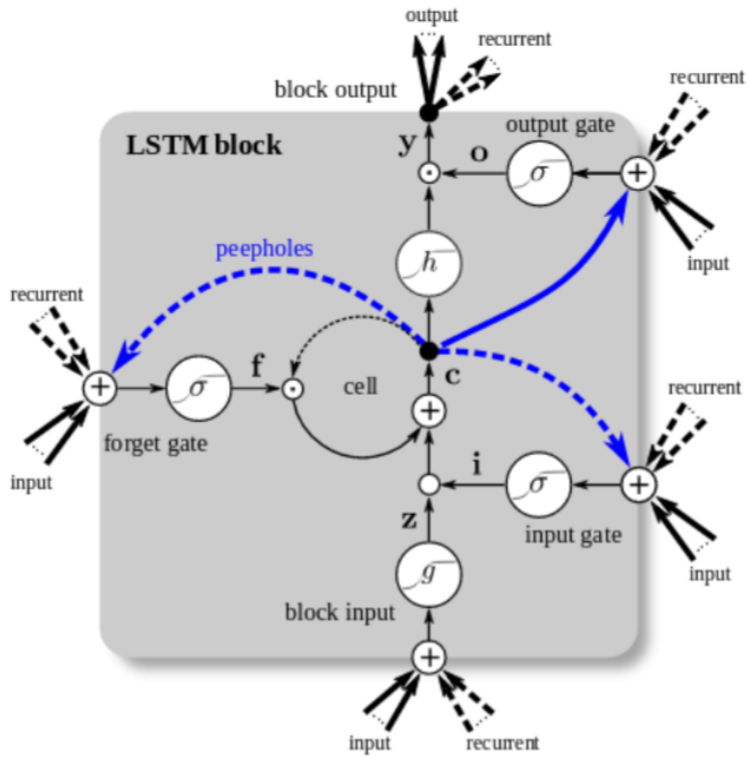
$$\mathbf{i}^t = \sigma\left(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i\right)$$

$$\mathbf{f}^t = \sigma\left(\mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f\right)$$

$$\mathbf{c}^t = \mathbf{i}^t \odot \mathbf{z}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1}$$

$$\mathbf{o}^t = \sigma\left(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o\right)$$

$$\mathbf{y}^t = \mathbf{o}^t \odot h\left(\mathbf{c}^t\right)$$

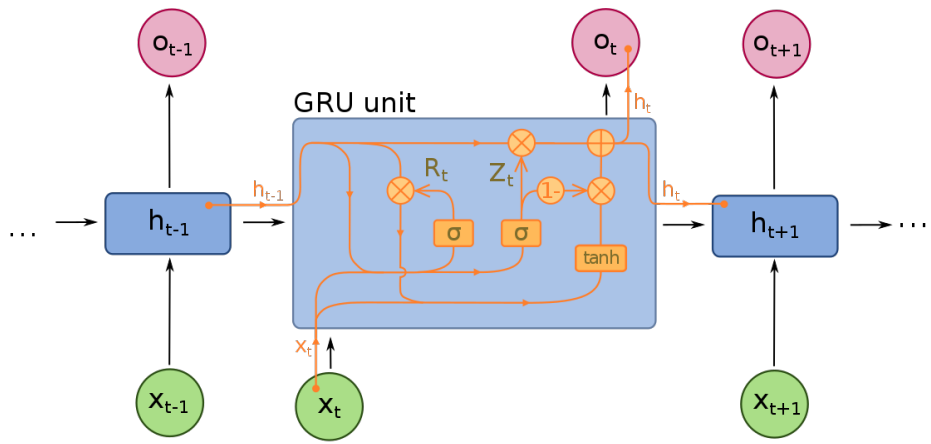


aktywacja bramek uzależniona dodatkowo od poprzedniego stanu $\mathbf{c}^{t-1}$ poprzez połączenia „podglądające” $\mathbf{p}_i$, $\mathbf{p}_o$, $\mathbf{p}_f$

Sepp Hochreiter, Jürgen Schmidhuber, „Long short-term memory”, 1997

Gated Recurrent Unit

- GRU (Kyunghyun Cho et al., 2014) brak jednostki wewnętrznej $\mathbf{c}$ i bramki wyjściowej
- bramki: resetu r_t i aktualizacji z_t
- mniej parametrów względem LSTM, porównywalna jakość wyników dla wielu problemów



Źródło: https://en.wikipedia.org/wiki/Recurrent_neural_network

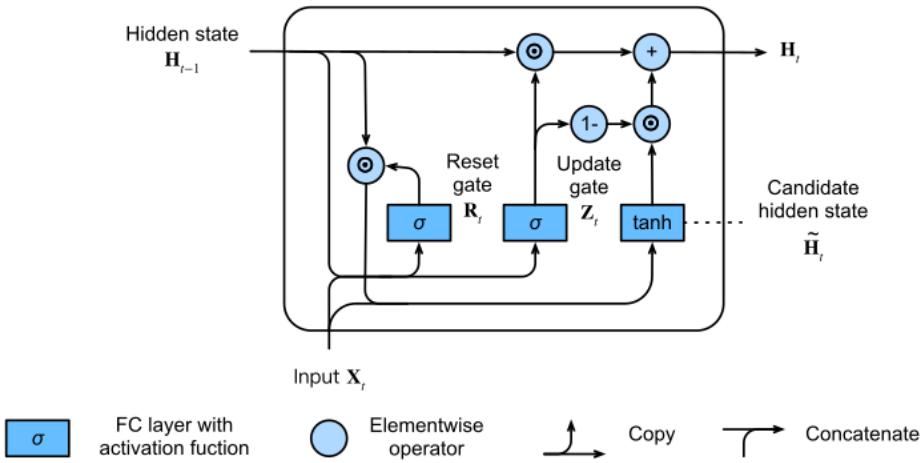
Gated Recurrent Unit

$$\mathbf{z}^t = \sigma\left(\mathbf{W}_z\mathbf{x}^t + \mathbf{U}_z\mathbf{h}^{t-1} + \mathbf{b}_z\right)$$

$$\mathbf{r}^t = \sigma\left(\mathbf{W}_r\mathbf{x}^t + \mathbf{U}_r\mathbf{h}^{t-1} + \mathbf{b}_r\right)$$

$$\tilde{\mathbf{h}}^t = \tanh\left(\mathbf{W}_h\mathbf{x}^t + \mathbf{U}_h\left(\mathbf{r}^t \circ \mathbf{h}^{t-1}\right) + \mathbf{b}_h\right)$$

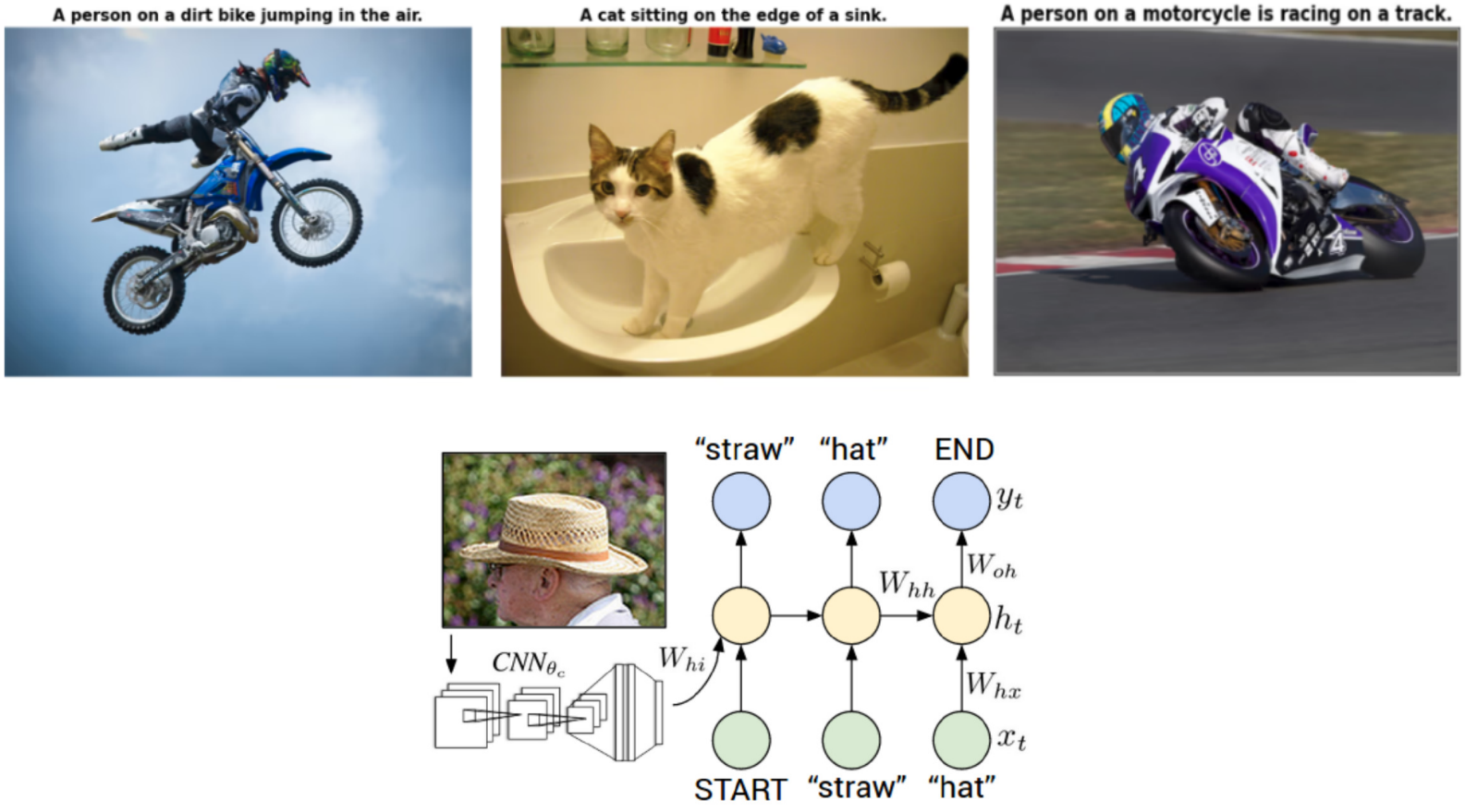
$$\mathbf{h}^t = \left(1 - \mathbf{z}^t\right) \circ \mathbf{h}^{t-1} + \mathbf{z}^t \circ \tilde{\mathbf{h}}^t$$



- stan proponowany $\tilde{\mathbf{h}}^t$, dla $\mathbf{r}^t = 0$ sprowadza się do warstwy MLP
- bramka aktualizacji $\mathbf{z}^t$ określa stopień wpływu poprzedniego stanu $\mathbf{h}^{t-1}$ oraz obecnego stanu proponowanego $\tilde{\mathbf{h}}^t$
- bramka resetu odpowiada za relacje krótkie w czasie, a bramka aktualizacji - za długie

Źródło grafiki: https://d2l.ai/chapter_recurrent-modern/gru.html

Zastosowania: generator opisów zdjęć

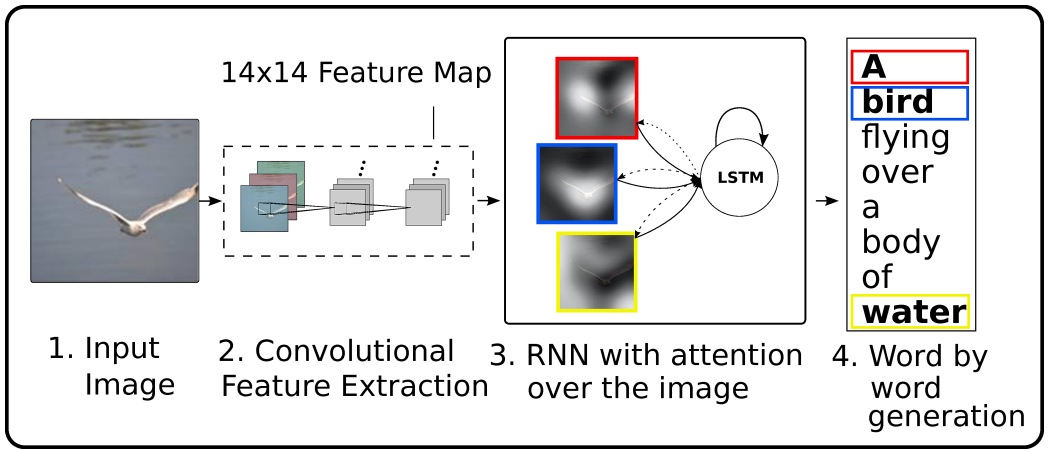


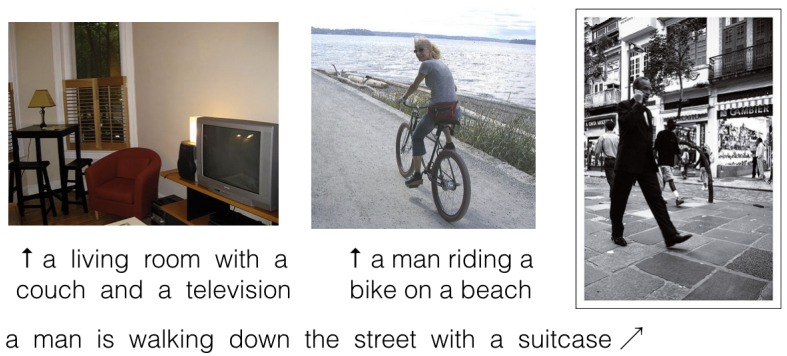
Multimodal Recurrent Neural Network ☞ NeuralTalk2

Karpathy A., Fei-Fei L. „Deep Visual-Semantic Alignments for Generating Image Descriptions", 2015

Zastosowania: generator opisów zdjęć

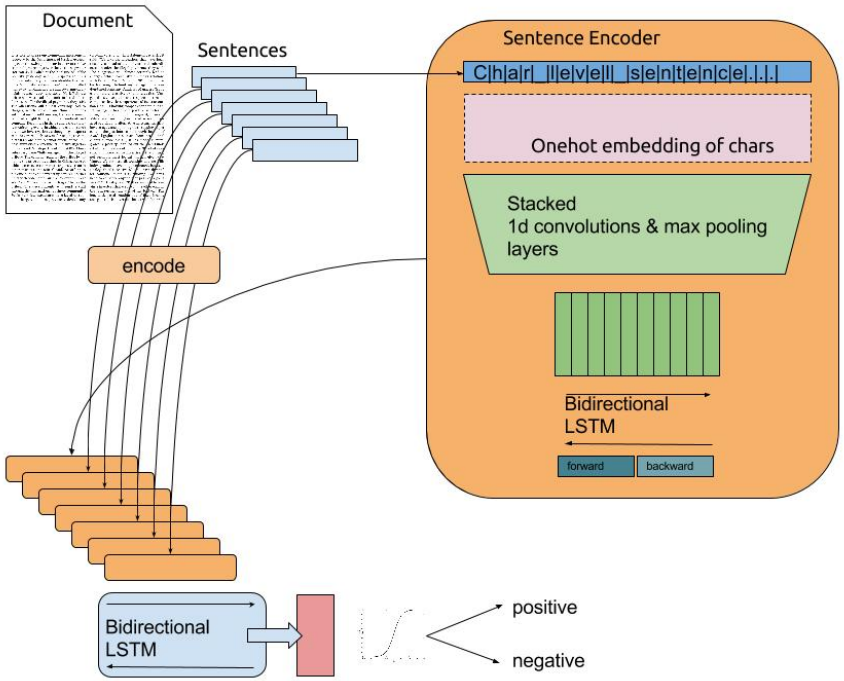
Show, attend and tell (Xu et al., 2015)





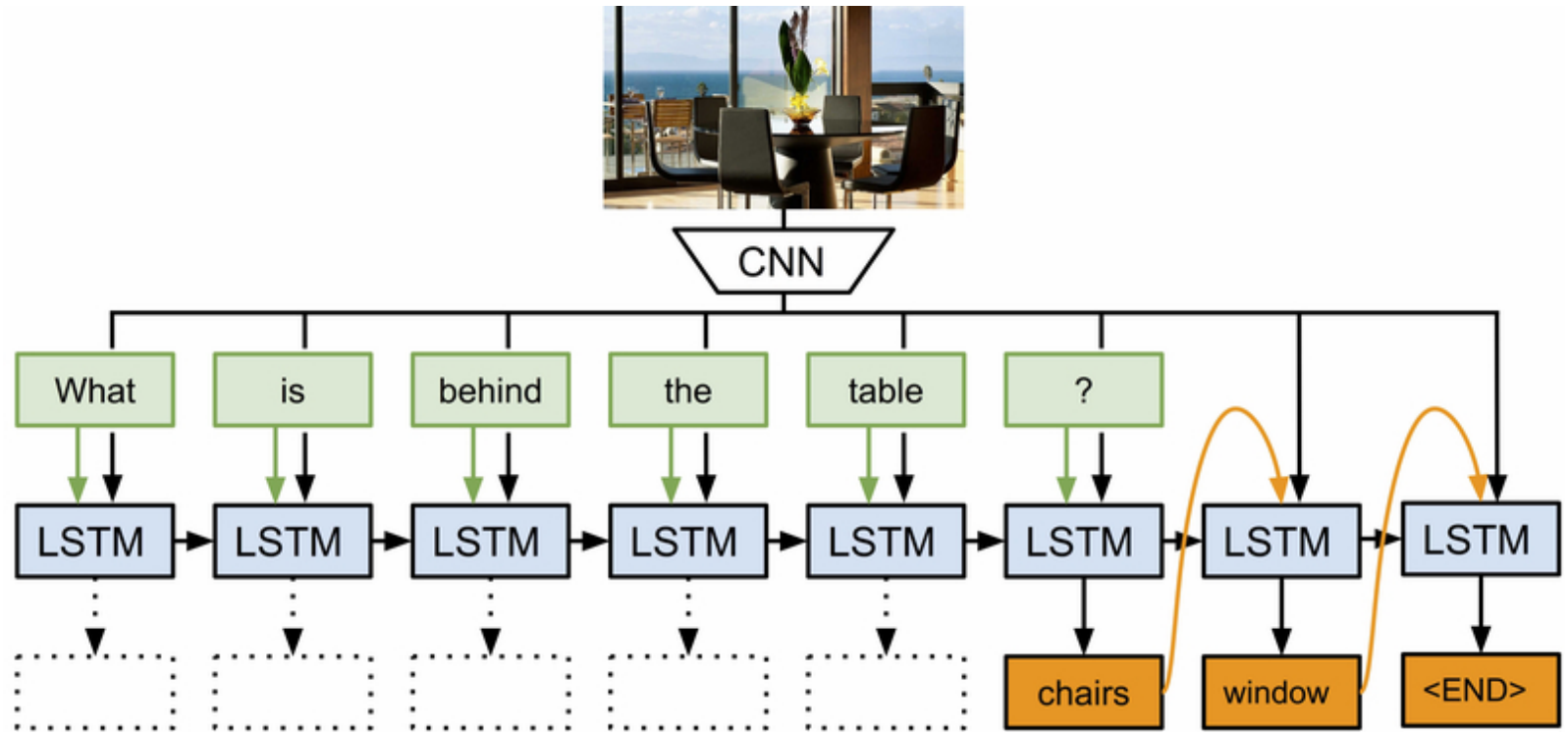
Mechanizm uwagi pozwala skupić się na wybranych fragmentach obrazu podczas generowania opisu

Zastosowania: analiza sentymentów



https://offbit.github.io/how-to-read/

Zastosowania: odpowiedzi na pytania



Uczenie nienadzorowane

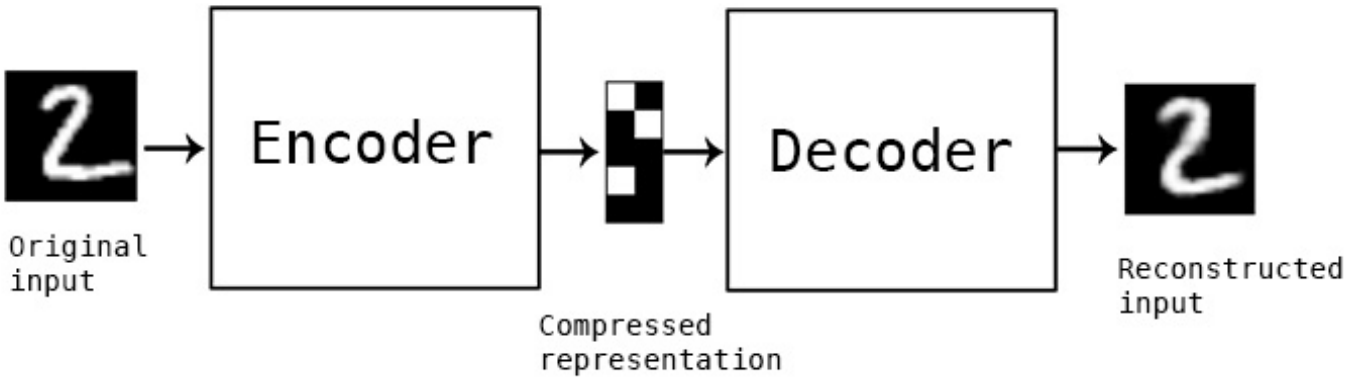
Autoenkodery

Modele generatywne

Uczenie nienadzorowane

- Uczenie **nadzorowane** (*supervised*) - dane wejściowe $\mathbf{X}$ posiadają spodziewane wyjście (etykiety) $(\mathbf{X}, \mathbf{Y})$
- Uczenie **nienadzorowane** (*unsupervised*) używa wyłącznie danych wejściowych $\mathbf{X}$. Dane bez etykiet są powszechnie występujące.
  - trening automatycznie wykrywa istotne cechy w danych,
  - modeluje rozkład prawdopodobieństwa danych
- Modele sieci:
  - autoenkodery (AE), splotowe AE (CAE), Variational AE (VAE)
  - Restricted Boltzmann Machines (RBM)
  - Generative Adversarial Networks (GAN)
- Zastosowanie:
  - kodowanie sygnału, wykrywanie istotnych cech, usuwanie szumu i rekonstrukcja sygnału, głęboka klasteryzacja
  - generowanie sygnałów, transfer stylu
  - inicjowanie głębokich sieci DNN (stacked autoencoders)

Autoenkodery



- autoenkodery uczą się odtwarzać sygnał wejściowy
- warstwy pośrednie kodują specyficzne cechy danych
- kompresja stratna - wymagamy aby model nie był jednostkowym odwzorowaniem
- AE uczą się reprezentacji danych (ekstrakcja cech), modelują rozkład prawdopodobieństwa sygnału
- koder i dekoder mogą być sieciami neuronowymi, całość uczona wsteczną propagacją błędu

Rys: ☞ Building Autoencoders in Keras

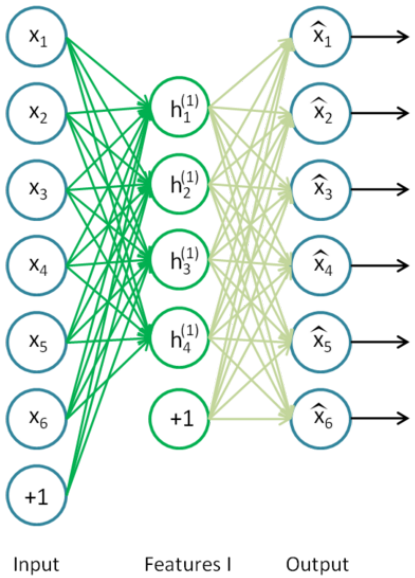
Autoenkodery

Encoder:

$$\mathbf{h} = f(\mathbf{x})$$

Representation trick

$$= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$



Decoder:

$$\hat{\mathbf{x}} = g(\mathbf{h})$$

$$= \sigma(\mathbf{W}'\mathbf{h}(\mathbf{x}) + \mathbf{b}')$$

- **Autoencoder** - sieć jednokierunkowa, której celem jest rekonstrukcja sygnału wejściowego
- warstwa ukryta $\mathbf{h}$: koduje sygnał wejściowy, detektor cech, skompresowana reprezentacja danych
- często wymaga się aby $\mathbf{W}' = \mathbf{W}^T$, współdzielone wagi kodera i dekodera

Źródło grafiki: http://ufldl.stanford.edu

Koszt rekonstrukcji

Funkcja rekonstrukcji $L(\mathbf{x}, \hat{\mathbf{x}})$

- MSE dla danych wejściowych rzeczywistych

$$L(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{2} \sum_k (\hat{x}_k - x_k)^2$$

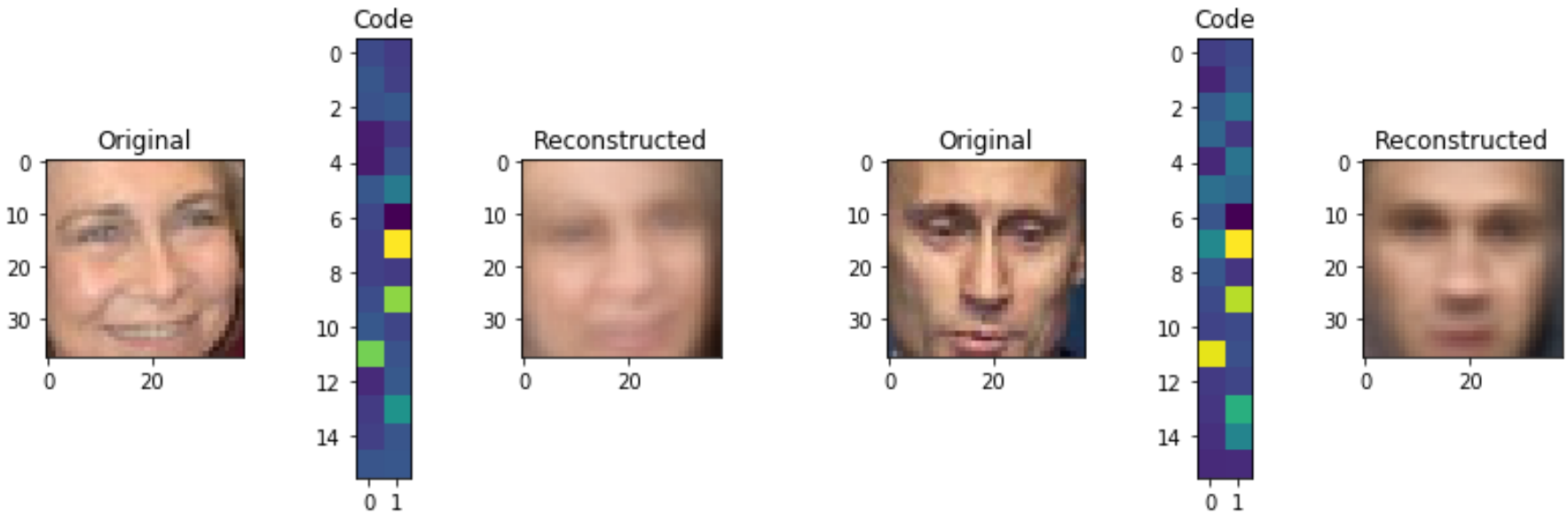
  - liniowa aktywacja warstwy wyjściowej $\hat{\mathbf{x}} = \mathbf{W}'\mathbf{h} + \mathbf{b}'$
- Cross Entropy dla danych binarnych (lub z zakresu $[0, 1]$)

$$L(\mathbf{x}, \hat{\mathbf{x}}) = -\sum_k \left(x_k \log \hat{x}_k + (1 - x_k) \log \left(1 - \hat{x}_k\right) \right)$$

- trening wsteczną propagacją błędu
- ograniczenia zapobiegające tworzeniu odwzorowania jednostkowego: niekompletny AE, regularyzowany AE, kurczliwy AE, ...

Niekompletny autoenkoder

- **Niekompletny AE** (*undercomplete*) - rozmiar warstwy **h** mniejszy od rozmiaru wejścia,
- kompresja sygnału do rozmiaru **h**
- dla liniowego dekodera wynik jest równoważny z PCA, wagi neuronów tworzą składowe główne maksymalizujące wariancję
- Przykład: input-output: 32x32 (1024 piksele) ,
  hidden (code size): 32



Źródło grafiki: ☞ Manash's blog

Regularyzowany AE

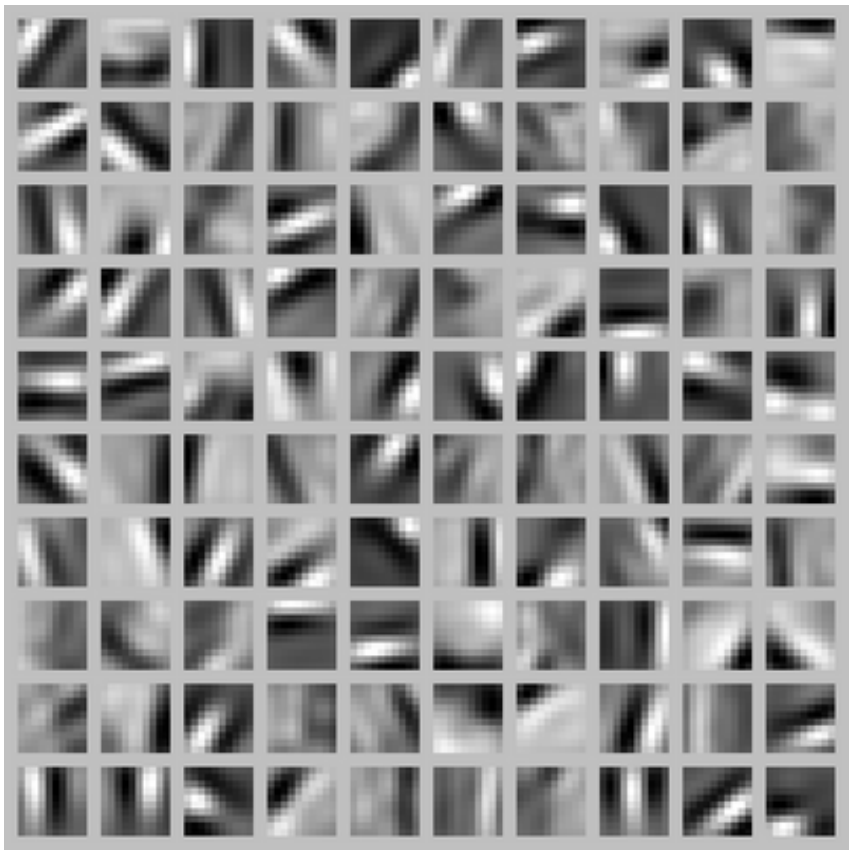
- wielkość **h** decyduje o pojemności sieci i jakości odwzorowania. Dla niekompletnych AE jest ograniczenie zwiększania rozmiaru warstwy ukrytej
- **Nadkompletny** AE (*overcomplete*) gdy rozmiar **h** większy od rozmiaru sygnału wejsciowego **x**,
  - brak kompresji
  - nie gwarantują uzyskania wartościowych reprezentacji bez dodania odpowiedniej regularyzacji wymuszającej odpowiednią reprezentację w **h** (np. rzadkość)
- **Rzadki autoencoder** z czynnikiem kary za brak rzadkiej reprezentacji

$$L(\mathbf{x}, \hat{\mathbf{x}}) + \lambda \sum |h_i|$$

  - w celu uzyskania rzadkich aktywacji można też użyć ReLU (Glorot, 2011)

Przykład

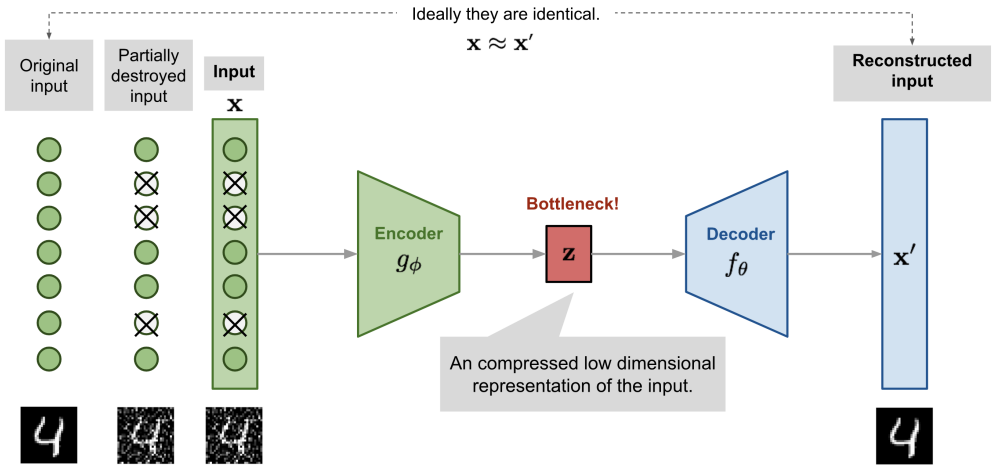
Cechy uzyskane w warstwie ukrytej przez autoenkoder rzadki (regularyzacja L_1) trenowany na naturalnych obrazach (bez etykiet!)



Oja, E. (1982). A simplified neuron model as a principal component analyzer. Journal of mathematical biology, 15(3), 267–73.

Autoenkoder z odszumianiem (DAE)

- **Denoise autoencoder (DAE)** usuwa szum (lub inne zniekształcenia) sygnału wejściowego.
- Sygnał wejściowy $\tilde{\mathbf{x}}$ jest zniekształconą kopią $\mathbf{x}$
- AE uczy się odtwarzać oryginalny sygnał bez deformacji

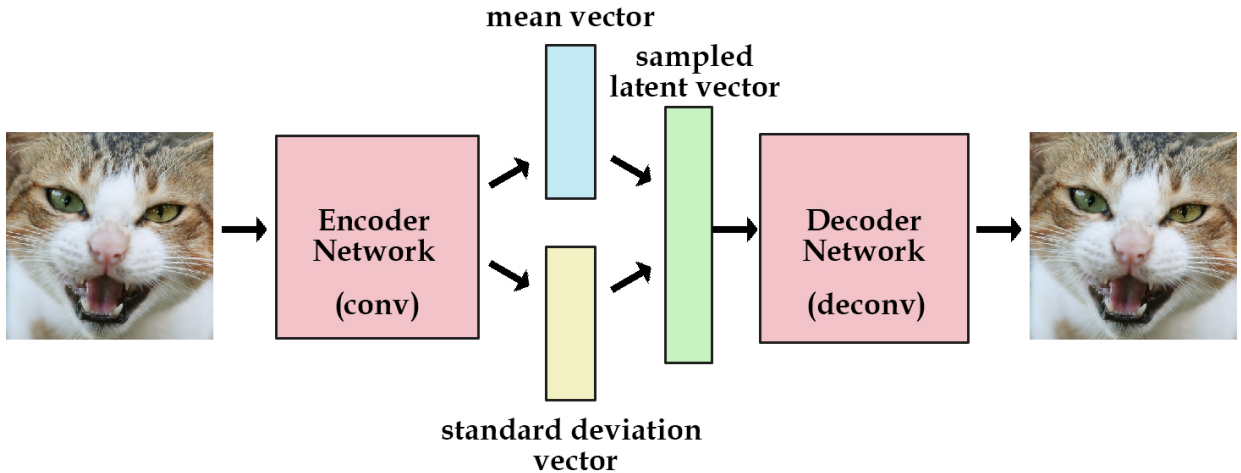




src: ☞ *Building Autoencoders in Keras*
☞ *From Autoencoder to Beta-VAE*

Wariacyjny AE

- **Variational autoencoders (VAE)** jawnie modeluje rozkład gęstości danych, kodowana reprezentacja to parametry funkcji gęstości, np. σ_i, μ_i rozkładu normalnego $N(\mu_i, \sigma_i)$



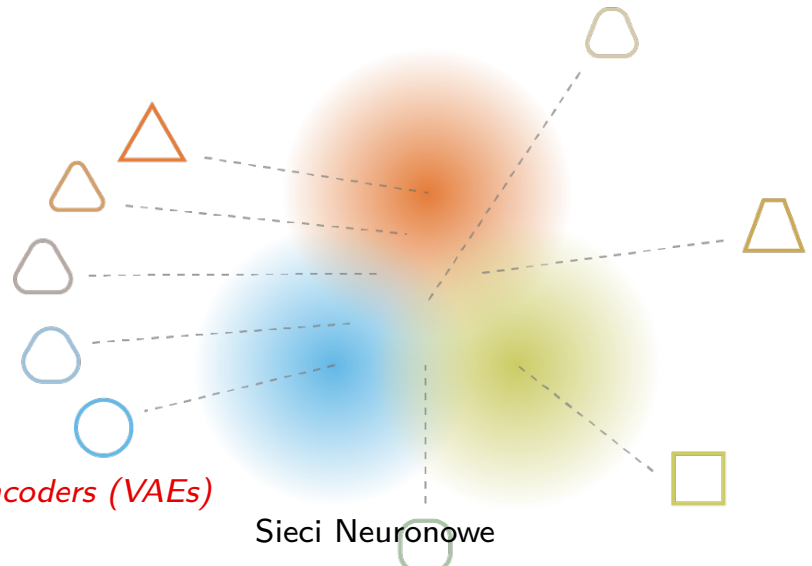
- sygnał wejściowy dekodera jest próbkowany z rozkładu określonego przez parametry warstwy kodującej
- model generatywny - umożliwia próbkowanie danych wyjściowych, tworzenie nowych sygnałów o charakterystyce zbliżonej do danych treningowych

Kullback–Leibler divergence

- czynnik regularyzujący, dodawany do funkcji kosztu (np. MSE), wymuszający odpowiedni rozkład danych w warstwie kodującej (KL divergence), np. preferujący rozkład $N(0, 1)$

$$\sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1$$

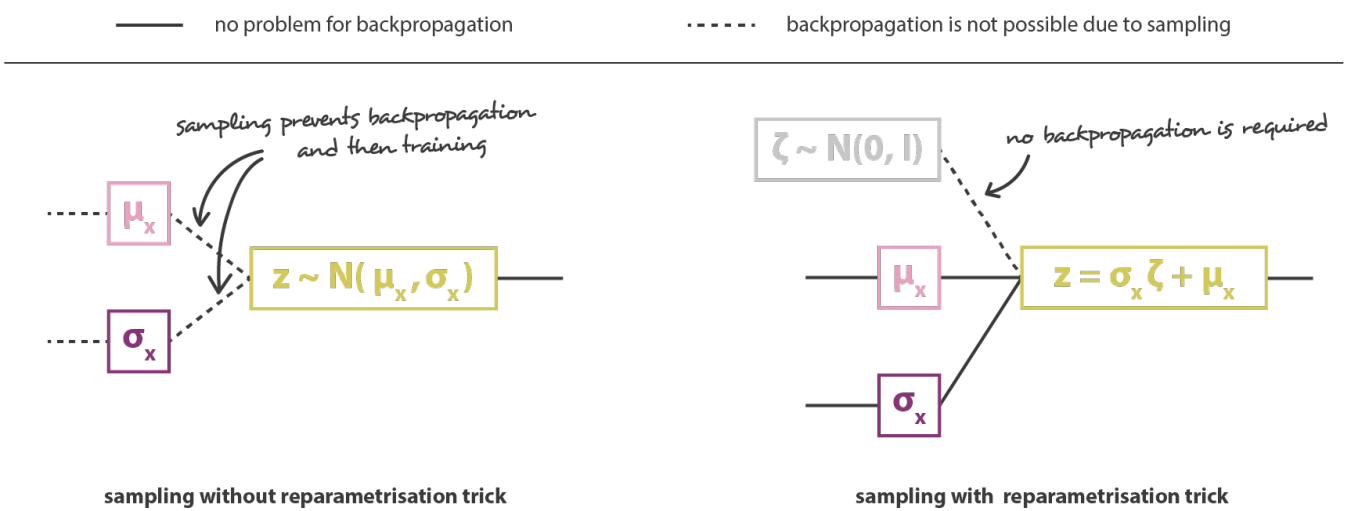
- osiąga minimum dla $\sigma_i = 1, \mu_i = 0$
- wymusza rozkład zakodowanych danych w centrum przestrzeni ukrytej, minimalizacja kosztu rekonstrukcji konkuruje z regularyzacją próbując rozdzielić grupy różnych danych



☞ *Understanding Variational Autoencoders (VAEs)*

Reparameterization trick (stochastyczne BP)

W jaki sposób zrealizować wsteczną propagację błędu?

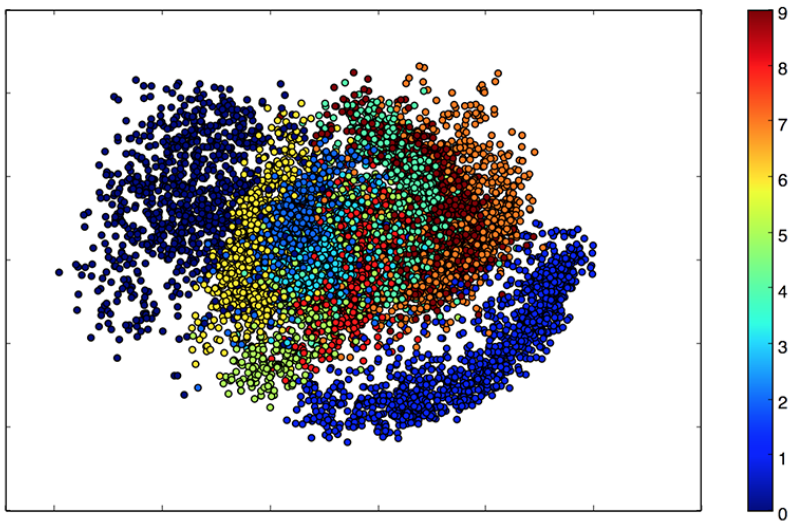


$$\mathbf{z} = \mu_{\mathbf{x}} + \sigma_{\mathbf{x}}\xi$$

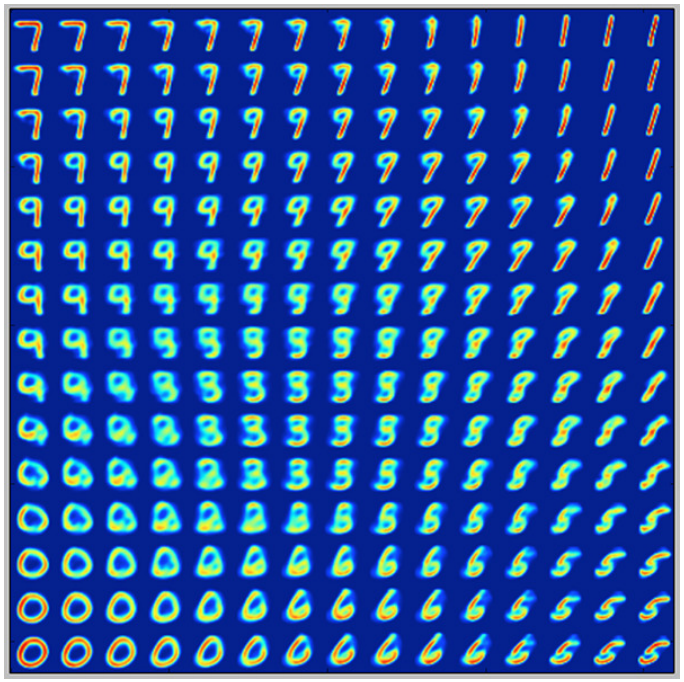
Próbka $\xi \sim \mathcal{N}(0, 1)$ stanowi kolejne wejście sieci neuronowej jednak nie posiadające parametrów optymalizacyjnych (nie propagujemy błędu tym wejściem).
Całą sieć można więc trenować wsteczną propagacją.

VAE na MNIST

wizualizacja danych treningowych w 2 wymiarowej warstwie ukrytej kodera



generowanie cyfr dla siatki 15x15 wartości 2 wymiarowej warstwy ukrytej dekodera

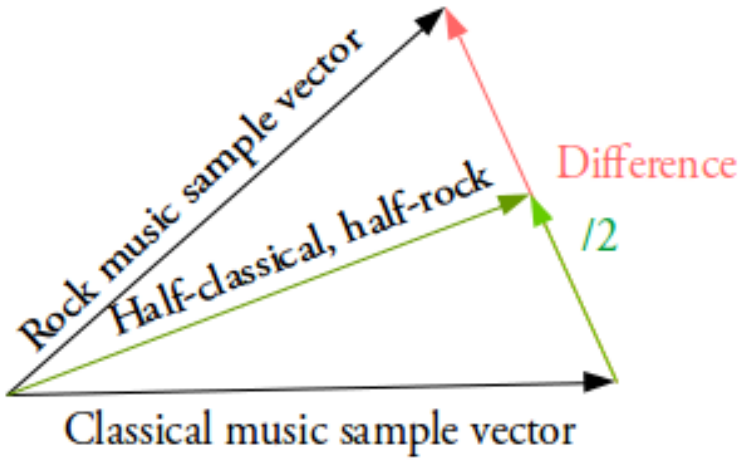


Architektura: Input-512-2-(μ, σ)-2-512-Output

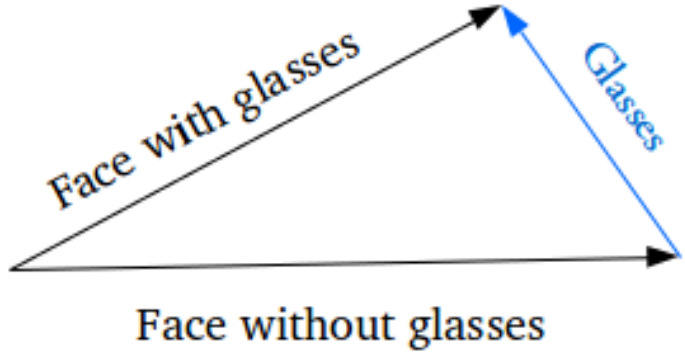
src: ☞ *Building Autoencoders in Keras*

Interpolowanie i generowanie pojęć

Arytmetyka w przestrzeni zmiennych utajonych



Środek między wektorami μ kodującymi pojęcia

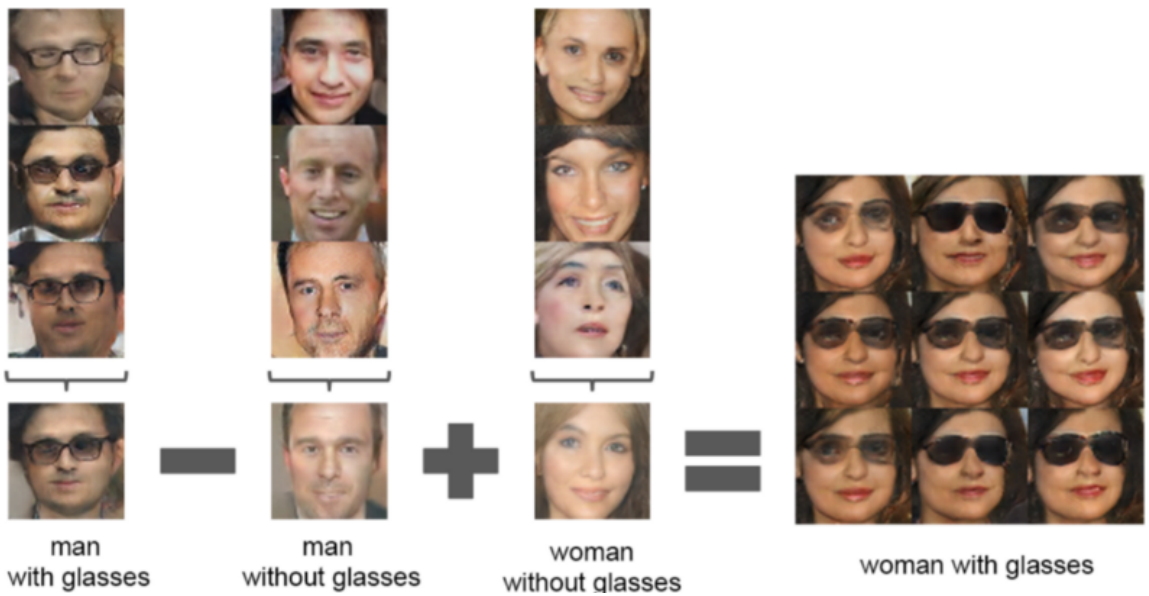


Różnica pomiędzy pojęciami może być dodana do innego pojęcia

☞ *Irhum Shafkat, Intuitively Understanding Variational Autoencoders*

Ukryta reprezentacja pojęć

- wyuczona reprezentacja danych koduje pojęcia
- arytmetyczne operacje na wektorach kodujących odpowiadają relacjom semantycznym pomiędzy pojęciami



Przykład: generowanie twarzy

Wejście



Rekonstrukcja



Losowe

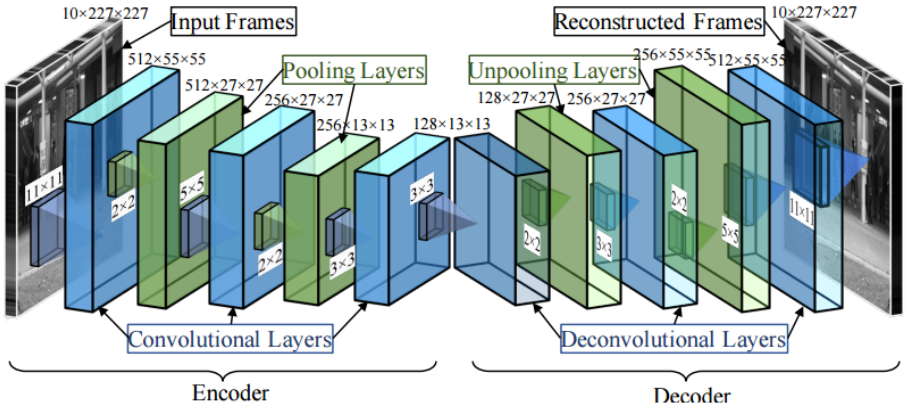


☞ Deep Feature Consistent Variational Autoencoder

☞ *Variational auto-encoder trained on celebA*

Inne typy autoenkoderów

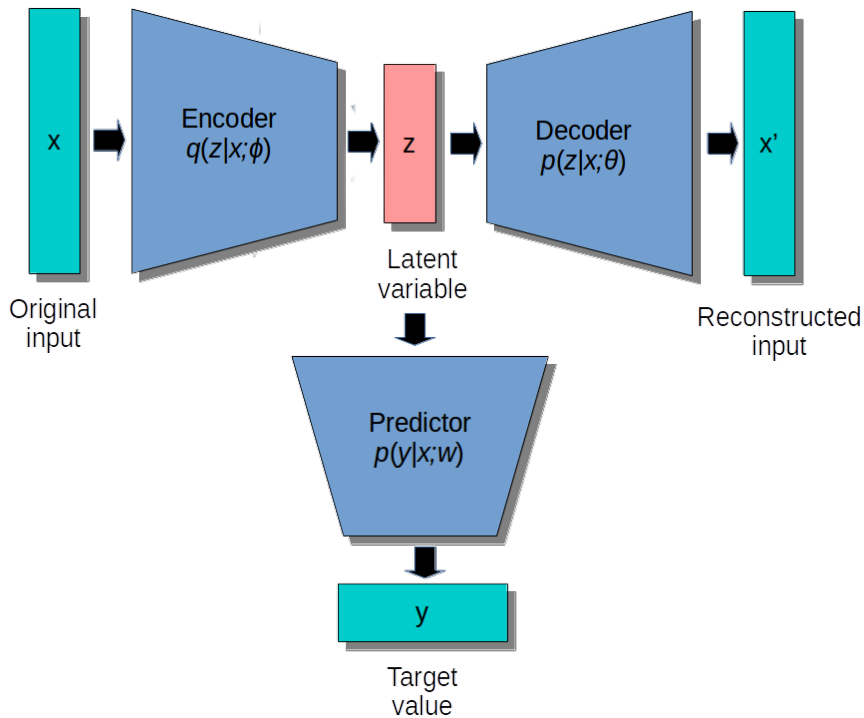
- Convolutional AE (CAE), dekoder jest symetryczną względem enkodera siecią dekonwolucyną



- seq2seq: model RNN endoder-decoder do modelowania sekwencji
- głębokie autoenkodery (DAE) - więcej warstw sprzyja kompresji danych o złożonych rozkładach
- stacked autoencoders (SAE) - tworzenie głębokich sieci z uczonych sekwencyjnie autoenkoderów

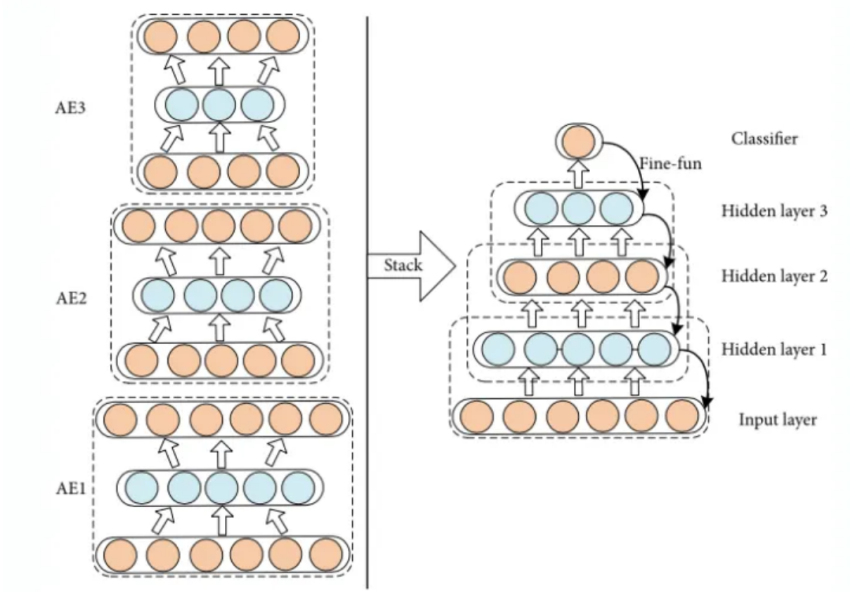
Uczenie pół-nadzorowane (samouczenie)

Uczenie pół-nadzorowane (**semi-supervised**) lub samouczenie (**self-taught**): najpierw trenujemy AE na dużym zbiorze danych w sposób nienadzorowany, następnie reprezentacja danych uzyskana z kodera wykorzystywana jest do uczenia płytkiego klasyfikatora na małym zbiorze z etykietami



Guo et al., 2017, Deep convolutional autoencoder

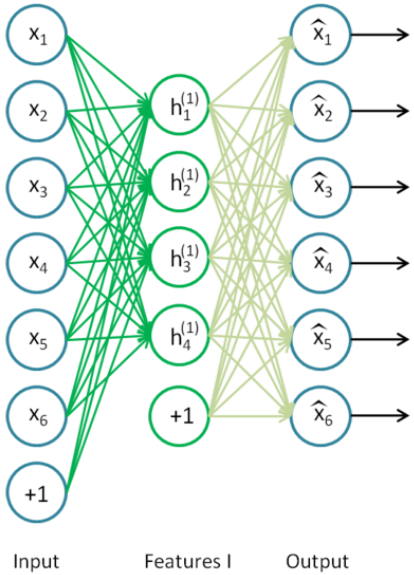
Stacked autoencoders

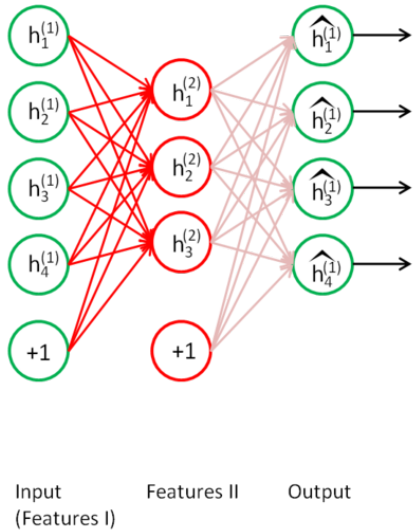


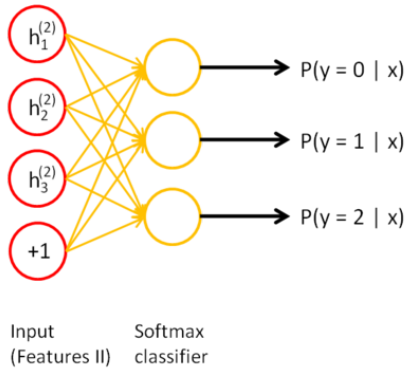
- *Stacked Autoencoder (SAE)*: stos kilku warstw autoenkoderów ułożonych jedna na drugiej
- każda warstwa autoenkodera jest trenowana osobno, najczęściej w trybie uczenia nienadzorowanego, aby nauczyć się coraz bardziej abstrakcyjnych reprezentacji danych

☞ *Deep Clustering with Convolutional Autoencoder*

Inicjalizacja głębokich sieci za pomocą SAE



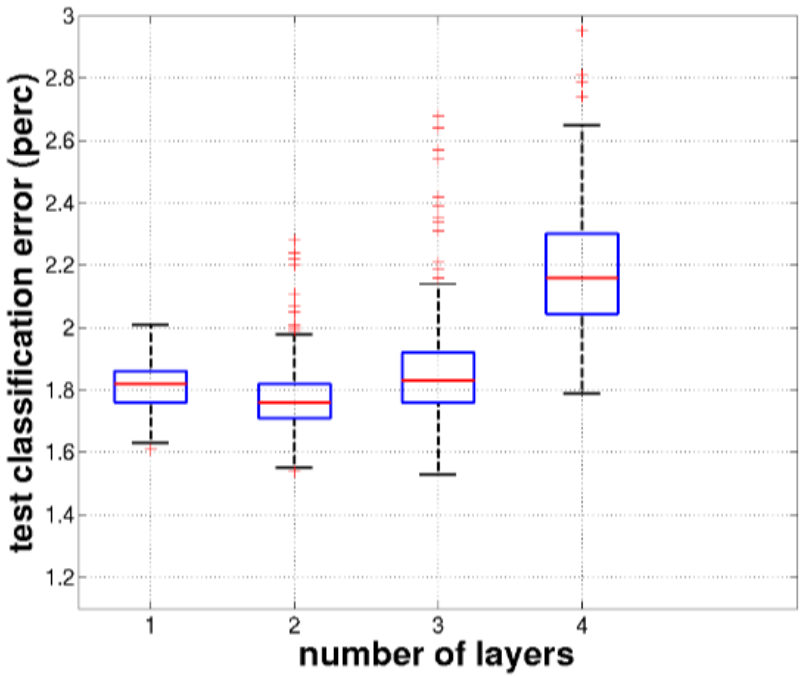


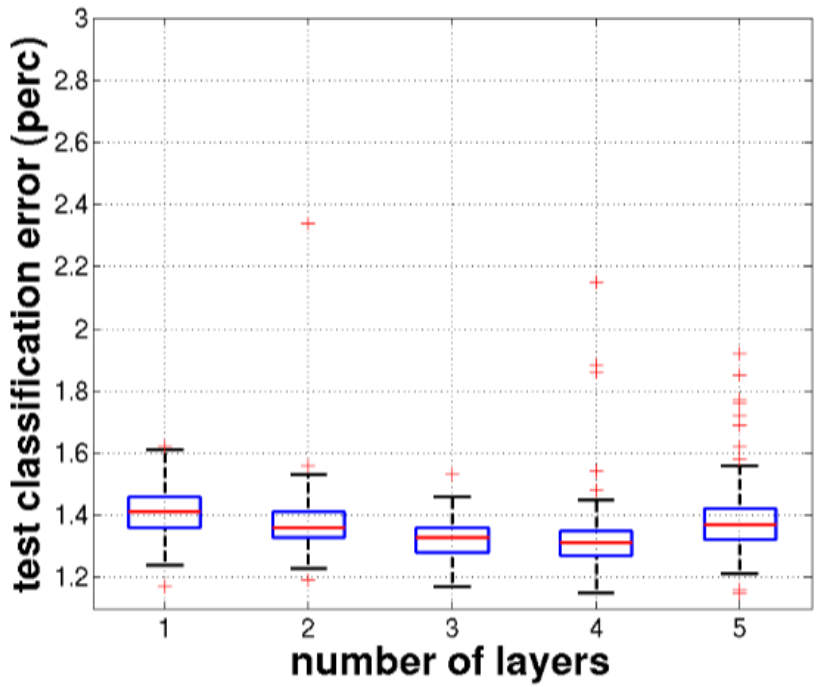


1. *Greedy layer-wise pre-training*:
   - trening nienadzorowany autoenkodera wsteczną propagacją
   - usuwamy warstwę wyjściową (dekoder) i uczymy kolejny AE, którego wejściem jest sygnał z warstwy ukrytej poprzedniego AE przy niezmiennych wartościach wag w poprzednich warstwach
   - powtarzamy procedurę dla każdej kolejnej warstwy
2. *fine tuning*: dodajemy w pełni połączoną warstwę wyjściową (lub kilka warstw) i uczymy całą sieć w sposób nadzorowany

Pre-trening DNN

Pre-trening poprawia generalizację, działa jak regularyzacja

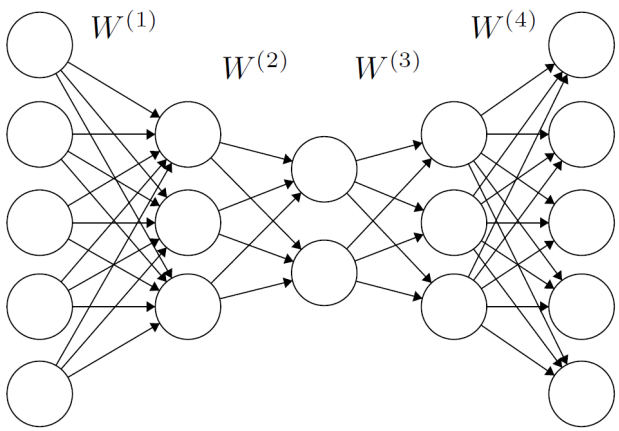




Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

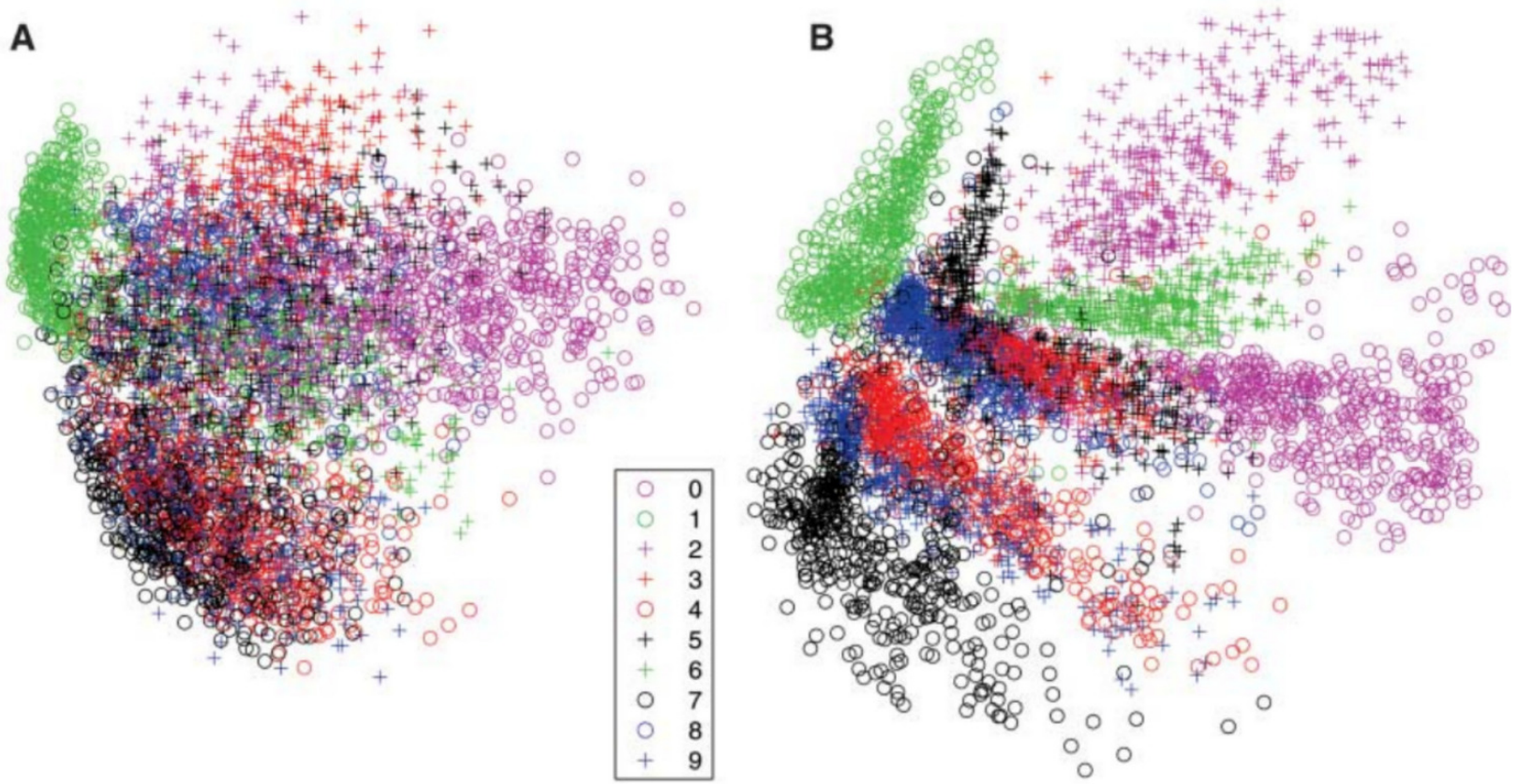
Głębokie autoenkodery

- dodatkowa warstwa ukryta zwiększa możliwości enkodera w odwzorowaniu kodu (uniwersalny aproksymator jest w stanie nauczyć się dowolnego kodowania)
- dodanie warstw pozwala zmniejszyć złożoność reprezentacji trudnych problemów
- głębokie AE pozwalają uzyskać większą kompresję (Hinton 2006)
- niekompletne głębokie AE - żadna warstwa ukryta nie powinna być mniejsza niż warstwa kodująca



Projekcja do 2D (dwa neurony kodujące)

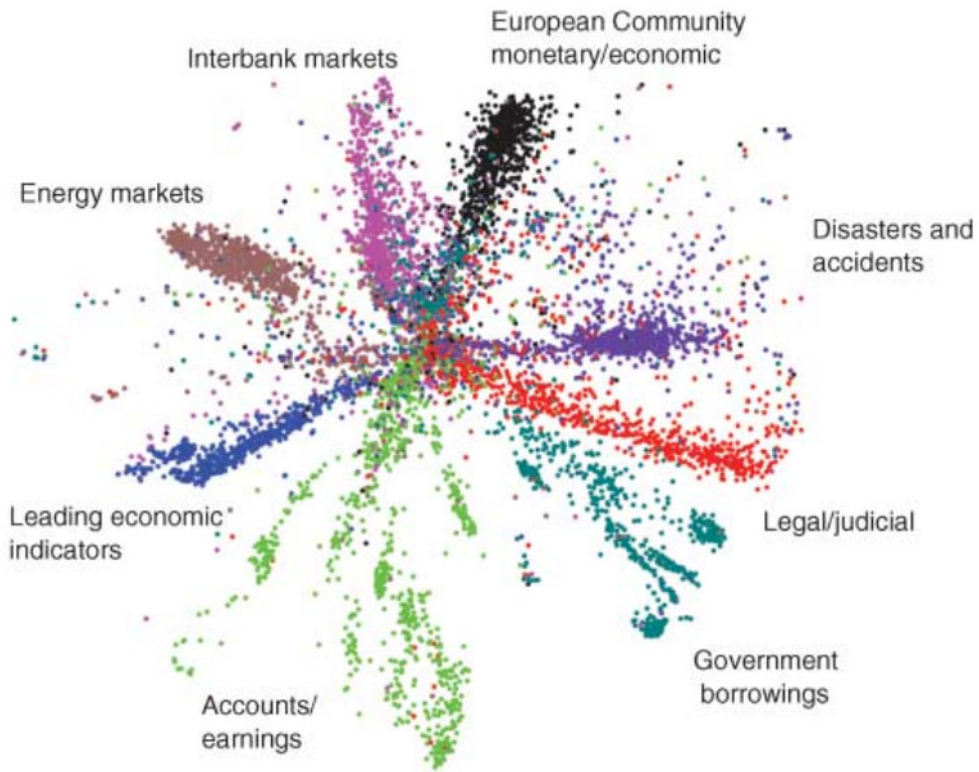
MNIST: PCA vs. Deep AE (784-1000-500-250-2)



G. Hinton, R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science 2006

Projekcja do 2D (dwa neurony kodujące)

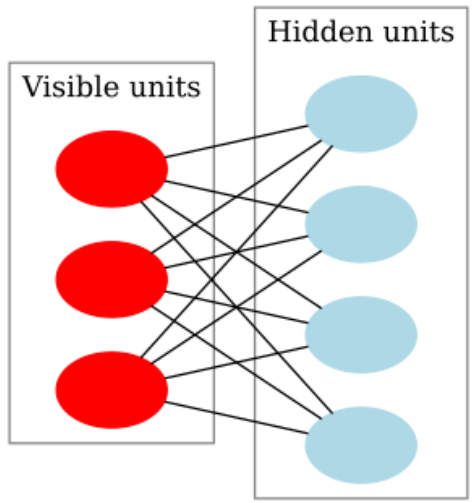
Wizualizacja dokumentów: architektura 2000-500-250-125-2



G. Hinton, R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science 2006

Restricted Boltzmann Machine

- **Restricted Boltzmann Machines (RBM)** (Hinton 2000), wcześniej Harmonium (P. Smolensky, 1986), to szczególny przypadek Maszyny Boltzmanna
- stochastyczna sieć neuronowa modelująca rozkład prawdopodobieństw nad zbiorem swoich wejść
- jest ograniczona (*restrictive*), ponieważ nie posiada połączeń między jednostkami w tej samej warstwie (brak połączeń wewnątrz warstw)
- algorytm uczenia: **dywergencja kontrastowa** (*contrastive divergence*)



Źródło grafiki: Wikipedia

Budowa RBM

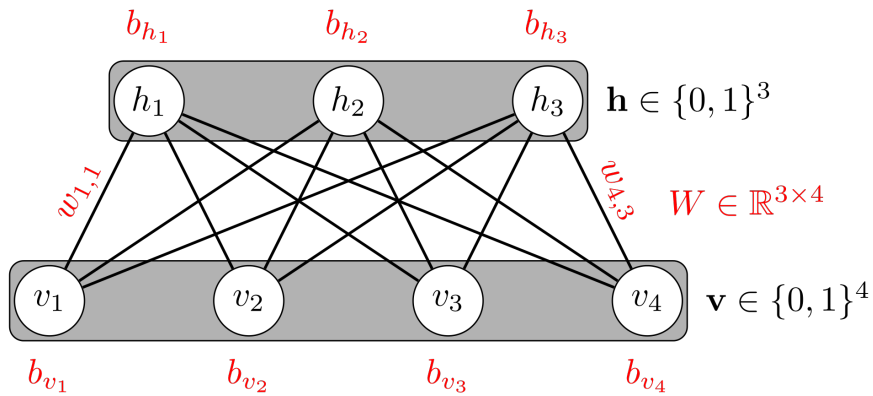
- binarne jednostki umieszczone w dwóch warstwach: widzialnej **v** i ukrytej **h**
- sygnał wejściowy podawany do jednostki widzialnej $\mathbf{v} = \mathbf{x}$
- energia układu

$$E(\mathbf{x}, \mathbf{h}) = -\sum a_i x_i - \sum b_i h_i - \sum\sum x_i w_{ij} h_j$$

gdzie w_{ij} to wagi połączeń między warstwami, a_i i b_i to wyrazy wolne związane z warstwą widzialną i ukrytą

- prawdopodobieństwo rozkładu

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{x},\mathbf{h})}$$



src: https://kwonkyo.wordpress.com/

RBM

- rozkład brzegowy

$$P(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}$$

- brak połączeń wewnątrz warstw, więc stany w warstwach są niezależne. Prawdopodobieństwo aktywacji:

$$P(\mathbf{h}|\mathbf{x}) = \prod_{j=1}^{n} P(h_j|\mathbf{x}), \qquad P(\mathbf{x}|\mathbf{h}) = \prod_{i=1}^{m} P(x_i|\mathbf{h})$$

$$P(h_i = 1|\mathbf{x}) = \sigma(\mathbf{w}_i\mathbf{x} + b), \qquad P(x_k = 1|\mathbf{h}) = \sigma(\mathbf{h}^T\mathbf{w}_k + c)$$

- model generuje próbki widocznych danych przez iteracyjne próbkowanie jednostek ukrytych i widocznych (*Gibbs sampling*)

Contrastive divergence CD

- cel treningu: maksymalizacja prawdopodobieństwa rozkładu danych $P(\mathbf{x})$

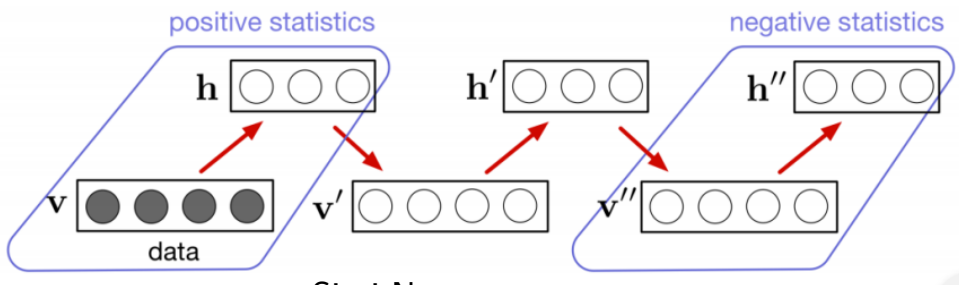
$$L = \frac{1}{N} \sum \log P(\mathbf{x})$$

- algorytm optymalizacji *Contrastive divergence* (Hinton)
  - dla każdego $\mathbf{x}$ wejściowego wygeneruj $\mathbf{x}'$ z k krotnego samplowania Gibbsa
  - aktualizacja

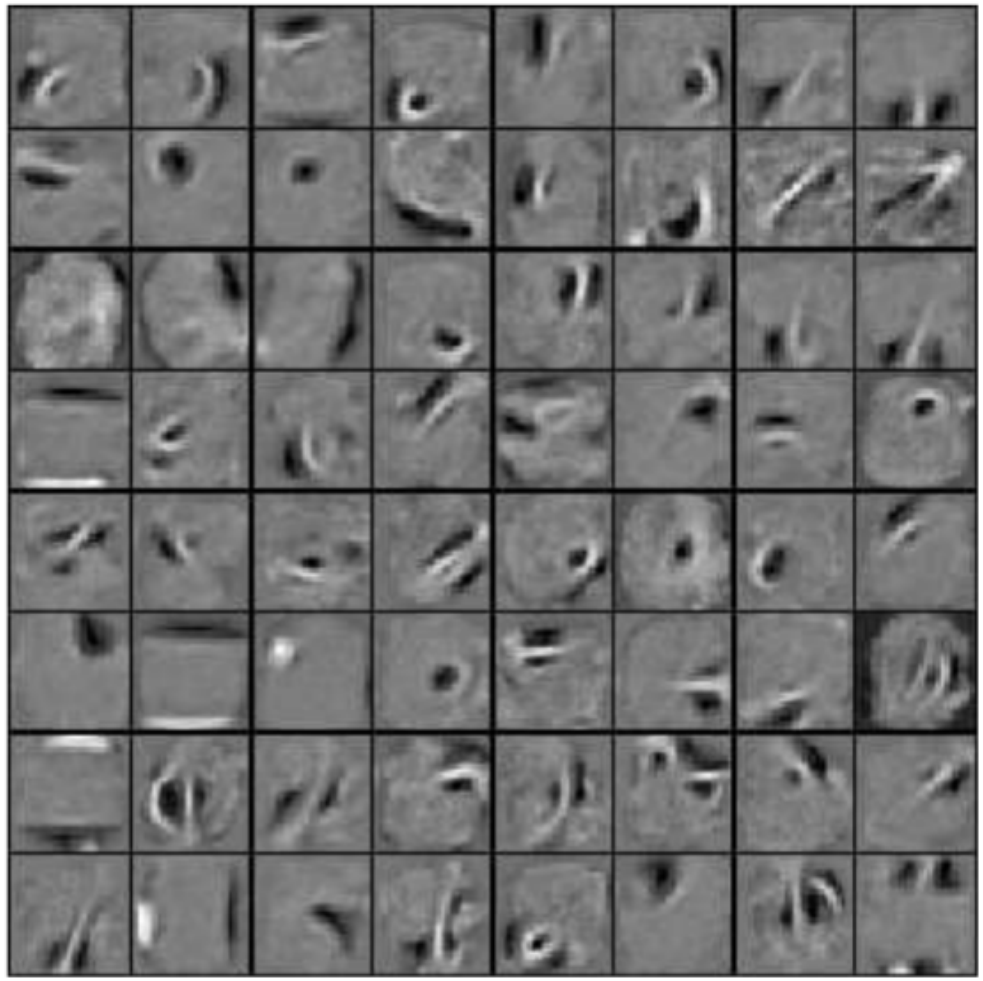
$$\Delta \mathbf{W} = \epsilon(\mathbf{x}\mathbf{h}^T - \mathbf{x}'\mathbf{h}'^T)$$

$$\Delta \mathbf{a} = \epsilon(\mathbf{x} - \mathbf{x}'), \qquad \Delta \mathbf{b} = \epsilon(\mathbf{h} - \mathbf{h}')$$

- RBM uczy się przybliżać rozkład danych, minimalizując różnicę między obserwowanymi a rekonstruowanymi próbkami
- w praktyce $k = 1$ (pojedyncze próbkowanie) daje dobre rezultaty



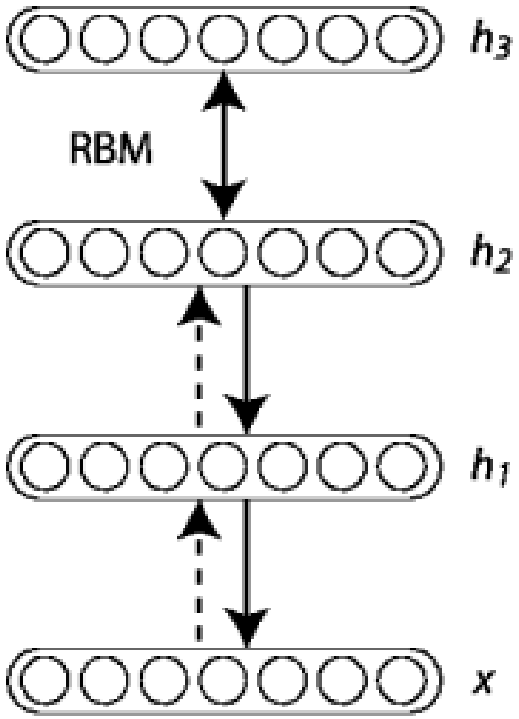
Filtry RBM na MNIST



Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

Deep Belief Networks (DBN)

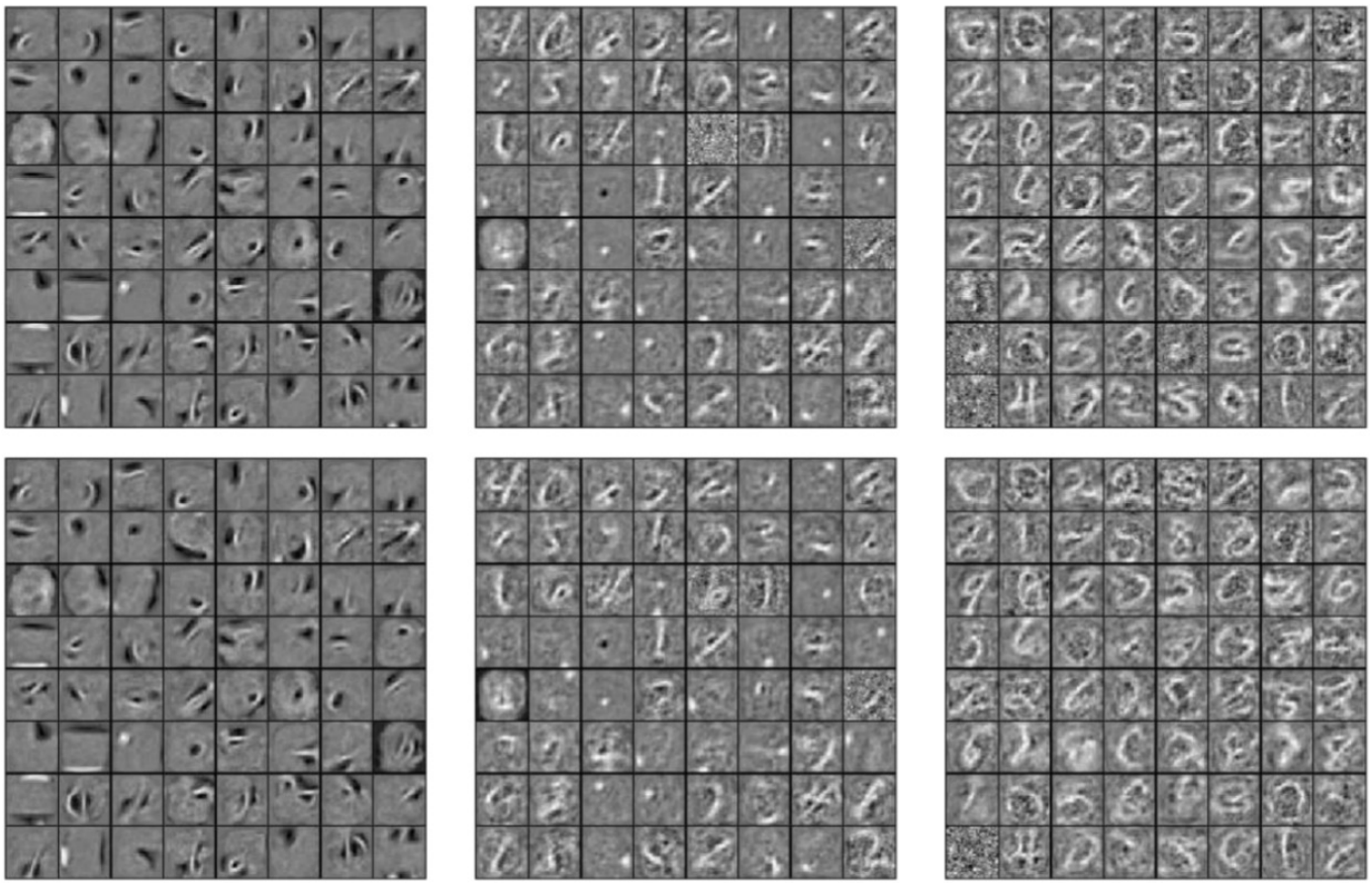
- DBN (Teh, Hinton 2006) wielowarstwowa sieć złożona z nałożonych na siebie RBM
- uczy się głębokiej, hierarchicznej reprezentacji danych, każda kolejna warstwa analizuje sygnał widoczny w warstwie ukrytej poprzedniego RBM
- uczenie zachłanne: kolejne warstwy dodawane i uczone pojedynczo
- DBN to pierwszy efektywny model głęboki
- to nie jest sieć jednokierunkowa



src: http://deeplearning.net/tutorial/DBN.html

Filtry DBN na MNIST

pre-training

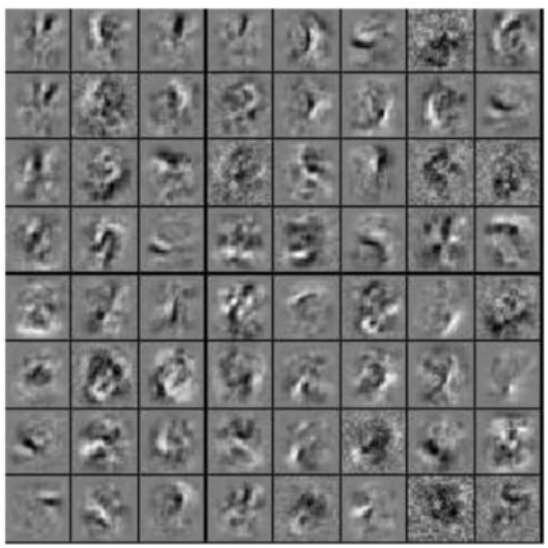
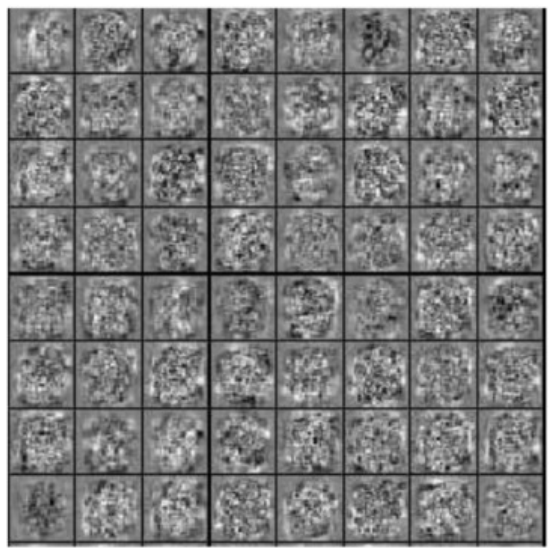
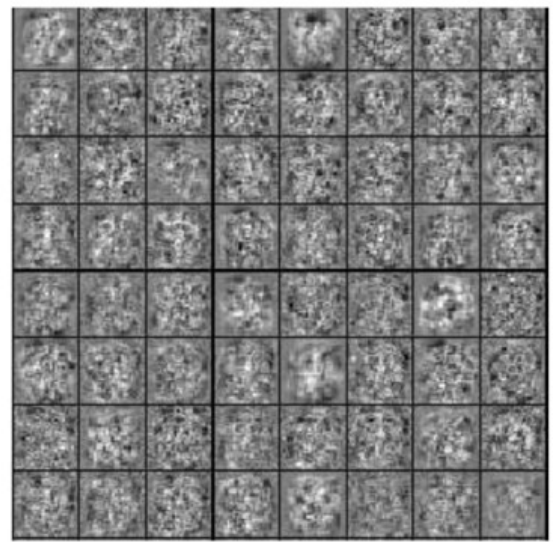


fine tune

Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

Filtry DBN na MNIST

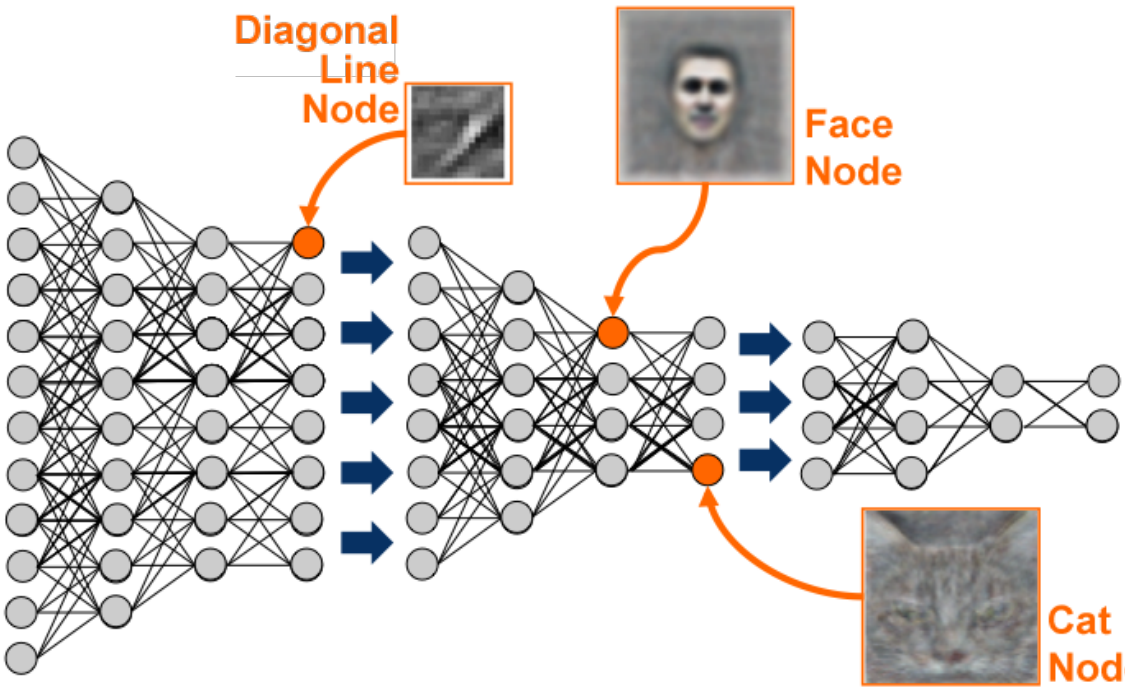
without pre-training

Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

Czym jest kot?

Sieć DBN trenowana na obrazach 200x200 z 10M losowych nagrań wideo z YouTube. Dane bez etykiet - nie było żadnego celu treningu. Trening rozproszony na 1000 maszynach (16K CPU) przez 3 dni.

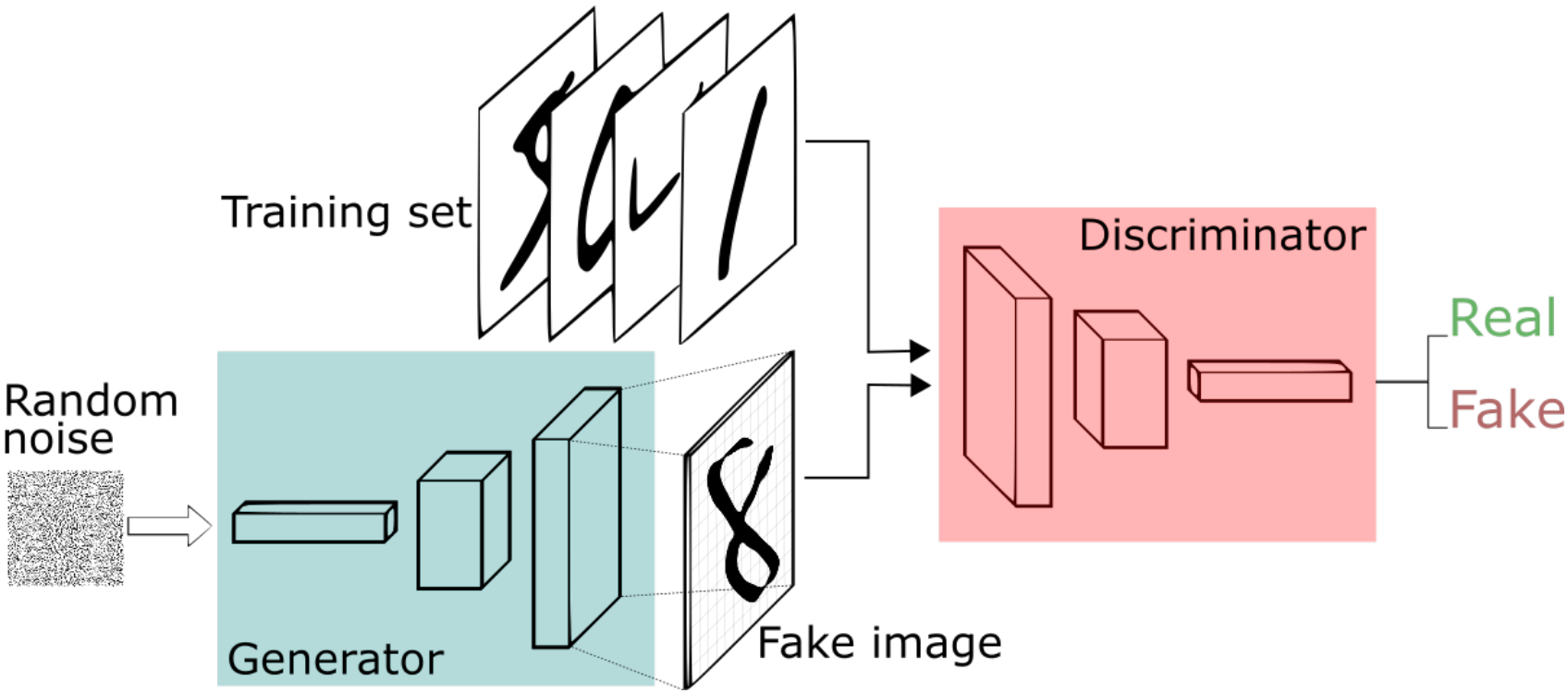


Pojedyncze neurony reagują na prezentację twarzy lub kota.

Le, 2013, Deep Belief Network

Generative Adversarial Network (GAN)

Goodfellow 2014



src: https://deeplearning4j.org/generative-adversarial-network

Generative Adversarial Network (GAN)

- GAN (Goodfellow 2014) - dwie sieci (generująca i oceniająca) rywalizujące ze sobą w grze o sumie zerowej
- model GAN uczy się generować dane o właściwościach zbliżonych do zbioru treningowego, np. zdjęć o realistycznych cechach
- sieć **oceniająca** (dyskryminująca, np. CNN) stara się odróżnić prawdziwy sygnał od wygenerowanego przez sieć generującą
- sieć **generująca** (np. sieć dekonwolucyjna) tworzy sygnał z pewnego rozkładu starając się „oszukać" sieć oceniającą, trening dąży do maksymalizacji błędu dyskryminacji
- obie sieci uczone wsteczną propagacją, sieć generująca tworzy coraz bardziej realistyczne obrazy, sieć oceniająca specjalizuje się w rozróżnianiu coraz subtelniejszych różnic pomiędzy obrazami prawdziwymi i wygenerowanymi

Trening GAN

Funkcja kosztu dyskryminatora - binarna entropia krzyżowa (spodziewana odpowiedź 1 dyskryminatora dla obrazu prawdziwego)

$$L_D = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\mathbf{x})} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_z(\mathbf{z})} \log(1 - D(G(\boldsymbol{z})))$$

gdzie $p_{data}(\mathbf{x})$ rozkład danych prawdziwych, $p_z(\mathbf{z})$ rozkład danych generowanych

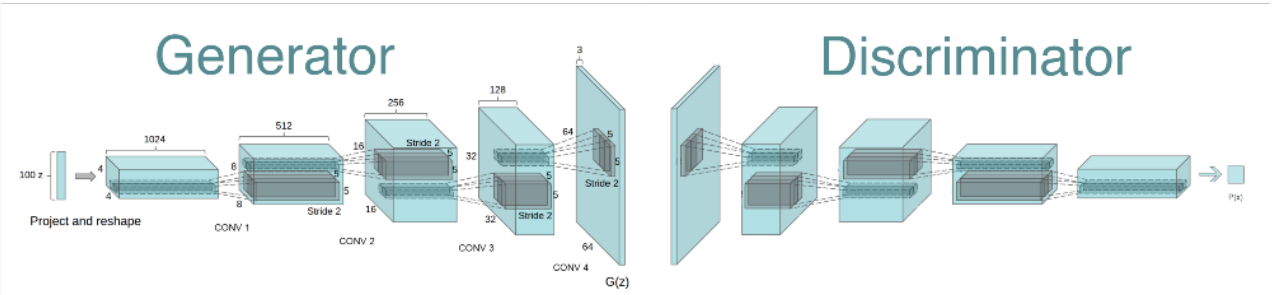
Funkcja kosztu generatora

$$L_G = -\frac{1}{2}\mathbb{E}_z \log D(G(\boldsymbol{z}))$$

maksymalizuje prawdopodobieństwo popełnienia błędu przez dyskryminator (spodziewana odpowiedź 1 dyskryminatora dla obrazu fałszywego)

DCGAN - Deep Conv GAN

Generowanie obrazów za pomocą splotowej sieci GAN

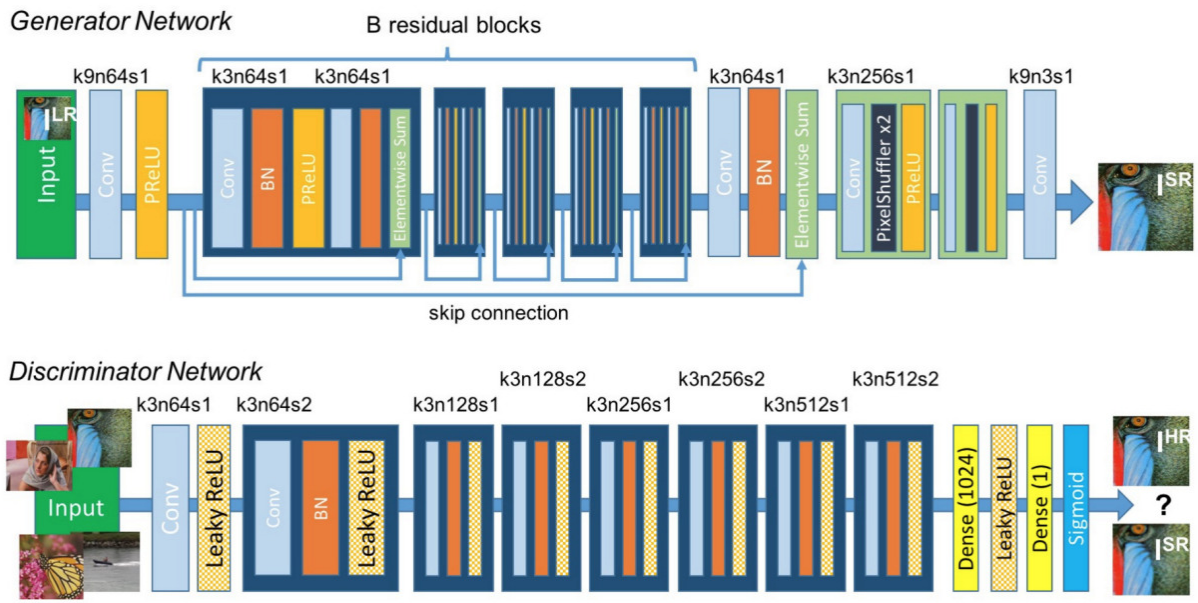




Radford, A., et al. (2015) Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

SRGAN - super resolution GAN



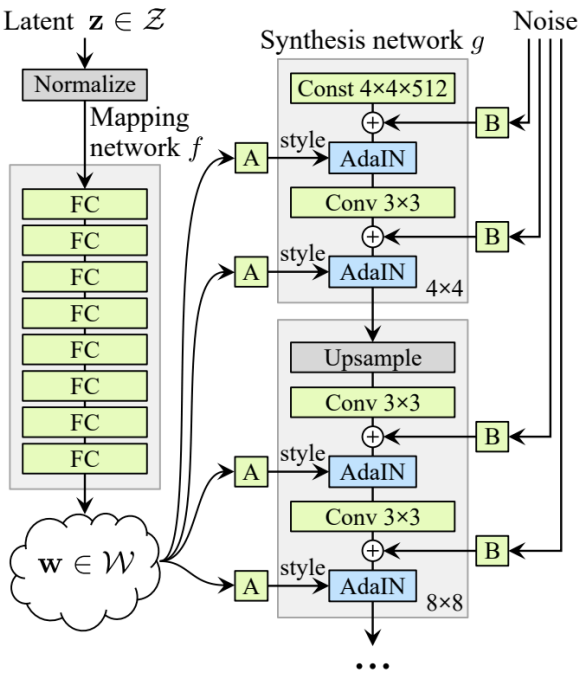


Rys: ☞ *Understanding and building Generative Adversarial Networks(GANs)*

Figure 4: Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer.

StyleGAN

Generator





Demo: ☞ Generator twarzy

T. Karras, S. Laine, T. Aila (2019) A Style-Based Generator Architecture for Generative Adversarial Networks

Inne zastosowania

- rekonstrukcja obrazów 3D ze zdjęć
- generowanie tekstu, synteza mowy, ...
- modyfikacje obrazów: zmiana twarzy, mimiki, fryzury, postarzanie osób na zdjęciach
- transfer stylu
- generowanie scen w grach wideo, zwiększanie rozdzielczości tekstur
- bezpieczeństwo: podnoszenie skuteczności algorytmów w wykrywaniu ataków cybernetycznych, wykrywanie fałszerstw, wykrywanie anomalii
- generowanie dane na potrzeby uczenia maszynowego
- generowanie obrazów z tekstu, generowanie nagrań wideo
- ☞ Curated list of awesome GAN applications and demo

Mechanizm uwagi

Transformery

Standardowa warstwa z uwagą

Wejście: dwie sekwencje $X \in \mathbb{R}^{T \times D}$ i $X' \in \mathbb{R}^{T' \times D}$, gdzie D to wymiar reprezentacji a T i T' to długości sekwencji

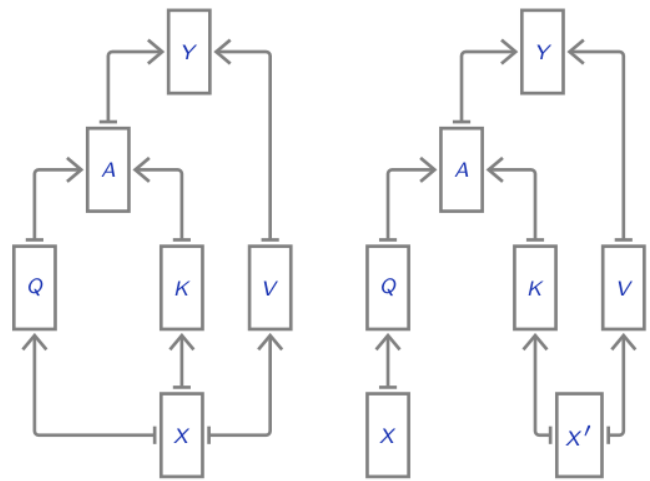
Projekcja do sekwencji zapytań Q (*queries*), K kluczy (*keys*) i V wartości (*values*)

$$Q = XW_Q \qquad K = X'W_K \qquad V = X'W_V$$

gdzie $W_Q \in \mathbb{R}^{D \times d_k}$, $W_K \in \mathbb{R}^{D \times d_k}$, $W_V \in \mathbb{R}^{D \times d_v}$

Macierz uwagi $A \in \mathbb{R}^{T \times T'}$

$$A = \text{softmax}_{\text{row}}\left(\frac{QK^\top}{\sqrt{d_k}}\right)$$



Sekwencja wyjściowa

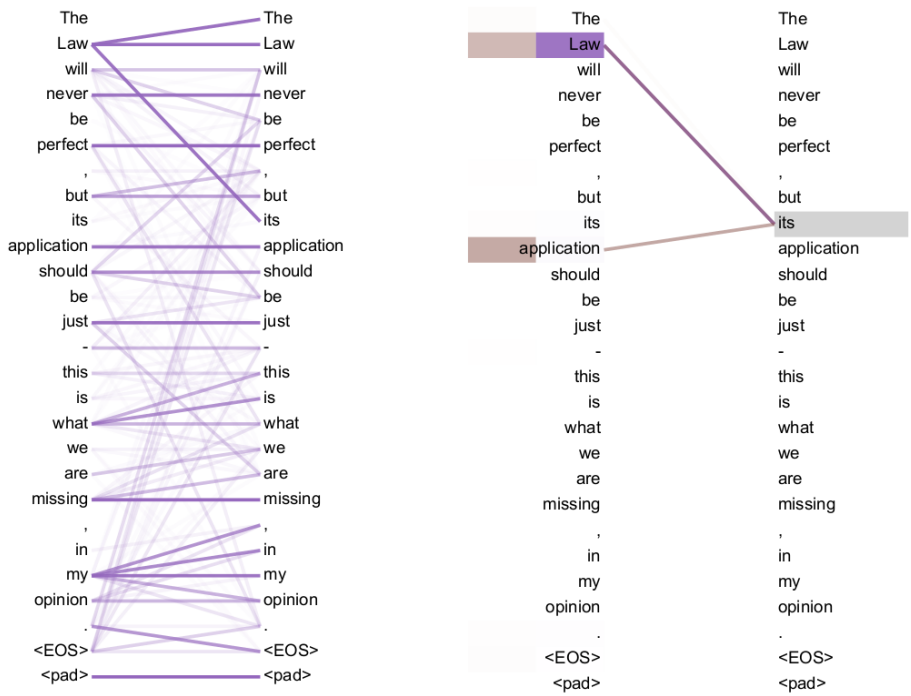
$$Y = AV$$

- samo-uwaga (*self attention*), jeżeli $X = X'$
- uwaga krzyżowa (*cross attention*), jeżeli $X \neq X'$

Uwaga to wszystko czego potrzebujesz

Uwaga (*attention*) jest uśrednieniem sekwencji **wartości** $V \in \mathbb{R}^{T' \times d_v}$ skojarzonych z sekwencją **kluczy** $K \in \mathbb{R}^{T' \times d_k}$ dopasowanych do sekwencji **zapytań** $Q \in \mathbb{R}^{T \times d_k}$

$$\text{Attention}(Q, K, V) = AV = \text{softmax}_{\text{row}}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$



☞ BertViz

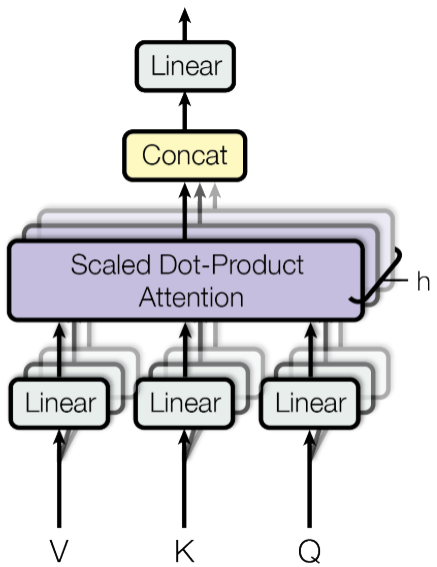
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In NIPS, 2017.

Uwaga o wielu głowach (*Multi-head attention*)

Uwaga z wieloma głowami pozwala skupić uwagę modelu na informacjach zawartych w różnych reprezentacjach (osobne projekcje Q_i, K_i, V_i) i w różnych pozycjach sekwencji

$$\text{MultiHead}(Q, K, V) = \text{Concat}\,(Y_1, \ldots, Y_{\text{h}})\, W_H$$

gdzie $W_H \in \mathbb{R}^{hd_v \times D}$



W przypadku standardowego transformera ustala się $d_k = d_v = D/h$, dzięki redukcji wymiaru każdej głowy całkowity koszt obliczeń pozostaje zbliżony do pojedynczej warstwy uwagi o pełnej wymiarowości

Standardowa operacja uwagi jest **niezmiennicza** względem permutacji kluczy i wartości:

$$\text{Attention}(Q, \sigma(K), \sigma(V)) = \text{Attention}(Q, K, V)$$

oraz **ekwiwariantna** względem permutacji zapytań, tzn. wynikowy tensor jest permutowany w ten sam sposób:

$$\text{Attention}(\sigma(Q), K, V) = \sigma(\text{Attention}(Q, K, V))$$

gdzie $\sigma(X)$ oznacza permutację wierszy macierzy

W konsekwencji *self-attention* i *cross-attention* są ekwiwariantne względem permutacji elementów w sekwencji X, a *cross-attention* jest niezmiennicza względem permutacji elementów w sekwencji X'.

Dlaczego samo-uwaga?

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

n - długość sekwencji, d - wymiar reprezentacji, k - rozmiar filtru splotu

- warstwy *self-attention* są szybsze od warstw rekurencyjnych gdy długość sekwencji n jest mniejsza od wymiarowości reprezentacji d

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In NIPS, 2017.

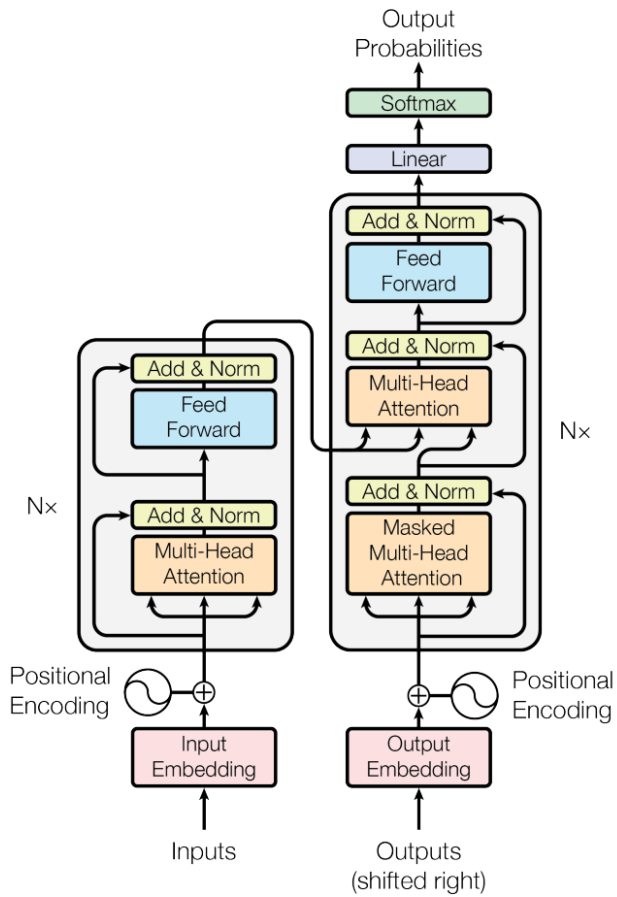
Transformer (Vaswani, 2017)

Transformery używają samo-uwagi do modelowania sekwencji w czysto jednokierunkowej sieci (bez warstw rekurencyjnych). Stosowane do budowy modeli językowych, np. BERT, GPT

Pierwotny model transformera $N = 6$
Model GPT-3 ma $N = 96$

Enkoder: bloki kodujące z samo-uwagą, połączenia skrótowe między warstwami

Dekoder: bloki dekodujące z samo-uwagą, w każdym bloku dodatkowa warstwa z uwagą krzyżową ponad wyjściem bloków enkodera, działa w trybie autoregresji - przewiduje kolejne słowo używając poprzednie słowo jako wejście



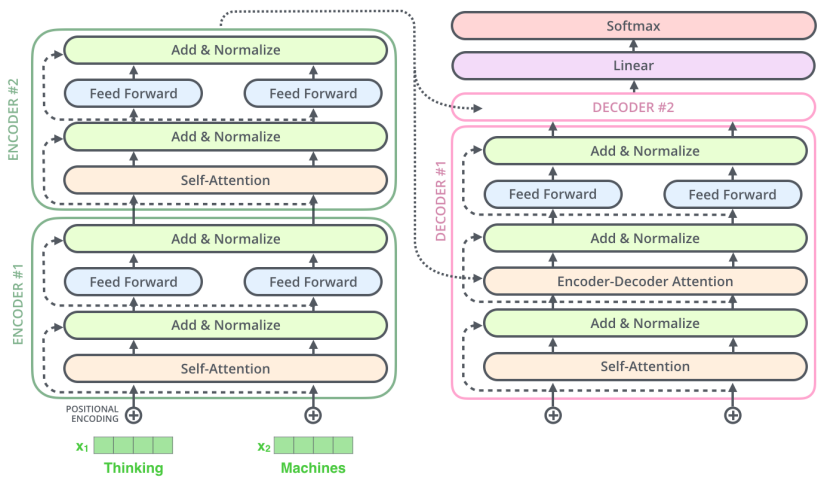
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In NIPS, 2017.

Transformer: kluczowe elementy

- mechanizm uwagi (*attention mechanism*) - pozwala modelowi skupić się na różnych częściach sekwencji wejściowej podczas generowania każdego elementu sekwencji wyjściowej
- warstwy z wieloma głowami uwagi (*multi-head attention layers*) - pozwalają modelowi skupić się na różnych aspektach sekwencji wejściowej jednocześnie
- warstwy feed-forward - pozwalają modelowi przetwarzać informacje na różnych poziomach abstrakcji
- normalizacja warstw (*layer normalization*) - pomaga stabilizować i przyspieszać trening modelu
- połączenia skrótowe (*residual connections*) - pomagają w propagacji gradientów i zapobiegają zanikowi gradientów w głębokich sieciach

Działanie transformera

- Wejście enkodera: sekwencja elementów (np. słów w zdaniu) w reprezentacji wektorowej (embedingi)
- Wejście dekodera: sekwencja embedingów przesunięta o jedno miejsce w prawo (tzw. *teacher forcing*) oraz sekwencja wyjściowa enkodera
- Dekoder generuje sekwencję wyjściową, przewidując kolejne elementy sekwencji wyjściowej na podstawie poprzednich elementów sekwencji wyjściowej i całej sekwencji wejściowej enkodera (zob. ☞ wizualizacja działania)

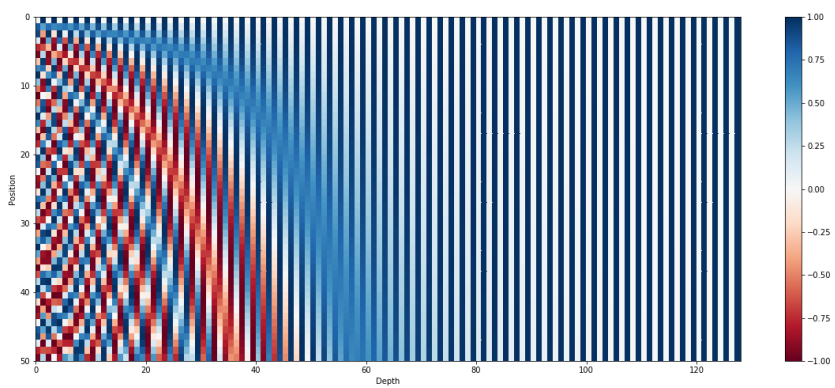


Wektor pozycyjny (Positional encoding)

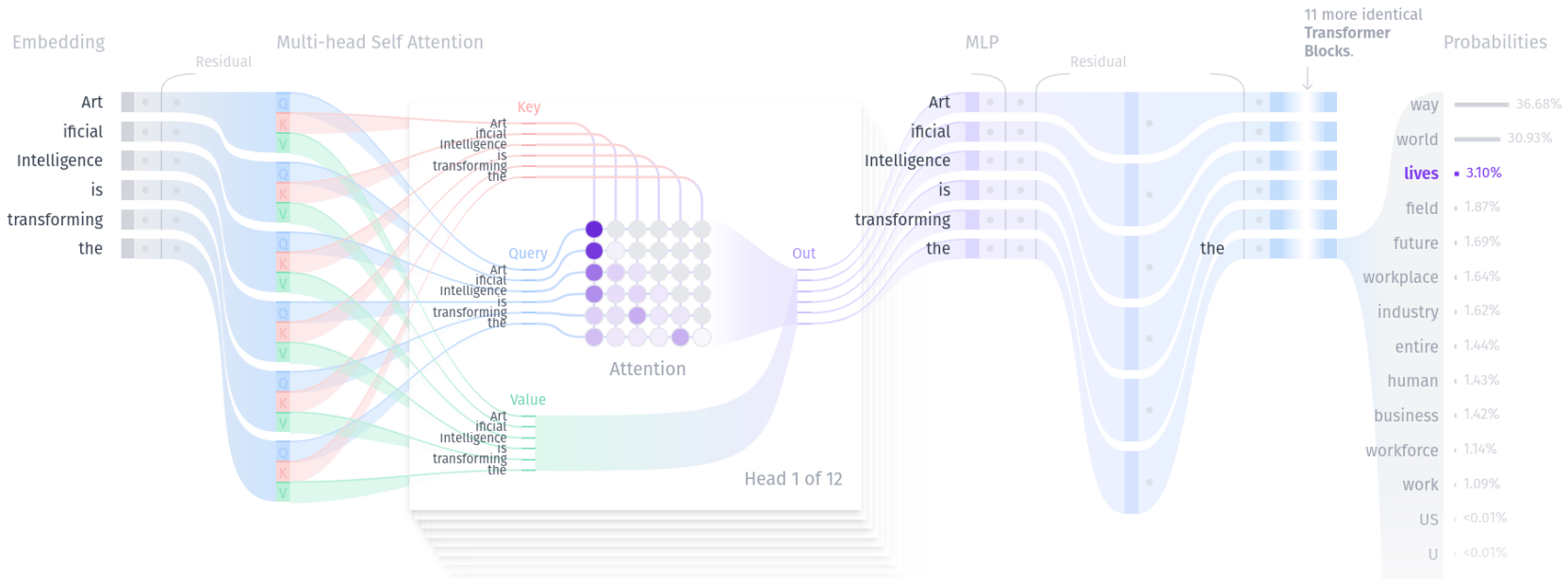
- transformer nie ma wbudowanego mechanizmu do modelowania kolejności elementów w sekwencji, więc dodaje się do wejścia enkodera i dekodera wektor pozycyjny, który jest sumowany z embedingiem elementu sekwencji
- wektor pozycyjny jest zwykle reprezentowany jako funkcja sinusoidalna o różnych częstotliwościach dla różnych wymiarów reprezentacji lub jako wektor uczony, który jest optymalizowany podczas treningu modelu
- każdy wymiar reprezentacji jest kodowany inną częstotliwością, co pozwala modelowi rozróżnić różne pozycje w sekwencji

Przykład kodowania pozycyjnego:

$$E_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$
$$E_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$



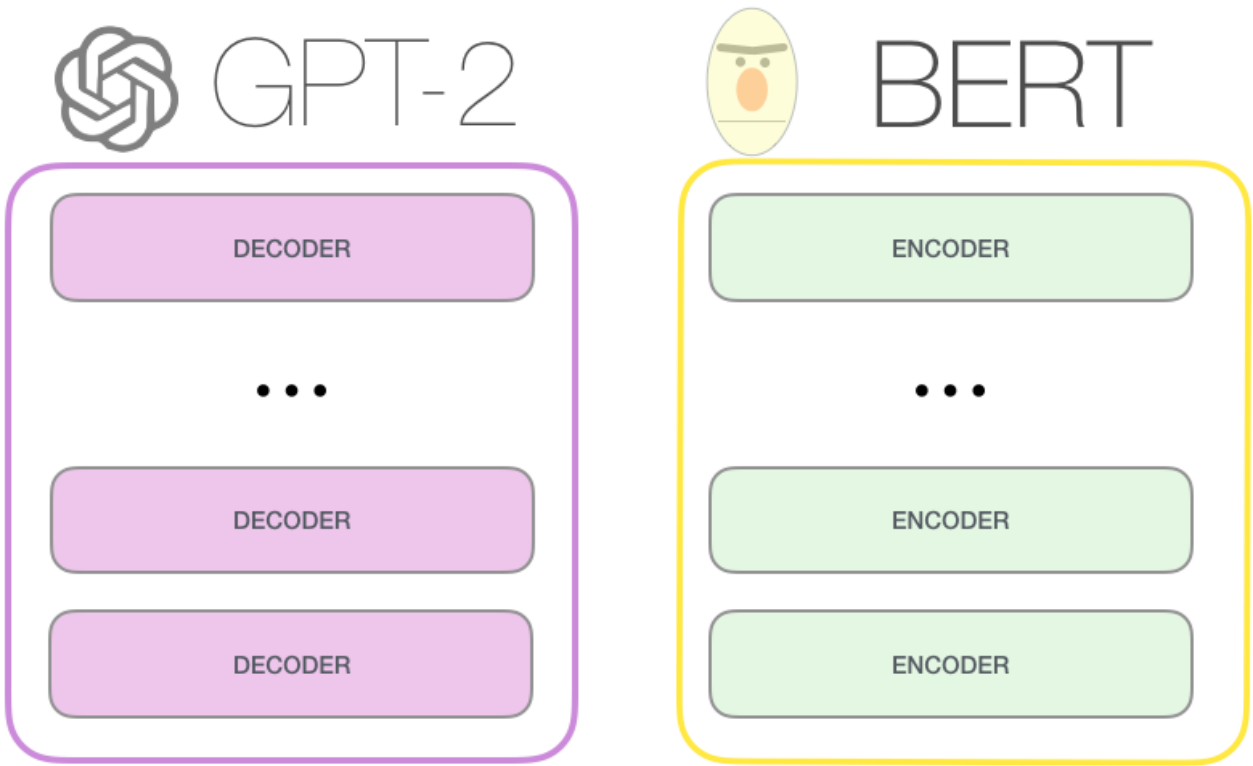
Transformer



☞ Transformer Explainer

BERT i GPT

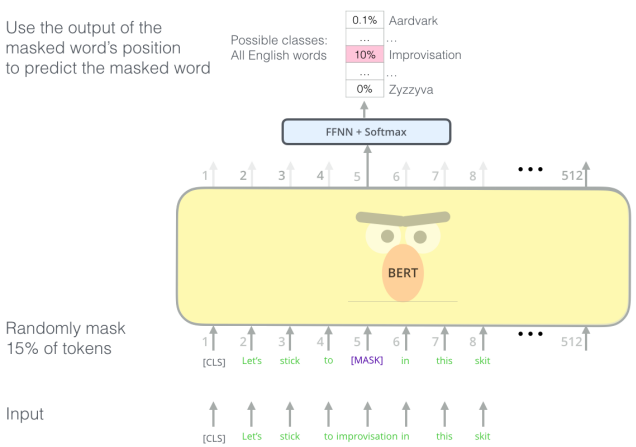
- BERT (Bidirectional Encoder Representations from Transformers) od Google (2019)
- GPT (Generative Pre-trained Transformer) od OpenAI

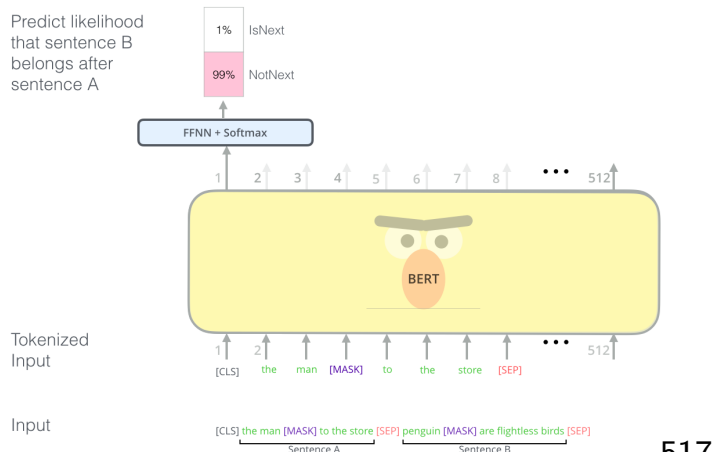


☞ *Attentional neural networks*

BERT

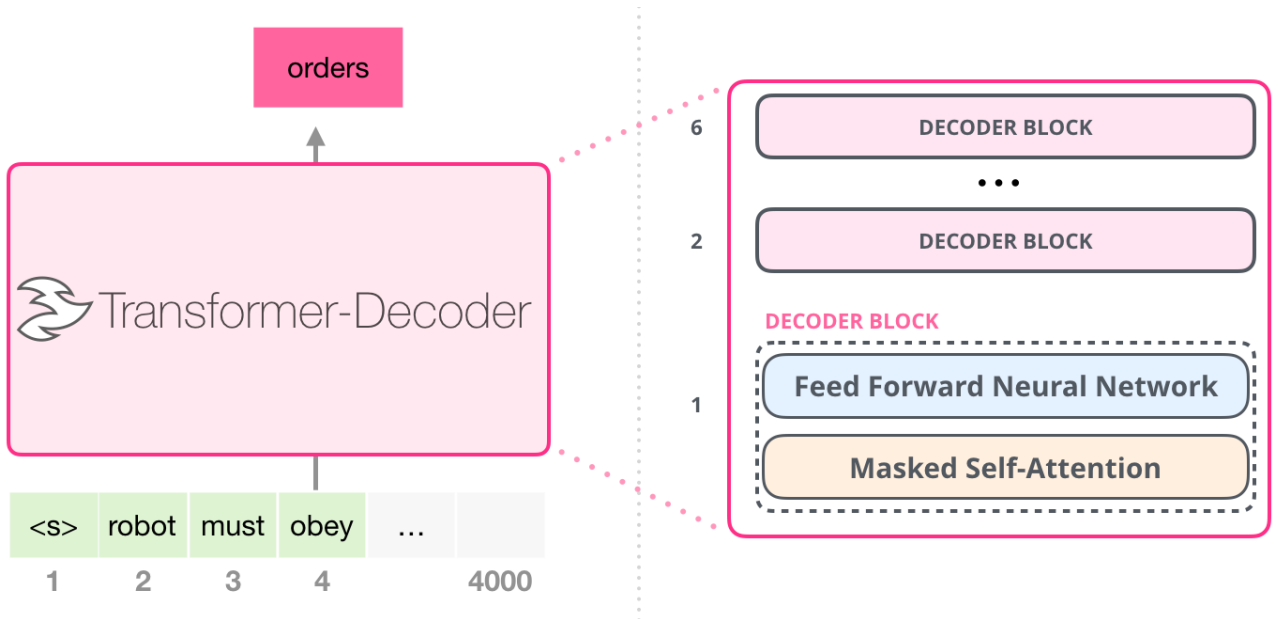
- BERT to model językowy oparty na architekturze transformera (używa tylko enkodera), zaprojektowany do zrozumienia kontekstu słów w zdaniu poprzez analizę zarówno lewego, jak i prawego kontekstu (*bidirectional*)
- maskowanie słów (*masked language modeling*), gdzie losowo maskuje się niektóre słowa w zdaniu i model musi przewidzieć, jakie to słowa
- przewidywanie relacji między zdaniami (*next sentence prediction*), gdzie model musi określić, czy dwa zdania są ze sobą powiązane





GPT

- GPT to model językowy oparty na architekturze transformera (używa tylko dekodera), zaprojektowany do generowania tekstu poprzez analizę kontekstu słów w zdaniu, ale tylko z lewej strony (*unidirectional*)
- model jest trenowany w trybie autoregresyjnym, gdzie przewiduje kolejne słowo w zdaniu na podstawie poprzednich słów



Visual Transformer (ViT)

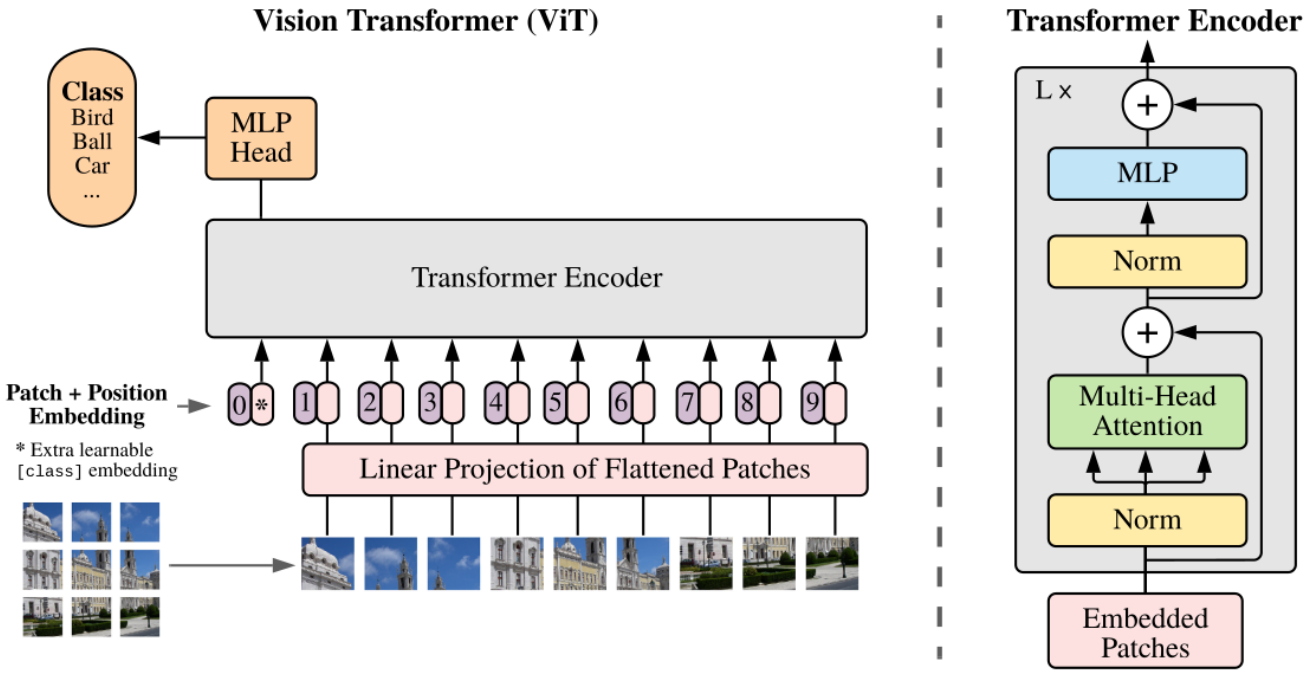


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In ICLR, 2021.

ViT

Wejście: sekwencja N łatek obrazu o rozmiarze $P \times P$ pikseli, gdzie $N = HW/P^2$ (dla obrazu o rozmiarze $H \times W$) i C kanałach (np. 3 dla RGB)

$$\mathbf{x} \in \mathbb{R}^{H \times W \times C} \qquad \longrightarrow \qquad \mathbf{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$$

gdzie $N = HW/P^2$

Liniowa transformacja do reprezentacji D wymiarowej: (patch embeddings)

$$\mathbf{z}_0 = \left[\mathbf{x}_{\text{class}};\, \mathbf{x}_p^1\mathbf{E};\, \mathbf{x}_p^2\mathbf{E};\, \cdots ;\, \mathbf{x}_p^N\mathbf{E}\right] + \mathbf{E}_{pos}$$

Enkoder transformera

$$\mathbf{z}'_\ell = \text{MSA}\left(\text{LN}\left(\mathbf{z}_{\ell-1}\right)\right) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \ldots L$$

$$\mathbf{z}_\ell = \text{MLP}\left(\text{LN}\left(\mathbf{z}'_\ell\right)\right) + \mathbf{z}'_\ell, \quad \ell = 1 \ldots L$$

Głowa MLP klasyfikująca

$$\mathbf{y} = \text{LN}\left(\mathbf{z}_L^0\right)$$

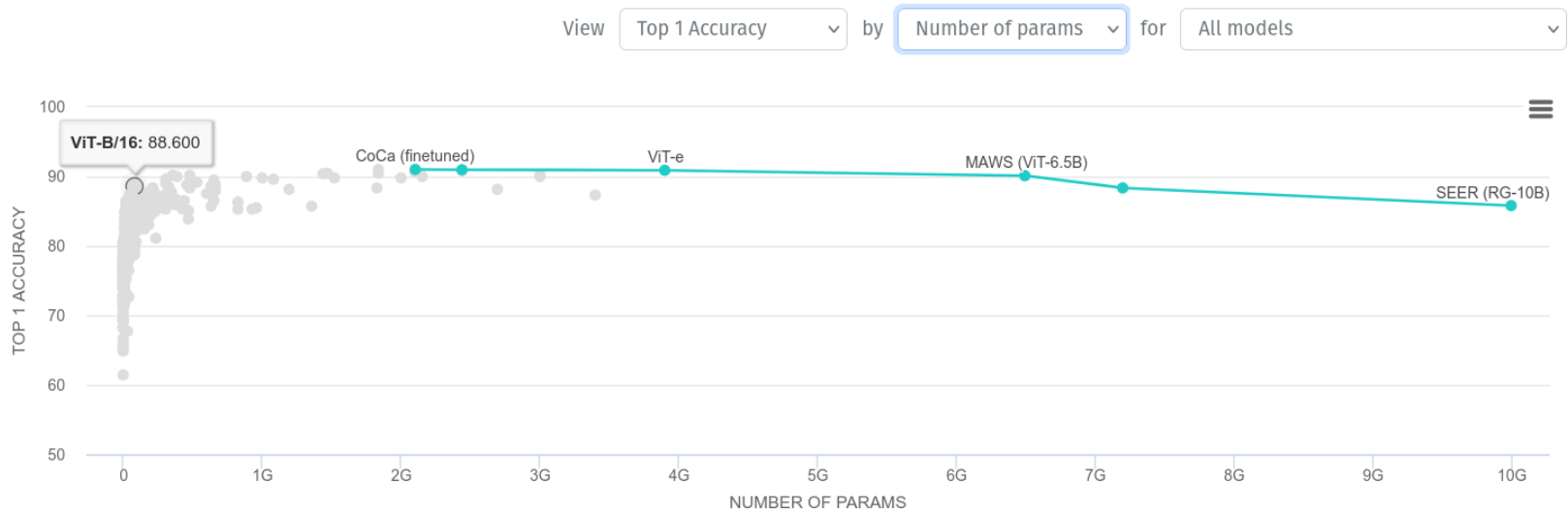
Image Classification: state of the art

Trend	Dataset	Best Model	Paper	Code	Compare
	ImageNet	CoCa (finetuned)			See all
	CIFAR-10	ViT-H/14			See all
	CIFAR-100	EffNet-L2 (SAM)			See all
	STL-10	VIT-L/16 (Spinal FC)			See all
	ObjectNet	CoCa			See all
	MNIST	Branching/Merging CNN + Homogeneous Vector Capsules			See all
	iNaturalist 2018	OmniVec2			See all
	SVHN	E2E-M3			See all
	ImageNet ReaL	Baseline (ViT-G/14)			See all
	Clothing1M	LRA-diffusion (CC)			See all

☞ Browse State-of-the-Art

Image Classification on ImageNet

Leaderboard | Community Models | Dataset



☞ Image classification on Imagenet