Sztuczne sieci neuronowe

Marek Grochowski

Plan

1. Perceptrony, sieci wielowarstwowe jednokierunkowe (MLP)
2. Metody uczenia sieci i algorytm wstecznej propagacji błędu
3. Radialne funkcje bazowe (RBF) i metody aproksymacji
4. Samoorganizacja, sieci SOM
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

Literatura I

[1] Osowski S., *Sieci neuronowe w ujęciu algorytmicznym*, Wydawnictwo Naukowo-Techniczne, Warszawa 1996

[2] Tadeusiewicz R., *Sieci neuronowe*, Akademicka Oficyna Wydawnicza RM, Warszawa 1993

[3] Ryszard Tadeusiewicz, Tomasz Gąciarz, Barbara Borowik, Bartosz Lepe, *Odkrywanie właściwości sieci neuronowych przy użyciu programów w języku C#*, Polska Akademia Umiejętności, 2008

[4] J. Żurada, M. Barski, W., *Jędruch Sztuczne sieci neuronowe*, Wydawnictwo Naukowe PWN 1996

Literatura II

[5] Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org

[6] Michael Nielsen, *Neural Networks and Deep Learning*, http://neuralnetworksanddeeplearning.com/

[7] Denny Britz, *Deep Learning Glossary*, http://www.wildml.com/deep-learning-glossary/

Inteligencja obliczeniowa

Computational Intelligence (CI) zajmuje się rozwiązywaniem problemów, które nie są efektywnie algorytmizowalne.

Problemy niealgorytmizowalne:

- zagadnienie jest zbyt złożone, np. problemy NP-trudne
- modelowany proces może zawierać trudne do zdefiniowania niejasności lub moze być stochastyczny z natury
- procesu nie da się opisać przez zrozumiałe zasady
- warunki mogą się zmieniać, algortym musi się dostosowac do nowyej sytuacji

Cechą wielu systemów CI jest rozwiązywanie zadań na podstawie znanych przykładów, uczenie się z empirycznych danych zamiast programowania rozwiązania. Systemy uczące się (machine learning, ML) oraz sztuczne sieci neuronowe (artificial neural networks, ANN) są ważnymi elementami CI.

Problemy efektywnie nieal gorytmizowalne

Problemy NP-trudne – Liczba kroków algorytmu dla złożonych sytuacji rośnie w sposób szybszy niż jakikolwiek wielomian liczby elementów (złożoności specyfikacji problemu).

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)

Definicje AI i CI (W. Duch):
CI: percepcja i sterowanie: zachowania sensomotoryczne – sieci neuronowe i uczenie maszynowe;
AI: wyższe czynności poznawcze: logika, język, rozumowanie, rozwiązywanie problemów.

Artificial Intelligence (AI) to część CI posługująca się symboliczną reprezentacją wiedzy, zajmuje się rozumowaniem, tworzeniem systemów ekspertowych.

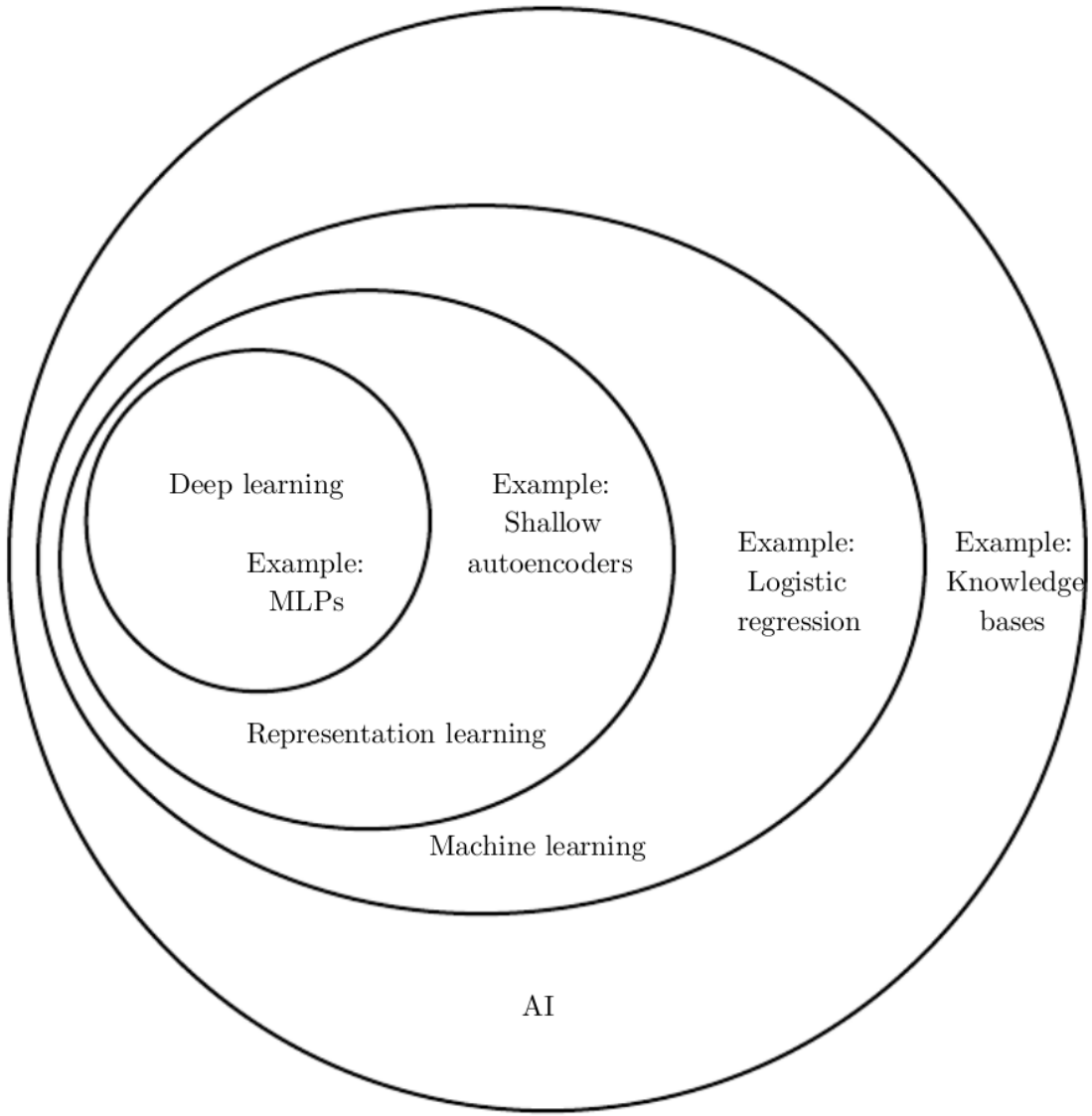
Obecnie „sztuczna inteligencja” używana jest na określenie wszystkiego, co się kojarzy z inteligencją maszyn.

Uczenie maszynowe

- **Uczenie maszynowe** (Machine learning, ML) - sytemy, które uczą się rozwiązywać problemy na podstawie dostarczonych przypadków uczących (zebranych danych)
- **Generalizacja** - dobrze wyuczony model działa poprawnie także dla przypadków, których nie było w danych treningowych
- Gdy dane się zmienią system można dostosować trenując go na nowych danych

Rysunek: https://xkcd.com/

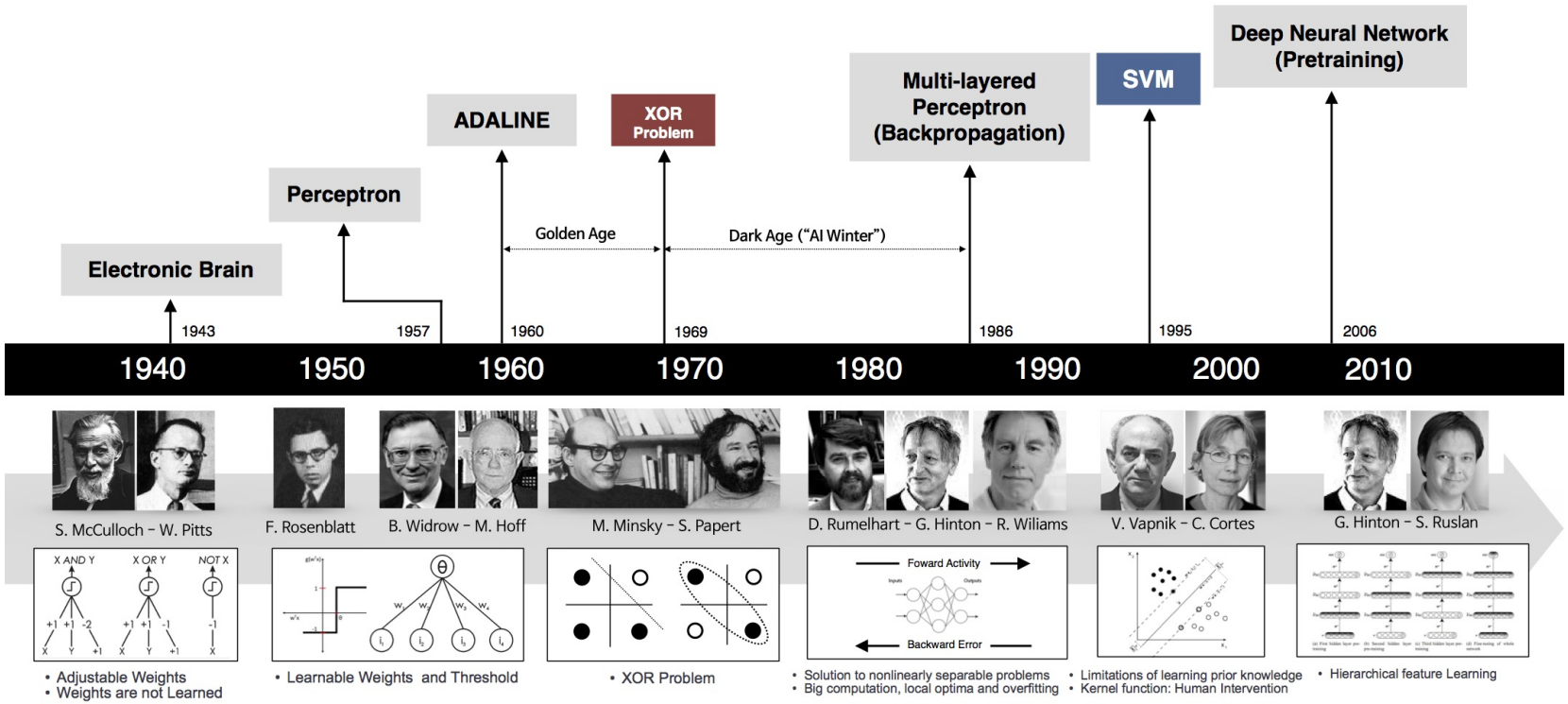
ANN i sztuczna inteligencja (AI)



Gooffellow, 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 za pomocą metod inspirowanych systemami biologicznymi -> Sztuczne Sieci Neuronowe (Artificial Neural Networks, ANN)



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

Rys historyczny I

1938 N. Rashevsky, neurodynamika - sieci neuronowe jako układ dynamiczny

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

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

1958 F. Rosenblatt, Perceptron - sieć jako funkcja matematyczna
J. von Neuman, The computer and the brain

1960 B. Widrow, M. Hoff, Adeline

1967 K. Steinbuch, E. Schmitt, Macierze uczące się

1969 M. Minsky, S. Papert, książka „Perceptrony” - wykazali ograniczenia sieci liniowych

1973 Chr. von der Malsburg, samoorganizacja w układzie nerwowym

Rys historyczny II

1982 J. Hopfield, model Hopfielda
T. Kohonen, samoorganizacja map topograficznych mózgu

1983 K. Fukushima, S. Miyake, T. Ito, neokognitron, głęboka sieć neurowa

1985 D. Ackley, G. Hinton, T. Sejnowski, maszyny Boltzmana

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

1987 G. Carpenter, S. Grossberg, model ART; W. Freeman, Chaos w mózgu

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

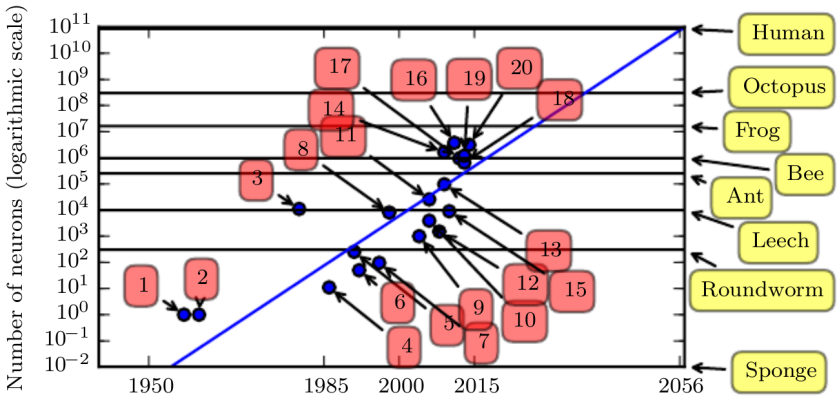
1995 V. Vapnik, SVM i „koniec ery Data Mining”

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

1998 Y. LeCun i inn, sieci konwolucyjne w analizie obrazów.

2005 GPU do sieci konwolucyjnych, bardzo duże sieci, głębokie uczenie

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 Fallside, 1991)
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)

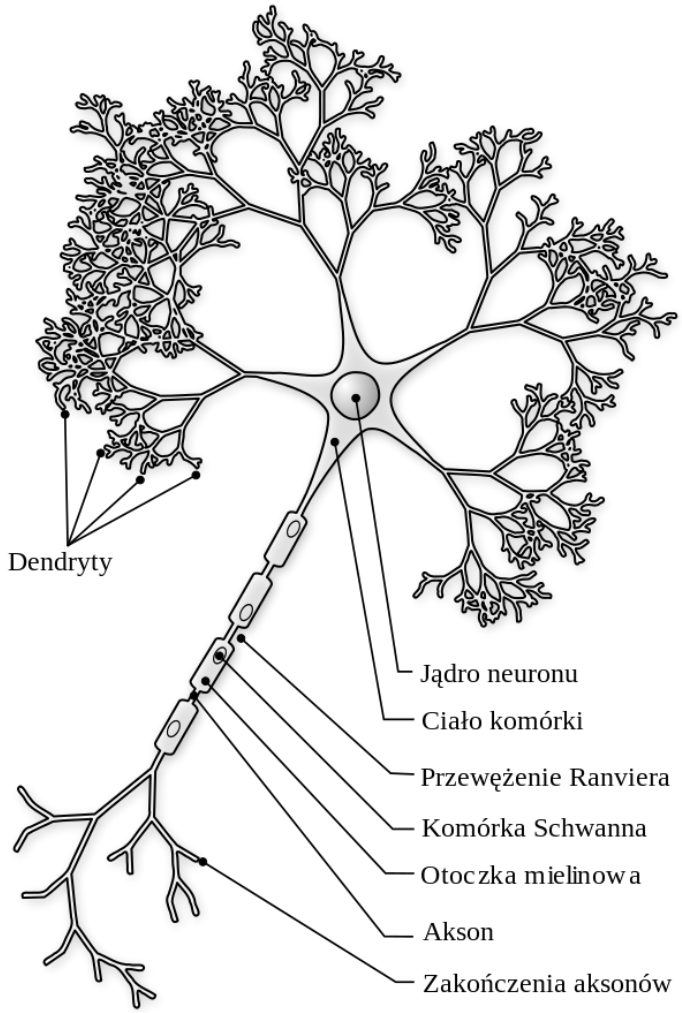
Liczba neuronów podwaja się co 2.5 roku

Gooffellow, 2016 [1]

Mózg i sieć neuronowa

- 10^{11} neuronów (100 miliardów), do 100Hz
- gęstość połączeń około 10^4 synaps na neuron
- Receproty - dostarczają sygnały wejsciowe (bodźce wewnętrzen i sygnał ze zmysłów)
- Sygnał wyjściowy z układu nerowego steruje efektorami (reakcja na bodziec)
- obliczenia równoległe
- układy biologiczne zbyt złożone - jedynie ogólne zasady organizacji i sposobu funkcjonowania układów biologicznych neuronów mogą służyć za inspirację
- Sztuczne sieci neuronowe to model matematyczny, który można widzieć jako złożenie wielu (nieliniowych) funkcji i często nie ma 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), sumuje napływające w danym momencie sygnały
- Akson (wyjście), emituje sygnał, gdy potencjał w somie osiągnie odpowiedni próg
- Synapsy - styki pomiędzy dendrytami i aksonami, pobudzenie napływające z axonu powoduje uwolnienie chemicznych neurotransmiterów, które wpływają na zachowanie połączeń przyłączonych neuronów
- Działanie neuronów biologicznych jest bardzo złożonym procesem, neurony w sztucznych sieciach są dużym uproszczeniem

☞ Schematic animation of spike generation in neural cell (J. Piersa)

Logika progowa

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

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

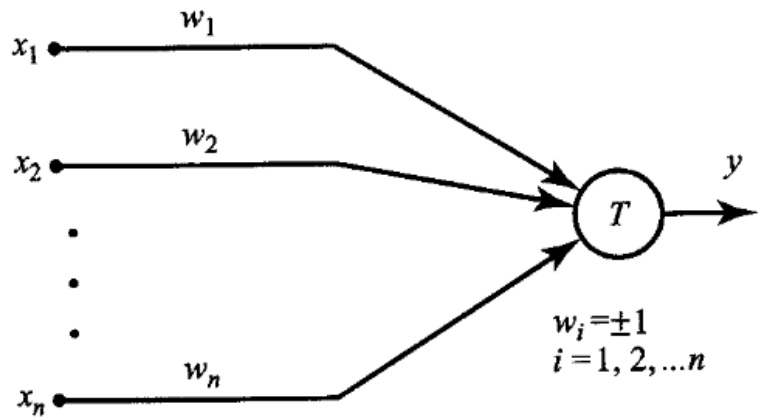
Sumowanie przestrzenne i czasowe potencjałów zgromadzonych na błonie komórkowej daje całkowitą aktywację elementu:

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

Aktywacja, razem z progiem T wzbudzenia, określa sygnał wyjściowy

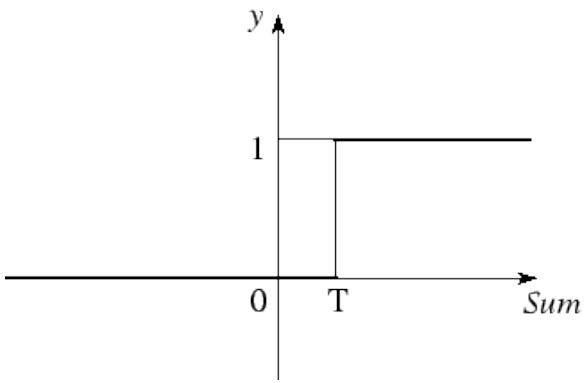
$$o(t) = f(a, T)$$

Neuron McCulloch, Pitts (1943-49)



wyjście neuronu w chwili $k+1$

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



grafika: www.wikipedia.org

Neuron McCulloch, Pitts (1943-49)

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

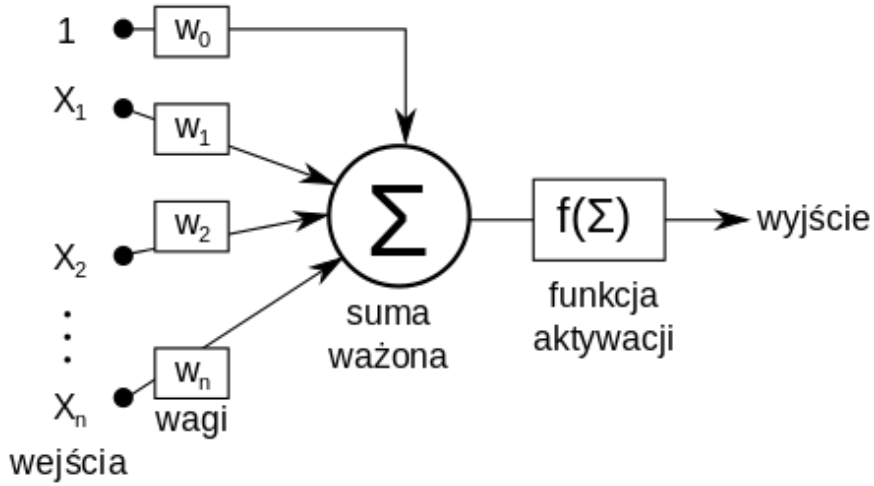
$$x_i = \{0, 1\}$$

- sygnały dochodzą przez synapsy pobudzające i hamujące

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

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

Ogólny schemat neuronu



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

gdzie

$\mathbf{x} = [1, x_1, x_2, \ldots, x_n]$

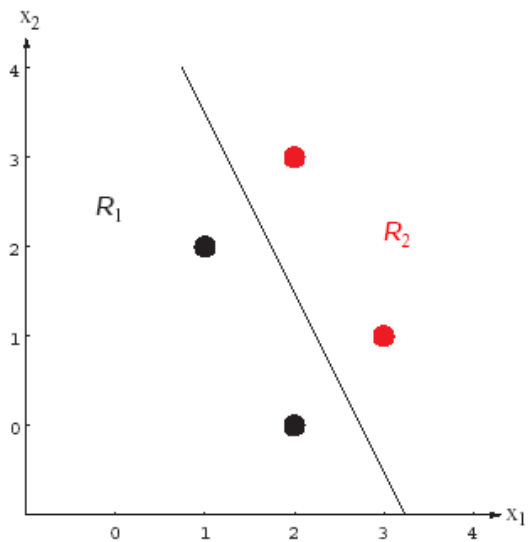
$\mathbf{w} = [w_o, w_1, w_2, \ldots, w_n]$

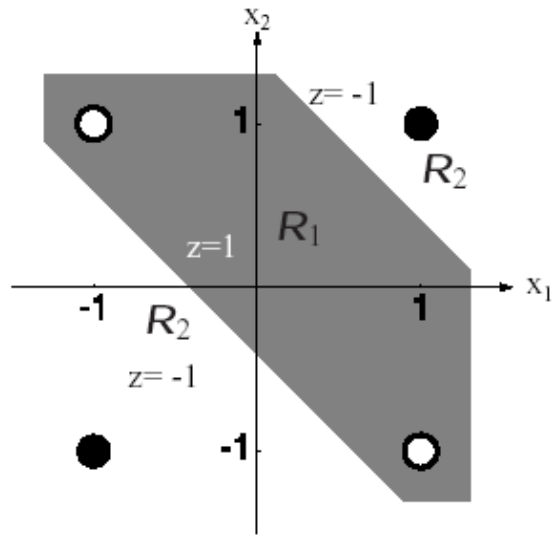
w_0 - wyraz wolny (bias)

Interpretacja geometryczna - separowalność

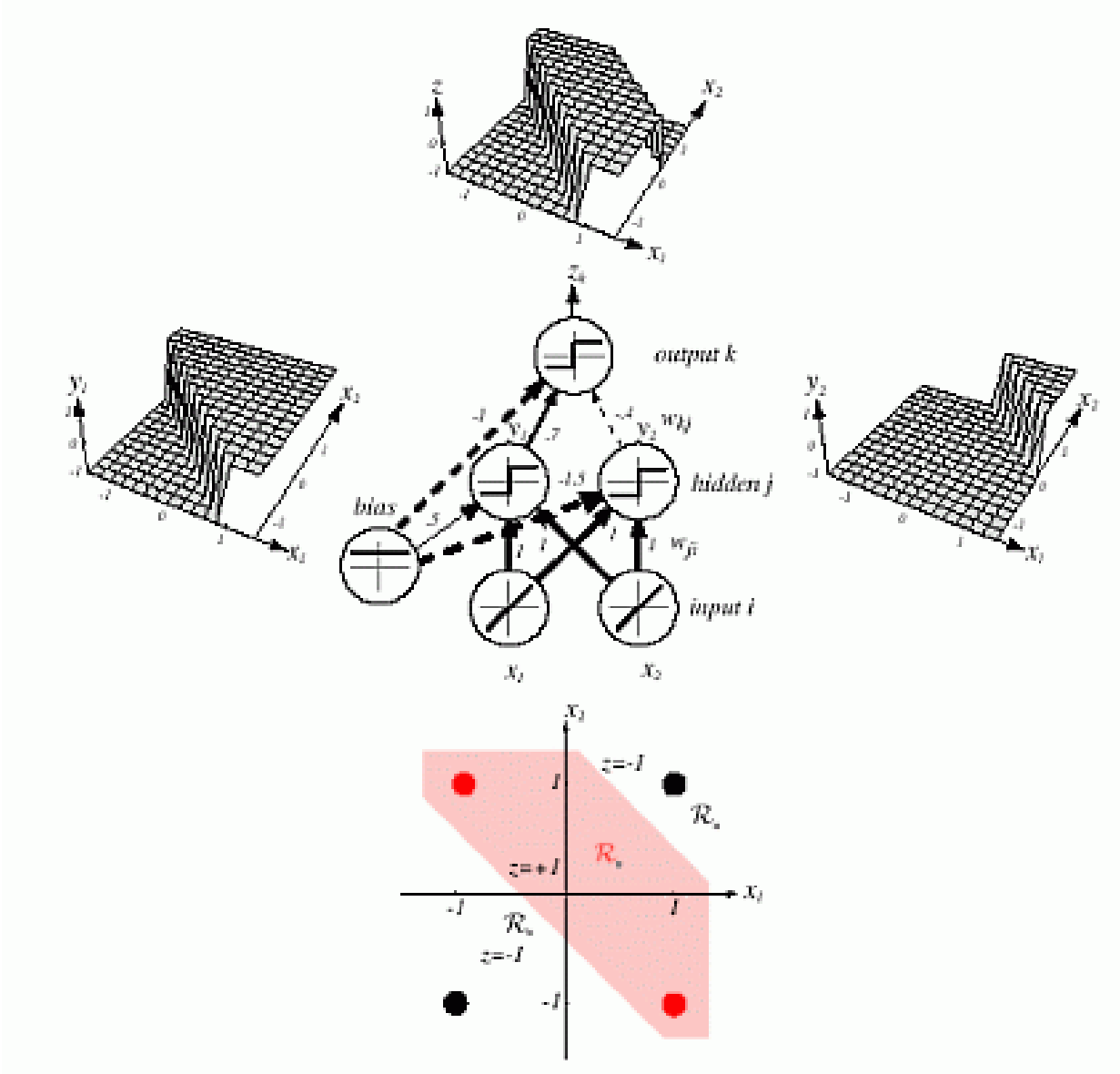
Neuron z progiem realizuje sepaowalność liniową, hiperpłaszczyzna dzieli przestrzeń na dwie części

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





XOR


output k
hidden j
bias
input i
w_{kj}
w_{ji}
x_1
x_2
z_k
y_1
y_2
z=-1
z=+1
z=-1
$\mathcal{R}_a$
$\mathcal{R}_b$
$\mathcal{R}_a$

Reguła Hebba (1949)

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

„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.”

Uczenie

Plastyczność układu (uczenie się): wagi synaptyczne w_i zmieniają się powoli w czasie.

Reguła Hebba

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

Reguła delta (Widrowa-Hoffa)

$$\Delta w_i = \eta(y - o) x_i$$

gdzie y to sygnał nagrody

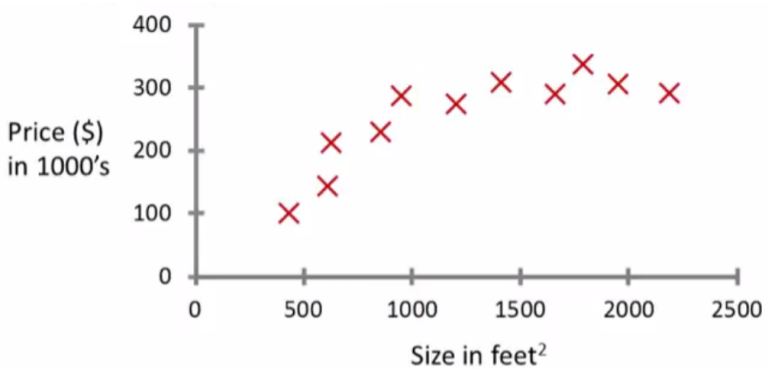
Rodzaje ucznia

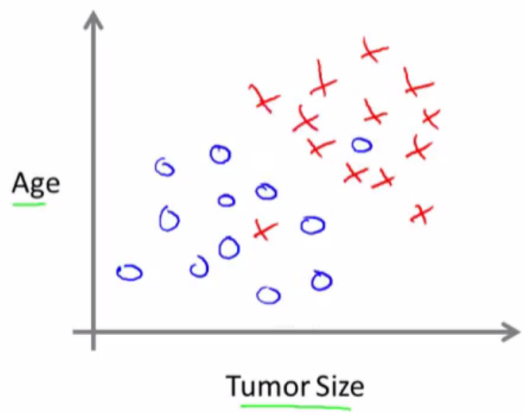
- **Uczenie bez nadzoru** (unsupervised learning)
  uczenie wyłącznie na podstawie sygnały wejściowego **X**, odkrywanie ciekawych struktur w przestrzeni danych wejściowych.
  Uczenie spontaniczne, odpowiada uczeniu w okresie niemowlęcym.
- **Uczenie nadzorowane** (supervised learning) - dla każdego sygnału wejściowego **x** dana jest pożądana odpowiedź **y**.
  Uczenie „szkolne".
- **Uczenie z krytykiem**, z „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 jest **x** jest etykietą klasy (np. nazwa obiektu na zdjęciu)





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

Uczenie nadzorowane

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

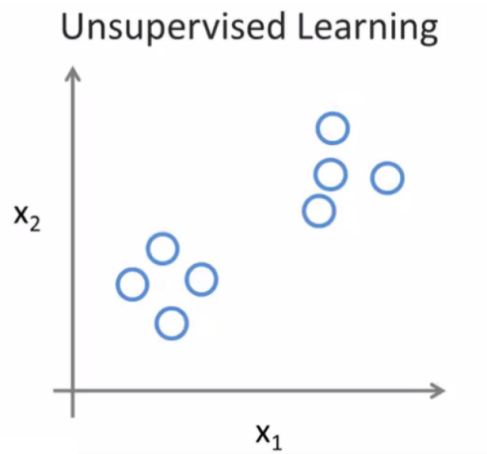
$$(f(\mathbf{x}; \mathbf{W}) - \mathbf{y})^2$$

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

Uczenie nienadzorowane

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

- **klasteryzacja** (analiza skupień) oparta na jakiejś mierze podobieństwa, szukanie struktur.

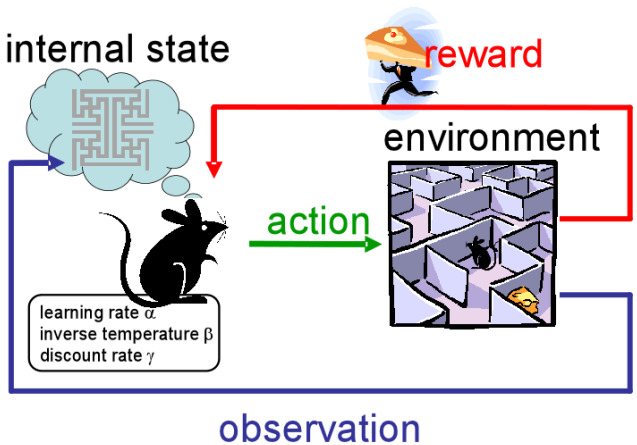


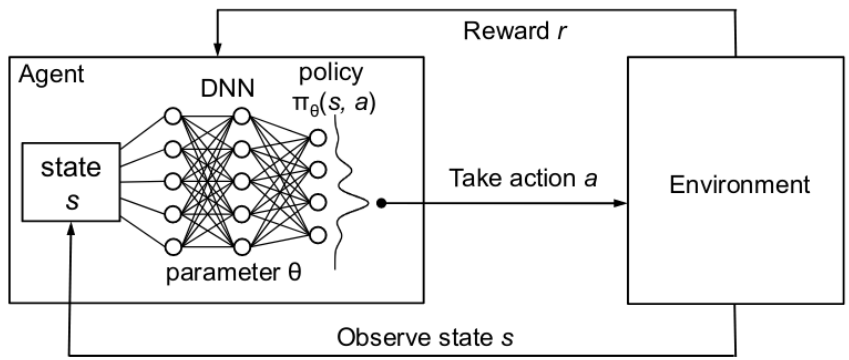
- Wykrywanie cech i regularności w danych, tworzenie przydatnych reprezentacji danych, kompresja (kodowanie) sygnału, redukcja wymiarowości, modelowanie sygnału, redukcja szumu

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

Uczenie z krytykiem

- Sygnał wyjściowy jest zazwyczaj akcją (sekwencją akcji) podejmowanych przez agenta
- Nagroda (lub kara) jest dostarczana z opóźnieniem (np. przegrana partia w szachy)

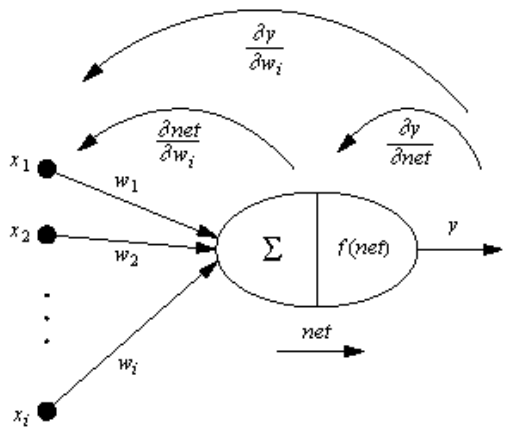




Rysunek: Hongzi Mao, „Resource Management with Deep Reinforcement Learning"
Aneek Das, The very basics of Reinforcement Learning

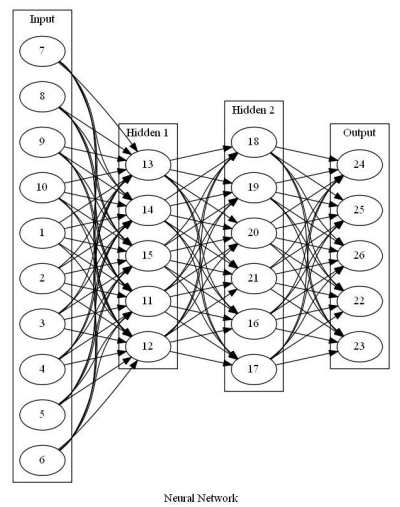
Główne aspekty modeli neuronowych I

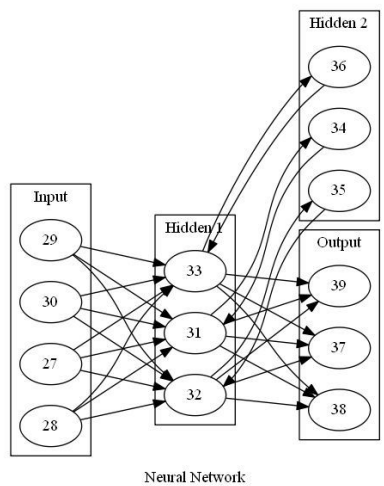
1. Sposób modelowania pojedynczego neuronu
   - Stan aktywacji (wzbudzenia) poszczególnych neuronów, sposób aktywacji tych neuronów, funkcja opisująca sygnał wyjściowy elementu,
   - neurony: progowe, liniowe, nieliniowe (np. sigmoidalne)

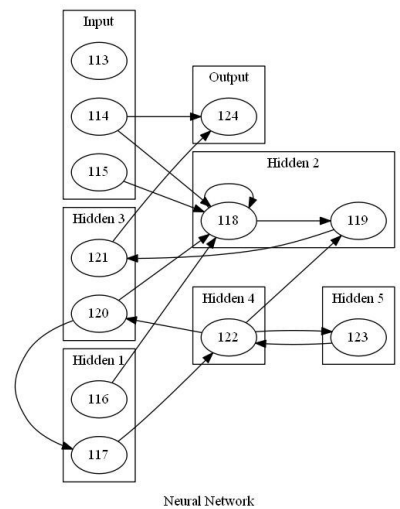


Główne aspekty modeli neuronowych II

2. Sposób propagacji sygnałów przez sieć neuronową
   - sieci jednokierunowe (feedforward)
   - sieci ze zprzężeniami zwrotnymi, seici rekurencyjne (reccurent)
3. Topologia połączeń elementów (architektura sieci):
   - budowa warstwowa (warstwa wejściowa, wyjściowa, warstwy ukryte), hierarchiczna
   - sieci jednowarstwowe, wielowartswowe, głebokie







Główne aspekty modeli neuronowych III

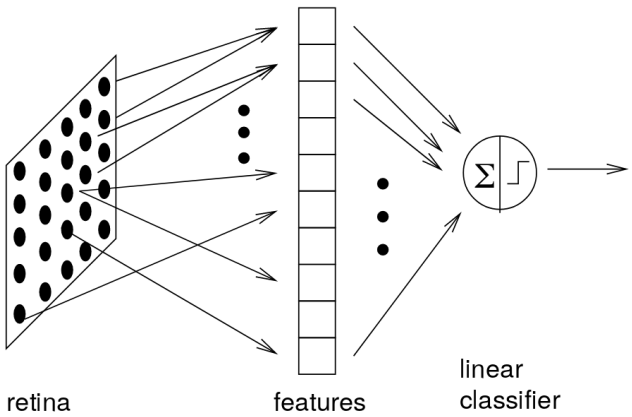
4. Reguły uczenia - reguły modyfikacji parametrów opisujących neurony i połączenia pomiędzy nimi, algorytm optymalizacji, funkcja kosztu
5. Otoczenie, w którym działa sieć neuronowa: sposób prezentacji danych, rodzaj danych wejściowych, reprezentacja sygnału wyjściowego (np. kodowanie etykiet klas)
6. Realizacja techniczna: język programowania, wykorzystanie GPU, zrównoleglenie obliczeń

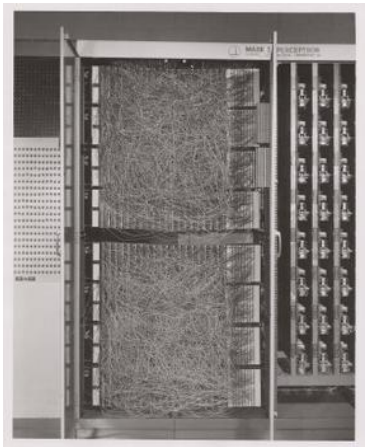
Perceptron

Reguły uczenia

Perceptron Rosenblatta (1958)

Klasyfikator neuronowy Mark I wzorowany na biologicznej percepcji (układ wzrokowy)





Trzy warstwy:

- S-units: wejście, siatkówka oka, np. fotokomórki 20 x 20
- A-units: asocjacyjne, zbierające dane z większych obszarów (obliczanie cech), 512
- R-units: liniowy klasyfikator uczący, 8

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 (ekstrakcja cech)

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

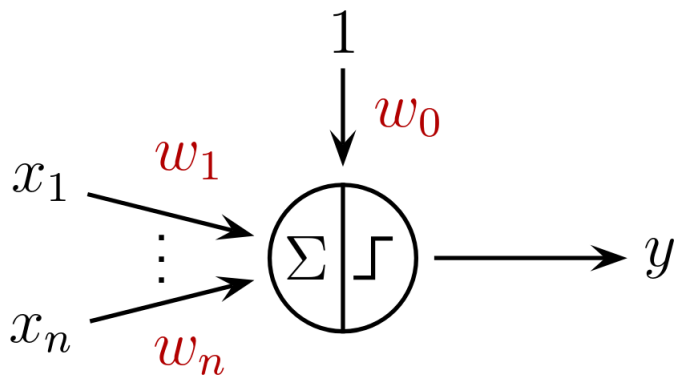
- sygnał wyjściowy

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

- wagi w_{ij} - potencjometry, uczenie - mechaniczna regulacja ich wartości

Perceptron współczesny

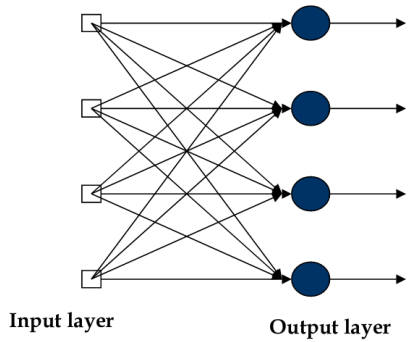
- Zwykle przez „perceptron” rozumie się teraz jeden neuron z wieloma wejściami (bez jednostek S, bo tu nie ma adaptacji).
- **Perceptron prosty**



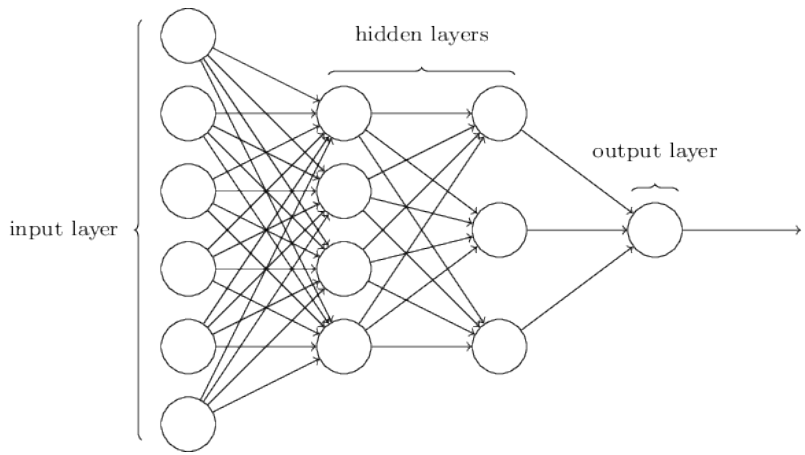
$$y = f\left(\sum_i w_i x_i + w_0\right)$$

Rys: Martin Riedmiller, Machine Learning (lectures)

Single layer perceptron



Multilayer perceptron (MLP)

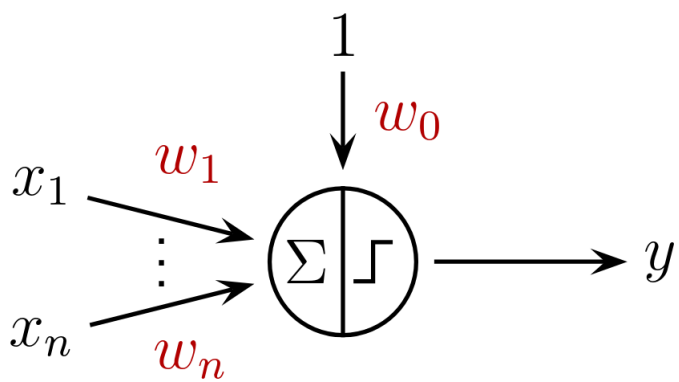


Rys: M. Bennamoun, Neural Computation (lecture)
computersciencewiki.org

Neuron binarny

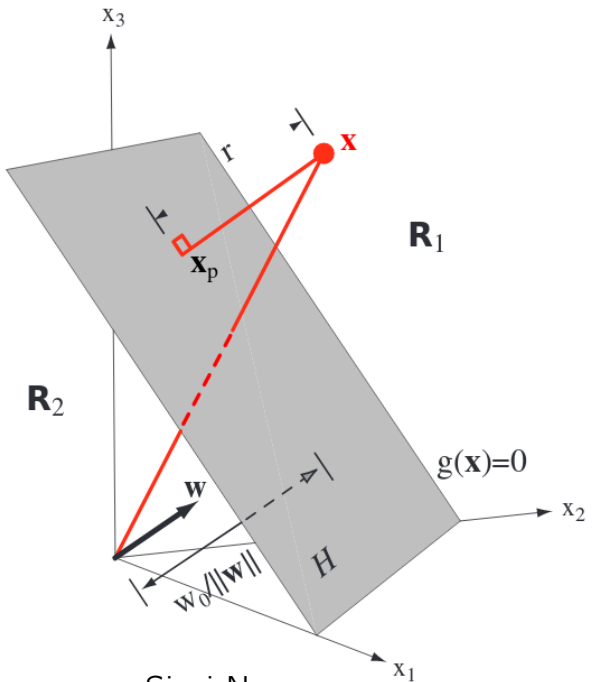
Perceptron prosty (binarny)

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



Interpretacja geometryczna

- $x_1, \ldots, x_n$ to punkty 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
- perceptron dzieli wektory x_i na te leżące poniżej płaszczyzny decyzyjnej ($y < 0$) oraz leżące powyżej płaszczyzny ($y \geq 0$)



Rys: Duda and Hart, Pattern Classification

Zadanie klasyfikacji

- **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**: dobrać wagi $\mathbf{w}$ i wartość progową 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. $\mathbb{P} \cap \mathbb{N} = \emptyset$

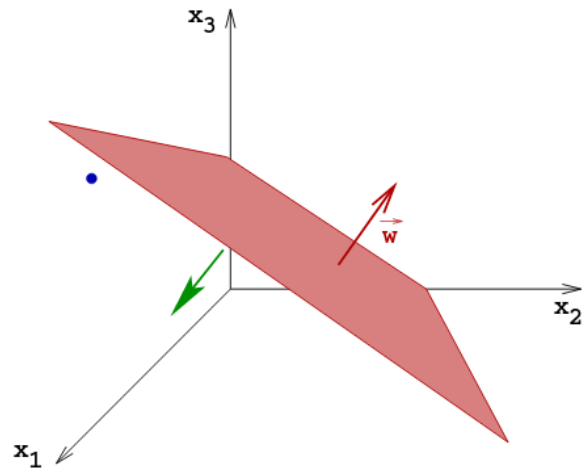
Uczenie perceptronu - idea

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 zeiększeniem w_0

Rys: Riedmiller, Machine Learning (lectures)

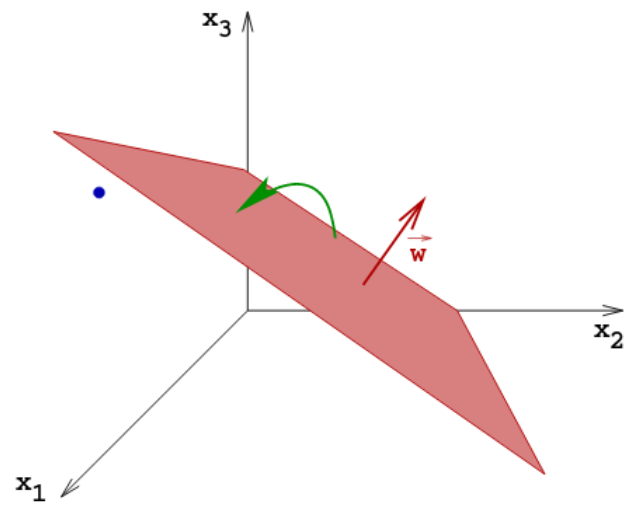
Uczenie perceptronu - idea

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

Algorytm 1 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 wagi $\mathbf{w}$ oraz wartość progową w_0 (np. małe losowe wartości w okolicy 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** $\mathbf{w}, w_0$

Algorytm uczenia perceptronu

- Jeżeli istnieje rozwiązanie (problem jest **liniowo separowany**) to algorytm je odnajdzie w skończonej licznie kroków
- Możliwe cykle: gdy wektor wag się powtórzy to problem jest nie rozwiązywalny
- Liczba błędów nie maleje monotoniczne - kolejna modyfikacja 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

Algorytm 2 Algorytm kieszonkowy

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

Output: perceptron klasyfikujący przypadki do dwóch klas

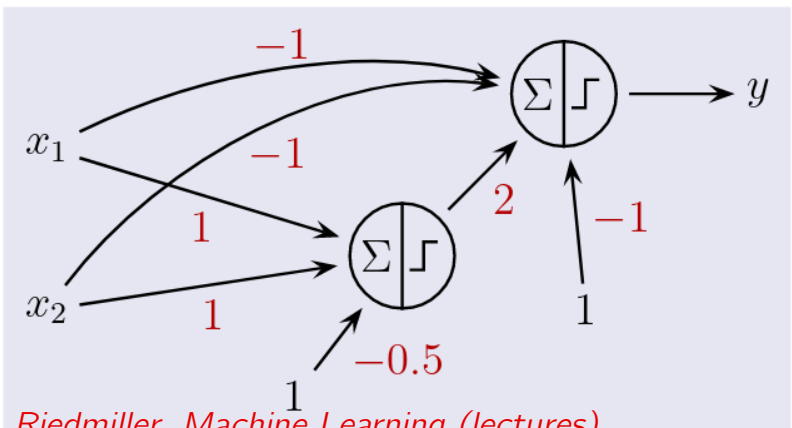
1: zainicjuj losowo wagi $\mathbf{w}$ i w_0
2: $t \leftarrow 0, \quad t' \leftarrow t, \quad \mathbf{w}' \leftarrow \mathbf{w}, \quad w_0' \leftarrow w_0$
3: **for** losowo wybranego $\mathbf{x} \in \mathbb{P} \cup \mathbb{N}$ **do**
4: **if** $\mathbf{x}$ poprawnie klasyfikowany **then**
5: $t \leftarrow t + 1$
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

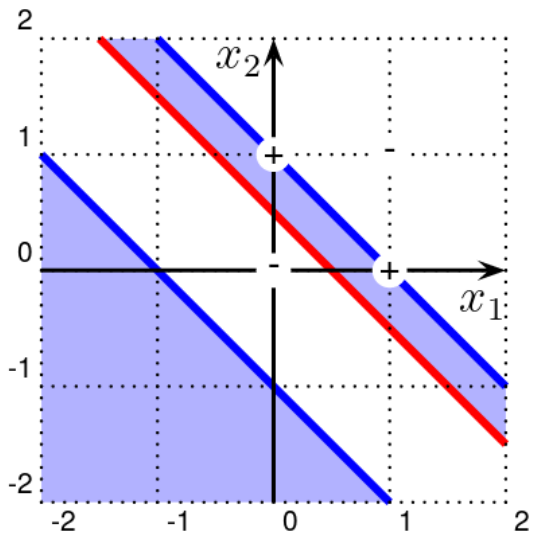
- Algorytm kieszonkowy ma za zadanie znaleźć najlepsze możliwe rozwiązanie nawet w przypadku problemów, które nie są liniowo separowalne
- Zapamiętuje tylko wagi ostatniego poprawnego przypadku, więc istnieje możliwość zignorowania wcześniejszego, lepszego rozwiązanie. Przeciwdziała temu alg. z zapadką, jednak wymaka on więcej nakładów obliczeniowych.
- **Algorytm kieszeniowy z zapadką** - modyfikacja algorytmu kieszeniowego, gdzie zapamiętywany jest zwycięzca tylko wtedy, gdy klasyfikuje poprawnie więcej przypadków treningowych

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



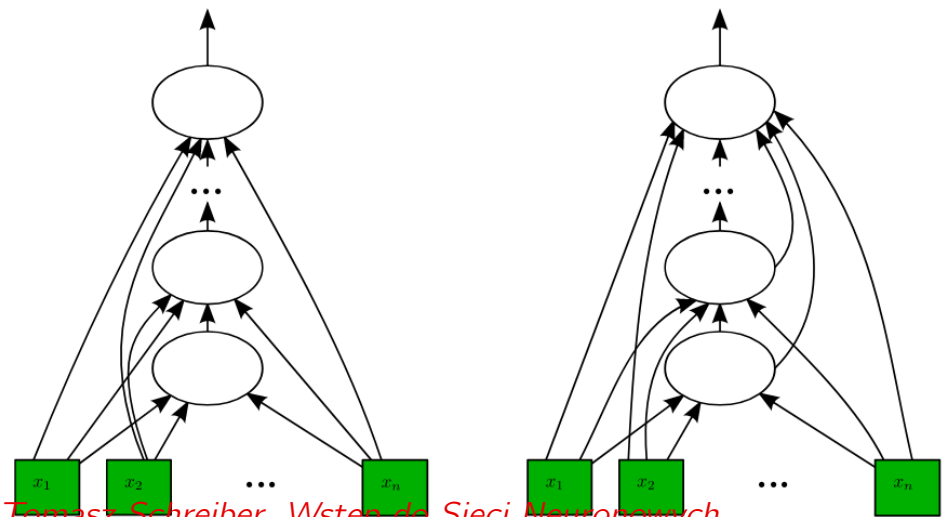
Rys: Riedmiller, Machine Learning (lectures)



Algorytm wieżowy i piramidalny

Przykład algorytmów rozrastających się

1. Wytrenuj pojedynczy perceptron np. alg. kieszonkowym na danych treningowych
2. Jeżeli nie uzyskano zadowalającego rezultatu to dodaj perceptron, którego wejściem będzie wyjście poprzedniego neuronu (alg. **wieżowy**) lub wyjścia wszystkich poprzednich neuronów (alg. **piramidalny**). Wróć do punktu 1.

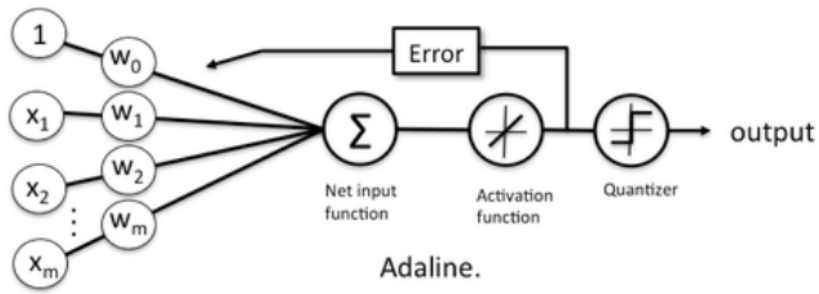


(a) Sieć wieżowa. (b) Sieć piramidalna.

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

Adaline - Adaptive Linear Element

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



Reguła uczenia Widrowa-Hoffa

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

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

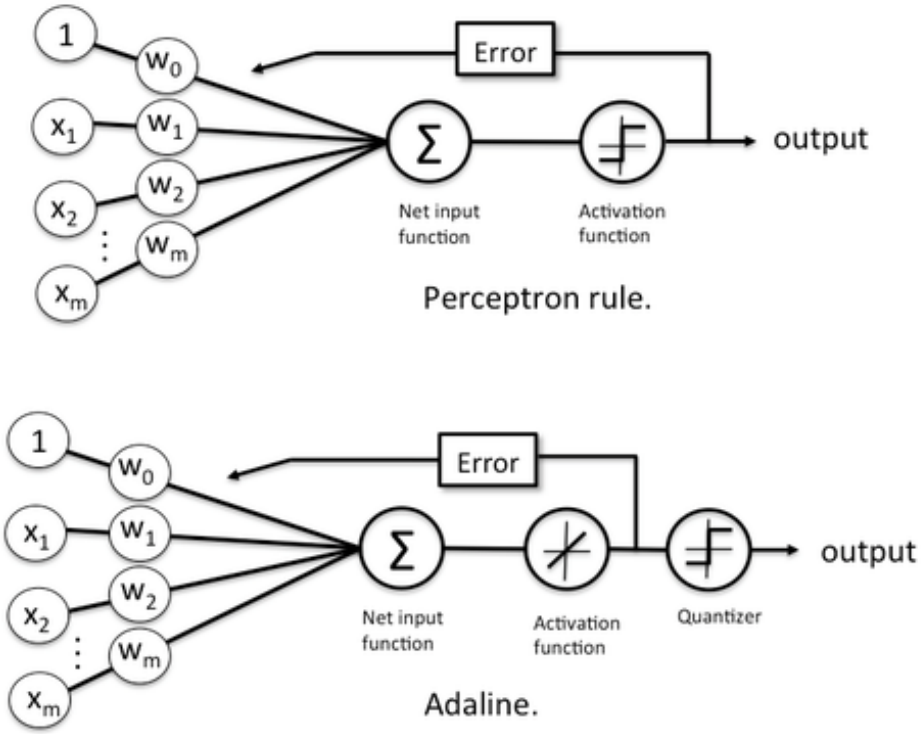
Reguła uczenia

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

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

- Ponieważ funkcja celu jest ciągła to możliwe jest trening za pomocą metody największego spadku gradientu
- dla neuronów liniowych reguła jest równoważna regule delta a Adeline rozwiązuje problem aproksymacji, wartości $y_i \in \mathbb{R}$
- dla neuronów z wyjściem progowym Adeline staje się klasyfikatorem, gdzie $y_i = \{-1, +1\}$
- Madeline - wielowarstwowa wersja Adeline

Adeline vs. Perceptron rule



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

Reguła delata

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

$$\Delta w_k = \lambda \nabla_{w_k} E \cdot x_k$$

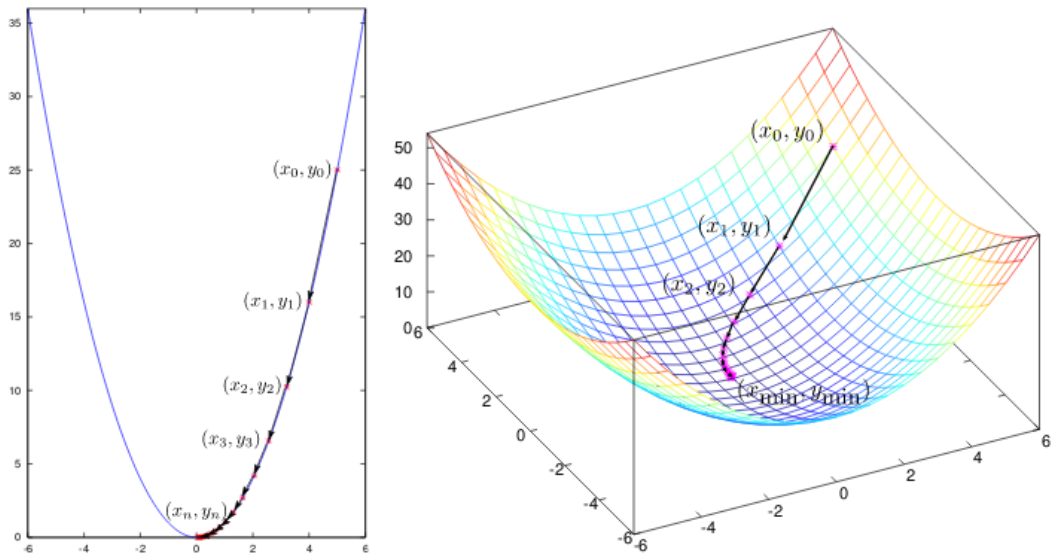
- minimalizacja błędu kwadratowego E w kierunku największego spadku gradientu

$$\Delta w_k = \lambda \left(y - f(\mathbf{x})\right) f'(\mathbf{x}) x_k$$

$\lambda > 0$ współczynnik uczenia
y_i pożądany sygnał, $y_i \in \mathbf{R}$ dla aproksymacji, $y_i = \{-1, +1\}$ lub $y_i = \{0, 1\}$ dla klasyfikacji binarnej

Algorytm spadku gradientu

$$w \leftarrow w - \lambda \nabla E(w)$$



(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

- 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 zależy od punktu startowego
- Kryterium stopu algorytmu:
  - ilość kroków uczenia lub inne ograniczenie czasowe
  - osiągnięcie zadowalającego poziomu dokładności $E < \epsilon$
  - niewielkie zmiany $||\Delta w|| < \epsilon$
- stała uczenia λ, gdy za duża rozwiązanie może być pominięte, gdy za mała uczenie będzie powolne.
  Nie musi być wartością stałą, np. może być zmniejszana w czasie treningu.

Funkcje aktywacji I

- **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

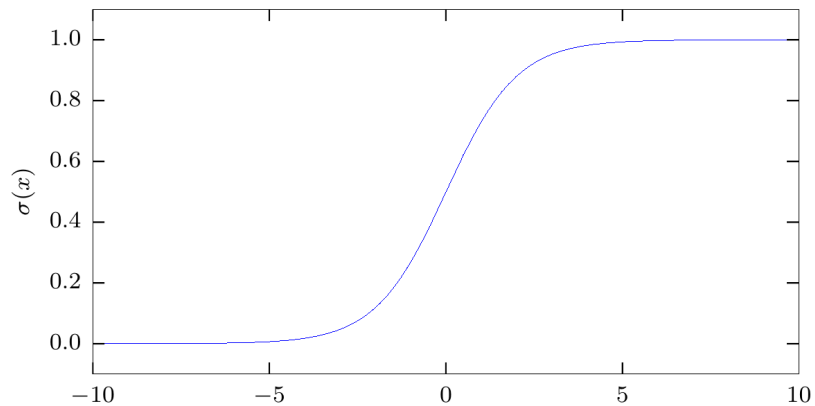
$$f(x) = \begin{cases} 1, & \text{gdy} \quad x \geq a \\ -1, & \text{gdy} \quad x < a \end{cases}$$

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

Funkcje aktywacji II

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

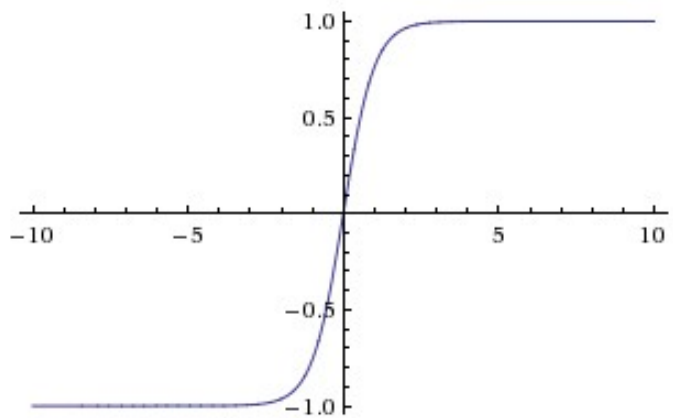
$$f(x) = \frac{1}{1 + e^{-x}} \qquad f'(x) = f(x)(1 - f(x))$$



Funkcje aktywacji III

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

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

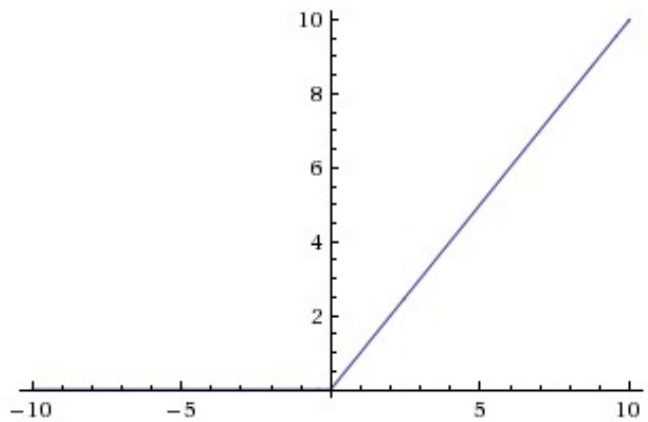


Funkcje aktywacji IV

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

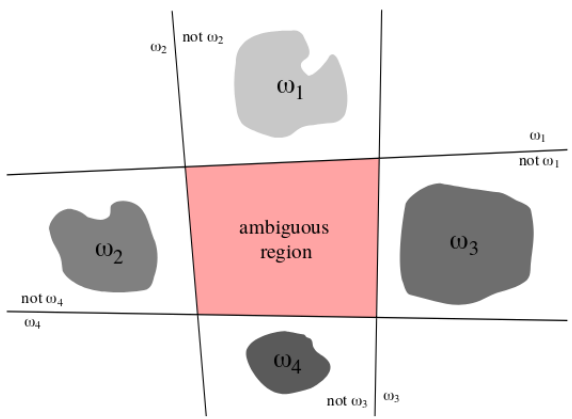
$$f(x) = \max(0, x) = \begin{cases} x, & \text{gdy} \quad x \geq 0 \\ 0, & \text{gdy} \quad x < 0 \end{cases}$$

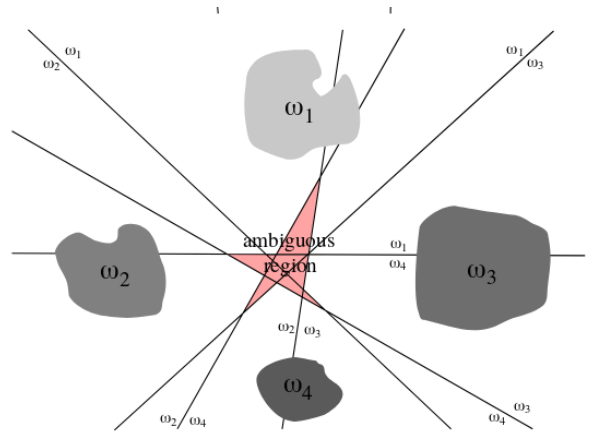
$$f'(x) = \begin{cases} 1, & \text{gdy} \quad x > 0 \\ 0, & \text{gdy} \quad x < 0 \end{cases}$$



Klasyfikacja wielu klas

- W przypadku wieloklasowym $y_i = 1, 2, 3, ..., k$
- Pojedynczy perceptron prosty może być nauczony aby separować przypadki z pojedynczej klasy od pozostałych (po jednym perceptronie na klasę), k klasyfikatorów binarnych
- Bardziej wymagające podejście $k(k-1)/2$ klasyfikatorów binarnych dla każdej pary klas

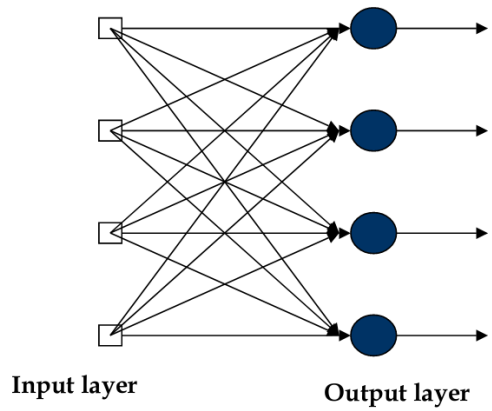




Rys: Duda and Hart, Pattern Recognition

Maszyna liniowa

- sieć jednowarstwowa - liczba wyjść równa liczbie klas



- funkcja dyskryminująca

$$f_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

- klasyfikacja: przypisanie wektorowi $\mathbf{x}$ klasy i odpowiadającej wyjściu f_i o największej wartości

$$f_i(\mathbf{x}) > f_k(\mathbf{x}) \quad \text{dla każdego} \quad i \neq k$$

Algorytm 3 Uczenie maszyny liniowej

Input: zbiór wektorów należących do jednej z k klas

Output: klasyfikator separujący wektory treningowe

1: zainicjuj losowo wagi $\mathbf{w}_i$
2: **for** losowo wybranego **x do**
3: **if** **x** należący do klasy i jest niepoprawnie przypisany do klasy j **then**
4: $\mathbf{w}_i \leftarrow \mathbf{w}_i + \mathbf{x}$
5: $\mathbf{w}_j \leftarrow \mathbf{w}_j - \mathbf{x}$
6: **return** $\mathbf{w}, w_0$

Jednokierunkowe sieci wielowarstwowe

Multilayer Perceptron (MLP)

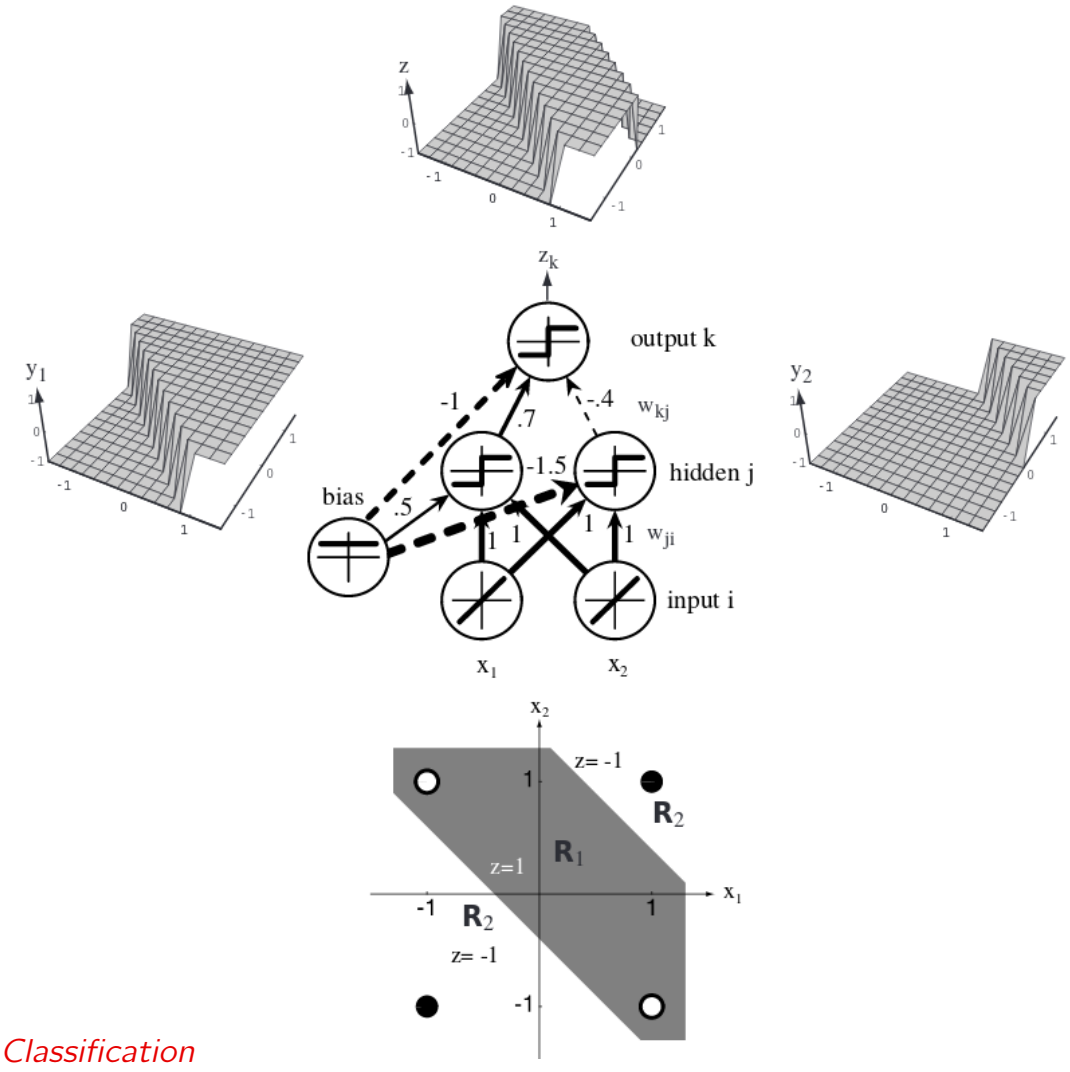
- Perceptrony jednowarstwowe potrafią rozwiązywać problemy separowalne liniowo
- Reguła delta dla sieci jednowarstwowej:

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

  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 może być dalej analizowany przez kolejne warstwy

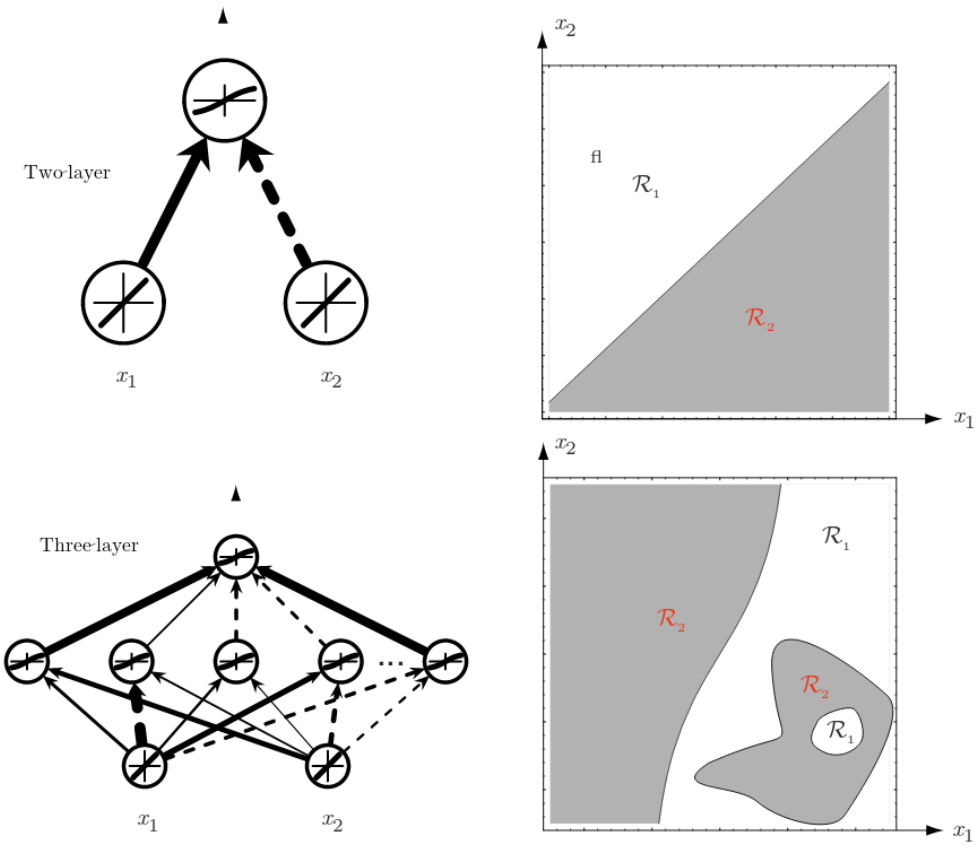
$$\mathbf{y} = f(\mathbf{x}) = f^{(M)} \left(f^{(M-1)} \left(\dots f^{(1)}(\mathbf{x}) \right) \right)$$

Jeszcze raz XOR



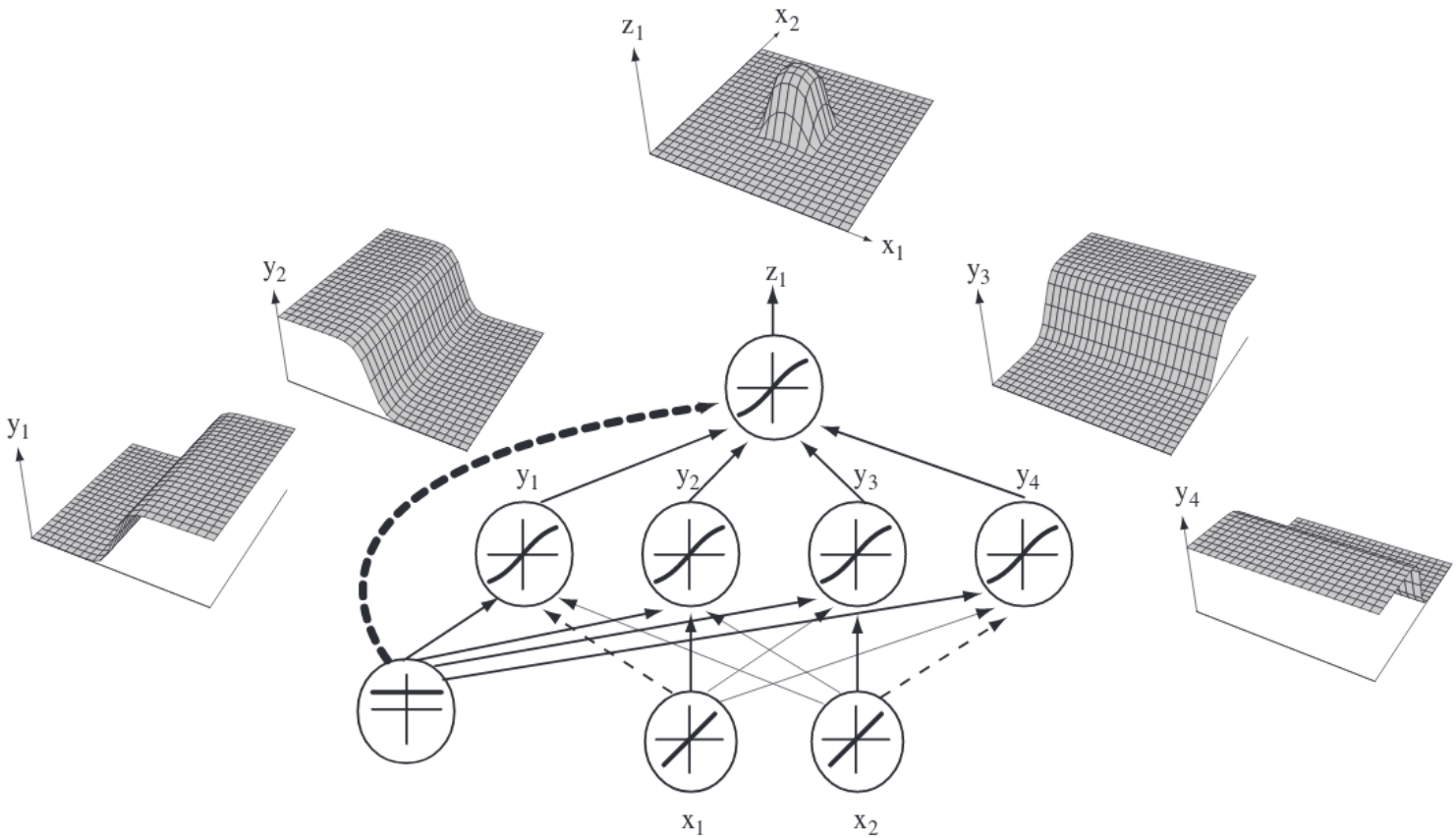
Rys: Duda, Hart, Pattern Classification

Warstwa ukryta a granice decyzji



Rys: Duda, Hart, Pattern Classification

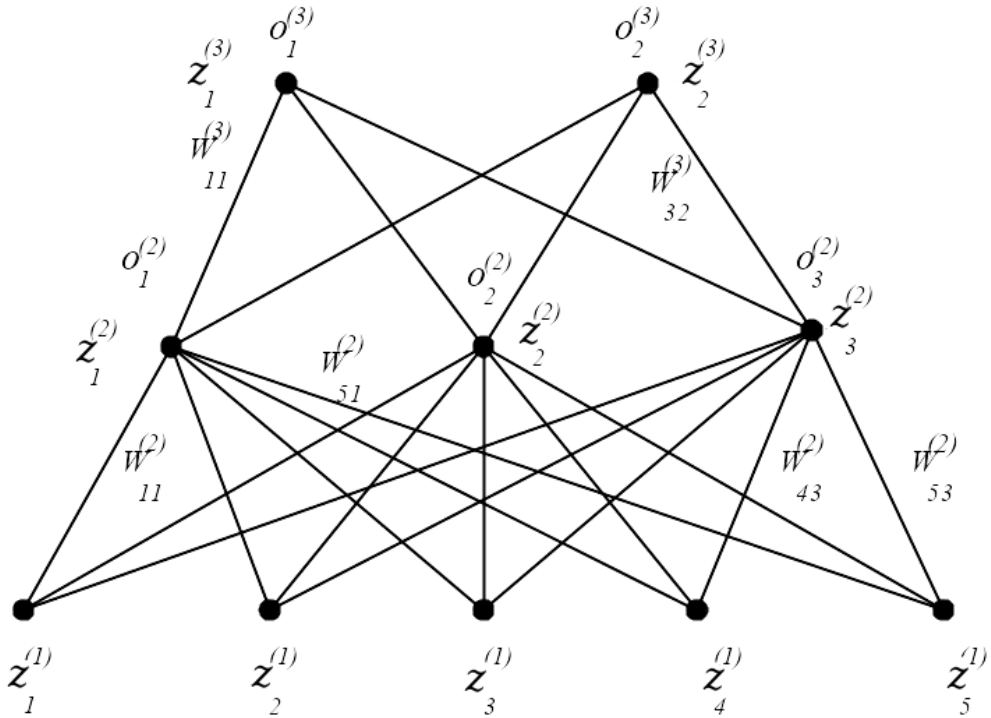
Przykład: sieć MLP 2-4-2



Rys: Duda, Hart, Pattern Classification

Perceprton 3 warstwowy

- Warstwy: wejściowa, ukryte, wyjściowa

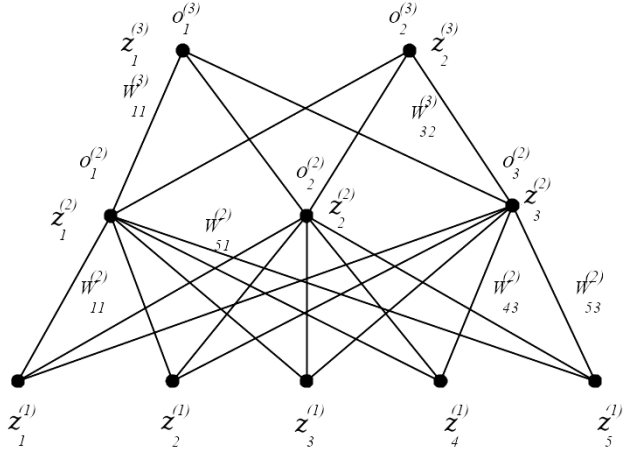


- Warstwy w pełni połączone (wagi tworzą macierz)

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

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

MLP - oznaczenia



M - liczba warstw ($M = 3$)

$w_{ij}^{(l)}$ - waga łącząca element i należący do warstwy $l-1$ oraz element j 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 wychodzący z elementu 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)

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 spadku gradientu:

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

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 dla 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 $M-1$:

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

gradient

$$\begin{aligned}
\frac{\partial f_i(\mathbf{x};\mathbf{W})}{\partial w_{jk}^{(M-1)}} &= \frac{\partial \sigma(z_i^{(M)})}{\partial z_i^{(M)}} \frac{\partial z_i^{(M)}}{\partial w_{jk}^{(M-1)}} = \sigma'\left(z_i^{(M)}\right) \frac{\partial \left(\sum_l w_{li}^{(M)} o_l^{(M-1)} \right)}{\partial w_{jk}^{(M-1)}} \\
&= \sigma'\left(z_i^{(M)}\right) w_{ki}^{(M)} \frac{\partial o_k^{(M-1)}}{\partial w_{jk}^{(M-1)}} \\
&= \sigma'\left(z_i^{(M)}\right) w_{ki}^{(M)} \sigma'\left(z_k^{(M-1)}\right) o_j^{(M-2)}
\end{aligned}$$

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

zmiana wag dla warstwy $M-1$

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

$$\Delta w_{0k}^{(M-1)} = \eta \delta_k^{(M-1)}$$

Struktura wzoru w kolejnych warstwach jest taka sama

Wsteczna propagacja błędu - podsumowanie

Funkcja realizowana przez sieć wielowarstwową

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

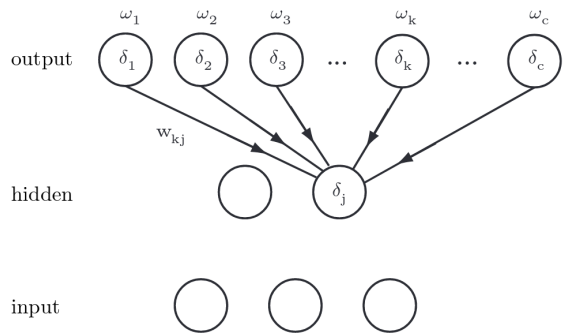
Aktualizacja wag w dowolnej warstwie K - uogólniona reguła delta

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

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

Sygnał błędu w warstwie K

$$\delta_i^{(K)} = \sum_j \delta_j^{(K+1)}w_{ij}^{(K)}$$



sygnał błędu w warstwie wyjściowej:

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

Algorytm wstecznej propagacji

Algorytm BP:

1. Zainicjuj wagi **W** małymi losowymi wartościami
2. Dopóki nie spłoniony warunek stopu wykonuj
   oblicz sygnał od wejścia do wyjścia

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

   oblicz sygnał błędu od warstwy wyjściowej do wejściowej

$$\delta_i^{(K)} = \sum_j \delta_j^{(K+1)} w_{ij}^{(K)}$$

   aktualizuj wagi

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

Algorytm wstecznej propagacji można w łatwy sposób rozszerzyć na:

- 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

- błąd uśredniony błąd całego zbioru treningowego
- nie praktyczne 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
- możliwe operacje na macierzach, łatwiejsze zrównoleglenie GPU/CPU

Własności MLP

- **Uniwersalny aproksymator:** sieć MLP z jedną warstwą ukrytą jest w stanie aproksymować dowolną funkcję ciągła 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

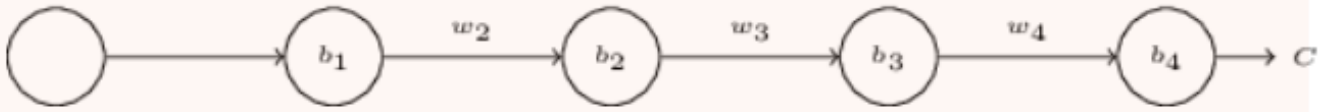
Problemy II

- Minima lokalne i plateau, wąskie „rynny” - np. wielokrotny start
- Wpływ nowych wzorców na już nauczone – zapominanie
- Dobór stałej uczenia
- Znikający gradient, eksplodujący gradient
- Przygotowanie danych uczących: normalizacja, standaryzacja, kodowanie wyjść, ...
- Ocena modelu: zbiory walidacyjne, testowe, kroswalidacja

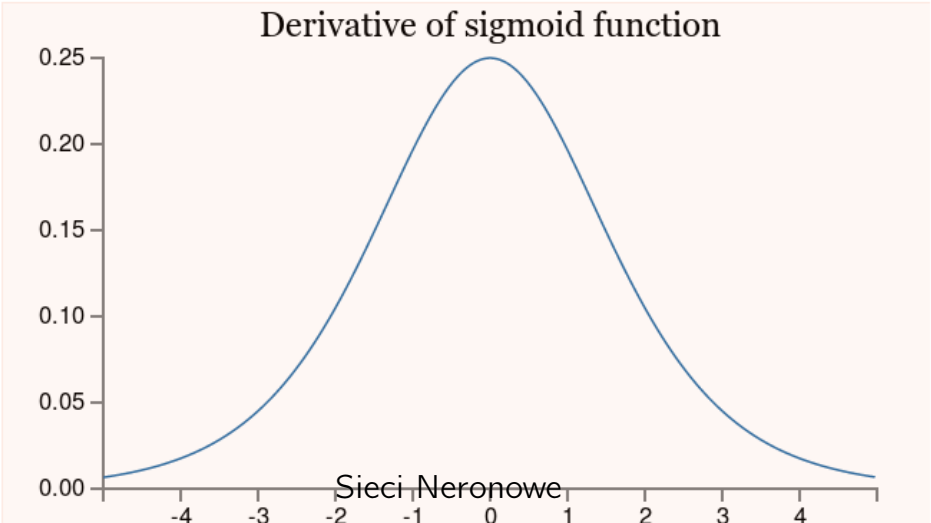
Znikający gradient

Szybkość uczenia (zmiany wag) drastycznie maleje w głębszych warstwach - sygnał błędu zanika
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

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 $|w_i \sigma'(z_i)| > 1$, 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*)

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

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,
Θ - parametry sieci,
y - oczekiwane wyjścia

Funkcja kosztu Cross Entropy

Cross entropy

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

- miara odległości miedzy rozkładami y i $\hat{y} \in (0, 1)$
- gdy $y \approx \hat{y}$ to funkcja kosztu bliska zeru
- $J > 0$

Gradient funkcji kosztu:

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

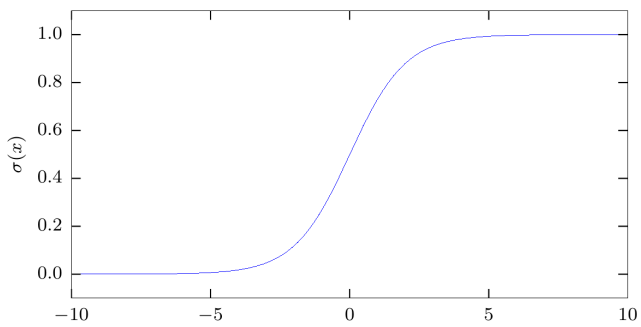
Funkcja liniowa w warstwie wyjściowej

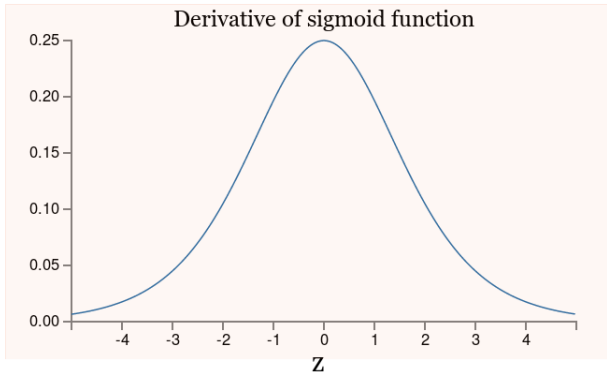
$$z = \mathbf{w}^T \mathbf{h} + b$$

- 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 2 klas, modelowanie rozkładu Bernoullego, binarne wartości wyjściowe

Funkcja logistyczna w warstwie wyjściowej

- podczas minimalizacji MSE błąd zanika, gdy funkcja się nasyca (także dla niepoprawnej odpowiedzi) $\frac{\partial \hat{y}}{\partial \Theta} \approx 0$

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

- problem znika przy zastosowaniu kosztu Cross Entropy (CE) Binarna funkcja kosztu: na każdym wyjściu oczekujemy stanu 0 lub 1

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

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

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, każde wyjście sieci związane jest z jedną klasą, wartości kodowane w postaci wektora binarnego *one hot*

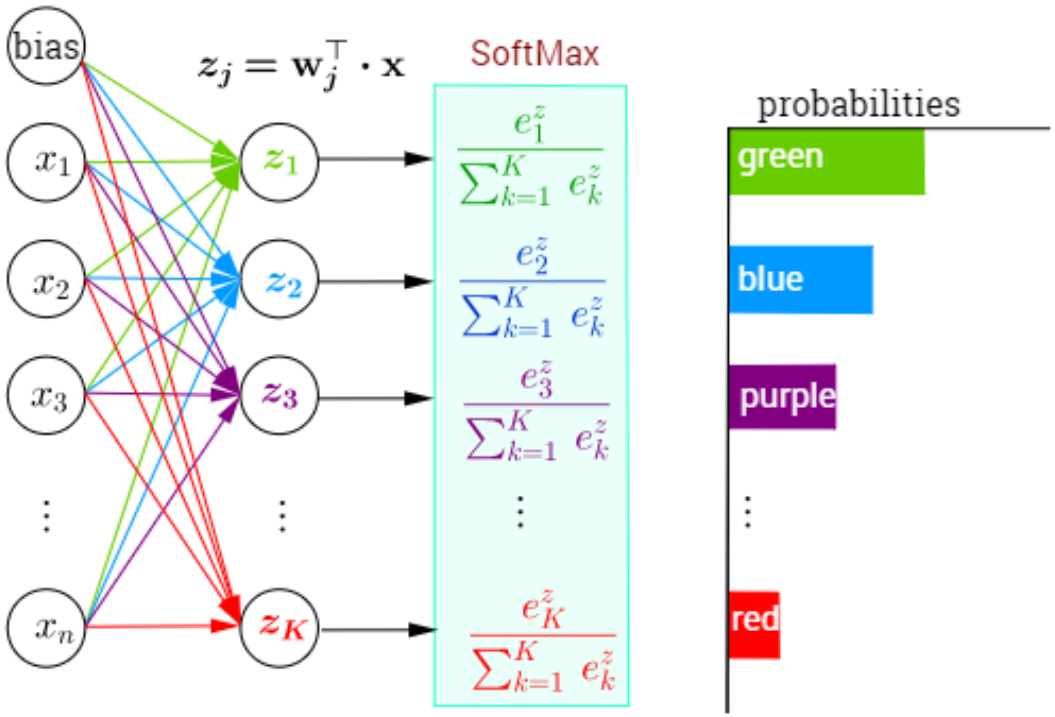
$$[0, \ldots, 0, 1, 0, \ldots, 0]$$

- wartości wyjściowe sieci reprezentują rozkład prawdopodobieństw zmiennej wyjściowej

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

Multi-Class Classification with NN and SoftMax Function

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



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 CE gdy $\hat{y}_i = softmax(\mathbf{z})_i$

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

- obliczenia się upraszczają i znika problem zanikania błędu dla nasyconych wyjść, który występował dla funkcji kosztu MSE i wyjść sigmoidalnych

- dobrze współgra z treningiem *cross entropy*, błąd nie znika nawet dla dużych wartości aktywacji z_i

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

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

$$softmax(x) = softmax(x + c)$$

- minimalizacja kosztu CE powoduje zwiększenie wartości z_i na jednym z wyjść i zmniejszenie wszystkich pozostałych

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

- z punktu widzenia neurobiologicznego softmax może być widziany jako realizacja mechanizmów hamowania występującego w grupach neuronów w korze, podobnie do mechanizmu *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

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

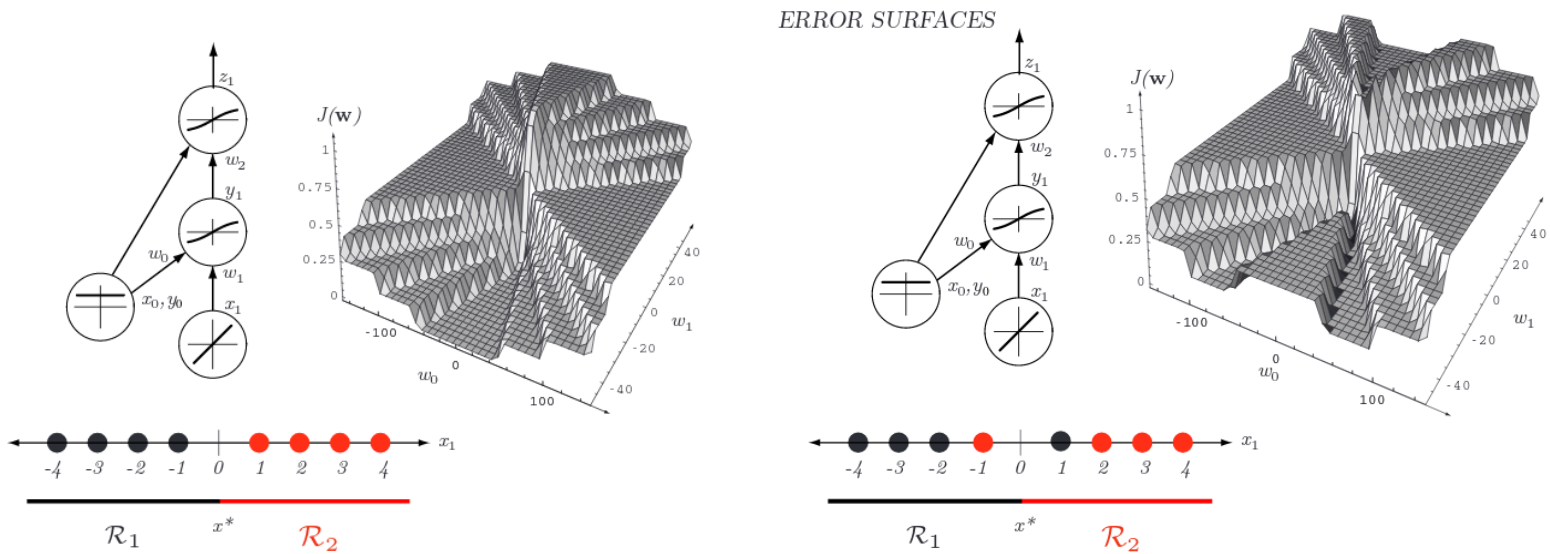
- funkcje ograniczone: sigmoidalne i tangens hiperboliczny powodują problemy z niestabilnym gradientem, uczenie może utknąć, gdy funkcja się nasyci
- obecnie najpopularniejszym typem aktywacji w sieciach jest ReLU

Demonstracje

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

Ulepszenia treningu sieci MLP

Powierzchnia błędu



Minima lokalne, globalne

Plateau - regiony o małej zmienności błędu względem wag

Jak omijać minima lokalne?

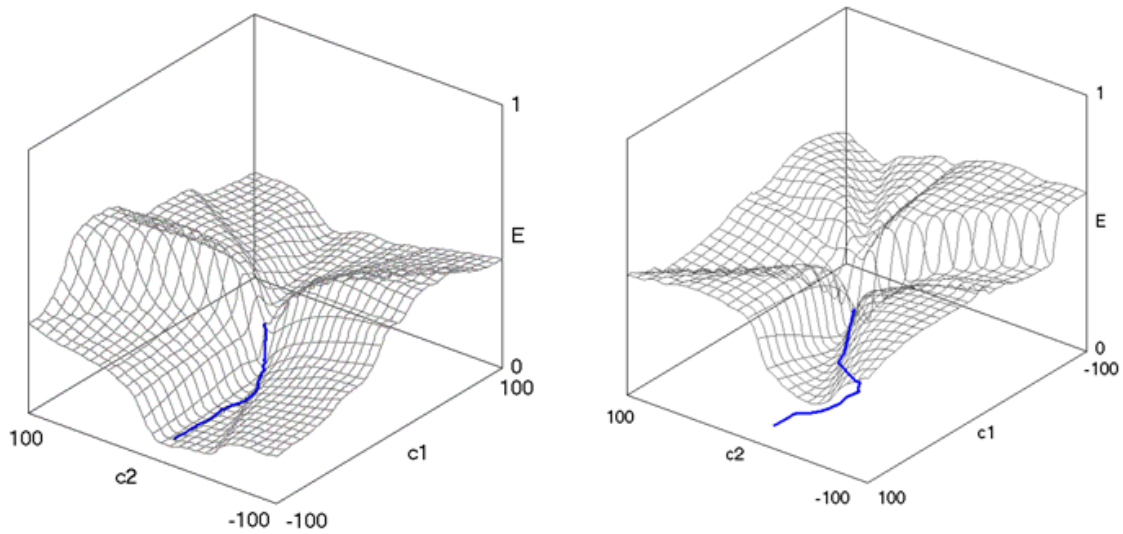
- Wielokrotny start z różnymi wartościami początkowymi - najprostsza ale skuteczna metoda
- Szum dodawany do wag lub do danych pozwala wygładzić funkcję błędu i uciec z płytszych minimów – formalnie jest to równoważne regularyzacji, czyli dodaniu dodatkowego członu wygładzającego do funkcji błędów.
- Losowa kolejność prezentowania przypadków
- Modyfikacje BP lub inne algorytmy optymalizacji

Metody globalnej minimalizacji

- Metody globalnej minimalizacji: wiele metod.
- Monte Carlo, symulowane wyżarzanie, metody multisympleksowe, minimalizacja Tabu, homotopia ...
- Dużo prac łączących algorytmy genetyczne z sieciami MLP
- Zalety: globalne, proste w realizacji, niektóre nie potrzebują gradientu, inne łączą zalety gradientowych z globalnymi.
- Wady: zwykle kosztowne i czasochłonne

Trajektorie zbieżności

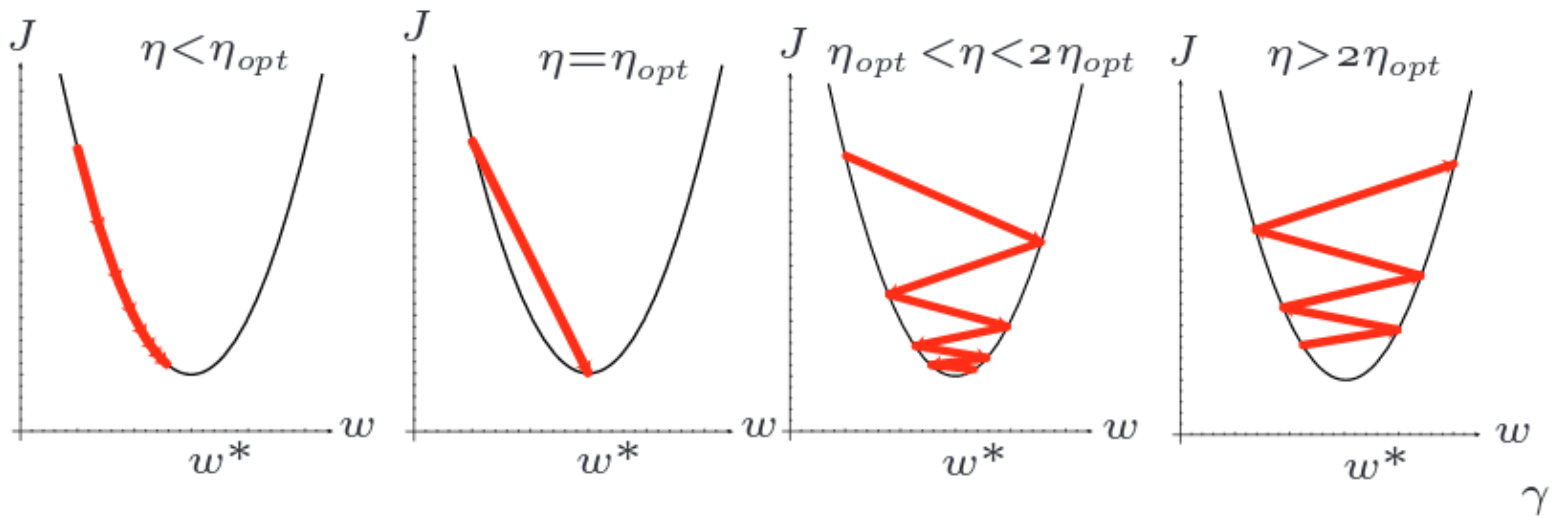
PCA na macierzy kowariancji wag w kolejnych krokach iteracji



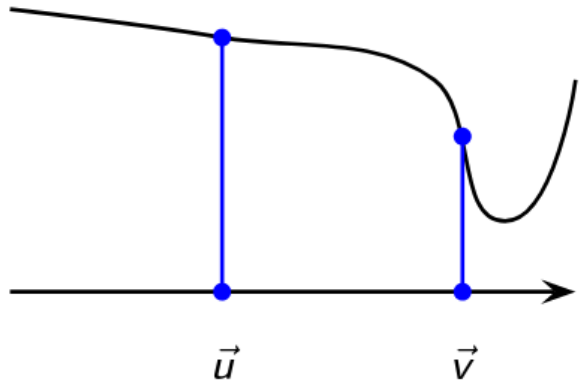
- wiele płaskowyżów i kanionów.
- dane leżące daleko od granicy mają mały wpływ na powierzchnie błędu
- przy końcu uczenia główna redukcja błędu wynika ze wzrostu wartości wag, czyli wyostrzania się sigmoid aż zrobią się prawie skokowe

Dobór kroku uczenia

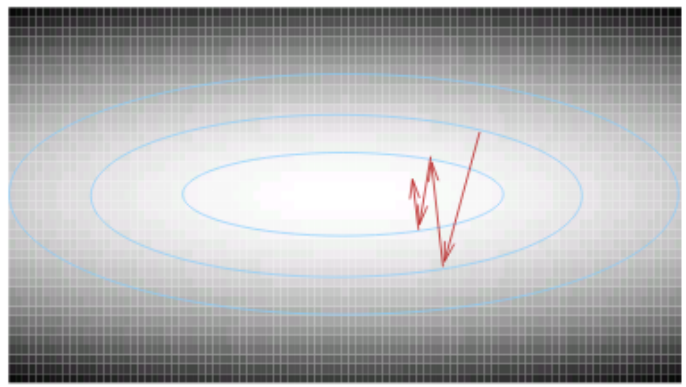
- zbyt mała wartość η - powolna zbieżność
- za duża wartość η - niestabilny trening, oscylacje
- wartość zmiana w czasie uczenia, różne podejścia:
  - zmniejszana w czasie treningu
  - zwiększana dla płaskich powierzchni błędu
  - dobierana niezależnie do warstwy lub pojedynczych neuronów i wag



Płaskie powierzchnie i strome doliny



Powierzchnia błędu w wielu wymiarach ma różne nachylenie



Rys: Riedmiller, Machine Learning

Trening z momentem

prędkość uczenia zależna od poprzednich wartości gradientów

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

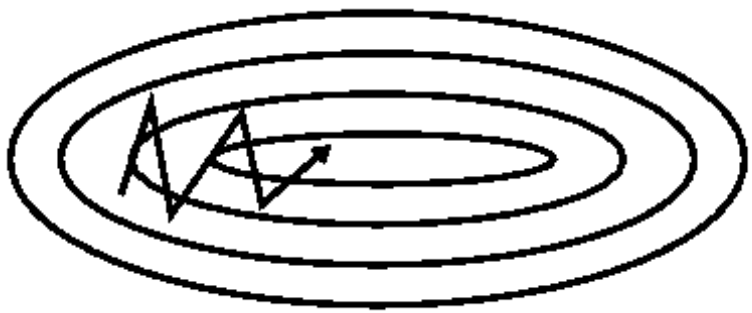
- zwiększa krok uczenia, gdy gradient nie zmienia kierunku. Przyśpieszenie na płaskich odcinkach wynosi w przybliżeniu

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

- redukuje oscylacje (spowalnia uczenie), gdy gradient zmienia kierunek
- gdy $\nabla E = 0$ wagi są nadal modyfikowane (bezwładność), może to ułatwić opuszczenie strefy „przyciągania” do minimum lokalnego
- $0 < \gamma < 1$, typowo współczynnik $\gamma = 0.9$

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 daje średnio aktywację neuronów w zakresie liniowym $[-1, 1]$ sigmoidy
- Analogicznie dla kolejnych warstw

Skalowanie wartości wejściowych

- Rożne skale mierzonych cech x_i, „dzikie” rozkłady dalekie od rozkładu normalnego, wartości odstające
- Dane wejściowe $\mathbf{x}_i$ powinny posiadać zbliżone zakresy wartości
- 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

Resilent BP (Riedmiller, Braun, 1992)

- radzi sobie z problemem znikających (oraz za dużych) gradientów
- wyłacznie znak gradientu (nie amplituda) uwzględniany w obliczeniach
- stała uczenia dobierana dla każdej wagi niezależnie

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

RPROP

- wymaga informacji z 2 ostatnich kroków uczenia
- η_{ij} rośnie, gdy brak zmiany kierunku gradientu
- η_{ij} maleje, gdy następuje zmiana kierunku

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

gdzie

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

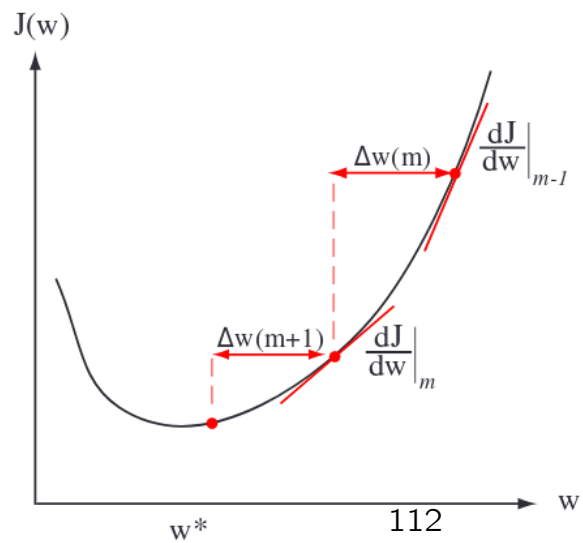
$$a = 1.2, \quad b = 0.5, \quad \eta_{min} = 10^{-6}, \quad \eta_{max} = 50$$

Quickprop (Fahlman, 1988)

- założenie: wagi są niezależne a funkcja błędu w okolicach minimum może być przybliżona parabolą
- przybliżenie za pomocą wartości i gradientów funkcji w 2 punktach $w(m)$ i $w(m-1)$

$$\Delta w(m+1) = \frac{\nabla_{ij} E(m)}{\nabla_{ij} E(m-1) - \nabla_{ij} E(m)} \Delta w(m)$$

- wagi mają niezależną zbieżność uczenia
- zbieżność kwadratowa
- trening może być niestabilny



Metody 2 rzędu

Rozwinięcie w szereg Taylora:

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

gdzie $\nabla^2 E(\mathbf{w}) = \mathbf{H}$ jest macierzą $n \times n$ pochodnych 2 rzędu (Hessian)

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

Gradient funkcji kosztu:

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

Minimum osiągane dla $\nabla E(\mathbf{w}+\mathbf{d}) = 0$, stąd

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

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

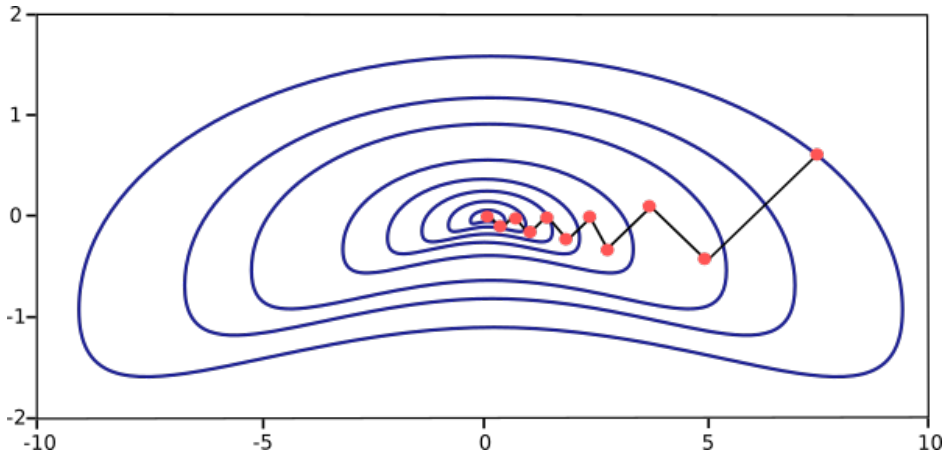
Metoda Newtona

Iteracyjna metoda 2 rzędu:

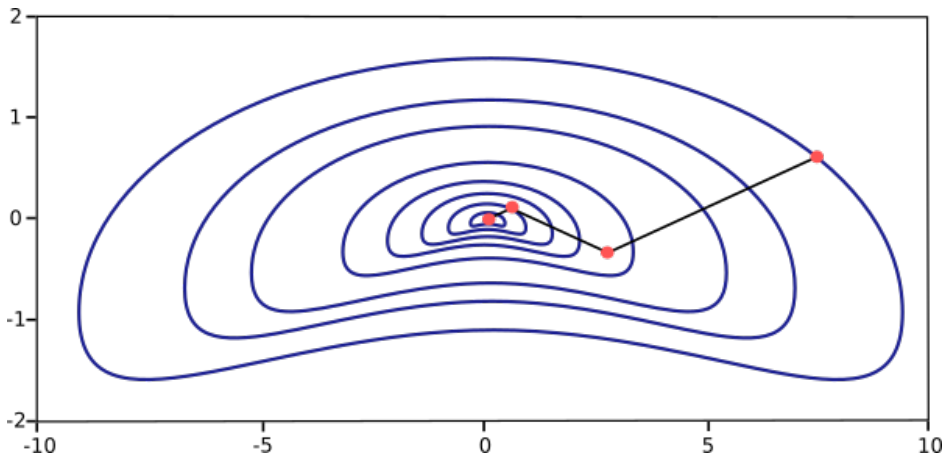
$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mathbf{H}^{-1}\nabla E(\mathbf{w}(t))$$

- metoda kosztowna czasowo $O(n^3)$ (odwracanie macierzy w każdej iteracji)
- zbieżność (kwadratowa) po kilku iteracjach
- metoda kosztowna pamięciowo, Hessian $O(n^2)$, gdzie n liczba wag
- praktyczne zastosowania tylko dla małych sieci
- $\mathbf{H}$ nie zawsze jest dodatnio określona i - stosuje się aproksymacje

Spadek gradientu



Metoda Newtona



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

Metody quasi-Newtona

Przybliżenia do 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 - przybliżenie do H^{-1} oraz iteracyjna metoda Newtona, kwadratowo zbieżna
  - Davidon-Fletcher-Power (DFP)
  - Broyden-Fletcher-Goldfarb-Shanno (BFGS).
- Metoda Levenberg-Marquardta oparta jest na przybliżeniu Gaussa-Newtona

Metoda Levenberg-Marquardta

Jakobian dla funkcji błędu $E = \sum e_i^2$

$$J_{ij} = \frac{\partial e_i}{\partial w_j}$$

Jakobian można policzyć korzystając z wstecznej propagacji.
Pochodna funkcji błędu:

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

Przybliżenie do Hesjanu:

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

gdzie współczynnik tłumienia μ zapewnia dodatnią określoność $\mathbf{H}$.
Aktualizacja wag:

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

Metoda Levenberg-Marquardta

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

- dla $\mu = 0$ mamy metodę Newtona
- dla dużych μ mamy metodę największego spadku z małym krokiem uczenia
- LM startuje z dużą wartością μ a następnie w trakcie uczenia μ jest zmniejszane
- bardzo szybko zbieżna metoda dla funkcji, które są sumą kwadratów (MSE)
- nie nadaje się do zastosowań z funkcją Cross Entropy
- nie praktyczne dla dużych danych, duży koszt pamięci

Metoda gradientów sprzężonych

Metoda **gradientów sprzężonych** (*conjugated gradients*) w kolejnych krokach iteracji poszukuje minimum wzdłuż kierunków sprzężonych do wszystkich poprzednich kierunków.
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

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

gdzie współczynnik sprzężenia β

- reguła Fletchera-Reevesa

$$\beta = \frac{(\nabla E(\mathbf{w}(m)))^2}{(\nabla E(\mathbf{w}(m-1)))^2}$$

- reguła Polaka-Ribiera

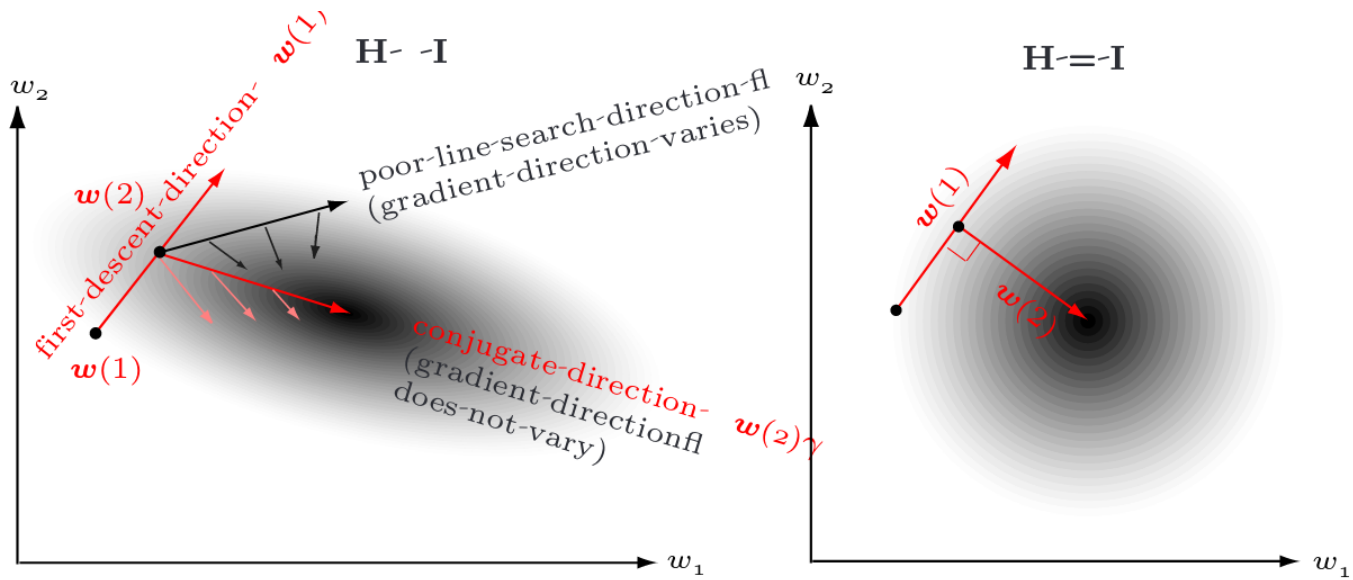
$$\beta = (\nabla E(\mathbf{w}(m)) - \nabla E(\mathbf{w}(m-1)))\frac{\nabla E(\mathbf{w}(m))}{(\nabla E(\mathbf{w}(m-1)))^2}$$

Metoda gradientów sprzężonych

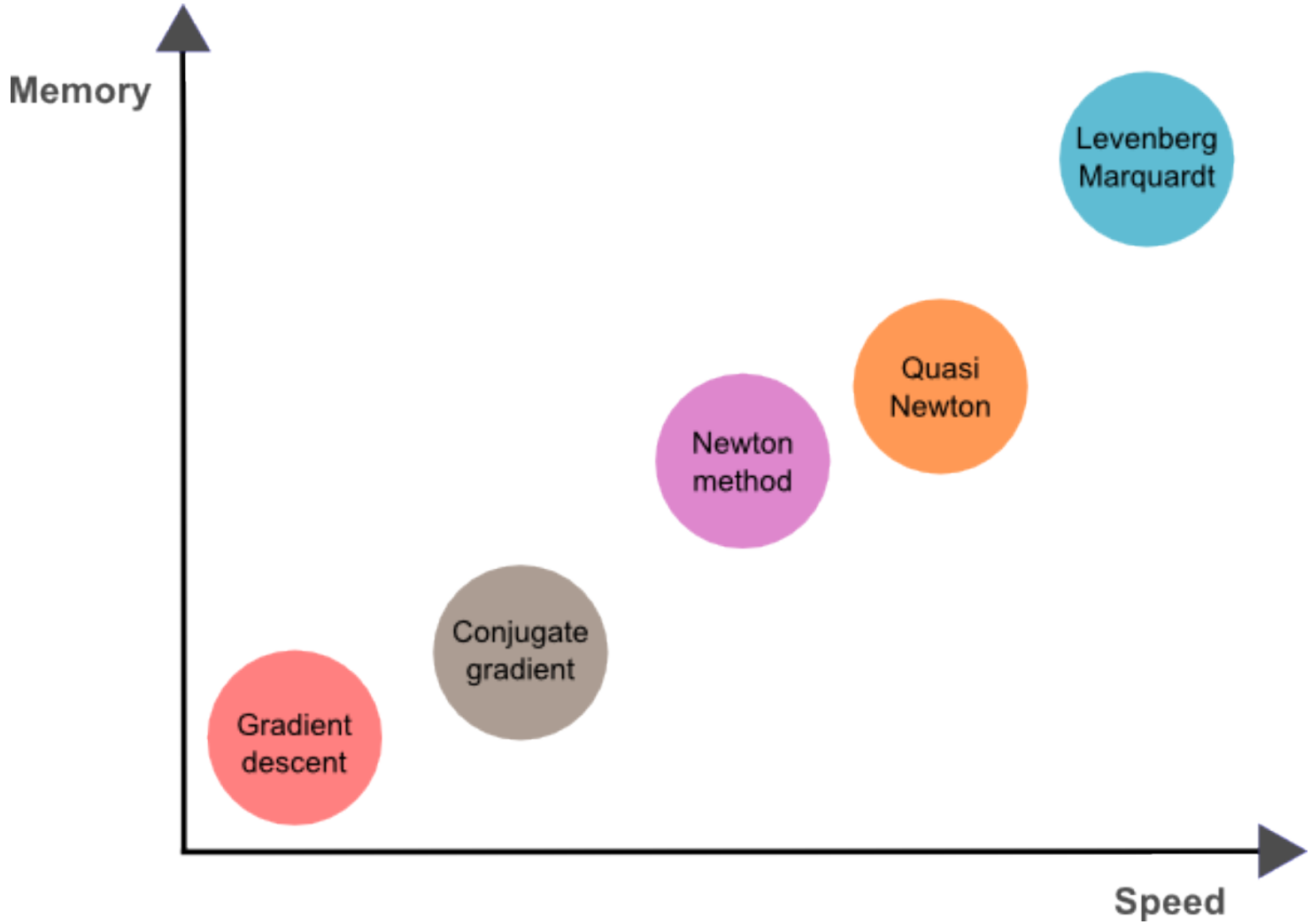
- pierwszy kierunek $\mathbf{w}(0)$ wyznaczamy za pomocą spadku gradientu
- w każdym kroku wyszukujemy minimum wzdłuż pojedynczego kierunku (liniowe szukanie) sprzężonego
- po n krokach (gdzie n jest liczbą optymalizowanych parametrów) metodę inicjujemy ponownie w ostatnio znalezionym punkcie
- dla kwadratowej funkcji kosztu w n wymiarach metoda CG osiąga minimum w n krokach
- zbieżność znacznie szybsza od GD bez konieczności wyznaczania $\mathbf{H}^{-1}$
- małe zapotrzebowanie pamięciowe

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

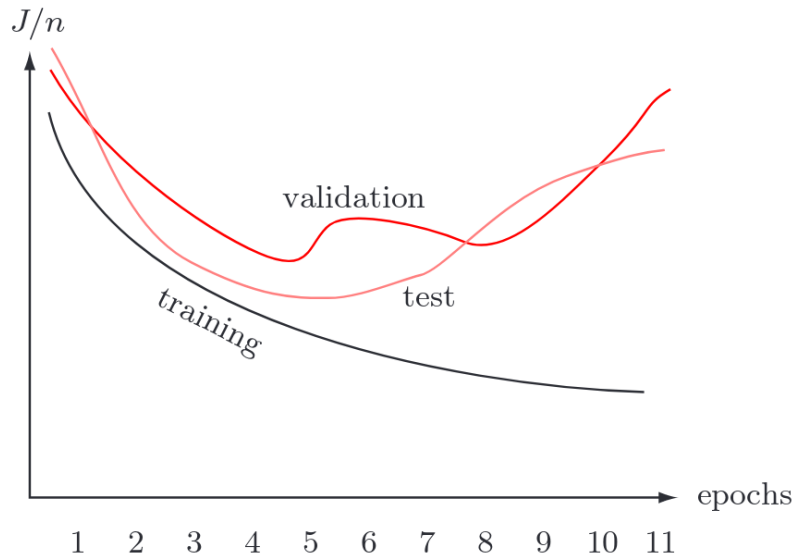


Rys: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network

Generalizacja i regularyzacja sieci MLP

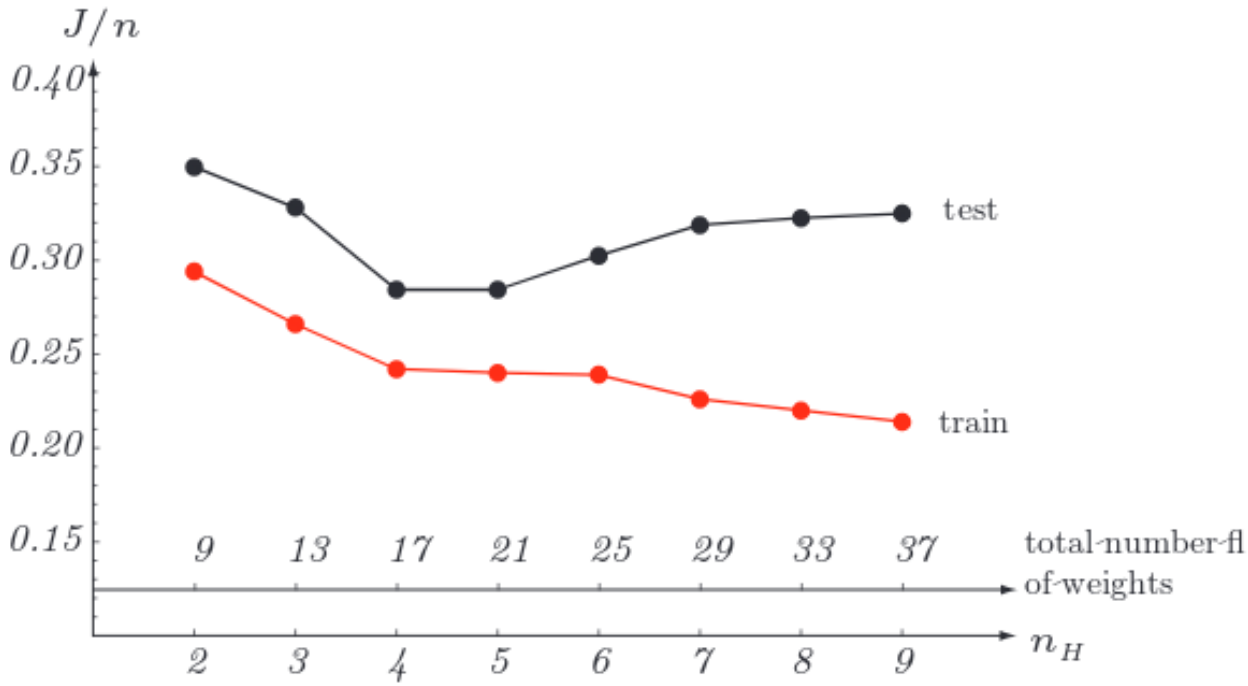
Generalizacja

- Na zbiorze treningowym błąd jest w stanie zawsze osiągnąć 0
- **Generalizacja** jest celem uczenia - uogólnienie reguły, która wytworzyła dane. Dążymy do minimalizacji błędu na danych, które nie zostały użyte w treningu.
- Zbiór **walidacyjny** - wyodrębniony fragment danych (np. ze zbioru uczącego) pozwala na ocenę błedu generalizacji - przydatny przy określaniu punktu zatrzymania treningu



Duda, Hart, Pattern Recognition

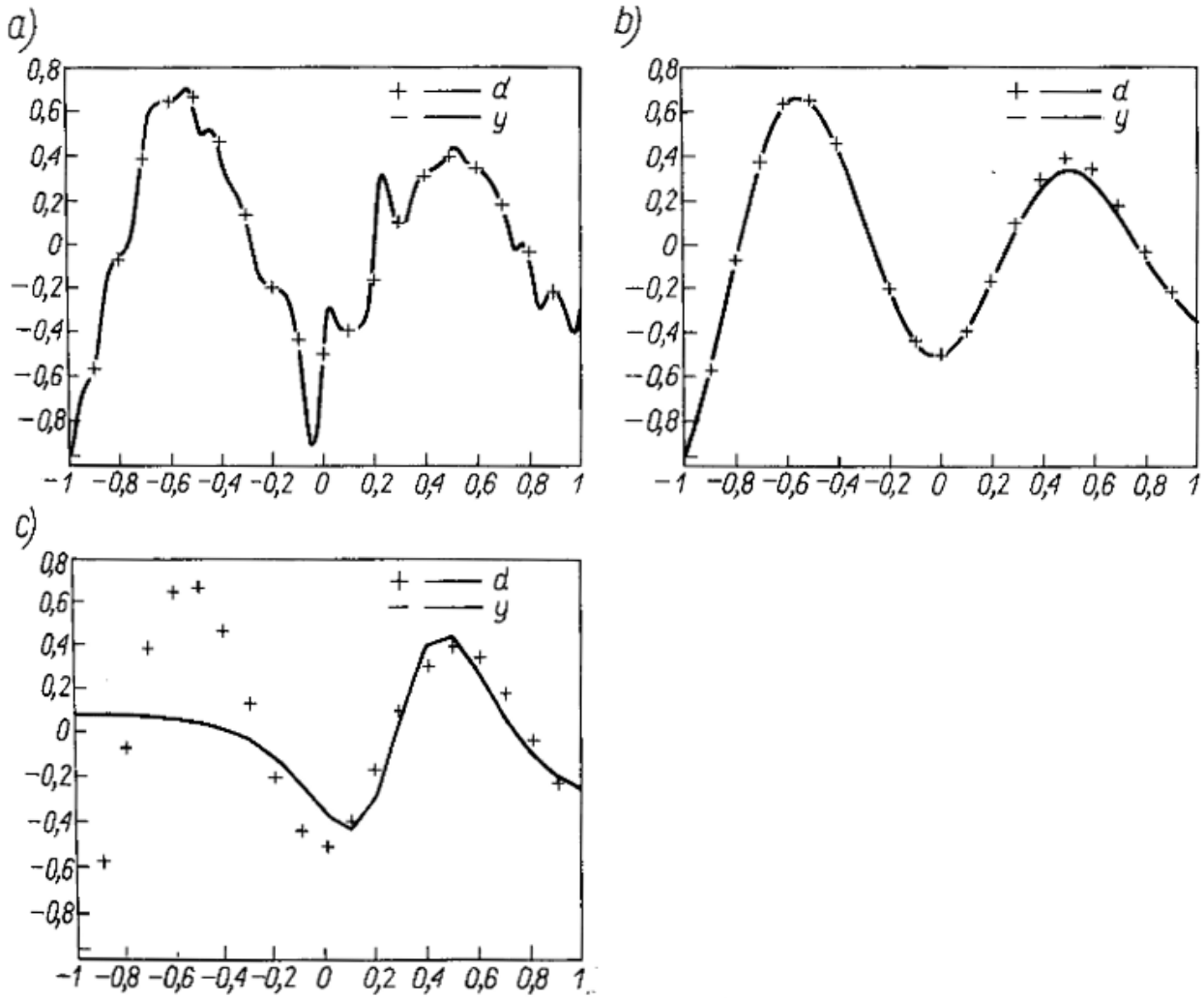
Rozmiar sieci



- ilość wag określa ilość stopni swobody - nie powinna przekraczać liczby przypadków uczących (np. $n/10$)
- za dużo wag - model uczy się na pamięć (przeuczenie)
- za mało wag - model zbyt prosty (niedouczony)

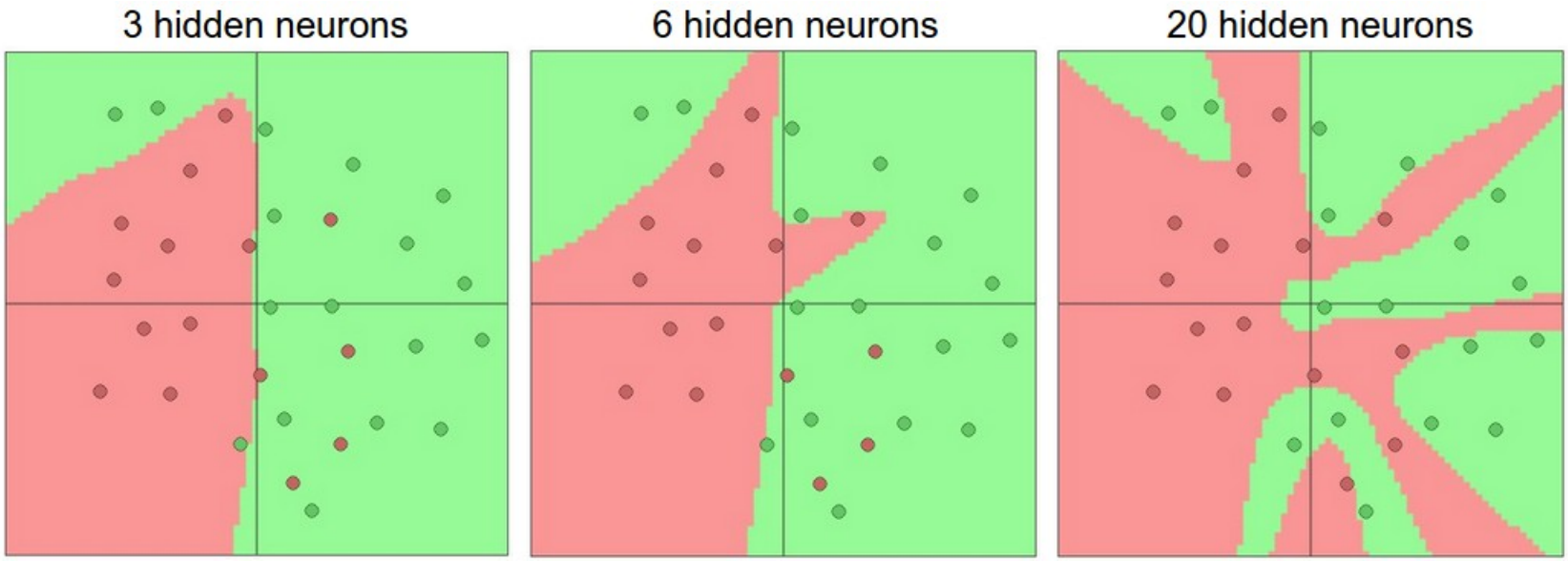
Rozmiar warstwy ukrytej

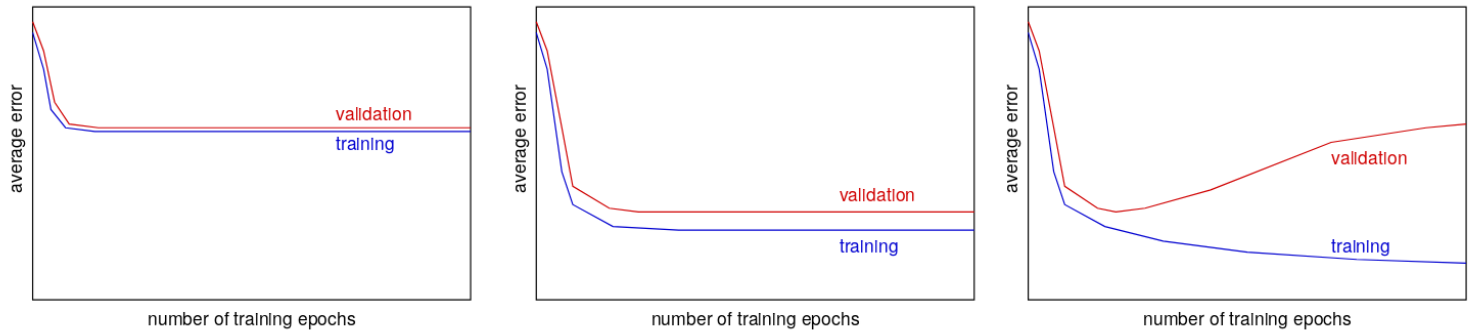
Zdolność uogólniania w problemie aproksymacji



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

Regularyzacja

Regularyzacja to metody, których celem jest poprawienie generalizacji

- dobór wielkości (liczby neuronów) modelu, preferowane najprostsze rozwiązania (brzytwa Ockhama)
  - z pośród wielu modeli wybierz ten o najmniejszym błędzie walidacyjnym i najmniejszej liczbie parametrów (neuronów)
  - metody konstrukcyjne (rosnące) - zacznij od najprostszego modelu i zwiększaj jego złożoność w kolejnych iteracjach
  - pruning - usuwanie nadmiarowych połączeń lub neuronów z wytrenowanej sieci
  - sieci ontogeniczne (rozrastające się i kurczące się)
- małe wartości wag są preferowane
  - 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

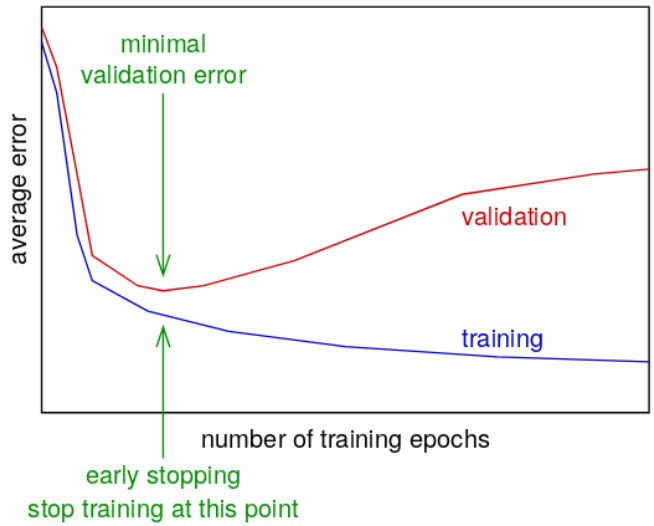
$$\bar{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda\Omega(\mathbf{w})$$

  gdzie $\lambda > 0$ steruje wielkością regularyzacji
- wczesne zatrzymanie treningu, zanim błąd walidacyjny wzrośnie
- zwiększenie liczby danych lub przetransformowanie cech do „przyjemniejszej” postaci
- dodanie szumu do wag lub danych treningowych
- selekcja cech - wypór tylko istotych cech do czuenia
- uśrednianie wyników z wielu modeli, boosting, baging

Trening i walidacja

- uczenie sieci minimalizuje błąd treningowy a nie błąd generalizacji
- zbiór walidacyjny pozwala oszacować błąd generalizacji w trakcie treningu
- zbiór testowy - uzywany wyłacznie do ewalucaji nauczonego modelu
- kroswalidacja - metoda pozwalająca na oszacowanie błędu generaliazcji i wariancji modelu poprzez powtaraznie treningu i ewaluacji na kolejnych podziałach zbioru uczącego
- k krotna kroswalidacja - dzieli zbiór na k częśći, gdzie $k-1$ jest używanych do treningu a reszta do testu
- wyznaczamy średni błąd z k treningów
- kroswalidacja leave-one-out - gdy N rózne liczbie wektorów uczących

Wczesne zatrzymanie



- przerwij trening, gdy błąd walidacyjny zacznie rosnąć, często wystarczy tylko kilka iteracji
- zapamiętaj model o najmniejszym błędzie walidacyjnym
- wymaga wydzielenia części zbioru treningowego, więc zmniejszenie liczby próbek uczących

Rys: Riedmiller, Machine Learning

Dobór liczby neuronów

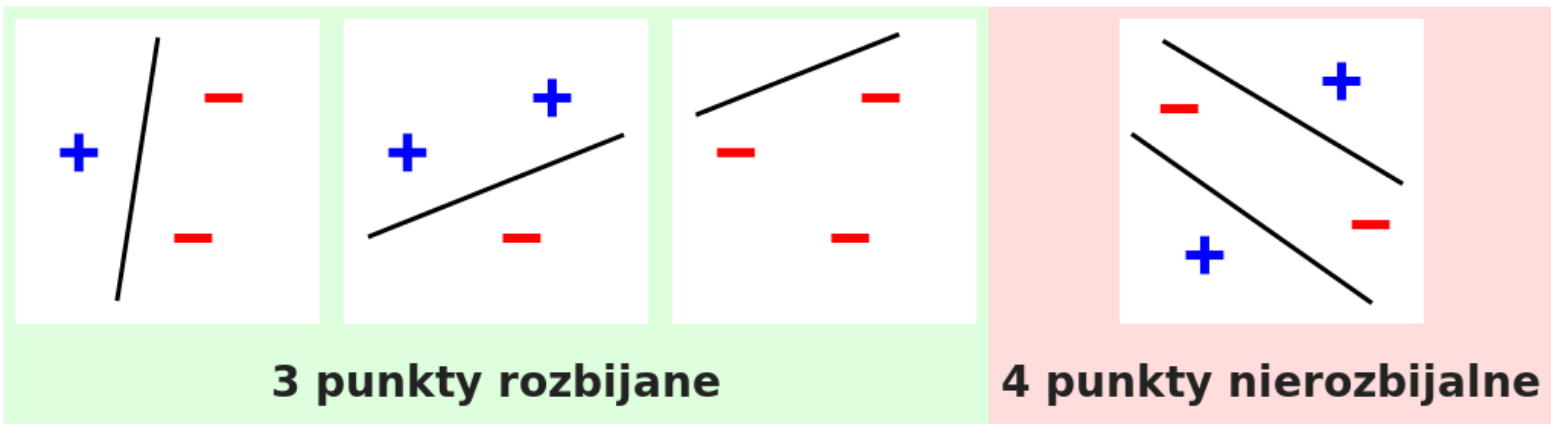
Probabilistyczne oszacowanie błędu generalizacji sieci MLP klasyfikującej

$$E_G(\mathbf{w}) \leq E_T(\mathbf{w}) + \epsilon\left(\frac{N}{D}, E_T\right)$$

gdzie
N - liczba wzorców uczących,
D - miara złożoności Vapnika-Chervonenkisa (VC dimension)
ϵ - przedział ufności maleje ze wzrostem $\frac{N}{D}$
Przeuczenie może zajść gdy $D > N$

Wymiar VC

Wymiar VC klasyfikatora to największa liczba punktów, którą klasyfikator jest w stanie rozbić, tj. podzielić przy dowolnej kombinacji etykiet (2^N możliwości)



Dla perceptronu $D = 3$

Wymiar VC dla MLP

Dla sieci MLP

$$2\left\lceil \frac{N_h}{2} \right\rceil d \leq D \leq 2N_w(1 + \log N_n)$$

N_h - ilość neuronów ukrytych
d - ilość cech (wymiar przestrzeni wejściowej)
N_w - liczba wag
N_n - liczba wszystkich neuronów

- w praktyce $D \approx N_w$
- liczba próbek powinna być kilkukrotnie (np. 10 razy) większa od wymiaru VC

Regularyzacja L_2

Regularyzacja L_2 (weight decay, regularyzacja Tichonova) - czynnik regularyzacyjny dążący 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

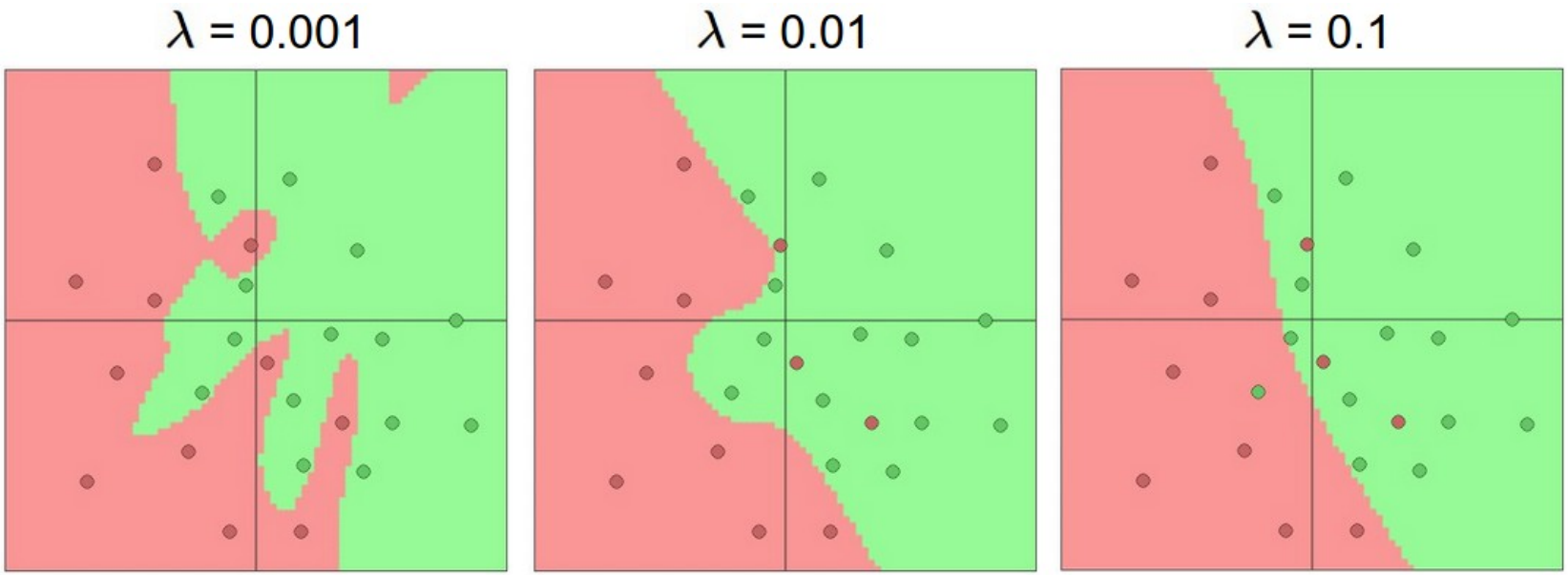
Zmniejszone są wszystkie wagi w trakcie uczenia

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

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/

Zmodyfikowany człon kary:

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

Minimalizacja E powoduje:

- zmniejszenie wartości wag w_{ij}
- eliminację neuronów dla których $\sum_k |w_{ik}|$ jest bliskie zera

Regularyzowana zmiana wag zależna jest od wartości w_{ij}

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

Dla neuronów z dużym $\sum_k |w_{ik}|$ czynnik regularyzacyjny zanika

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

Metody wrażliwościowe redukcji sieci

- Eliminacja wag na podstawie jej wpływu na wartość błędu
- Wagi o małej wartości $|w_i|$ są mniej istotne od dużych wag
- wrażliwość połączenia

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

- połączenia o najmniejszej wrażliowści 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 w szereg Tylora i przyjmując $\Delta E = 0$ (punkt zbieżności, sieć wytrenowana)

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

Miara ważności wagi

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

Wagi o najmniejszym S_i mogą być usunięte z wytrenowanej sieci.

Optimal brain Damage

1. Wytrenuj sieć dowolnym algorytmem uczenia
2. Wyznacz diagonalne elementy hesjanu oraz wartości wrażliwości S_i
3. Posortuj wagi zgodnie z rosnącymi wartościami S_i i obetnij te o najmniejszych wartościach.
4. Wróć do punktu 1

Redukcja neuronów o małej aktywności

- Neurony o niewielkiej aktywności (których sygnał wyjściowy nie zmienia się dla zbioru treningowego) można usunąć bez szkody dla generalizacji
- duża aktywność neuronu świadczy o jego przydatności
- modyfikacja funkcji kosztu (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

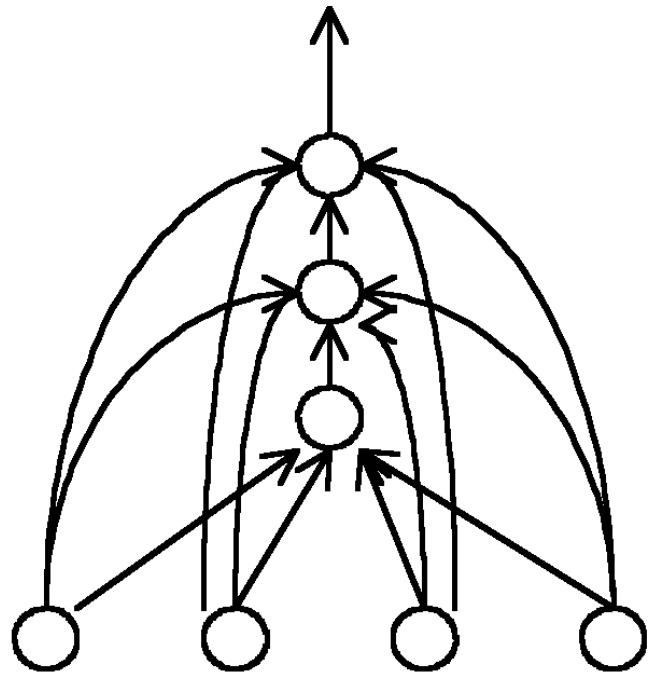
$$e = \frac{1}{1 + \Delta_{ij}^2}$$

Architektury konstruktywne (rozrastające się)

- automatycznie dobierają złożoność modelu do złożoności problemu
- od prostych do złożonych modeli, strategia zachłanna
- przedwczesne przerwanie uczenia nie musi być katastrofą
- (zazwyczaj) niski koszt obliczeniowy, np. w każdym cyklu douczamy tylko dodatkowy neuron, reszta połączeń jest „zamrożona”
- możliwość budowania sieci o różnorodnych funkcjach aktywacji

Algorytm wieżowy

1. dodaj neuron tworząc nową warstwę
2. trenuj do uzyskania zbieżności
3. zamróź wagi
4. wróć do punktu 1

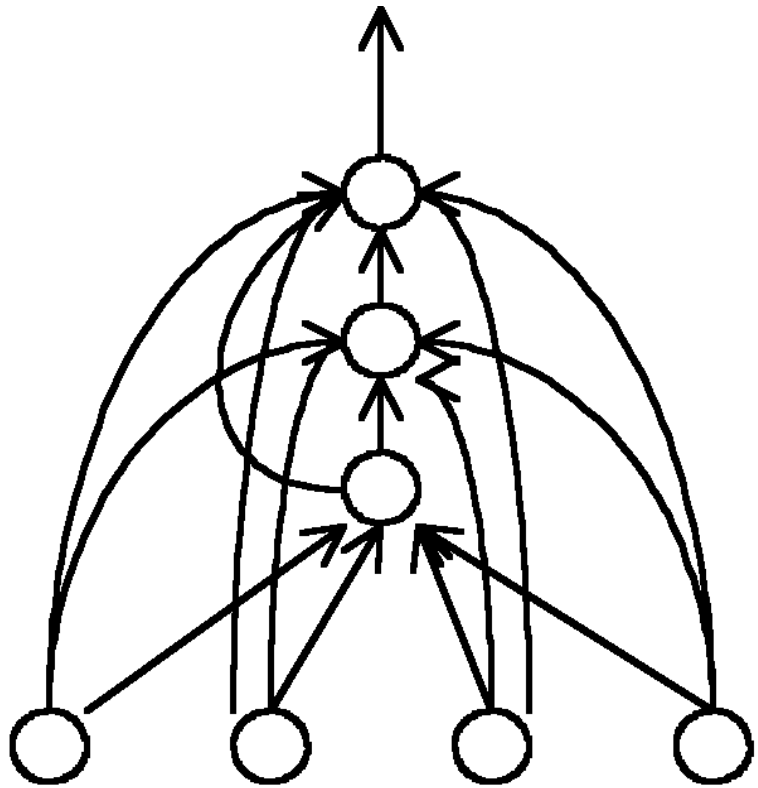


Zbiega się po skończonej liczbie kroków dla wypukłych danych.

Każda warstw usuwa przynajmniej jeden błąd, ale generalizacja może być kiepska.

Algorytm piramidalny

Uczenie - podobnie jak w algorytmie wieżowym.
Każdy neuron posiada wejście do wszystkich poprzednich warstw.



Korelacja kaskadowa (Fahlman i Labiere, 1989)

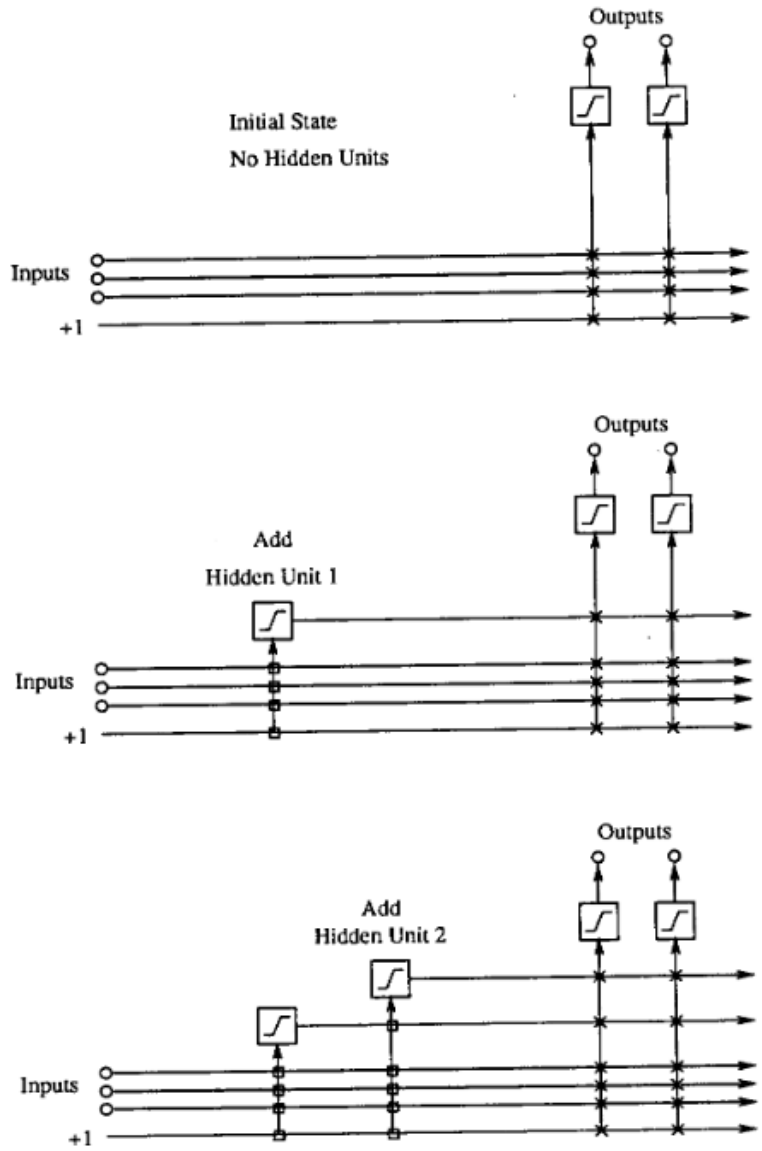
Sieć rozrastająca się:

- zaczynamy od treningu sieci bez warstwy ukrytej
- w kolejnych krokach dodawany jest neuron do warstwy ukrytej, wejścia neuronu połączone są ze wszystkimi wejściami sieci i wyjściami poprzednio dodanych neuronów ukrytych
- wagi neuronu dobierane są w procesie maksymalizacji korelacji nowego neuronu k z błędem wykazywanym przez neurony wyjściowe

$$C = \sum_{j=1}^{M} \left| \sum_{i=1}^{N} (o(\mathbf{x}_i) - \overline{o}) \left((y_{ij} - f_j(\mathbf{x}_i)) - (\overline{y}_{ij} - \overline{f}_j(\mathbf{x}_i)) \right) \right|$$

gdzie N - liczba wzorców, M - liczba wyjść sieci

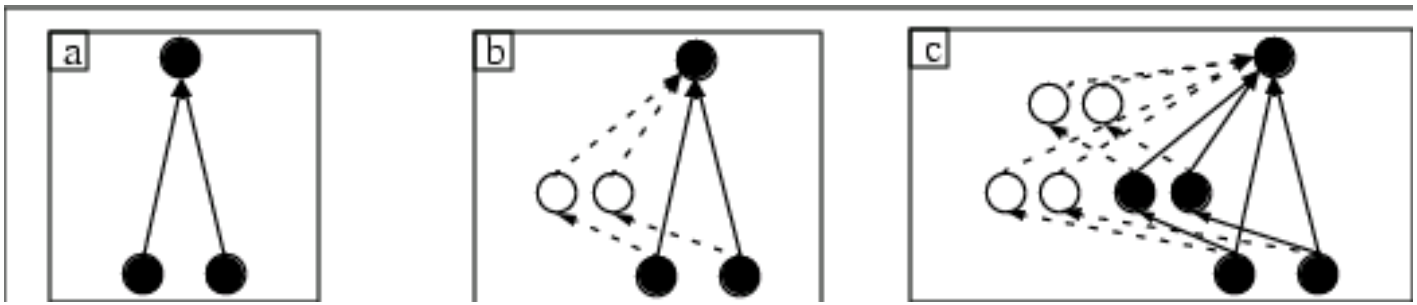
Korelacja kaskadowa



Kaskadowa korelacja

- wagi kandydata maksymalizujące korelację są „zamrażane” - douczaniu podlegają wyłącznie wagi wyjściowe sieci
- jeśli wyjścia neuronu kandydata są skorelowane dodatnio z błędem to w wyniku treningu wykształci wagi, które mają szansę zniwelować ten błąd
- w każdym kroku rozpatruje się zbiór kandydatów (od 5 do 10), mogą posiadać różne funkcje aktywacji (sigmoidalna, gaussowaka, itp.),
- kandydaci trenowani są równolegle konkurując ze sobą, wybiera się najlepszego (o największej korelacji)

Flex net (Mohraz, Protzel 1996)



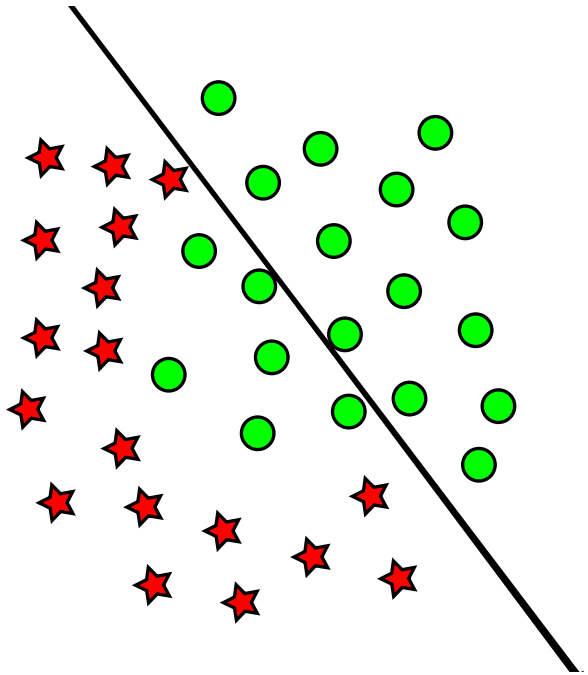
1. startuj bez warstwy ukrytej
2. dodaj kandydatów i trenuj sieci niezależnie
   Kandydaci mogą pojawić się w istniejących warstwach lub tworzyć nowe warstwy
3. z pośród kandydatów wybierz te, które najlepiej poprawiły wynik
4. wróć do punktu 2 aż do uzyskania zadowalających rezultatów

Klasyfikator cząstkowy

Niech $\mathbb{Q}^+ \cup \mathbb{Q}^- \subset \mathbb{X} \quad \wedge \quad \mathbb{Q}^+ \cap \mathbb{Q}^- = \emptyset$

Klasyfikator cząstkowy f_c

daje odpowiedź $+1$ dla co najmniej jednego obiektu ze zbioru $\mathbb{Q}^+$ oraz odpowiedź -1 dla wszystkich elementów ze zbioru $\mathbb{Q}^-$



Niech $\mathbb{R} = \{(\mathbf{x}_i, y_i) : f_c(\mathbf{x}_i) = +1\} \subset \mathbb{Q}^+$

Dla funkcji logicznych (wejścia binarne) zawsze istnieje klasyfikator cząstkowy w postaci perceptronu

Ogólna konstruktywistyczna metoda sekwencyjna

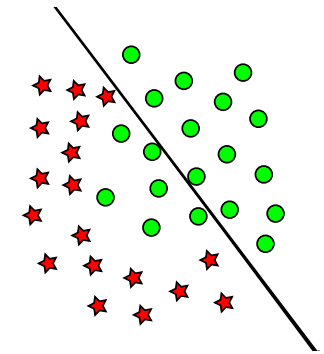
General Sequential Constructive Method (GSCM, Musseli, 1998)

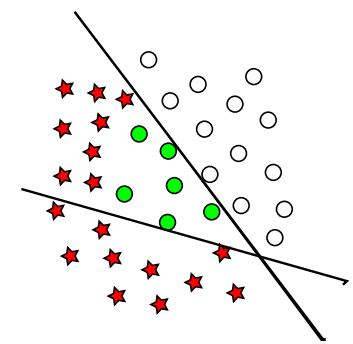
Algorytm 4 GSCM problem dwuklasowy

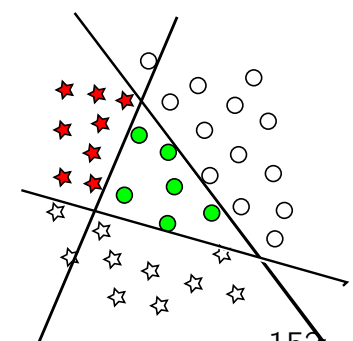
Input: Zbiór treningowy $\mathbb{D}$

Output: Sieć jednowarstwowa

1: $h \leftarrow 0$ ▷ pusta warstwa ukryta
2: **while** zbiór $\mathbb{D}$ zawiera przypadki z przeciwnych klas **do**
3: $h \leftarrow h + 1$
4: wybierz etykietę $d_h = \{-1, +1\}$
5: wytrenuj klasyfikator cząstkowy f_h dla klasy d_h
6: $\mathbb{D} \leftarrow \mathbb{D} \setminus \mathbb{R}_h$
7: Utwórz wynikową sieć zawierającą klasyfikatory cząstkowe w warstwie ukrytej







Przykładowe wagi połączeń do warstwy wyjściowej

$$u_0 = \sum_{j=1}^{h} u_j + d_{h+1}, \qquad u_j = d_j 2^{h-j} \quad \text{dla} \quad j = 1, \ldots, h$$

Przykłady algorytmów

Realizacja klasyfikatora cząstkowego za pomocą perceptronu:

Algorytm nieregularnego podziału (Irregular Partitioning IPA, Marchand, Golea, 1993)

1. startuje z pustego zbioru odseparowanych $\mathbb{R} = \emptyset$
2. dla kolejnych wektorów $\mathbf{x} \in \mathbb{Q}^+ \setminus \mathbb{R}$
   trenuje perceptron w poszukiwaniu hiperpłaszczyzny oddzielającej $\{\mathbf{x}\} \cup \mathbb{R}$ od $\mathbb{Q}^-$
   jeżeli istnieje separacja to $\mathbb{R} \leftarrow \mathbb{R} \cup \{\mathbf{x}\}$
3. zwróć ostatnio uzyskany perceptron

Przykłady algorytmów

Realizacja klasyfikatora cząstkowego za pomocą perceptronu:

Algorytm zamiany etykiet (Target Switch TSA, Campbell, Vicente, 1995)

1. wytrenuj perceptron dowolną metodą
2. jeżeli istnieje $\mathbf{x} \in \mathbb{Q}^-$, który jest źle klasyfikowany to znajdź najdalej oddalony od płaszczyzny decyzyjnej, błędnie klasyfikowany wektor z $\mathbb{Q}^+$ i zmień jego etykietę (przenieś do $\mathbb{Q}^-$)
3. powtarzaj aż do uzyskania perceptronu realizującego klasyfikator cząstkowy

Algorytmy rozrastające się - podsumowanie

- problem z przeuczeniem, kiedy zaprzestać rozrost? Za dużą sieć można przyciąć
- koszt obliczeń zależy od czasu uczenia pojedynczego KC
- złożone struktury danych mogą nie zostać wykryte przez dodanie pojedynczego neuronu, zachłanna strategia nie zawsze jest najlepsza
- algorytmy rosnące nie gwarantują najprostszych sieci
- niektóre tworzą specyficzne architektury
- sieci ontogeniczne - rosną i kurczą się w trakcie treningu
  - wyszukiwanie najlepszej architektury, stosuje się np. alg. genetyczne, ewolucyjne
  - zazwyczaj wymagające obliczeniowe

Rozmiar zbioru treningowego

- większa liczba danych zmniejsza ryzyko przeuczenia
- „there's is no data like more data"
- dodanie szumu do danych uodparnia sieć na drobne zmiany w przestrzeni wejściowej
- sztuczne zwielokrotnianie danych, często stosowane z dobrymi rezultatami
  - w klasyfikacji obrazów: zmiany kontrastów i nasycenia barw, przesunięcia, obroty, itp.
  - w analizie mowy: pogłos, szum otoczenia, zmiana głośności, przyspieszenie i zwolnienie mowy, itp..

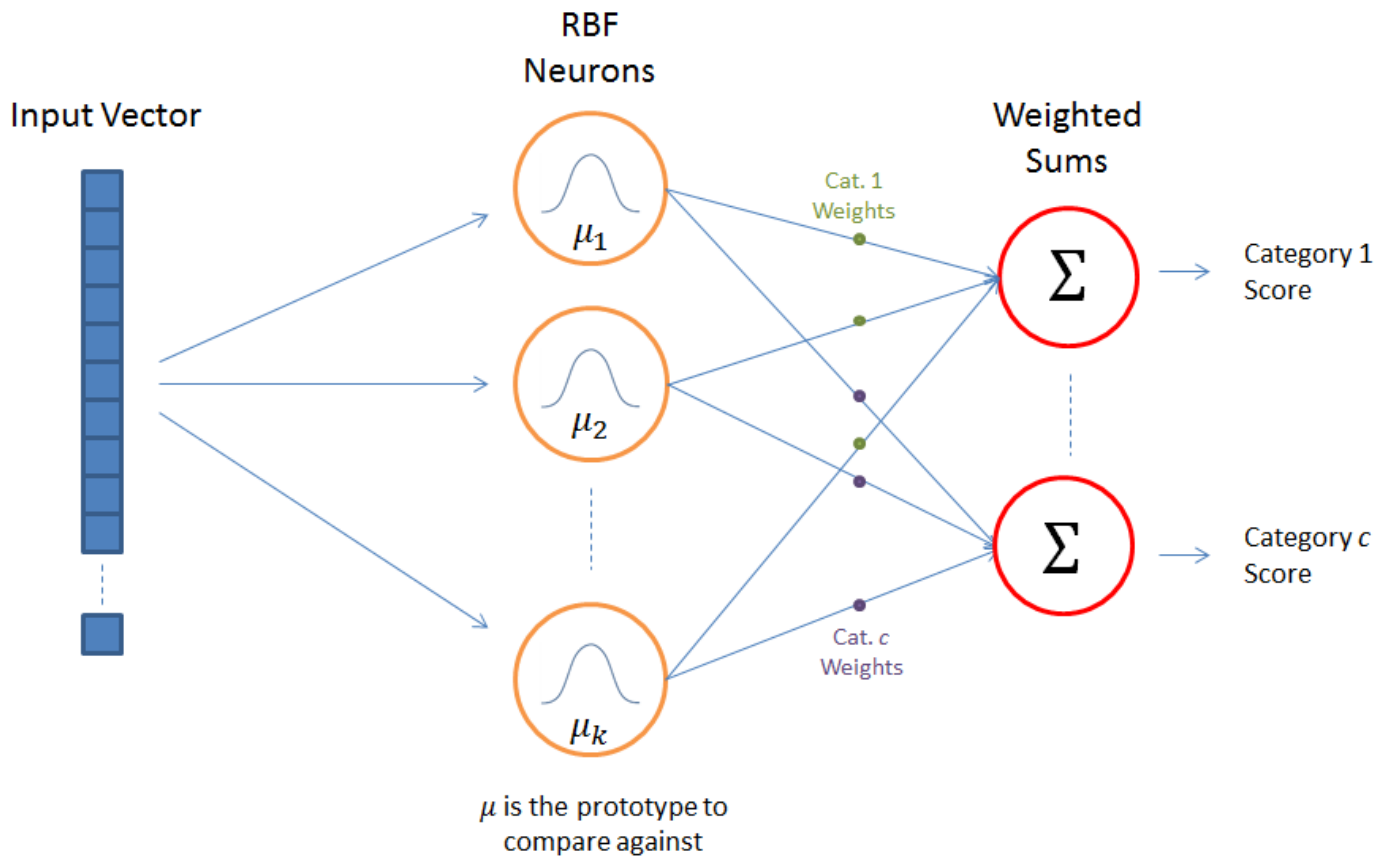
Selekcja i ekstrakcja cech

- dane treningowe często zawierają cechy, które nie wnoszą wiele w treningu modelu i mogą wpływać negatywnie na trening
- selekcja cech usuwa nieistotne zmienne zmniejszając rozmiar zbioru treningowego
- dodanie cech, które niosą wartościową informację poprawia generalizację
- istnieje wiele metod, najczęściej dobierane są do specyfiki problemu
- niezbalansowane klasy - częsty problem w klasyfikacji

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:

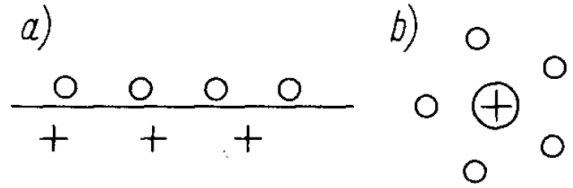
$$f(\mathbf{x}) = \sum_{i=1}^{K} w_i \varphi(\|\mathbf{x} - \mathbf{c}_i\|) + w_0$$

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

- jedna warstwa ukryta, neurony z radialnymi funkcjami
- funkcja bazowa (radialna) φ określa podobieństwo przypadku (prototypy)
- wagi połączeń neuronów ukrytych określają centrum $\mathbf{c}_i$ funkcji radialnej
- wyjścia liniowe - ważona suma pobudzeń neuronów radialnych

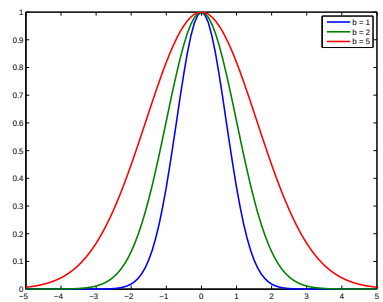
Funkcje radialne

- 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.
- funkcja radialna $\varphi(\|\mathbf{x} - \mathbf{c}\|)$ zanika w obszarach dalekich od centrum $\mathbf{c}$
- Przykład: funkcja Gaussa

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



- współczynnik dyspersji σ decyduje o zasięgu funkcji bazowej

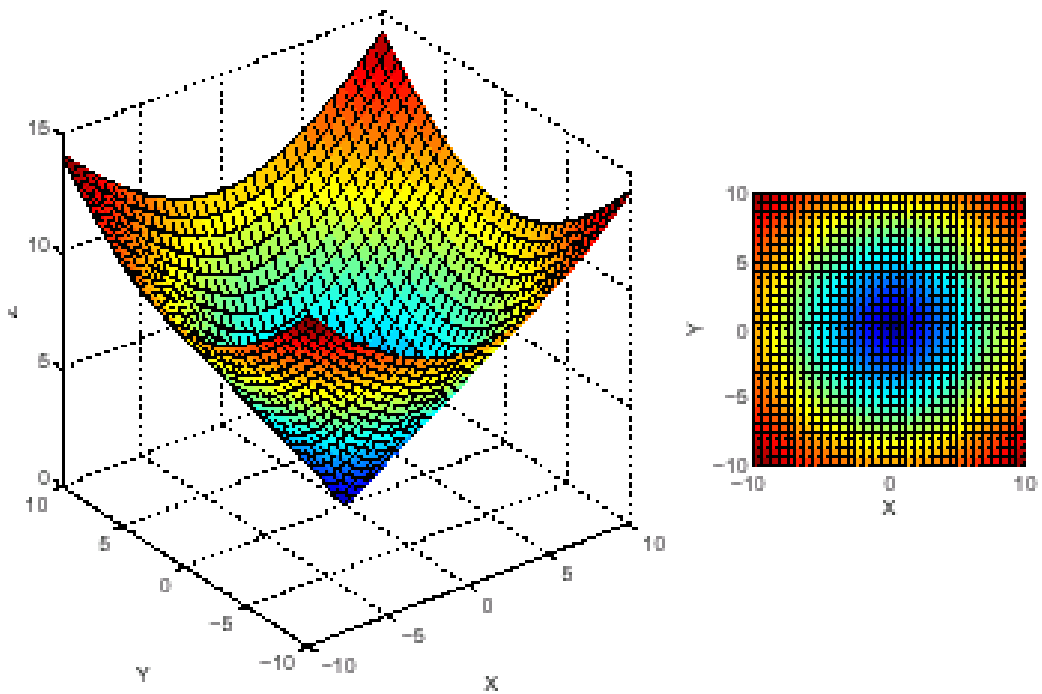
Przykłady funkcji radialnych

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

$h(r) = r$		liniowa
$h(r) = e^{-(\frac{r}{\sigma})^2}$		gaussowska
$h(r) = (\sigma^2 + r^2)^\beta,$	$1 > \beta > 0$	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

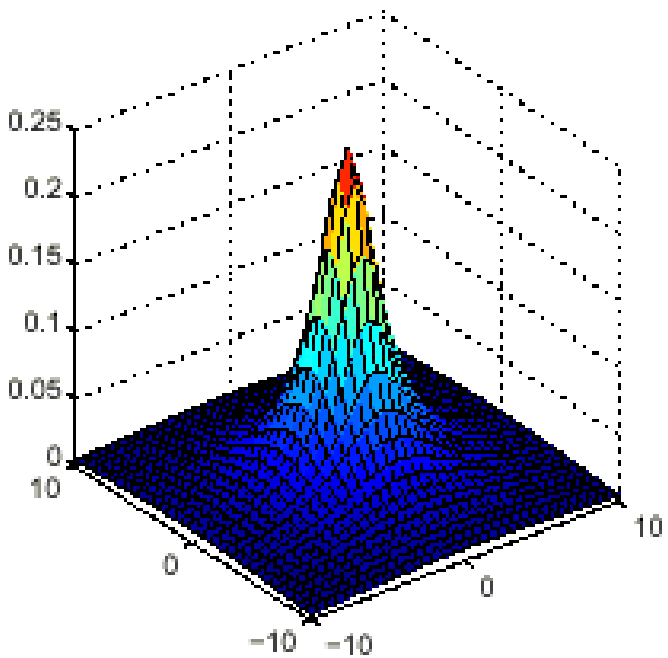
$$h_i(r) = r = \|\mathbf{x} - \mathbf{c}_i\|$$

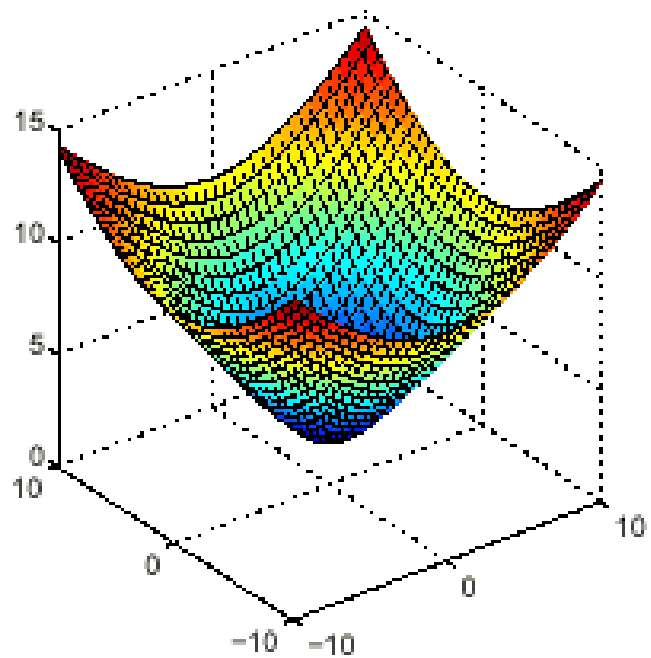


Funkcje wielokwadratowe

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

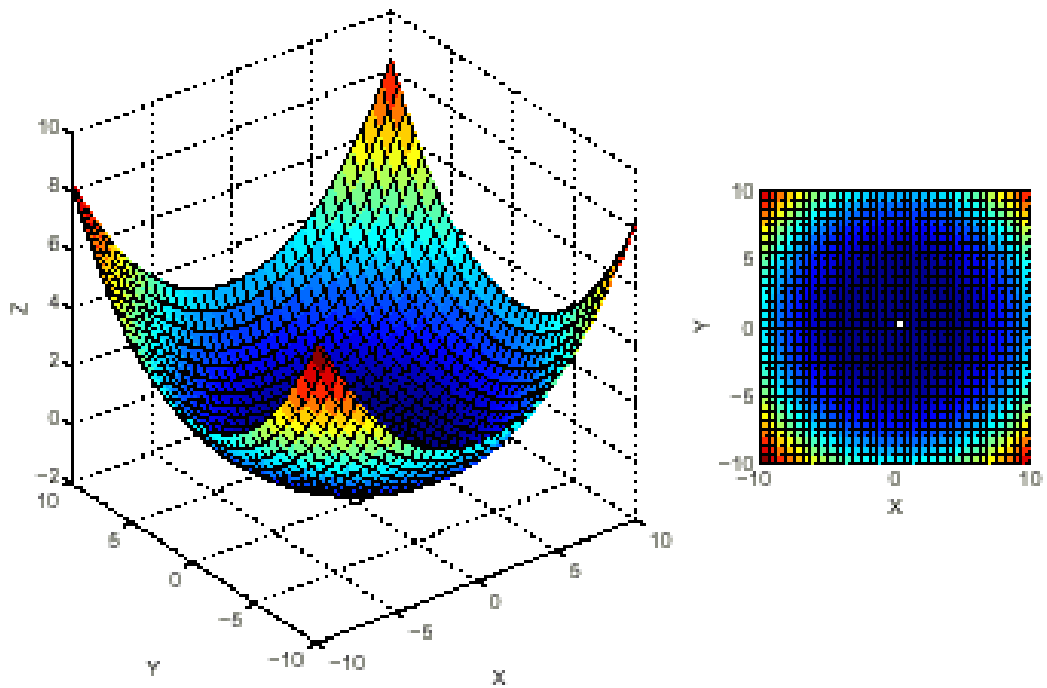
$$h(r) = (\sigma^2 + r^2)^{\beta}, \quad \beta = 0.5$$





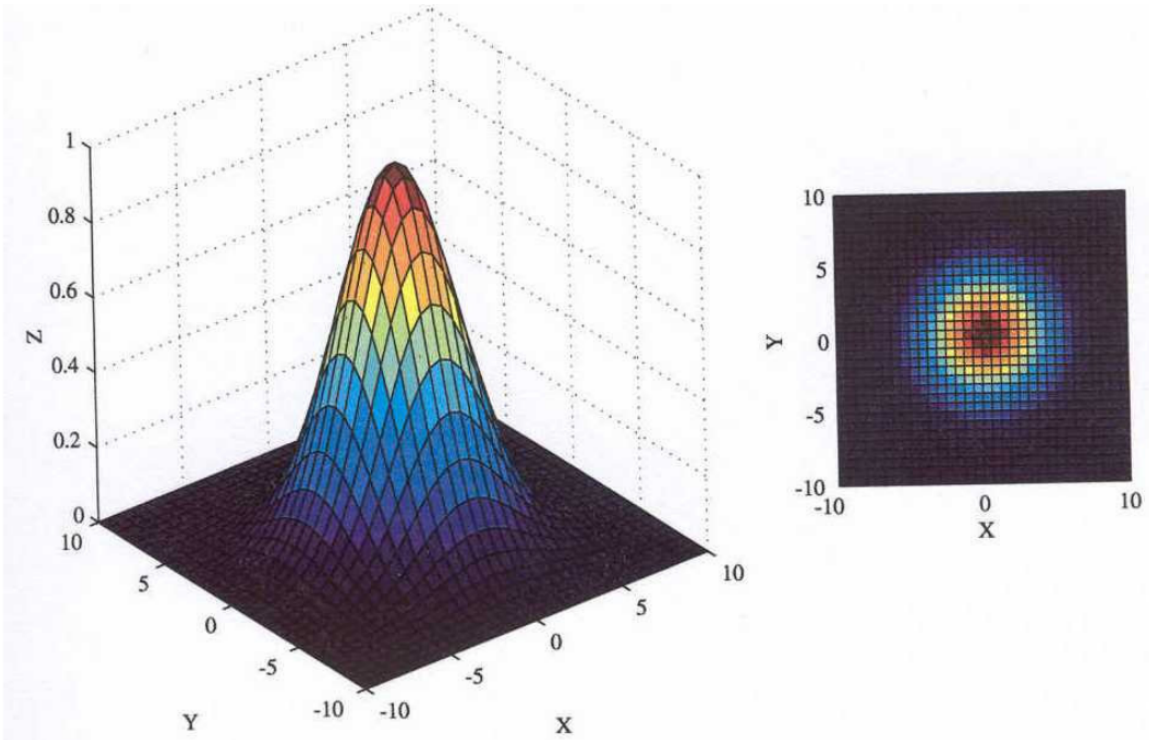
Funkcja cienkiej płytki

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



Funkcja Gaussa

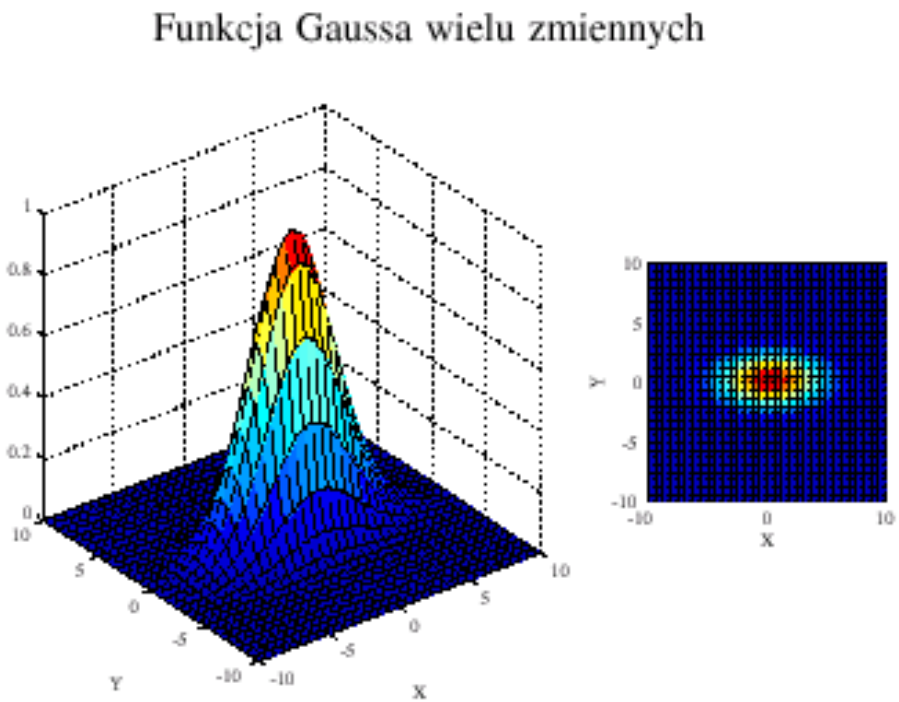
$$h(r) = e^{-\frac{-r^2}{2\sigma^2}}$$



Najczęściej używana wa roli funkcji bazowej RBF

Funkcja Gaussa i z pełną macierzą rozmycia

$$h(r) = e^{-\frac{-r^2}{2\sigma^2}} = e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}_i)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}_1)}$$



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

Uczenie RBF jako problem aproksymacji

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

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

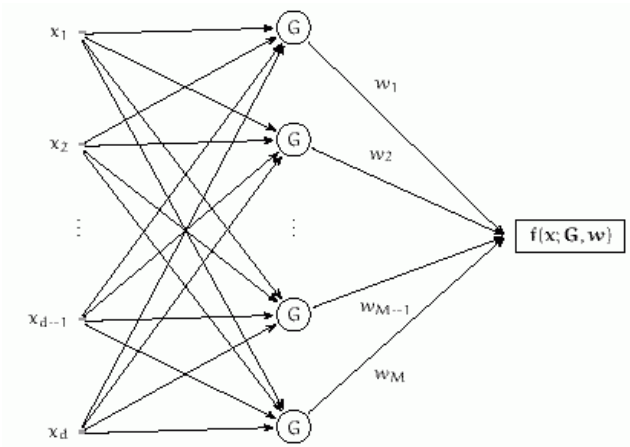
funkcja RBF

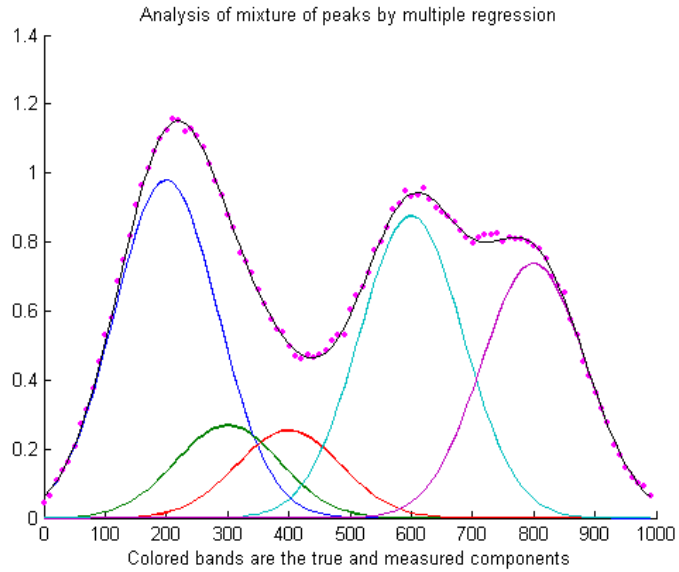
$$f(\mathbf{x}) = \sum_{i=1}^{K} w_i \varphi \left(\|\mathbf{x} - \mathbf{x}_i\| \right)$$

funkcja 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 $K = N$, funkcje bazowe o ustalonych centrach 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 radialnych f. bazowych macierz interpolacji $\mathbf{H}$ dodatnio określona (Light 1992)
- dla wąskich f. Gaussowskich istnieje idealne rozwiązanie, ale zła generalizacja

$$\mathbf{w} = \mathbf{y}$$

- problem źle określony, przewymiarowanie,
- hiperpłaszczyzna interpolacji nie jest gładka
- większe dyspersje σ i mniej funkcji ($K < N$)
  $\rightarrow$ lepsza generalizacja

Aproksymacja RBF

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

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

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

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

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

Rozwiązanie RBF

$$\mathbf{w} = \mathbf{H}^{+}\mathbf{y}$$

dokonując rozkładu

$$\mathbf{H}^{+} = \mathbf{U}\mathbf{S}\mathbf{V}^{T}$$

gdzie $\mathbf{U}$ i $\mathbf{V}$ są ortogonalne o wymiarach $N \times N$ oraz $K \times K$
Macierz $\mathbf{S}$ o rozmiarach $N \times K$ jest pseudodiagonalna, gdzie

$$s_1 \geq s_2 \geq \ldots \geq s_k \geq 0$$

Redukując liczbę kolumn do r naistotniejszych wartości

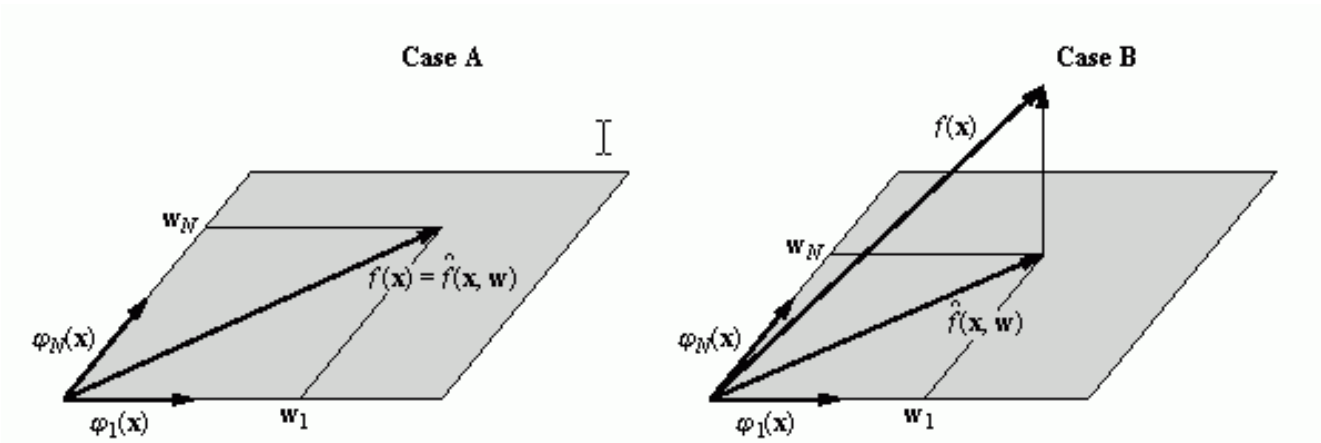
$$\mathbf{w} \approx \mathbf{V}_r \mathbf{S}_r^{-1} \mathbf{U}_r^{T} \mathbf{y}, \qquad \text{gdzie} \qquad s_r^{-1} = [\frac{1}{s_1}, \frac{1}{s_2}, \ldots, \frac{1}{s_r}]$$

- dobór wag w jednym kroku dla ustalonych funkcji radialnych
- macierze $\mathbf{V}_r$ i $\mathbf{U}_r$ są ortogonalne, więc problem jest dobrze uwarunkowany
- głównym problemem jest dobór centrów i rozmyć funkcji φ

Interpretacja geometryczna

Jeśli prawdziwa aproksymowana funkcja $f(x)$ leży w przestrzeni rozpiętej przez wektory bazowe $\Phi(\mathbf{x})$ to możliwe jest rozwiązanie bez błędu, w przeciwnym razie aproksymowana jest projekcja ortogonalna (błąd jest ortogonalny do p-ni bazowej).

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



Rozwiązanie z regularyzacją

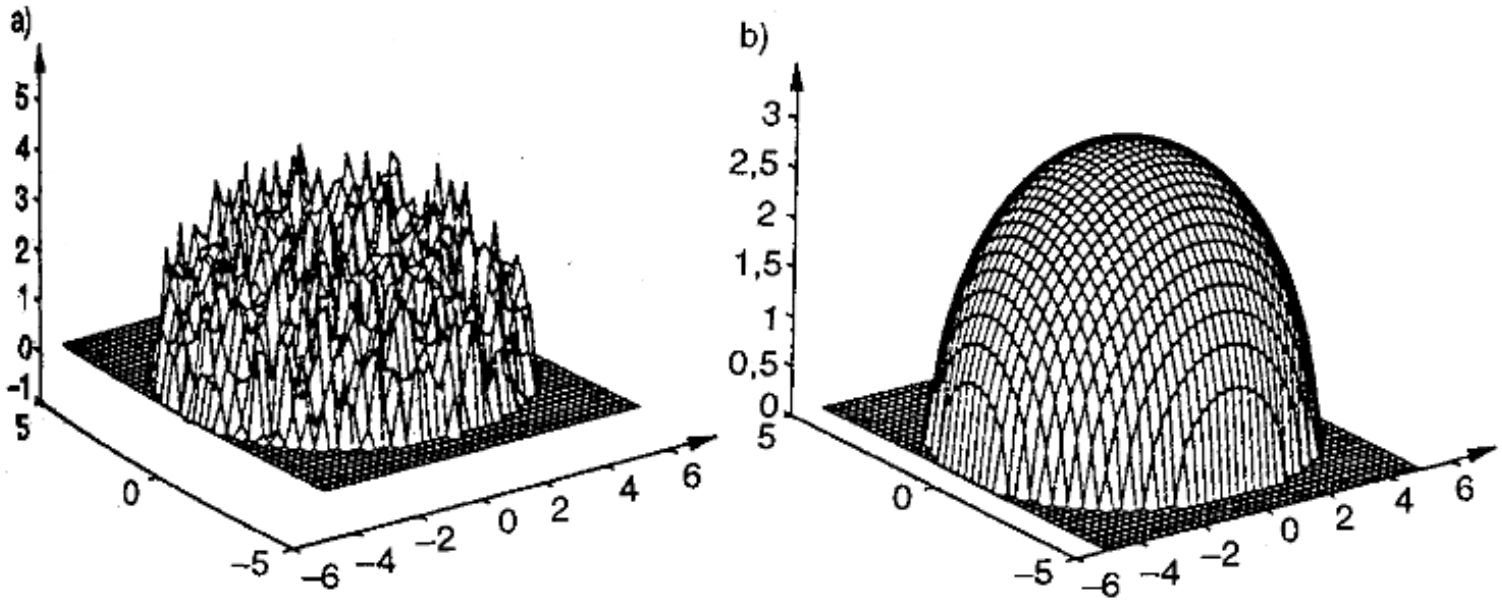
Minimalizacja funkcji błędu z członem regularyzacyjnym

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

gdzie operator $\hat{P}$ wymusza np. gładkość funkcji aproksymującej

$$\left\| \hat{P} F_{\mathbf{w}} \right\|^2 = \int_{\mathbf{R}^N} \left\| \frac{\partial^2 F_{\mathbf{w}}(\mathbf{x})}{\partial x_i \partial x_j} \right\| d\mathbf{x}$$

Wpływ regularyzacji



Nadmiarowa liczba funkcji bazowych o małej dyspersji bez regularyzacji i po regularyzacji (Ossowski 1996)

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

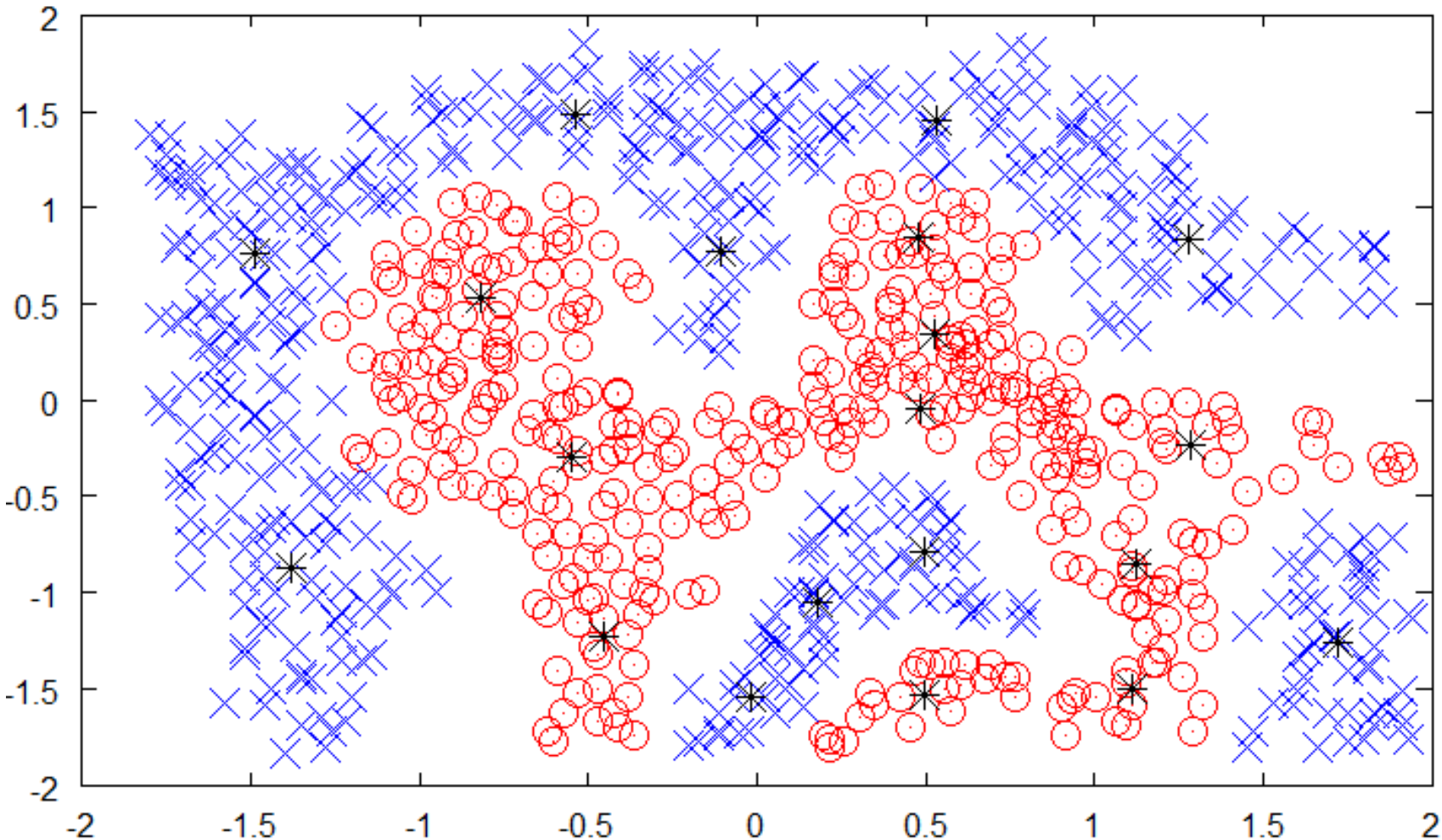
$$\begin{aligned} \mathbf{w}^T \Phi(\mathbf{x}_i) &\geq 0 \quad \text{dla} \quad \mathbf{x}_i \in \mathbb{C}_1 \\ \mathbf{w}^T \Phi(\mathbf{x}_i) &< 0 \quad \text{dla} \quad \mathbf{x}_i \in \mathbb{C}_2 \end{aligned}$$

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.

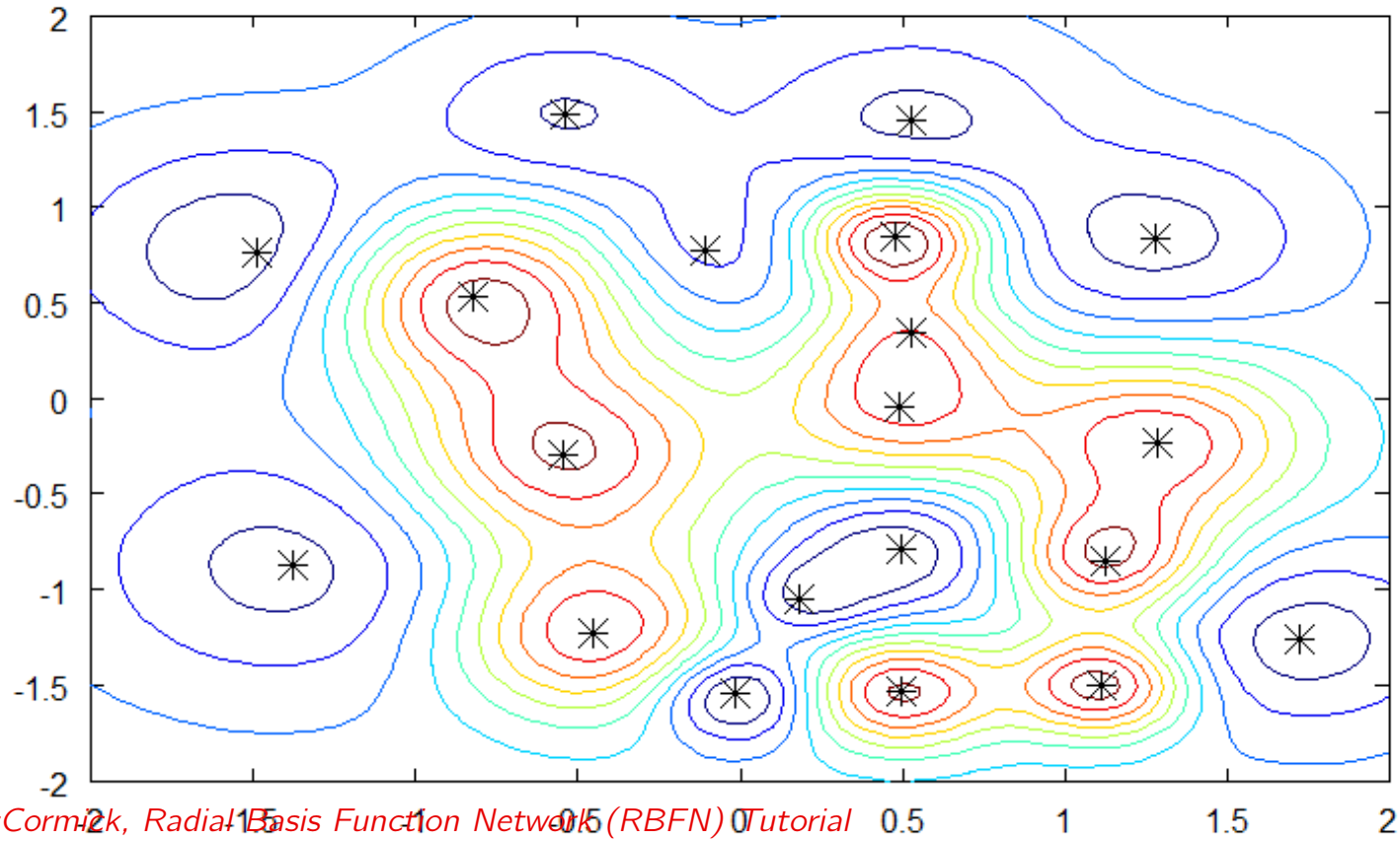
Przykład: 2 klasy



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

Przykład: 2 klasy

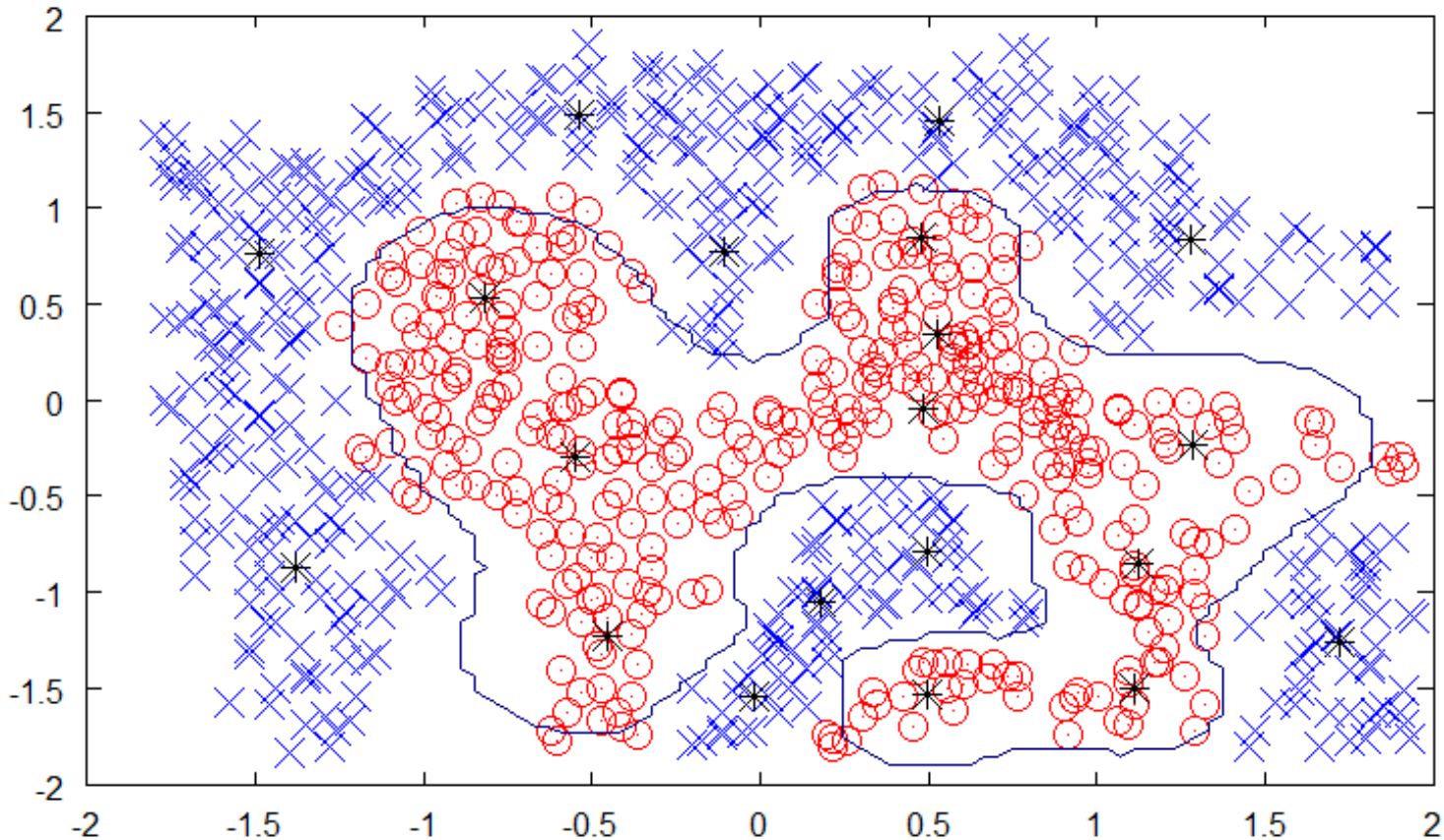
Wartości dla klasy 1, najczęściej wagi powiązane z funkcjami radialnymi dla klasy 1 mają wartości dodanie a dla klasy 2 - ujemne



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

Przykład: 2 klasy

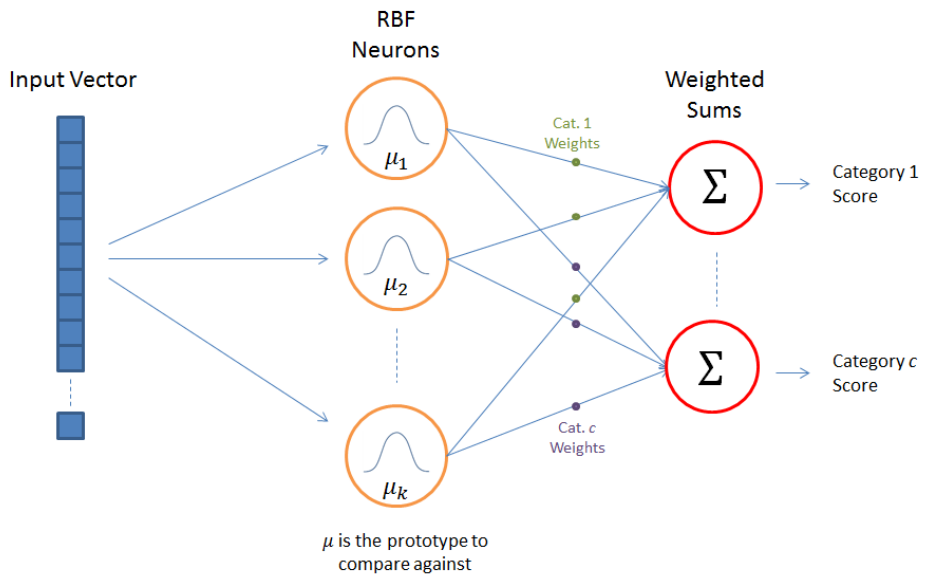
Granica decyzyjna



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

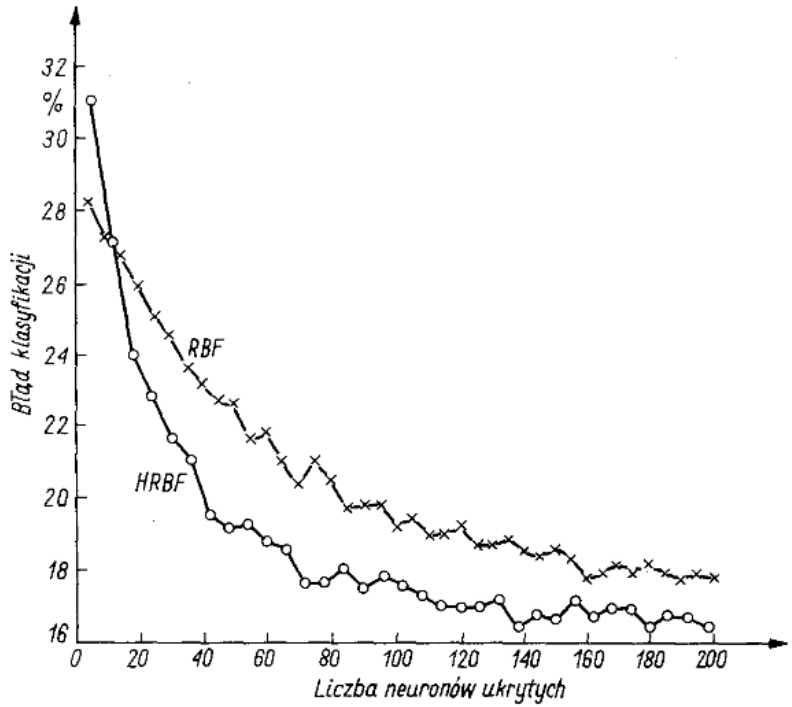
Sieć RBF

- Siec **GRBF** (Generalized RBF) gdy $K < N$ liczba funkcji radialnych jest mniejsza niż liczba wzorców 1
  - liczba funkcji bazowych kluczowa, powinna być mniejsza od liczby wzorców
- Siec **HRBF** (Hyper RBF) używa normy ważonej $\|\mathbf{x}\|_Q^2$ z pełną macierzą obrotów i skalowania $\mathbf{Q}$

$$\left\|\mathbf{x}-\mathbf{c}_i\right\|_{\mathbf{Q}}^2 = \left(\mathbf{x}-\mathbf{c}_i\right)^T \mathbf{Q}^T \mathbf{Q} \left(\mathbf{x}-\mathbf{c}_i\right)$$

posiada o wiele więcej parametrów do uczenia ale zwykle wymaga mniejszej liczby funkcji bazowych.

Dobór liczby parametrów sieci RBF



HRBF uzyskuje podobną poprawność do RBF przy mniejszej licznie neuronów

Ossowski, Sieci Neuronowe

Trening RBF

- Parametry nieliniowe funkcji transferu: centra, dyspersje; + wagi.
- Trening 2 etapowy:
  - najpierw ustawienie funkcji bazowych
  - potem rozwiązanie układu równań (pseudoinwersja)
- Możliwe jednoczesne uczenie centrów, dyspersji i wag, np. za pomocą metod gradientowych (wsteczna propagacja)
- Inicjalizacja początkowych centrów: losowa, klasteryzacja, samoorganizacja, probabilistyczna
- Inicjalizacja dyspersji: wartość stała, zależna od gęstości danych w różnych rejonach, średnie odległości od wektorów z innych klas

Ustalenie centrów

- 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
- w pobliżu granic decyzyjnych (bliskie przypadki z różnych klas) lub w centrach skupisk wektorów z danej klasy
- losowy wybór spośród wzorców treningowych - prosta ale skuteczna metoda
- douczanie pozycji centrów metodami nadzorowanymi lub nienadzorowanymi: klasteryzacja, samoorganicazcja, LVQ, ...

Ustalenie dyspersji

- wymagane gładkie odwzorowanie, rozmycie pełni rolę regularyzacji
- pola recepcyjne wszystkich funkcji bazowych powinny pokrywać cały obszar danych wejściowych
- pola recepcyjne 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.
  Przykładowa funkcja gaussowaka

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

- rozmycie równe średniej odległości od sąsiadujących centrów

Ustawienie dyspersji

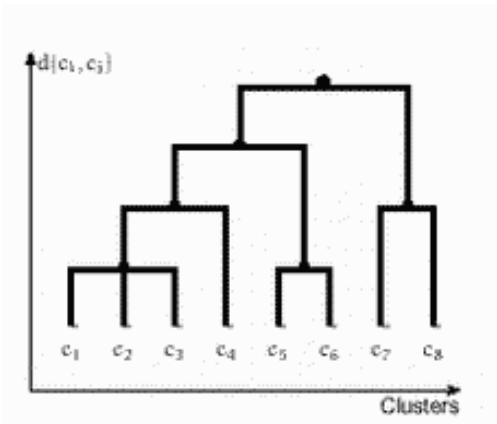
- σ_i dla każdej funkcji radialnej równe odległości euklidesowej od najbliższego sąsiedniego centrum klastra (Tarasenko, 1994)
- uwzględniając odległość od P najbliższych sąsiadów (gdzie zwykle $P \leq 3$) (Moody, 1989)

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

Incjalizacja RBF: klasteryzacja

Zastosuj algorytm klasteryzacji (analizy skupień) i przypisz centra funkcji radialnych centrom klastrów. Np. algorytm k-średnich, dendrogramy, łaczenie histogramów
Metoda dendrogramów:

1. przypisz każdy $\mathbf{x}_i$ do odrępnego klastra
2. połącz najbliższą parę $\mathbf{x}_i$ i $\mathbf{x}_j$ w jeden klaster
3. powtarzaj 2 dopóki nie uzyska się zadowalającej liczby klastrów lub najmniejsza odległość między klastrami przekroczy ustalony poziom



Incjalizacja RBF: klasteryzacja

Algorytm inicjalizacji centrów przez klasteryzację (k-średnich), rozmieszcza centra w centrach skupisk:

1. Rozmieść k centrów równomiernie lub losowo ze zbioru treningowego
2. Dla centrum $\mathbf{c}_i$ znajdź zbiór wszystkich punktów z leżących najbliżej tego centrum
3. Aktualizauj położenie centrum $\mathbf{c}_i$ jako średnią dla punktów znalezionego zbioru

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

4. Powtarzaj dwa ostatnie kroki aż do zbieżności

Inicjalizacja RBF: samoorganizacja

Algorytm inicjalizacji centrów przez samoorganizację (metoda on-line k-średnich):

1. Wybierz losowo k centrów skupisk
2. Dla każdego $\mathbf{x}$ ze zbioru treningowego, znajdź najbliższe centrum $\mathbf{c}_j$
3. Zmień położenie centrum $\mathbf{c}_j$ zwycięzcy

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

4. Powtarzaj dla kolejnych $\mathbf{x}$ aż do uzyskania zbieżności

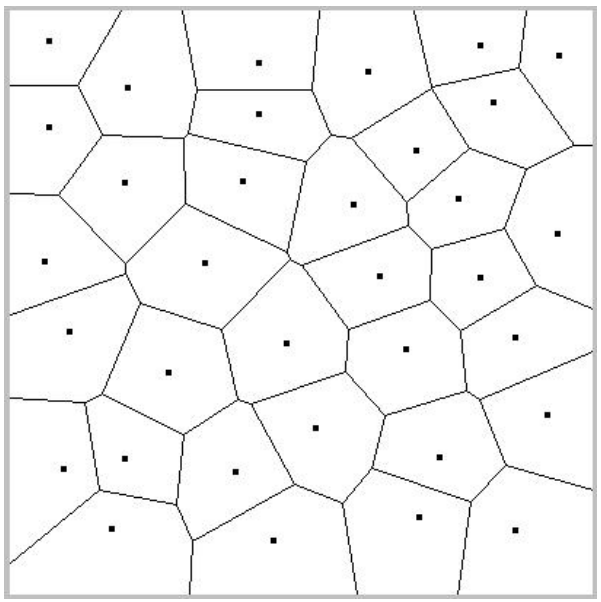
Stała uczenia η_k zanika w miarę wzrostu k, np:

$$\eta_k = \frac{\eta_0}{1 + \frac{k}{T}}$$

gdzie T - stała liczba epok

Diagram Voronoi

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

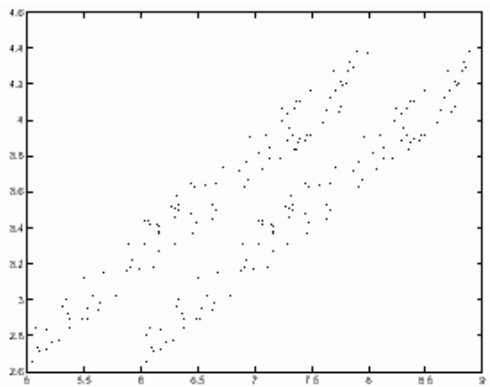


Uczenie - obroty i rozmycia

- Pełna macierz transformacji $\mathbf{Q}$ dla dużej liczby cech posiada za dużo parametrów
- Obroty funkcji zlokalizowanych są przydatne, więc wystarczy $Q_{ii} \neq 0$ i $Q_{ii+1} \neq 0$, która realizuje dowolne obroty
- Inna metoda: iloczyn funkcji radialnej z bicentralną

$$\varphi(\mathbf{x}) \cdot (\sigma(\mathbf{wx} + w_0) - \sigma(\mathbf{wx} + w_0))$$

- Uproszczenie: kąt obrotu ustalony po inicjaliacji położeń



Trening RBF: metoda probabilistyczna

Zał: rozkład równomierny danych w zbiorze treningowycm, diagonalne dyspersje $\mathbf{\Sigma}$

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

Centra oraz rozmycia optymalizawane są rownocześnie

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

$$\mathbf{\Sigma}_i(k+1) = \frac{\mathbf{\Sigma}_i(k) + \eta_k\left(\varphi_i(\mathbf{x})\|\mathbf{x} - \mathbf{c}_i(k)\|^2 - \mathbf{\Sigma}_i(k)\right)}{(1-\eta_k) + \eta_k\varphi_i(\mathbf{x})}$$

Współczynnik uczenia maleje w czasie $\eta_k = \frac{\eta_0}{k}$

Uczenie metodami gradientowymi

Minimalizacja funkcji błędu

$$E = \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 = \sum_{i=1}^{N} \left(y_i - \sum_{k=0}^{K} w_k \varphi_k(\|\mathbf{x}_i - \mathbf{c}_k\|) \right)^2$$

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} = -\eta (y - f(\mathbf{x})) \varphi(\|\mathbf{x} - \mathbf{c}_j\|)$$

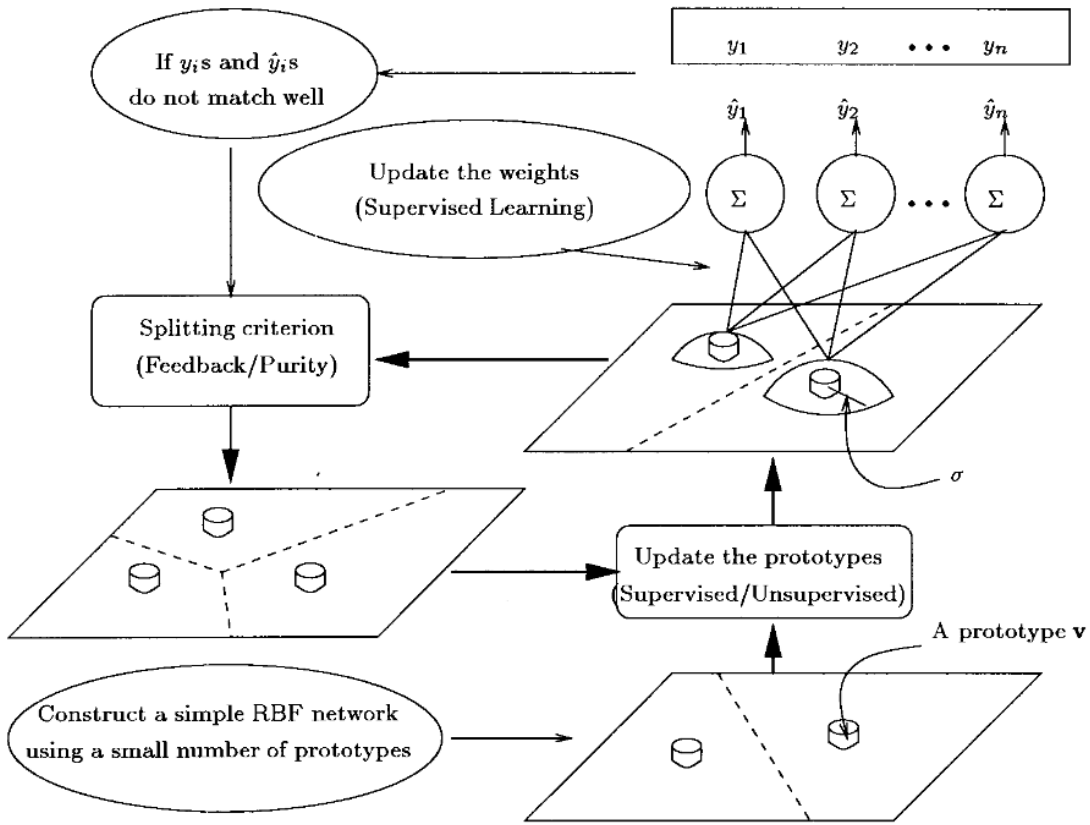
$$\Delta c_{ji} = -\eta \frac{\partial E}{\partial c_i} = -\eta (y - f(\mathbf{x})) w_j e^{-\frac{\|\mathbf{x} - \mathbf{x}_j\|^2}{2\sigma_j}} \frac{(x_i - c_{ji})}{\sigma_j^2}$$

$$\Delta \sigma_i = -\eta \frac{\partial E}{\partial \sigma_i} = \eta (y - f(\mathbf{x})) w_j e^{-\frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{2\sigma_j^2}} \frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{\sigma_j^3}$$

Konstruktywny RBF

- GAL (Growing and Learning)
- GrRBF (Growing Radial Basis Function)
- FEN (Function Estimation Networks)
- RAN (Resource Allocation Networks)
- Klasyfikatory Gaussowskie, sieci probabilistyczne ...

Growing RBF



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. Wstaw nową funkcję radialną w punkcie o największym błędzie

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

   gdzie $\mathbf{c}_m$ centrum odpowiadające największemu błędowi, $\mathbf{c}_n$ - najbliższe centrum względem $\mathbf{c}_m$
4. Powtarzaj 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 $t-1$ istnieje k neuronów, wówczas nowy neuron dodawnay jest z parametrami

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

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 $\|\mathbf{x} - \mathbf{x}\|$

SOM Self Orgnizing Map

Samoorganizacja i uczenie konkurencyjne

Uczenie nadzorowane i nienadzorowane

Uczenie nadzorowane:

- sygnał wejściowy $\mathbf{x}_i$ posiada oczekiwaną odpowiedź $\mathbf{y}$
- trening dąży do minimalizacji błędu pomiędzy odpowiedzią sieci a wartością oczekiwaną
- klasyfikacja, aproksymacja, ...

Uczenie nienadzorowane:

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

Mechanizmy samoorganizacji

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

Aspekty samoorganizacji

- trening dostosowuje wagi neuronów (a przez to ich aktywacje) do aktywności sygnałów uczących
- pewne grupy neuronów reagują na podobne wzorce
- większe sygnały -> większe wagi -> większe aktywacje -> większe różnice pomiędzy grupami neuronów reagujących na odmienne bodźce
- nadmiarowość danych (redundancja) - wielokrotne powtarzanie podobnych wzorców jest niezbędne w samoorganizacji

Samoorganizacja i uczenie nienadzorowane

Samoorganizujące się sieci neuronowe

- 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
- wektorowa kwantyzacja, kodowanie, kompresja - opisanie zbioru danych mniejsza liczbą wektorów kodujących (prototypów)
- mapy cech - organizacja danych w niskowymiarowych przestrzeniach (1D, 3D, 3D) przy zachowaniu porządku topologicznego

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 take most*) - łagodniejsza forma biorąca pod uwagę neurony o słabszym pobudzeniu, w pewnym otoczeniu $h(\mathbf{r}_i, \mathbf{r})$
- Zwykle zakładamy unormowanie wektorów wag $\mathbf{w}$ lub sygnału wejściowego $\mathbf{x}$

Neuron zwycięzca k o największym pobudzeniu

$$\mathbf{w}_k^T \mathbf{x} > \mathbf{w_j}^T \mathbf{x}, \qquad j = 1, \ldots, N$$

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

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

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

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

$$\min \|\mathbf{x} - \mathbf{w}\| = \max \mathbf{w}^T\mathbf{x}$$

stąd neuron zwycięzca

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

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 Manhatan L_1

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

- iloczyn skalarny (odległość kątowa)

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

- norma L_∞

$$d\left(x, w_j\right) = \max_k \left|x_k - w_{jk}\right|$$

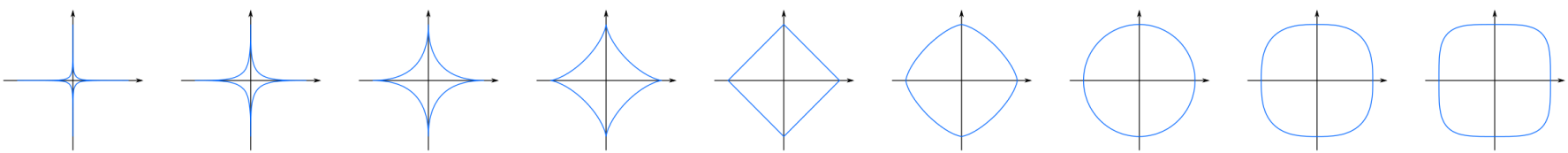
Odległość Minkowskiego

Uogólniona funkcja odległości

$$d(x, y) = \left(\sum_{k=1}^{K} (x_k - y_k)^p \right)^{\frac{1}{p}}$$

- $p = 1$ odległość Manhatan
- $p = 2$ odległość euklidesowa

Jednostkowy okręg w przestrzeni R^2



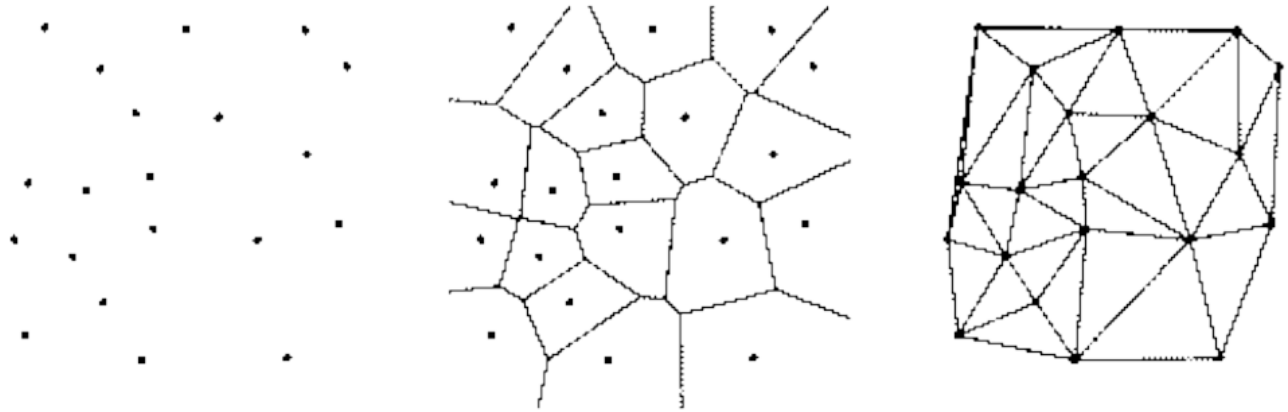
Rys: wikipedia.org

Voronoi i Delaunay

Punkty danych

granice decyzji Voronoia

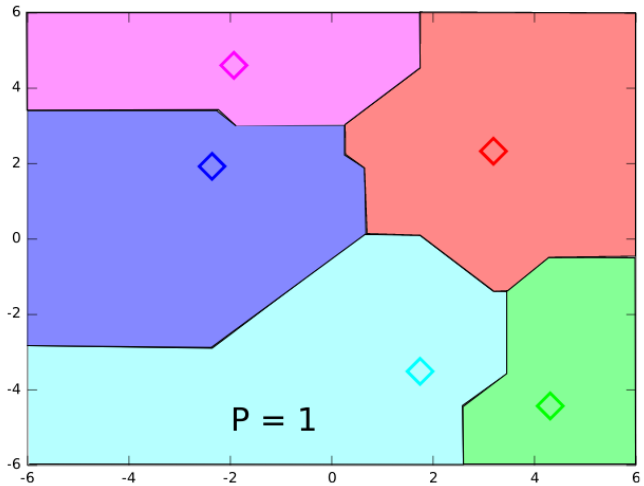
Triangulacja Delauny

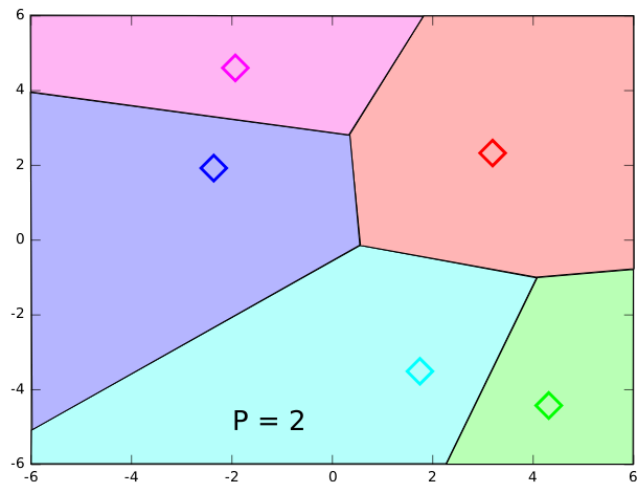


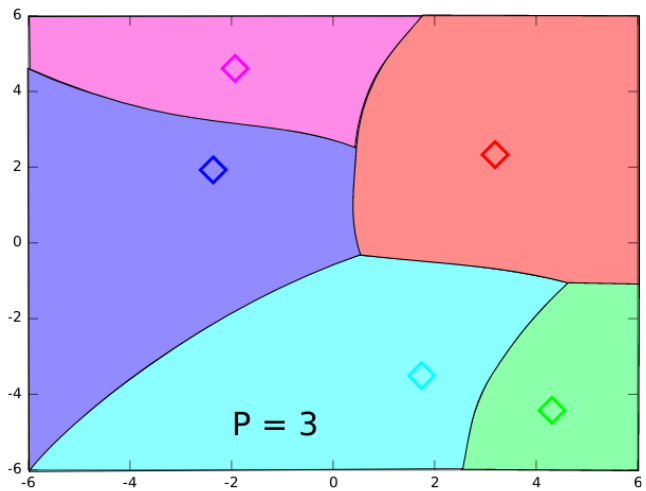
Neurony dzielą przestrzeń cech na rozdzielne obszary (obszary Voronoi), w danym obszarze tylko jeden neuron zwycięża konkurencję

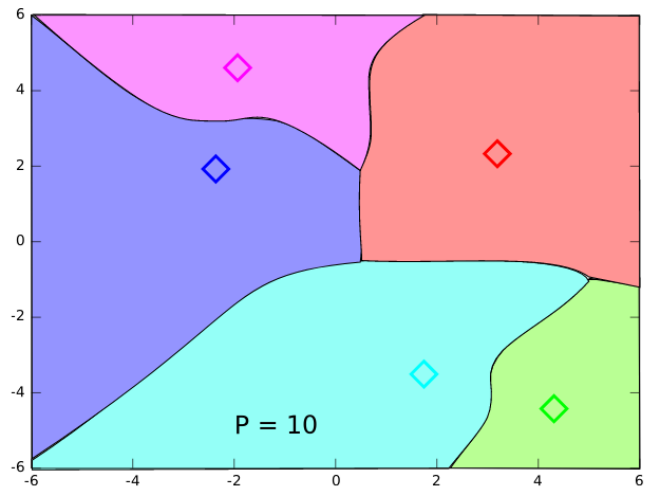
- Zbiór Voronoia - zbiór wektorów wewnątrz obszaru Voronoia.
- Łącząc neurony, których obszary Voronoia mają wspólną krawędź otrzymujemy traingulację Delaunaya.

Strefy wpływów





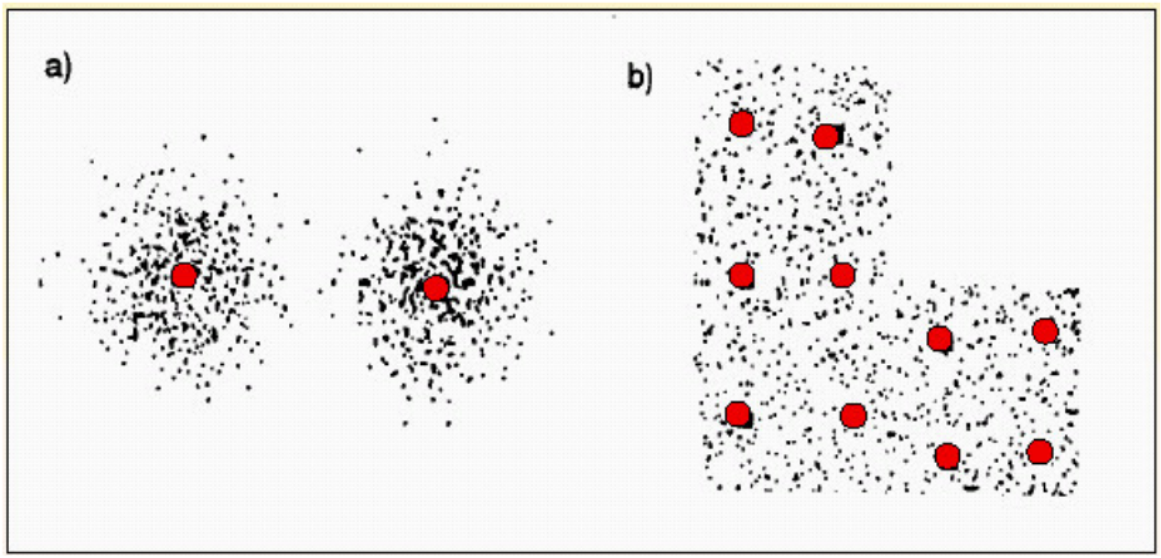




Rys: M. Czoków, J. Piersa, Wstęp do sieci neuronowych

Kwantowanie wektorowe

- **Kwantowanie wektorowe** - zakodowanie danych za pomocą wektorów prototypowych. Pojedynczy prototyp reprezentuje grupę wektorów wejściowych
- Księga kodów - zbiór prototypów kodujących (ich numery)
- Kompresja danych - liczba prototypów jest mniejsza od liczby wzorców



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)

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

4. Powtarzaj 2 i 3 aż do uzyskania zbieżności

Błąd kwantowania wektorowego dla k neuronów

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

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

Normalizacja wektorów

- Proces samoorganizacji prowadzi do spójnego podziału przestrzeni cech, gdy wektor **w** jest znormalizowany (po każdej adaptacji poddajemy go normalizacji)
- Gdy wektory **x** są znoralizowane, wóeczas $\mathbf{w}_i$ również w czasie treningu będą stawały sie automatycznie znormalizowane
- normalizacja

$$\hat{x}_i = \frac{x_i}{\sqrt{\sum_{i=1}^{d} x_i^2}}$$

- lub poprzez dodanie dodatkowego wejścia x_0 spełniającego

$$\sum_{i=0}^{d} x_i = 1$$

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)
- uczenie samoorganizujące dowolna metoda w celu minimalizacji błędu kwantyzacji
- podobne ramki pobudzają ten sam neuron zwycięski
- prezentacja wszystkich ramek $\mathbf{x}_i$ pozwala ustalić kolejność neuronów zwycięskich $1, 5, 1, 30, \ldots$, ich numery tworzą księgę kodów
- współczynnik kompresji

$$k_r = \frac{N_r n_x n_y T}{N_r \log_2 n + n n_x n_y t}$$

gdzie N_r liczba ramek, n - liczba neuronów, T liczba bitów potrzebna do reprezentacji wartości x_i, t liczba bitów potrzebna do reprezentacji wagi w_i

oryginał

VQ 1024 codebook

☞ Image Compression with Vector Quantization by Ivan-Assen Ivanov

Mapy ekwiprobabilistyczne

- Zasada jednakowego zniekształcenia: każdy neuron powinien mieć podobny wkład do końcowego błędu kwantyzacji (klasyfikacji).
- przecenia rejony mało prawdopodobnych danych i niedojenia rejony o dużym prawdopodobieństwie
- przy losowej minimalizacji część neuronów może znaleźć się w obszarach gdzie nie będzie pobudzeń (martwe neurony)
- powstają neurony bezużyteczne; jak wykorzystać całą sieć?
- mechanizm zmęczenia - neurony z sumieniem (Conscience Learning) uwzględnia częstość wygrywania poszczególnych neuronów pozwalając innym wygrywać konkurencję
- odpowiada chwilowemu zmęczeniu neuronu po wzbudzeniu w sieciach neuronowych biologicznych

Mechanizm zmęczenia z potencjałem

Potencjał p_i neuronu zmieniany

$$p_i(t+1) = \begin{cases} p_i(t) + \frac{1}{n} & \text{dla neuronu zwycięzcy} \\ p_i(t) - p_{min} & \text{dla pozostałych} \end{cases}$$

Neurony dla których $p_i(t) < p_{min}$ nie biorą udziału we współzawodnictwie

$$d(\mathbf{x}, \mathbf{w}_c) = \mathbf{min}\{d(\mathbf{x}, \mathbf{w}_i)\} \quad \text{dla} \quad p_i \geq p_{min}$$

- dla $p_{min} = 0$ nie występuje zmęczenie (standardowy algorytm Kohonena)
- dla $p_{min} = 1$ tylko jeden neuron w danej chwili uczestniczy w uczeniu a reszta odpoczywa
- potencjał p_i ograniczony do 1
- dobre wyniki dla $p_{min} \approx 0.75$

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 odległości
- po kilku początkowych cyklach, wszystkie neurony miały już szansę wygrać, mechanizm się

Gaz neuronowy (Schulten i Martinez 1991)

1. Losowa inicjalizacja N neuronów
2. W kolejnych krokach $t = 1, 2, 3, \ldots, T$ wybierz przypadkowy wektor $\mathbf{x}$
3. Zrób ranking neuronów posortowanych względem odległości od $\mathbf{x}$

$$\|\mathbf{w}_i - \mathbf{x}\| \leq \left\|\mathbf{w}_j - \mathbf{x}\right\| ; i < j$$

4. Adaptacja

$$\Delta \mathbf{w}_i = \eta_i(t) h_i(t) (\mathbf{v} - \mathbf{w}_i)$$

5. Zmniejsz eksponencjalnie obszar λ i stałą uczenia $\eta(t)$

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

gdzie $m(i)$ to miejsce neuronu w rankingu (dla zwycięzcy $m(i) = 0$)

Promień sąsiedztwa

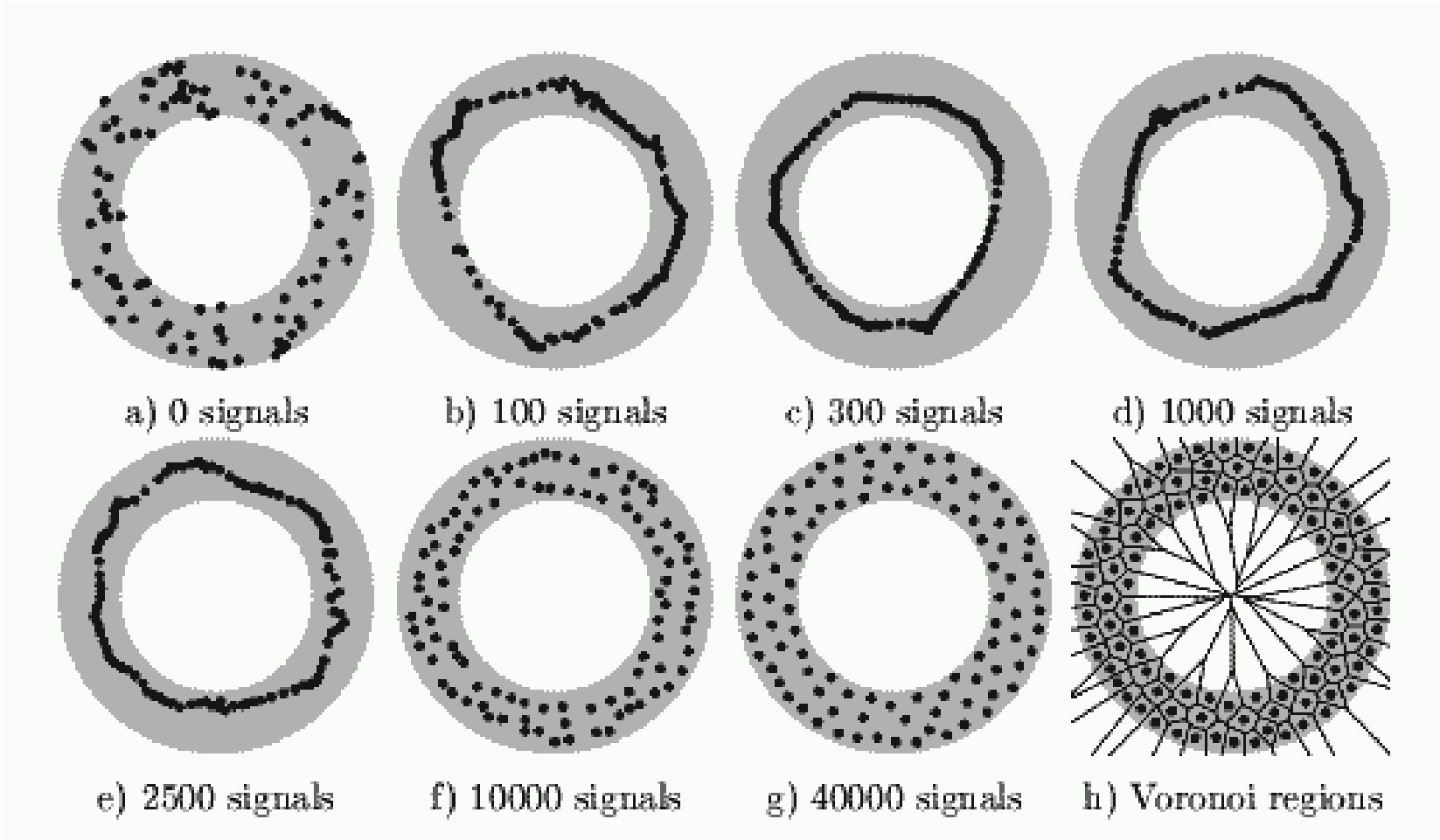
Promień sąsiedztwa maleje w trakcie uczenia od λ_0 do pewnej wartości λ_{min}

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

- dla $\lambda = 0$ uczenie WTA
- dla $\lambda > 0$ aktualizowane są wagi wszystkich neuronów z siłą zależna od pozycji w rankingu

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

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

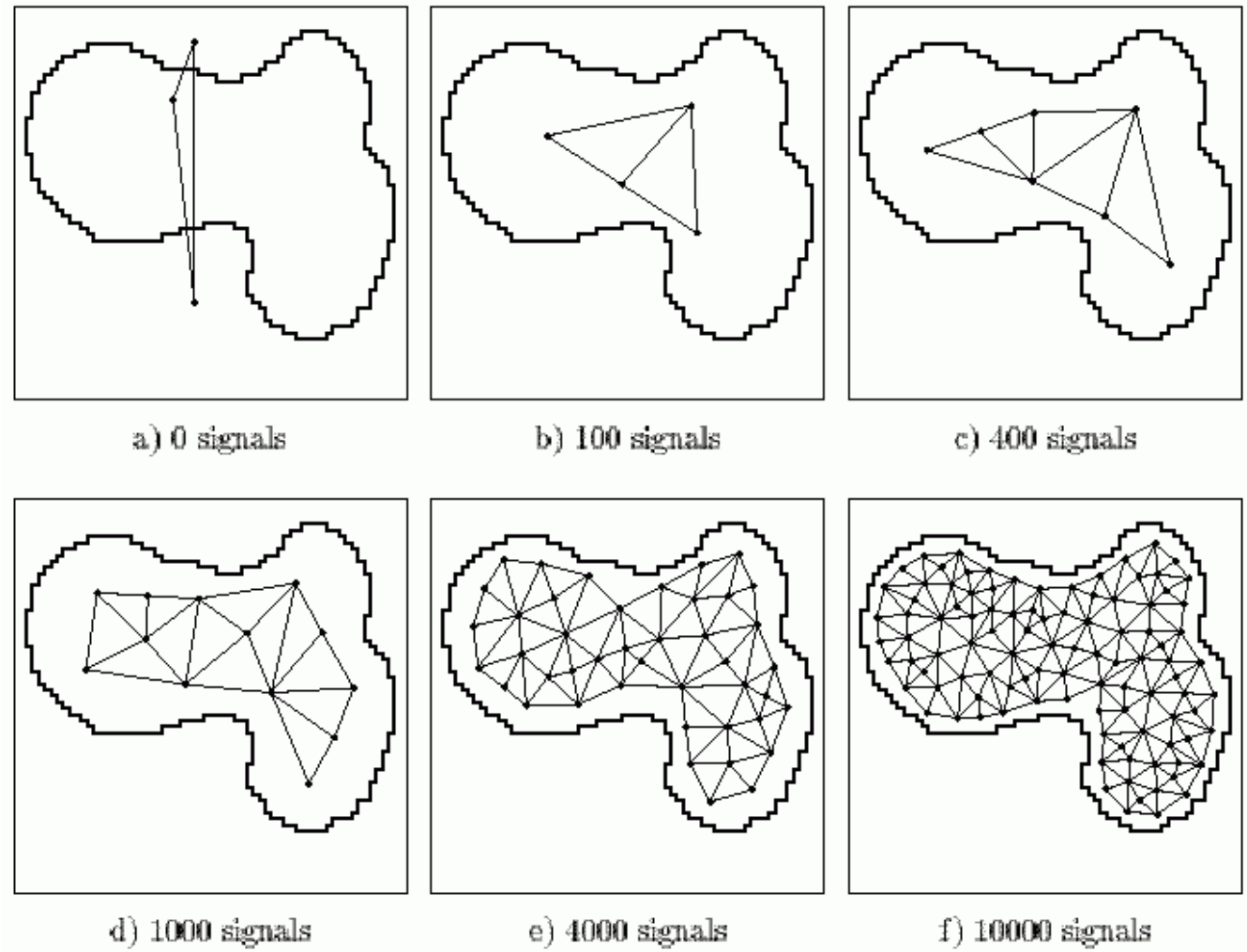

a) 0 signals
b) 100 signals
c) 300 signals
d) 1000 signals
e) 2500 signals
f) 10000 signals
g) 40000 signals
h) Voronoi regions

Growing Cell Structures

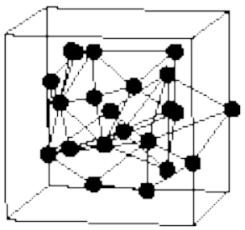
Konstruktywny SOM, zbliżony do Growning Neural Gas (Fritzkie)
Początkowa topologia: k-wymiarowy sympleks (k=1, 2, 3)
Dodaje się nowe neurony i usuwa stare.
Algorytm SOM, ale bez zmniejszania sąsiedztwa i adaptacji dokonuje się tylko dla zwycięzcy i bezpośrednich sąsiadów.

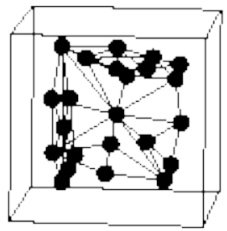
1. Dla danego $\mathbf{x}$ znajdź zwycięzcę c
2. Popraw wagi zwycięzcy oraz sąsiadów: $\Delta \mathbf{w}_i = \eta_s(\mathbf{x} - \mathbf{w}_i)$
3. Zwiększ licznik częstości τ_c o 1, zmniejsz wszystkie pozostałe o $-\alpha\tau_c$
4. Policz zrenormalizowane częstości $f_i = \frac{\tau_i}{\sum_j \tau_j}$
5. Po ustalonej liczbie epok T znajdź neuron o największej częstości f_i i wstaw pomiędzy ten neuron i najdalszego sąsiada nowy neuron tworząc lokalny sympleks; nowy wektor dla tego neuronu weź z interpolacji użytej pary.

Trening GCS

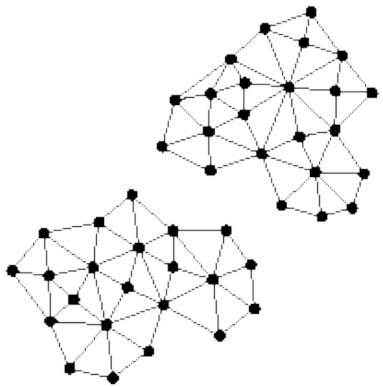

a) 0 signals
b) 100 signals
c) 400 signals
d) 1000 signals
e) 4000 signals
f) 10000 signals

GCS 2 obszary



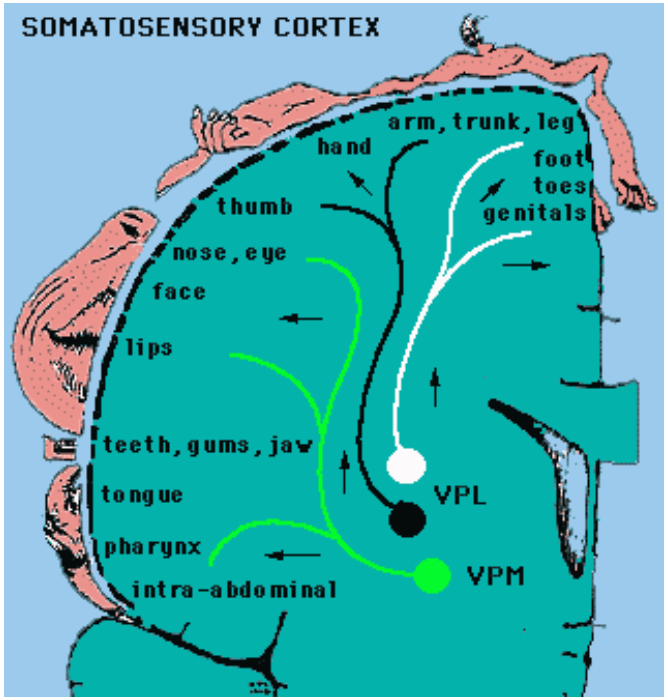


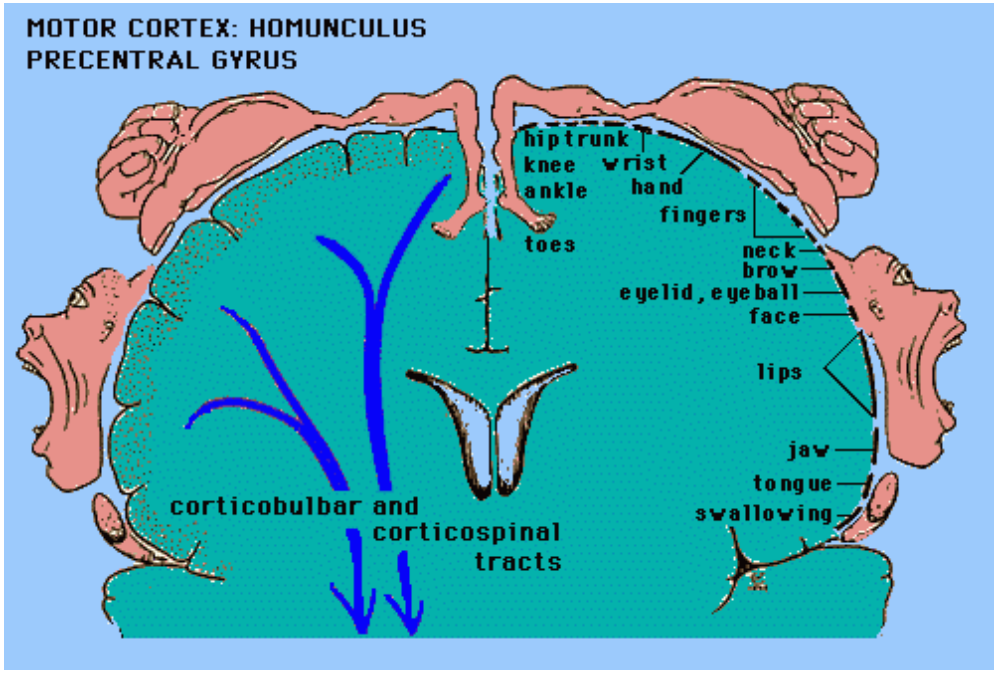
Sytuacja w 3-wym. przestrzeni danych - 2 oddzielone skupienia.



Sieć GCS rosnąca w dwóch wymiarach - odpowiednia topologia.

Mapy czuciowe i motoryczne





Inspiracje biologiczne SOM

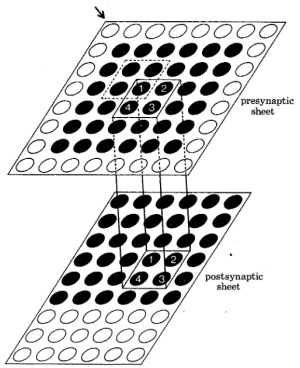
- przetwarzanie różnych danych zmysłowych i motorycznych (wzrok, słuch, dotyk, ...) odbywa się w różnych obszarach kory mózgowej
- neurony o podobnych funkcjach są obok siebie => mapy topograficzne
- istnieje topologiczna zależność w obliczeniach realizowanych przez mózg
- Przykłady
  - mapy somatosensoryczne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

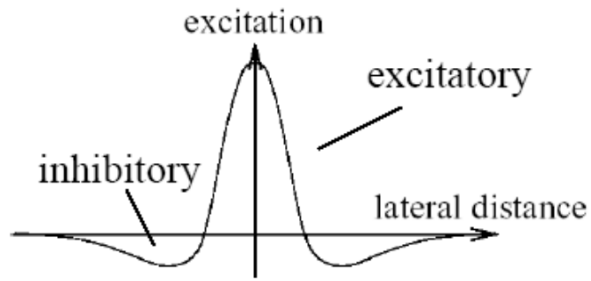
Modele samoorganiacji

- 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,

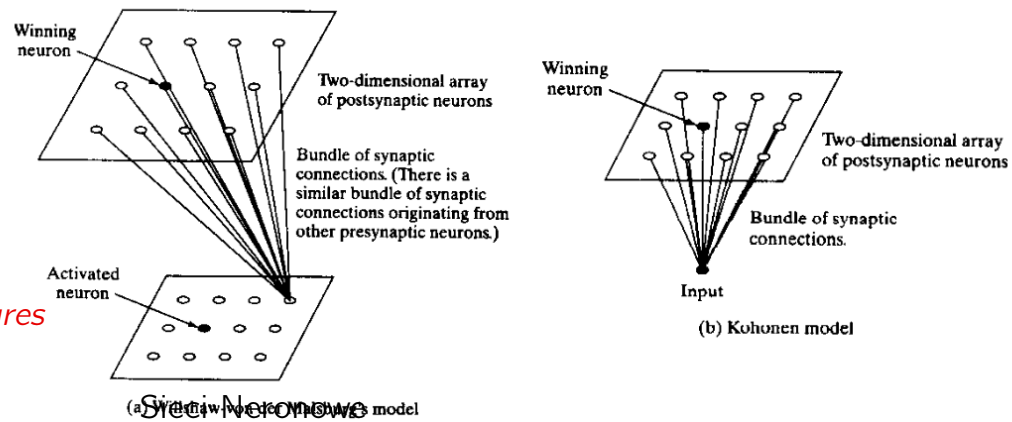




- aktywacja typu „Meksykańskiego kapelusza”
  - kooperacja - pobudzenia bliskiego zasięgu
  - konkurencja - hamowanie w dalszych regionach
- siatkówka (wejście) posiada narzuconą organizację topologiczną
- wybór zwycięskiego neuronu w wynikiem dynamiki neuronalnej

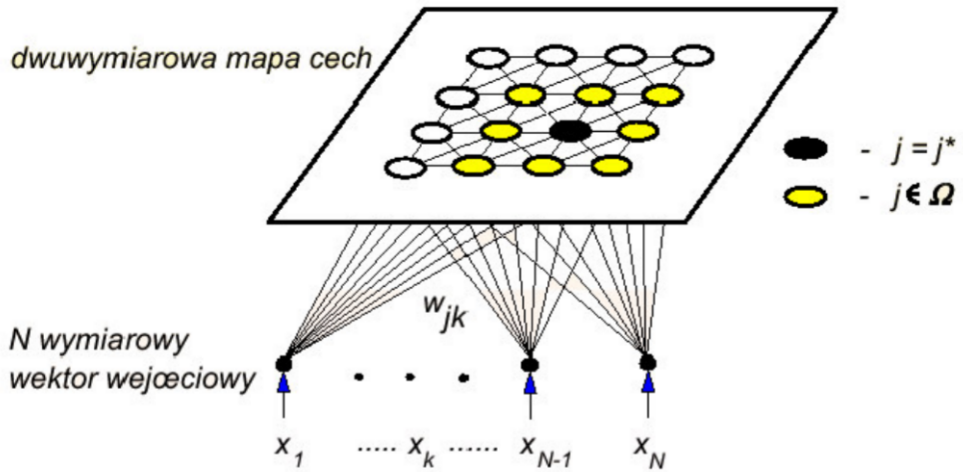
Sieś Kohonena

- Kohonen (1981) – najbardziej popularne uproszczenie, stąd SOM często utożsamiane z „siecią Kohonena”
- dwie fazy
  - 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 narzucoenej struktury topologicznej, dowolna wymiarowość wygnału 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 neurony powinny reagować na podobne wzorce

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)

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

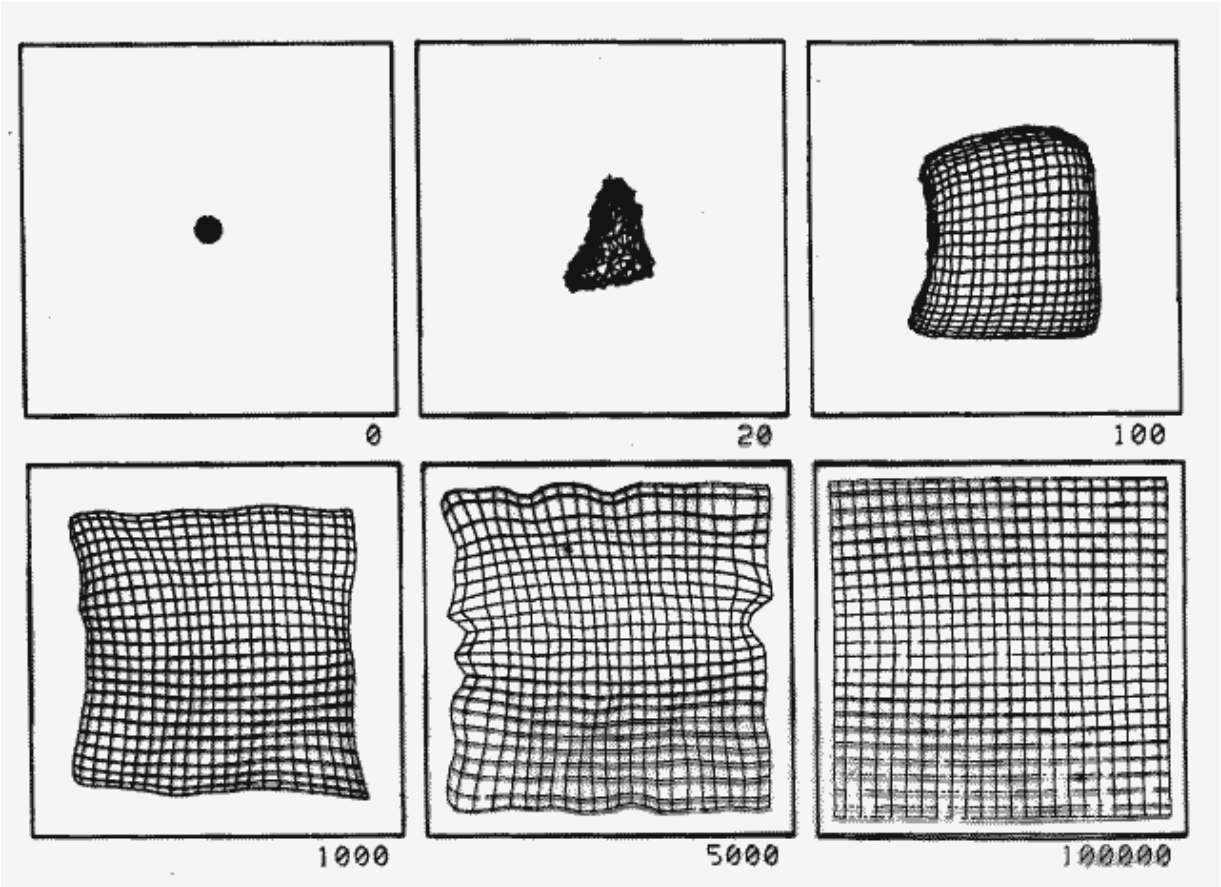
3. Przesuń wagi neuronu c i innych neuronów w sąsiedztwie $O(c)$ w stronę wektora $\mathbf{x}$

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

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

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

Przykład: 2D -> 2D

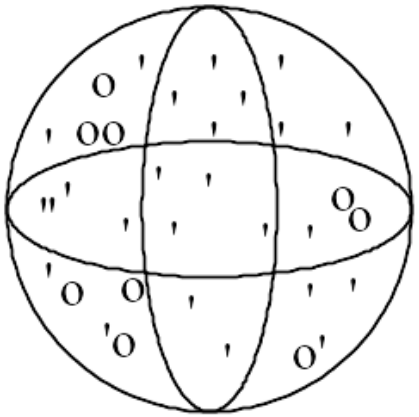


Rozkład jednostajny w kwadracie. SOM uczy się jednorodnego rozkładu. Początkowo wszystkie $\mathbf{w} \approx 0$

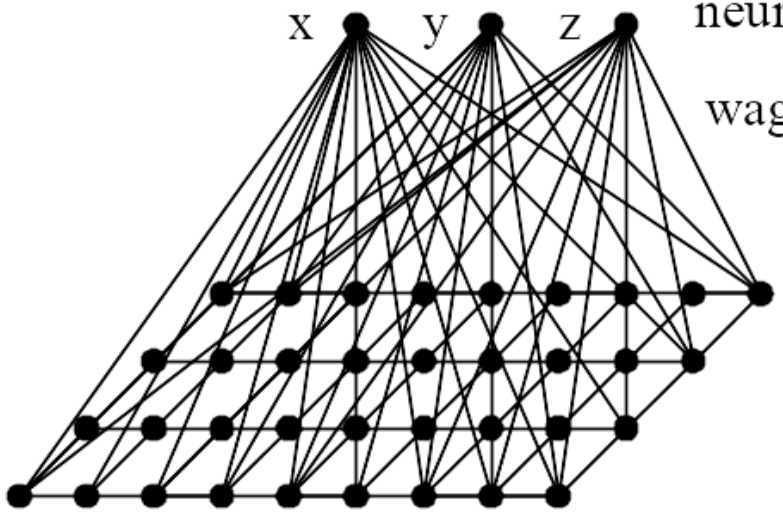
Sieć 2D, cecy 3D

o=dane

' = wagi sieci



przestrzeń cech

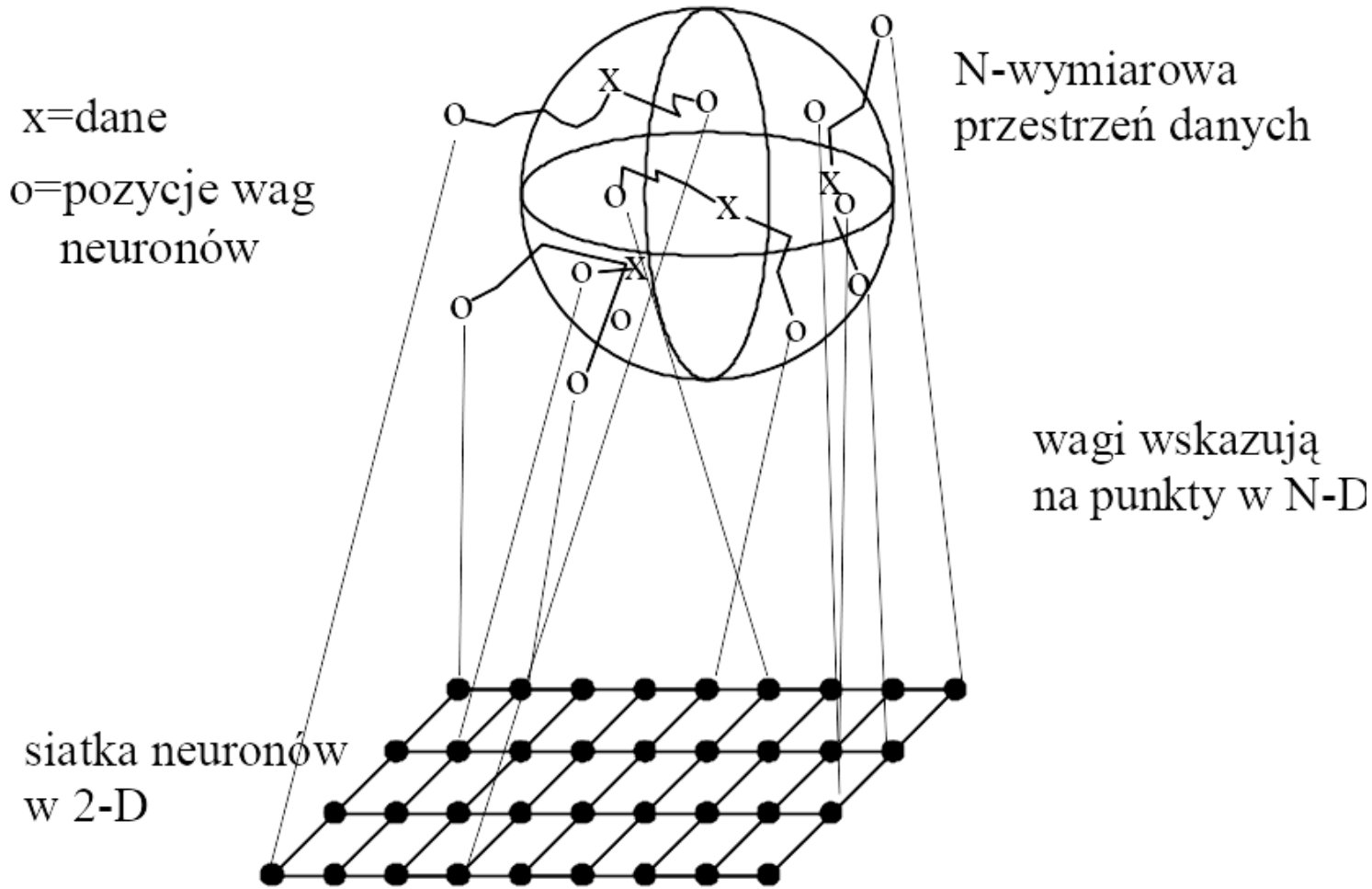


neurony wejściowe

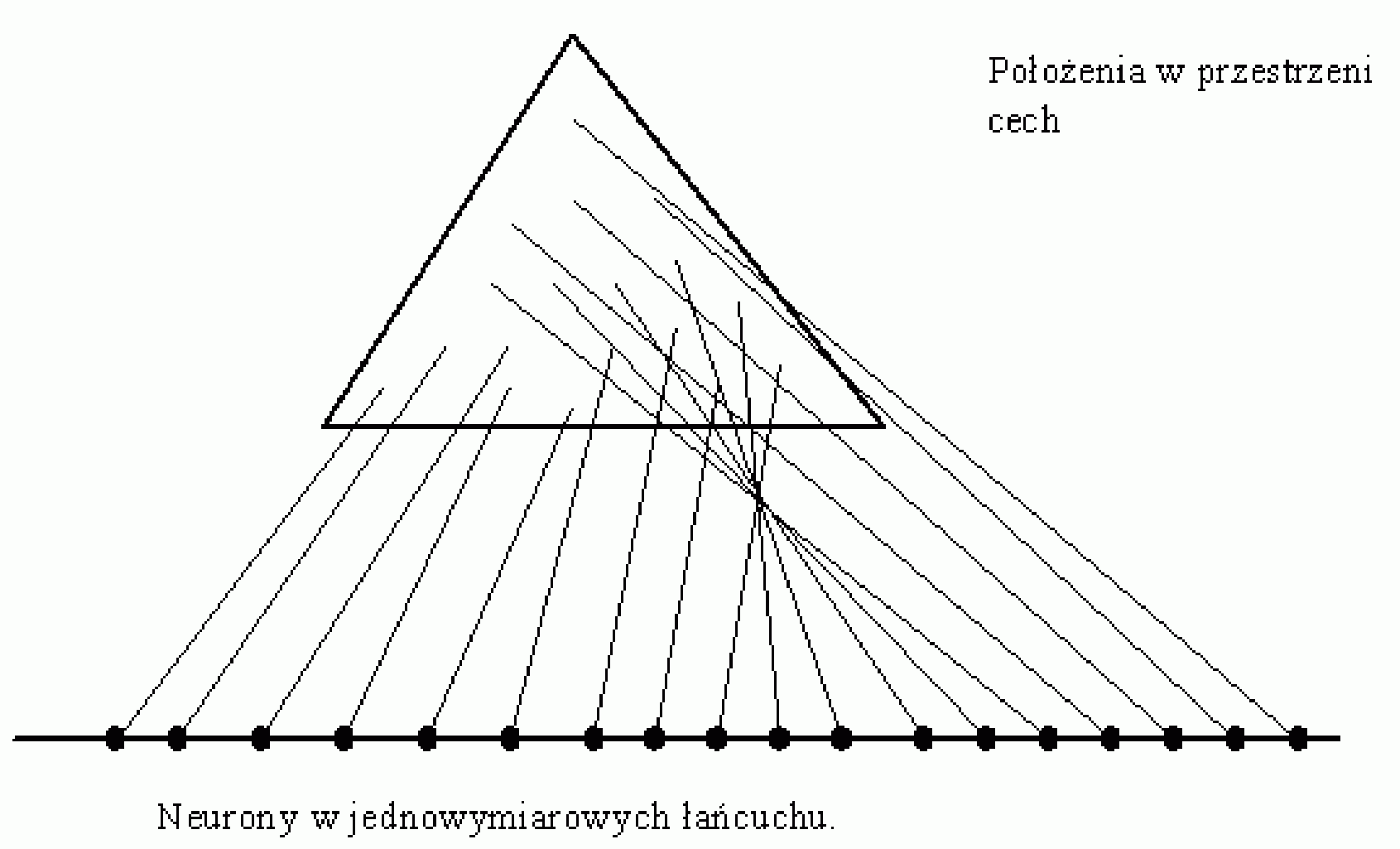
wagi przypisane połączeniom

2-D siatka neuronów

Uczenie sieci 2D

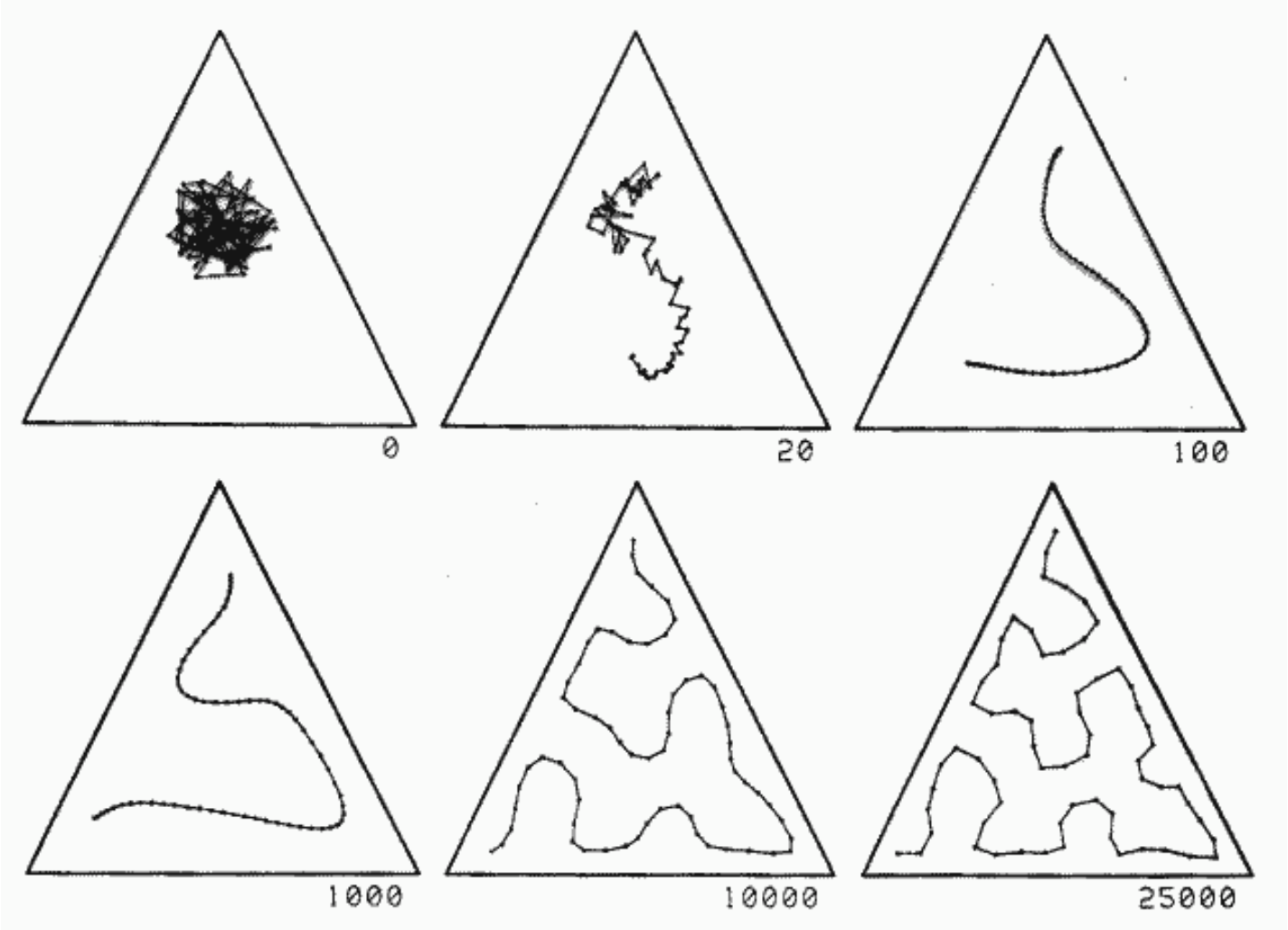


Sieć 2D to 1D

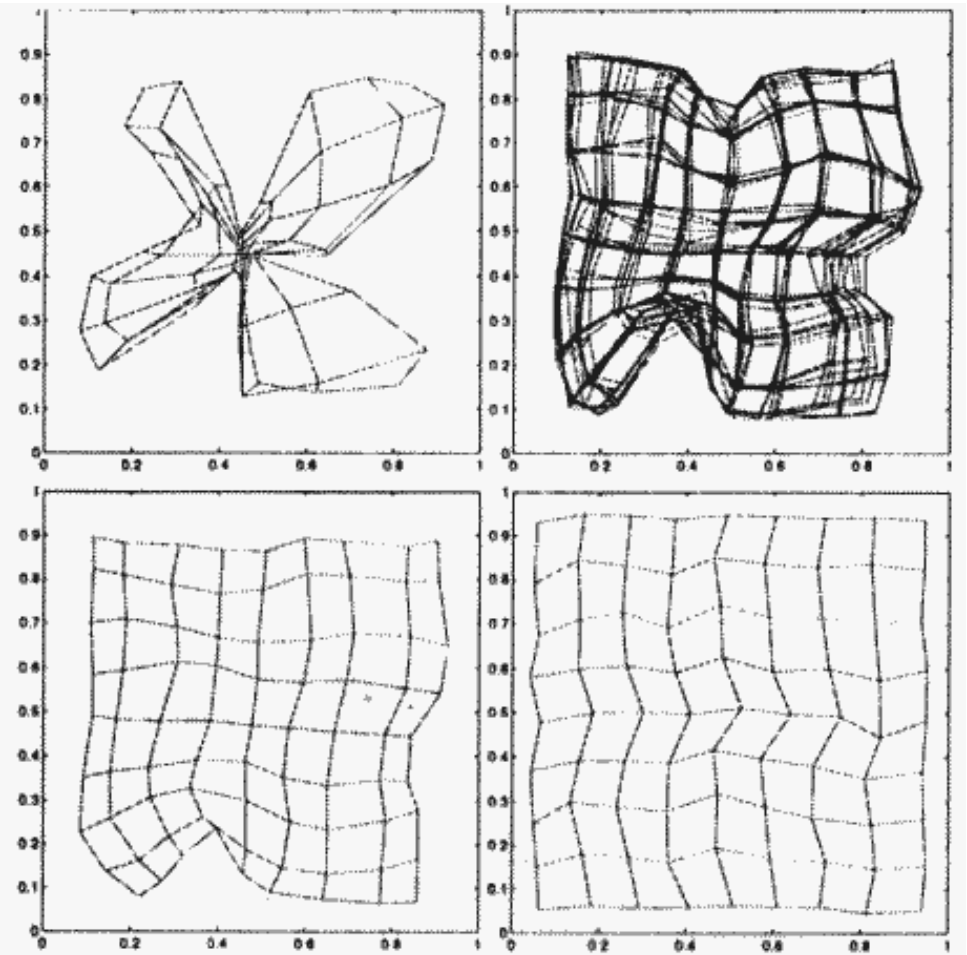

Położenia w przestrzeni cech
Neurony w jednowymiarowych łańcuchu.

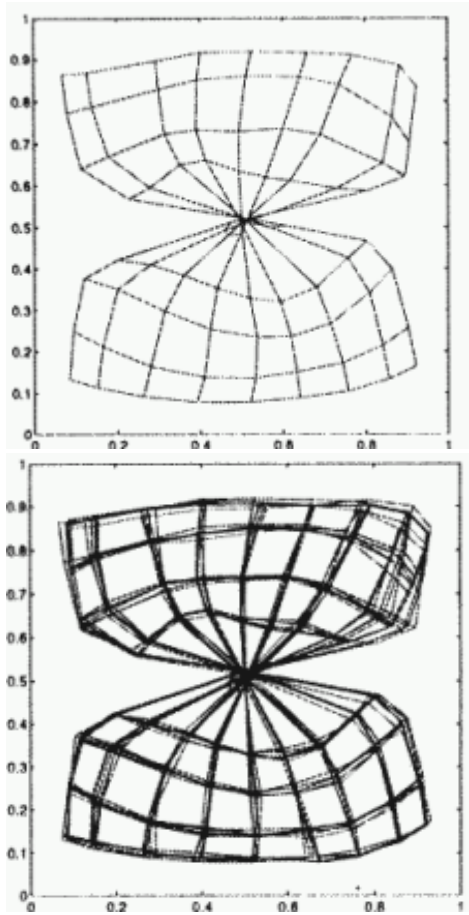
Przykład: fraktalne krzywe Peano

Krzywa Peano (lub Hilberta) - krzywa wypełniająca płaszczyznę



Zniekształcenia





Początkowe zniekształcenia mogą zniknąć lub pozostać.

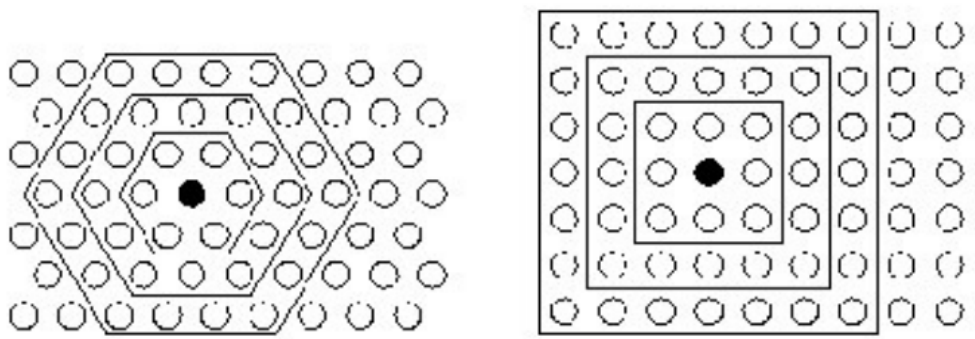
Rozmiar sąsiedztwa

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 $\sigma(t)$ maleje w czasie treningu, może by 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

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

zasięg funkcji maleje w czasie treningu

$$\sigma(t) = \sigma_0 \mathrm{e}^{-2\sigma_0 t / t_{\mathsf{max}}}$$

Dla $\sigma = 0$ funkcja sąsiedztwa obejmuje tylko wektor 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 neuronu i jest sumą po wszystkich neuronach:

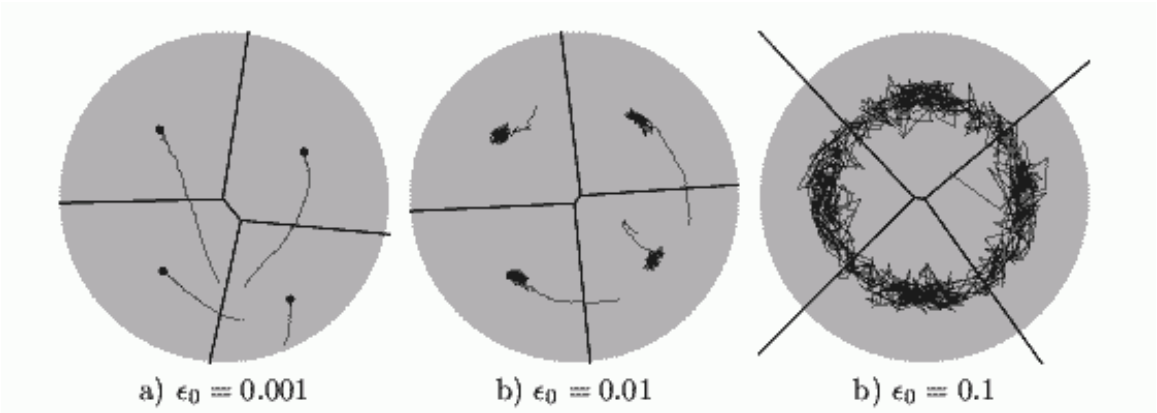
$$E_i(\mathbf{x}; \mathbf{w}, t) = \frac{1}{2} \sum_j h\left(\left|\mathbf{r}_i - \mathbf{r}_j\right|, t\right) \left\|\mathbf{x}(t) - \mathbf{w}_j(t)\right\|^2$$

Neuron-zwycięzca ma najmniejszy błąd lokalny:

$$c = \arg\min_i \sum_j h\left(\left|\mathbf{r}_i - \mathbf{r}_j\right|, t\right) \left\|\mathbf{x}(t) - \mathbf{w}_j(t)\right\|^2$$

Stała uczenia

Duża stała uczenia prowadzi do eksploracji znacznej części przestrzeni.



Stała uczenia maleje w trakcie uczenia

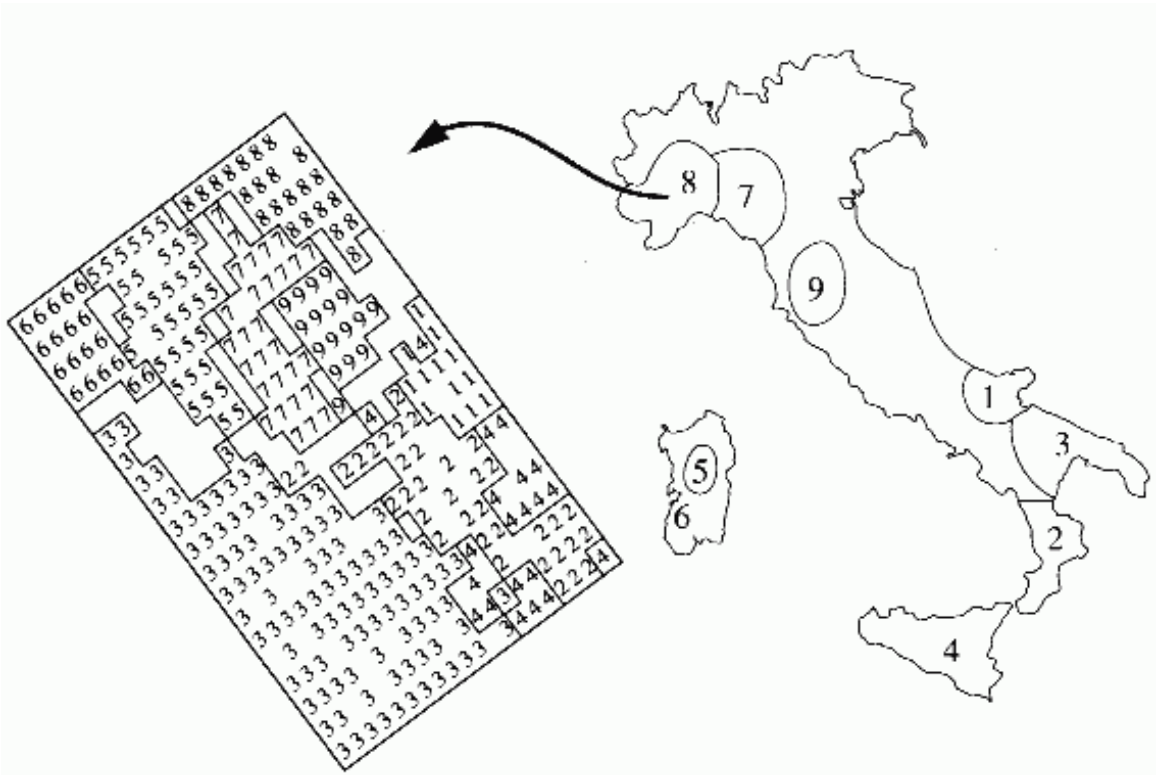
$$\eta(t) = \eta_0 e^{-\frac{t}{t_{max}}}$$

Własności SOM

- Powolna zbieżność algorytmu SOM, wymaga wielu iteracji
- Trudno coś udowodnić o zbieżności lub punktach stacjonarnych. Wyniki analityczne znane są tylko w 1D dla ciągłego czasu: wtedy wartości wag wzdłuż prostej porządkują się.
- W oryginalnym SOM nie ma funkcji błędu, nie ma więc gradientu!
- Sąsiednie neurony kodują sąsiednie obszary, ale sąsiednie obszary mogą być kodowane przez odległe neurony, relacje odległości nie zawsze są zachowane
- „Skręcone" konfiguracje przy zbyt szybkiej redukcji sąsiedztwa
- Jakość klasyfikacji: zwykle niska. Kohonen: SOM służy głównie do wizualizacji ... ale wizualizacja też kiepska, bo brak oceny pozwalającej na redukcję wprowadzanych zniekształceń.

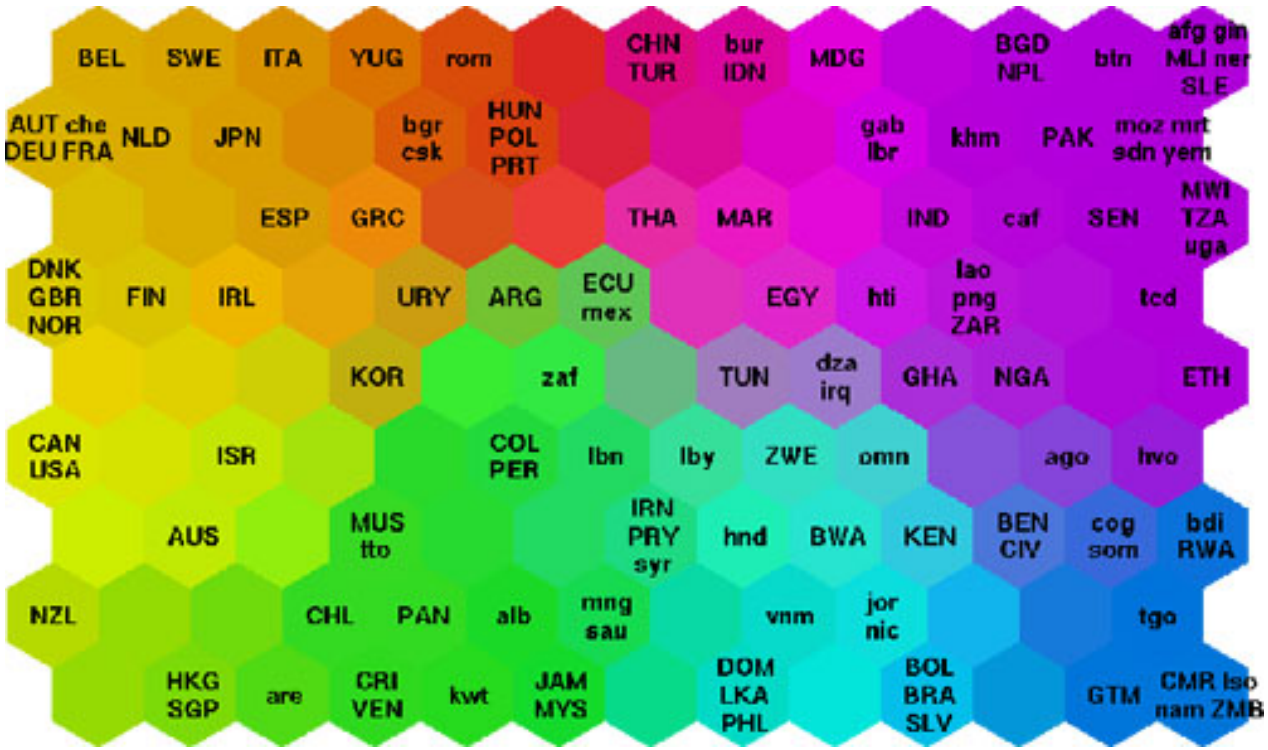
Przykład: Włoska oliwa

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



Przykład: Wodr powerty map

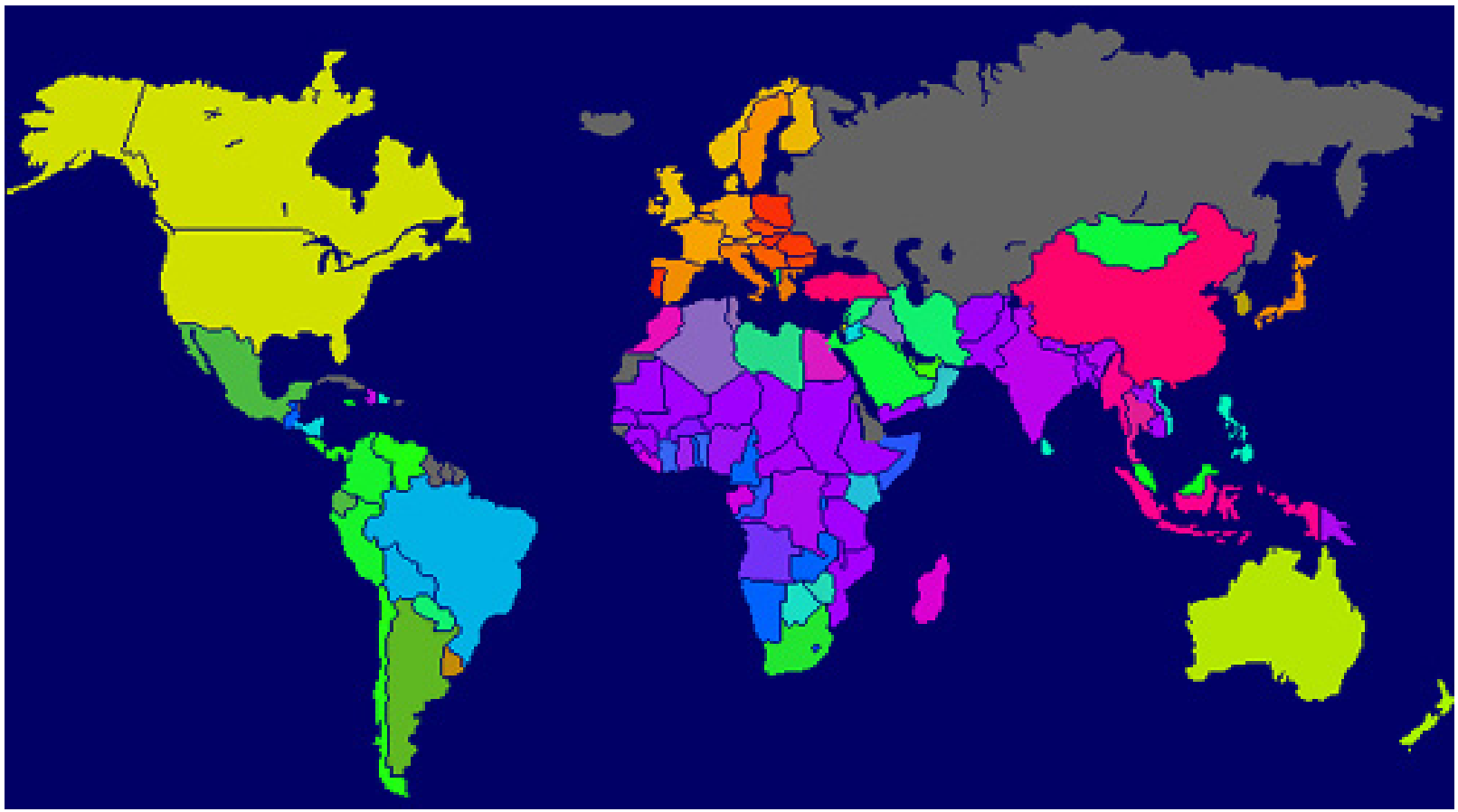
Dane: WorldBank data 1992 rok, 39 współczynników opisujących jakość życia w kraju (poziom służby zdrowia, edukacji, etc.)



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

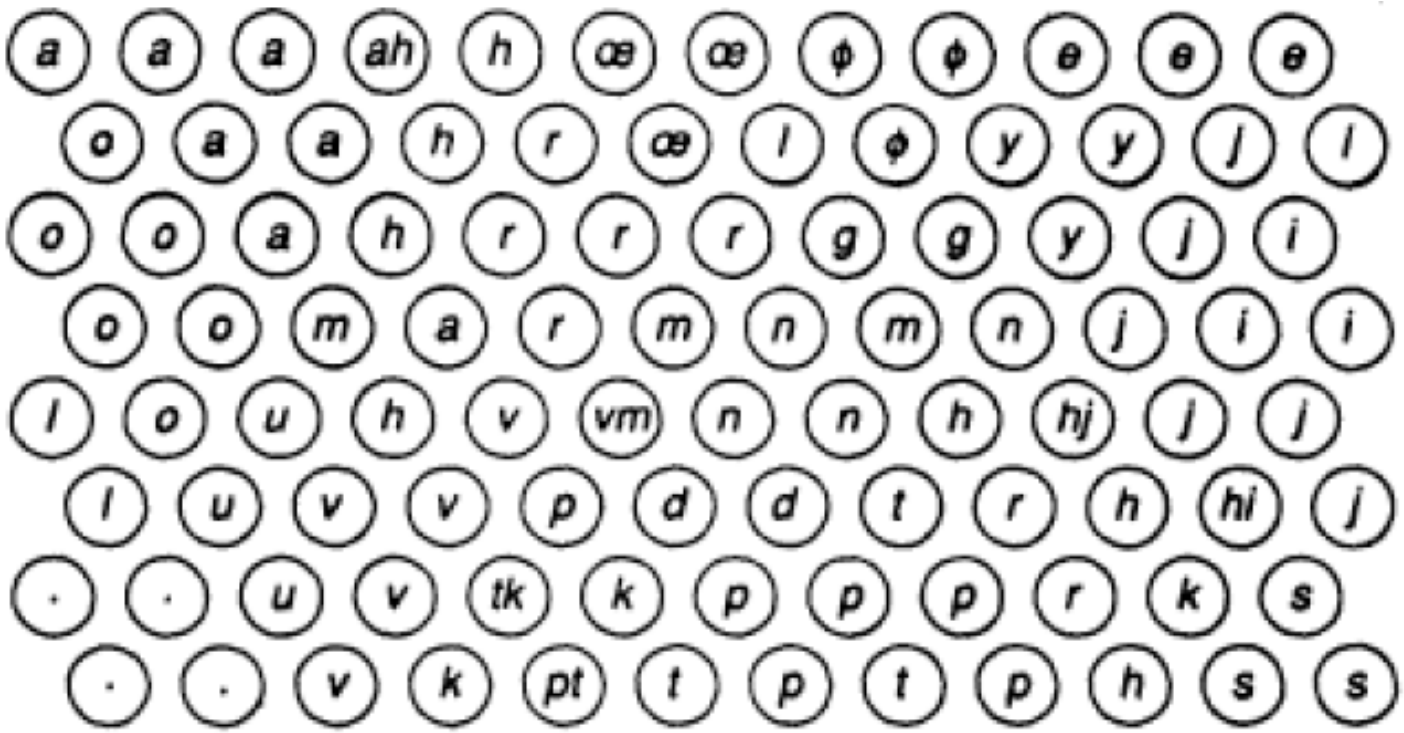
Przykład: Wodr powerty map

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 róznych obszarach kory (motoryzna, sensoryczna, wizyjna)
- Robotyka i AI: analiza danych z czujników, sterowanie ruchami robota, mapy orientacji przestrzennej
- Kategoryzacja dokumentów
- Analiza skupień genów, białek, cząteczek chemicznych, fonemów, dzwięków zwierząt, obiektów asrtonomicznych, dane ekonomiczne i finansowe
- Kompresja danych (obraz, dzwięk), filtrowanie informacji
- Medical and technical diagnostics.

- Analiza języka naturalnego: linguistic analysis, parsing, learning languages, hyphenation patterns
- Problemy optymaliacyjne: 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.

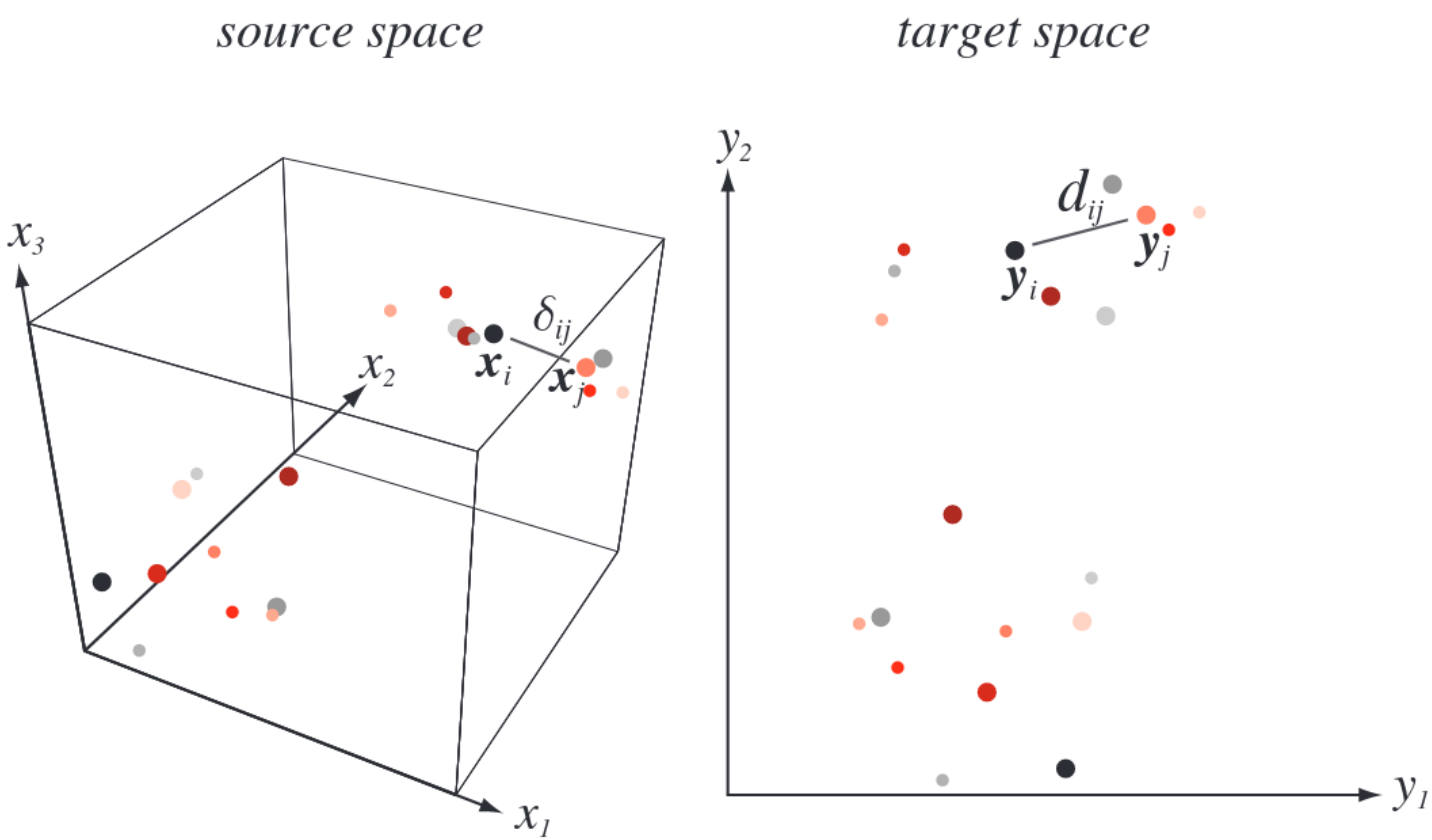
SOM vs MDS

Problemy SOM

- SOM bardzo popularne, prosta metoda wizualizacji
- nie gwarantuje wiarygodnej aproksymacji gęstości danych
- błąd kwantyzacji przydatny przy ocenie jakości odwzorowania ale nie mówi nic o jakości wizualizacji
- MDS, Multi-Dimensional Scaling (Thorton 1954, Kruskal 1964), zwane także mapowaniem Sammona (Sammon 1964)
  - redukcja wymiarowości próbą zachowania odległości
  - daje miarę zniekształceń topograficznych (minimalizacja funkcji stresu)
  - zazwyczaj redukcja do 2D, 3D w celu wizualizacji danych

Idea MDS

Szukamy mapowania $\mathbf{x} \to \mathbf{y} = M(\mathbf{x})$ minimalizującego pewną funkcję kosztu opartą o odległości δ_{ij} oraz d_{ij}



Rys: Duda, Hart, Pattern Classification

Algorytm MDS

Przestrzeń cech $\mathbf{x}_i \in \mathbb{R}^d$, $i = 1, 2, \ldots, n$
Odległość w przestrzeni cech

$$\delta_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$$

Przestrzeń docelowa $\mathbf{y}_i \in \mathbb{R}^K$, gdzie $K < N$
Odległość w przestrzeni docelowej

$$d_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|$$

Można użyć dowolnej metryki - najczęściej metryka Euklidesowa
W przypadki $K = 2$

$$E = \sum_{i<j}^{n} (\delta_{ij} - d_{ij})^2 = \sum_{i<j}^{n} \left(\delta_{ij} - \sqrt{(y_{i1} - y_{j1})^2 + (y_{i2} - y_{j2})^2} \right)^2$$

Funkcja E zależy od $2n$ parametrów $\mathbf{y}_i$
Jednak 3 współrzędne są redundantne: $\mathbf{y}_1 = (0, 0)^T$ wybór środka układu wsp., $\mathbf{y}_2 = (0, y_{22})$ wybór orientacji układu wsp.

Miary zniekształceń

$$E(\mathbf{r};\mathbf{w}) = \sum_{i<j}^{n} w_{ij}\left(\delta_{ij} - d_{ij}\right)^2 \geq 0$$

wagi w_{ij} mogą zależeć od odległości, np. maleć wykładniczo ze wzrostem δ_{ij}

Czynnik normalizacyjny

$$E(\mathbf{r}) = \frac{\sum_{i<j}^{n}\left(\delta_{ij} - d_{ij}\right)^2}{\sum_{i<j}^{n}\delta_{ij}^2 + \sum_{i<j}^{n} d_{ij}^2} \in [0,1]$$

wynosi 0 gdy idealne dopasowanie, 1 np. gdy $d_{ij} = 0$

Miary MDS

- Stres (Kruskal) błąd absolutny, mogą dominować duże odległości, zachowuje ogólną strukturę klastrów

$$E_1 = \sum_{i<j}^{n} (\delta_{ij} - d_{ij})^2 \geq 0$$

- Odwzorowanie Sammona, wpływ dużych odległości δ_{ij} jest zredukowany

$$E_2 = \frac{1}{c} \sum_{i<j}^{n} \frac{(\delta_{ij} - d_{ij})^2}{\delta_{ij}} \geq 0$$

gdzie

$$c = \frac{1}{\sum_{i<j} \delta_{ij}}$$

Miary MDS

- Błąd względny, wszystkie skale odległości traktowane są jednakowo

$$E_3 = \sum_{i=1}^{n} (1 - d_{ij}/\delta_{ij})^2 \geq 0$$

- Współczynnik alienacji (Guttman-Lingoes) – podobny do błędu względnego, trudniejszy do minimalizacji

$$E_4 = \sum_{i<j}^{n} (1 - \delta_{ij}/d_{ij})^2 \geq 0$$

Optymalizacja stresu

Sammon zaproponowala optymaliację za pomoca zmodyfikowanej metody Newtona

$$y_{pq}(k+1) = y_{pq}(k) - \eta \Delta_{pq}(k)$$

gdzie

$$\Delta_{pq}(k) = \frac{\partial E}{\partial y_{pq}} \Bigg/ \left| \frac{\partial^2 E}{\partial y_{pq}^2} \right|$$

składowa gradientu

$$\frac{\partial E}{\partial y_{pq}} = -\frac{2}{c} \sum_{j=1, j \neq p}^{n} \left(\frac{\delta_{pj} - d_{pj}}{d_{pj} \delta_{pj}} \right) (y_{pq} - y_{jq})$$

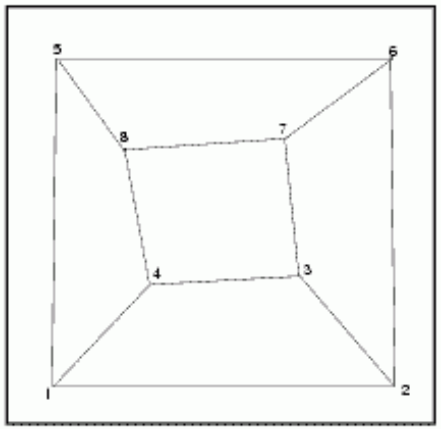
diagonalne składowe Hessianu

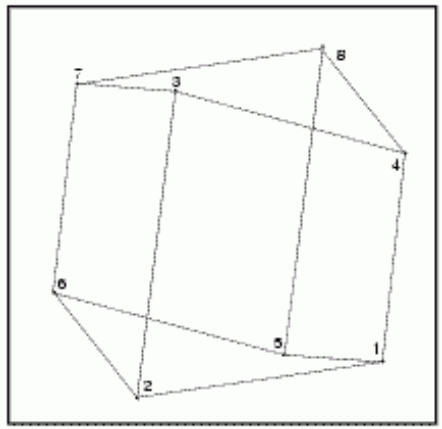
$$\frac{\partial^2 E}{\partial y_{pq}^2} = -\frac{2}{c} \sum_{j=1, j \neq p}^{n} \frac{1}{\delta_{pj} d_{pj}} \left[(\delta_{pj} - d_{pj}) - \frac{(y_{pq} - y_{jq})^2}{d_{pj}} \left(1 + \frac{\delta_{pj} - d_{pj}}{d_{pj}} \right) \right]$$

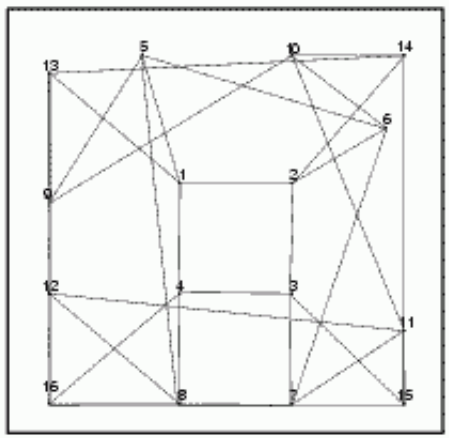
MDS i SOM

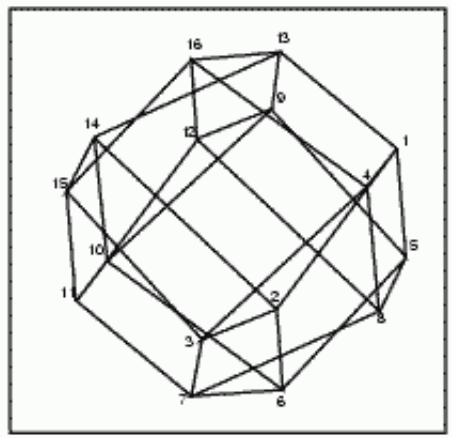
- w MDS brak funkcji mapującej $X \rightarrow Y$
- dla MDS dodanie nowego punktu wymaga ponownej optymalizacji
- SOM używany także w klasyfikacji
- MDS i SOM wrażliwe na szum w danych (nieistotne cechy)
- wyniki uzależnione od punktu startowego, każda optymalizacja może generować odmienne wyniki

Wizualizacja: hiperkostka

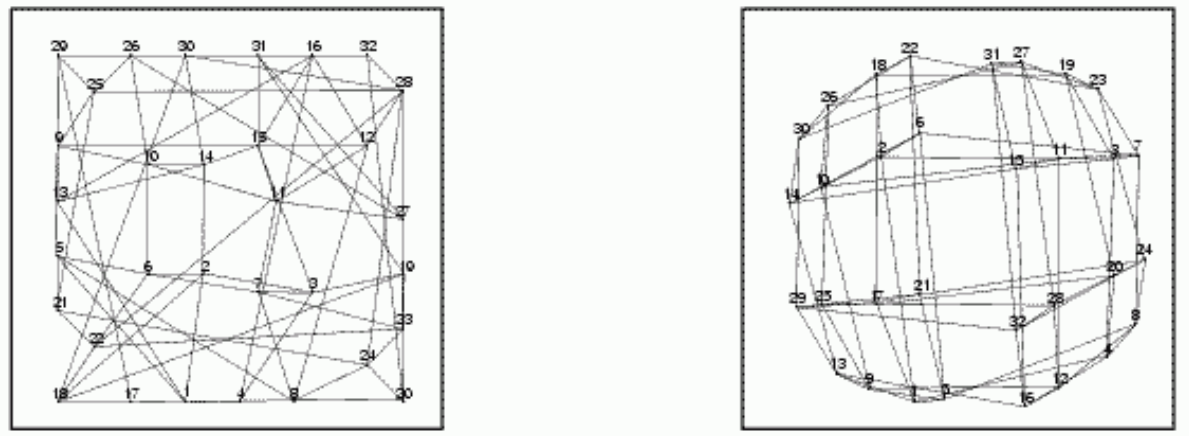


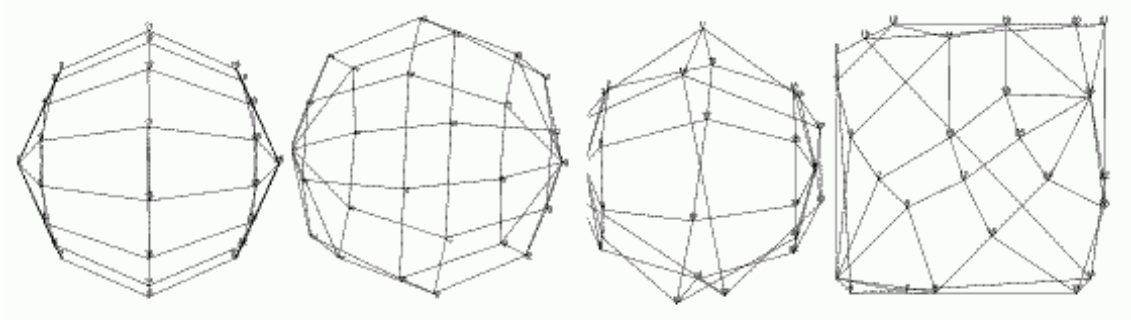






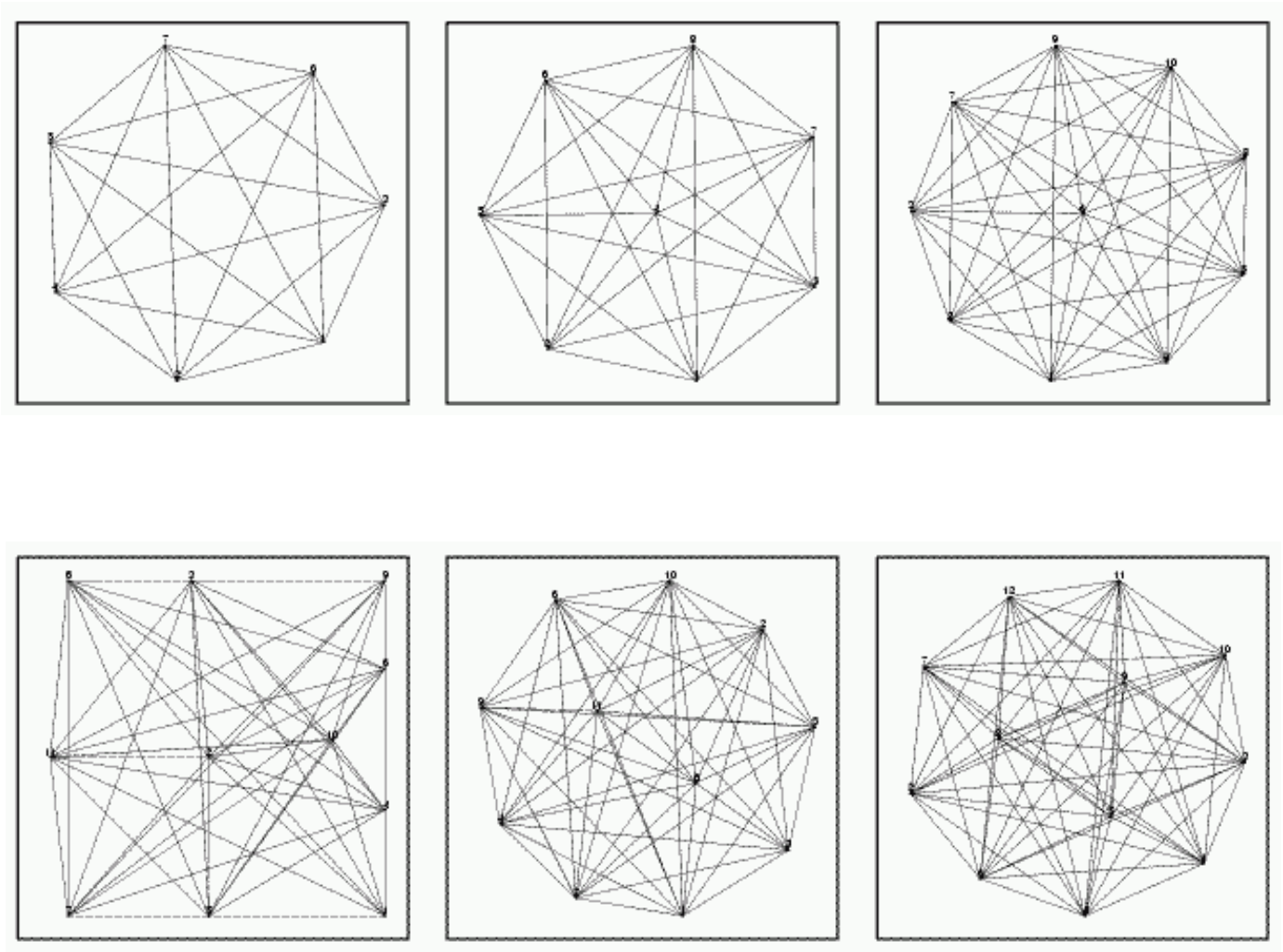
Wizualizacja: hiperkostka 5D + sfera w 3D



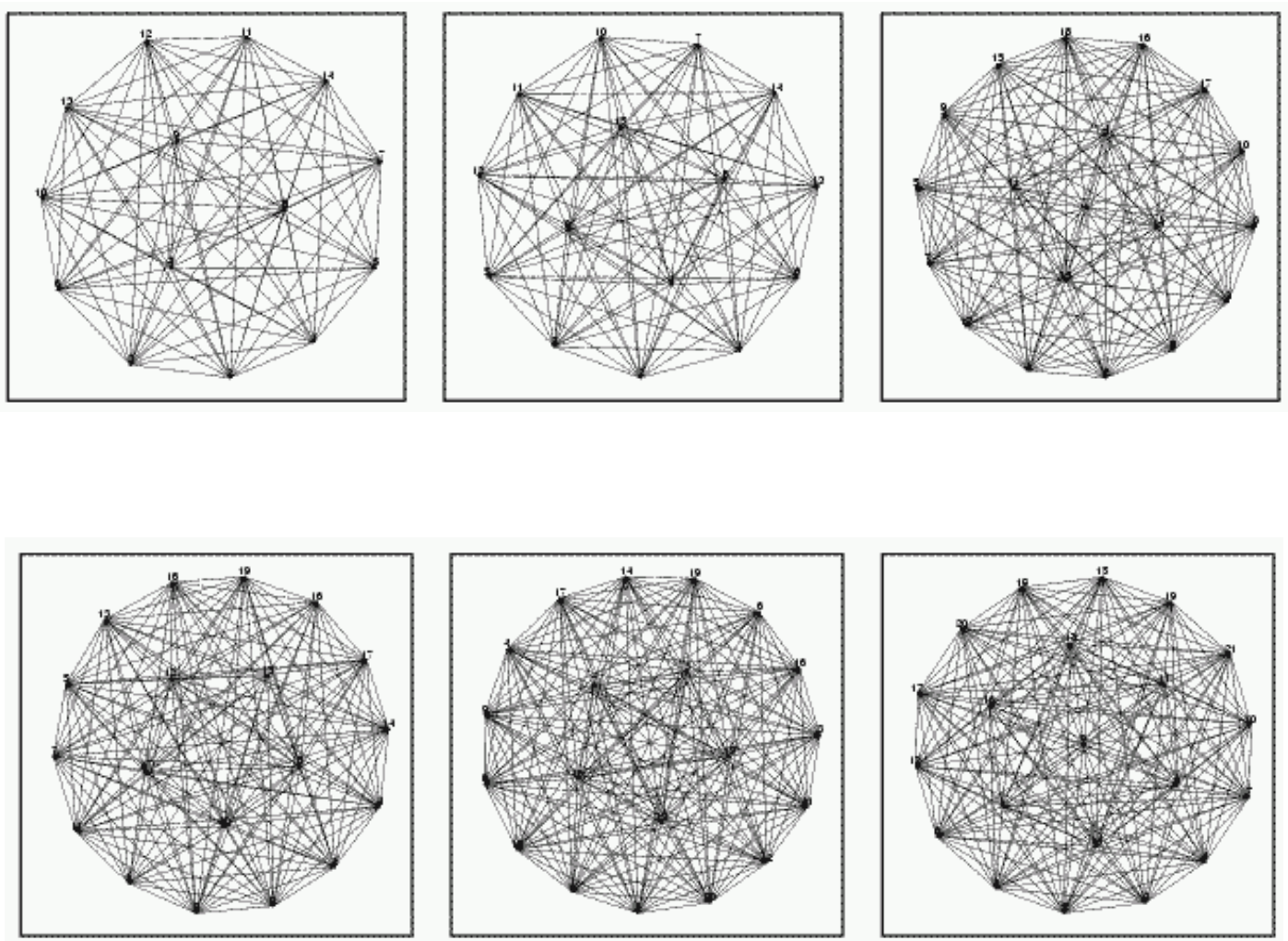


The two-dimensional representations of the 26 points on the sphere obtained by minimization of S, E, A, and by **SOFM** (left to right) with a 20 x 20 neurons map.

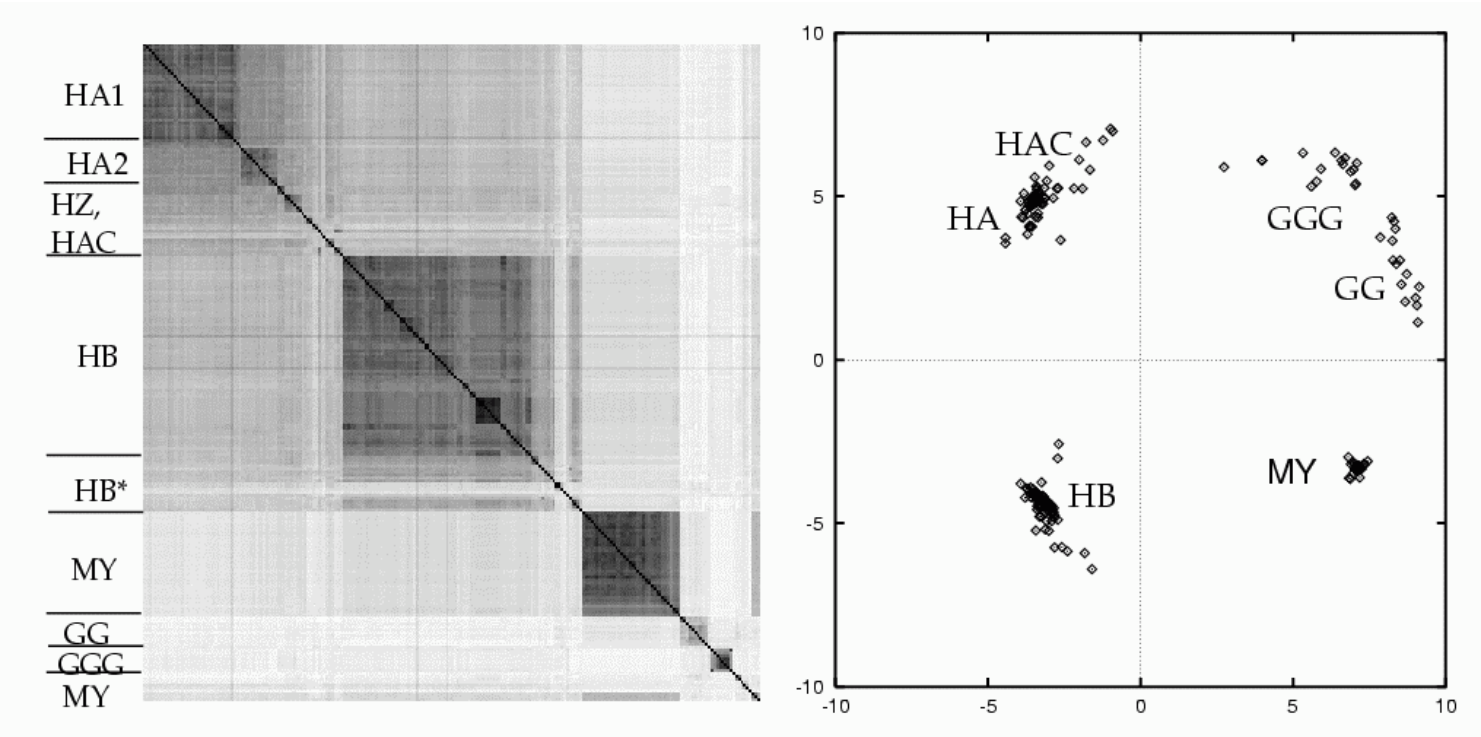
Wizualizacja: sympleks 6-11



Wizualizacja: sympleks 15-20

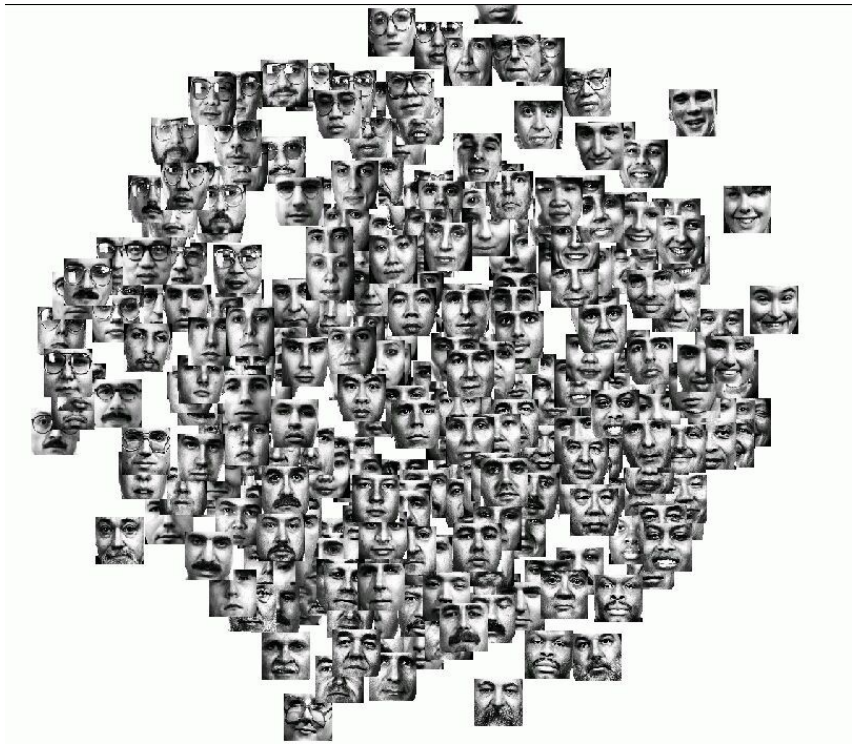


Sekwencje rodziny białek



sekwencja białek rodziny Globin, macierz podobieństwa, MDS odkrywa strukturę danych (Klock & Buhmann 1997).

Podobieństwo twarzy



300 twarzy (from Klock & Buhmann 1997).

Mapy semantyczne

- próba uchwycenia sensu słów i pojęć poprzez własności oraz relacji semantycznych
- Przykład: 8 ptaków i 8 ssaków
  dove, hen, duck, goose, owl, hawk, eagle,
  fox, dog, wolf, cat, tiger, lion, horse, zebra, cow.
- każde pojęcie opisane 13 cechami binarnymi
  size is: small, medium large;
  has 2 legs, 4 legs,
  has hair, hoofs, mane, feathers;
  likes to: hunt, run, fly, swim.
- utwórz zdania opisujące zwierzęta za pomocą wybranych cech
  Horse is big, has 4 legs, mane, hair, hoofs, likes to run.

Mapy semantyczne

	animal	dove	hen	duck	goose	owl	hawk	eagle	fox	dog	wolf	cat	tiger	lion	horse	zebra	cow
is	small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Mapy semantyczne

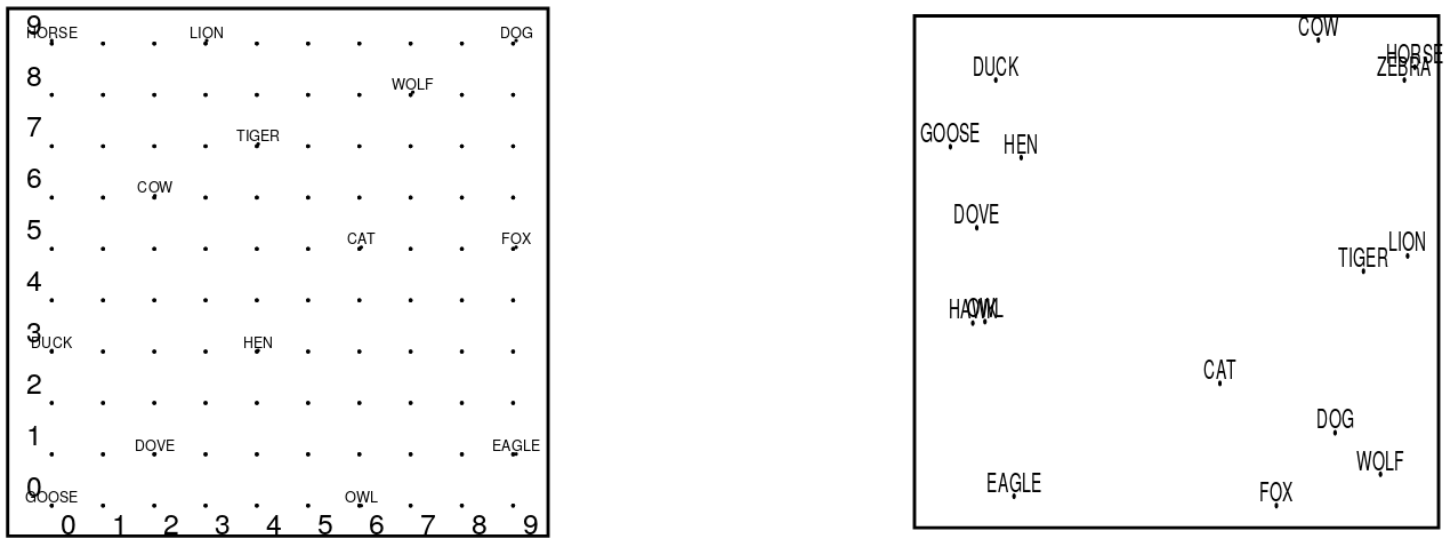


FIG. 1. The two-dimensional representations of the 13-dimensional semantic data obtained by SOM (left) with a 10 x 10 neurons map, a training of 10000 cycles, with final stress of 0.25, and the MDS (right) with final stress of 0.20 after 10 iterations.

SOM, Ritter and Kohonen 1989,
MDS, Naud & Duch (1996).

- Naturalna klasyfikacja zwierząt zachowująca ich podobieństwo
  - ssaki odseparowane od ptaków
  - małe zwierzęta od dużych
  - drapieżników od roślinożerców
- podobieństwo zwierząt jest powiązane z odległością na mapie

Samoorganizacja

Uczenie hebbowskie

Sieci hebbowskie

- uczenie konkurencyjne
  - reguła Kohonena
  - kwantyzacja wektorowa, SOM
- uczenie hebbowskie
  wykrywa powiązania korelacyjne między sygnałami
  - reguła Hebba, reguła Oja
  - dekompozycja składowych głownych PCA (Principal Component Analizys)
  - dekompozycja na składowe niezależne ICA (Independent Component Analizys) , separacja źródeł BSS (Bling Source Separation), nielinowe PCA, ...

Reguła Hebba (1949)

Neurons, that fire together, wire together.

Jeżeli neuron A jest cyklicznie 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
- jeżeli neurony nie są pobudzane jednocześnie (asynchronicznie) to połaczenie jest osłabiane

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

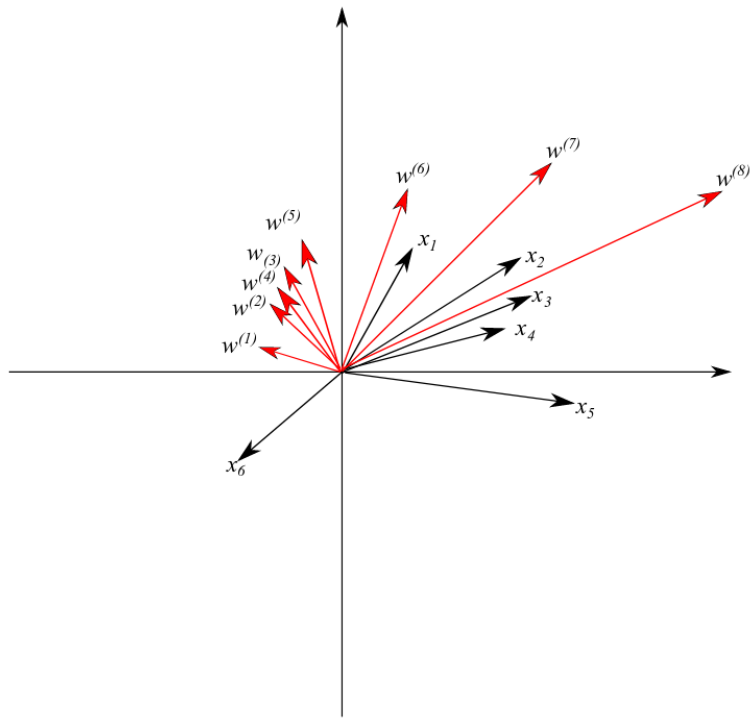
$$\Delta w_i = \eta y x_i$$

Decydujący wpływ na wartość wyjściową będą miały najczęściej występujące sygnały.

Uczenie hebbowskie prowadzi do wyznaczenia miary podobieństwa między sygnałem wejściowym x_i a rozkładem gęstości prawdopodobieństwa $p(x)$ sygnału.

Interpretacja geometryczna reguły Hebba

Wektor wag przemieszcza się w kierunku środka masy wektorów treningowych (kąt maleje w kierunku największej wariancji)



Trening prowadzi do nieograniczonego wzrostu wag (norma rośnie w każdym kroku)

Modyfikacje reguły Hebba

Zapobieganie nieograniczonemu wzrostowi wag

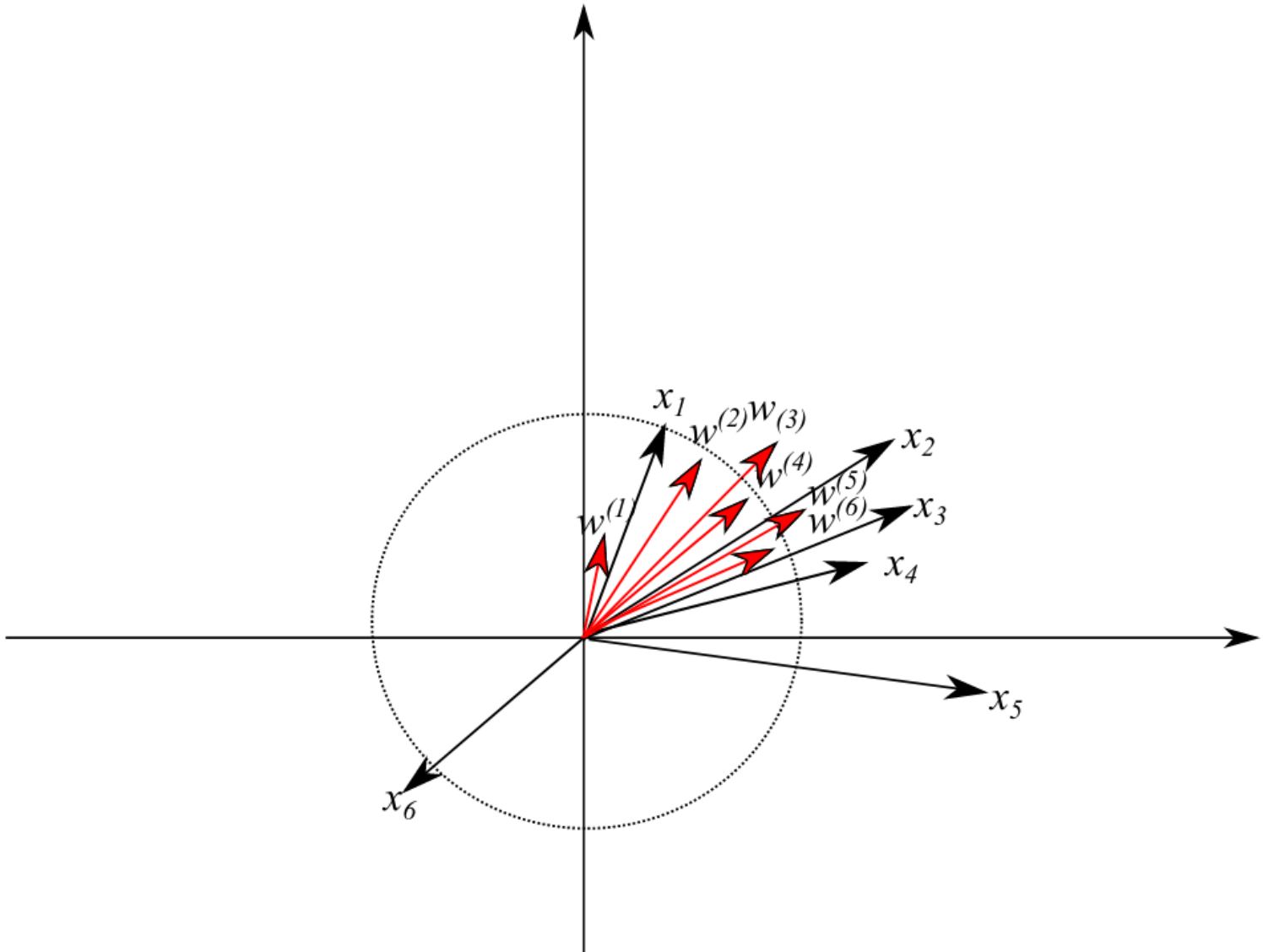
- normalizacja wag po każdym kroku uczenia
- dodanie czynnika normalizującego wagi, Oja (1982)

Reguła Oji

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

- dodanie wyrazu zanikania wagi proporcjonalnego do y^2
- adaptacja prowadzi do unormowania wag $\|\mathbf{w}\| = 1$
- adaptacja sprawia, że wektor wag zbliża się do wektora własnego macierzy kowariancji $E[\mathbf{X}\mathbf{X}^T]$ o największej wartości własnej
- wektor wag leży wzdłuż kierunku maksymalizaującego wartość oczekiwaną y^2, tj. wariancję sieci

Interpretacja geometryczna reguły Oji



Własności reguły Oji

Dla wyuczonego wektora wag wartość oczekiwana

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

stąd

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

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

Macierz kowariancji $E[\mathbf{x}\mathbf{x}^T] = \mathbf{R}_x$
Reguła Oji prowadzi do wyznaczenia wektora własnego $\mathbf{R}_x$

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

Analiza składowych głównych PCA

Transformacja liniowa

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

gdzie

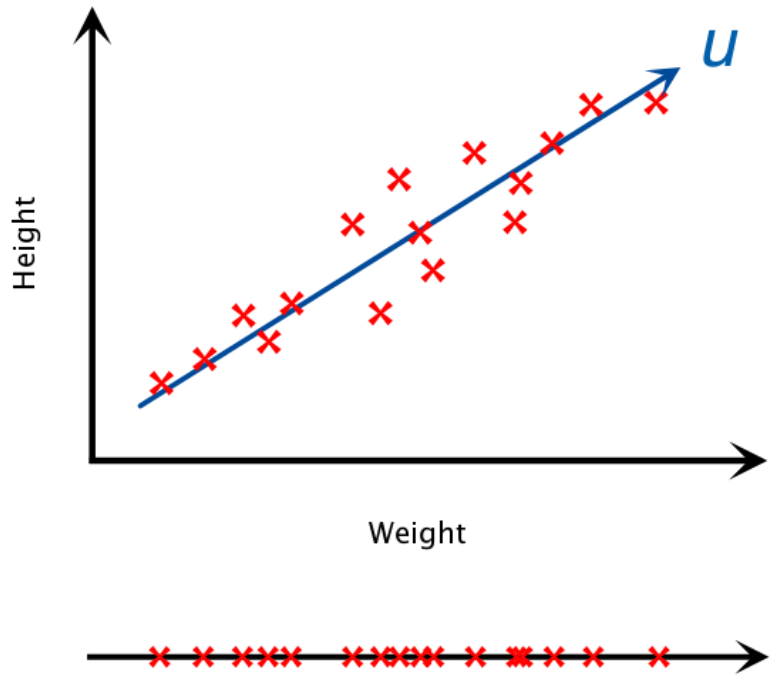
$$\mathbf{x} \in \mathbb{R}^N \qquad \mathbf{y} \in \mathbb{R}^K \quad \mathbf{W} \in \mathbb{R}^{K \times N} \qquad K < N$$

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 nieujemna $\Rightarrow$ wartości własne 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

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

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

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

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

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. Diagonalicacj 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

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

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 $\sum_{i=1}^{K} \lambda_i$

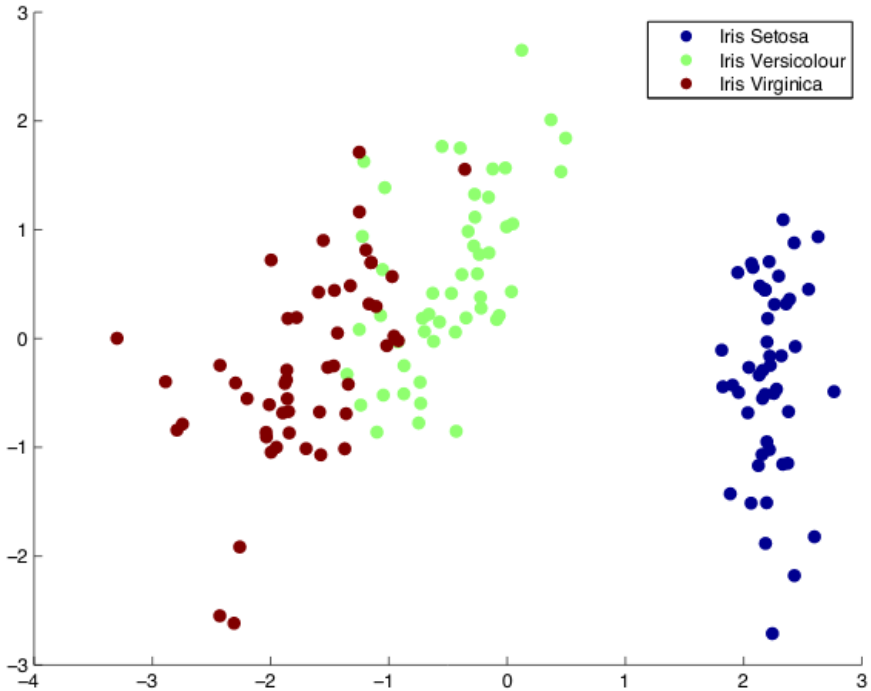
PCA

- 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 dużych wymiarów są zbyt złożone
- Metody adaptacyjne wyznaczania składowych głównych nie wymagają obliczania macierzy $\mathbf{R}_{xx}$
- Mogą te być stosowane w metodach on-line, gdy nie mamy dostępu do wszystkich danych

Przykład: PCA na danych Iris

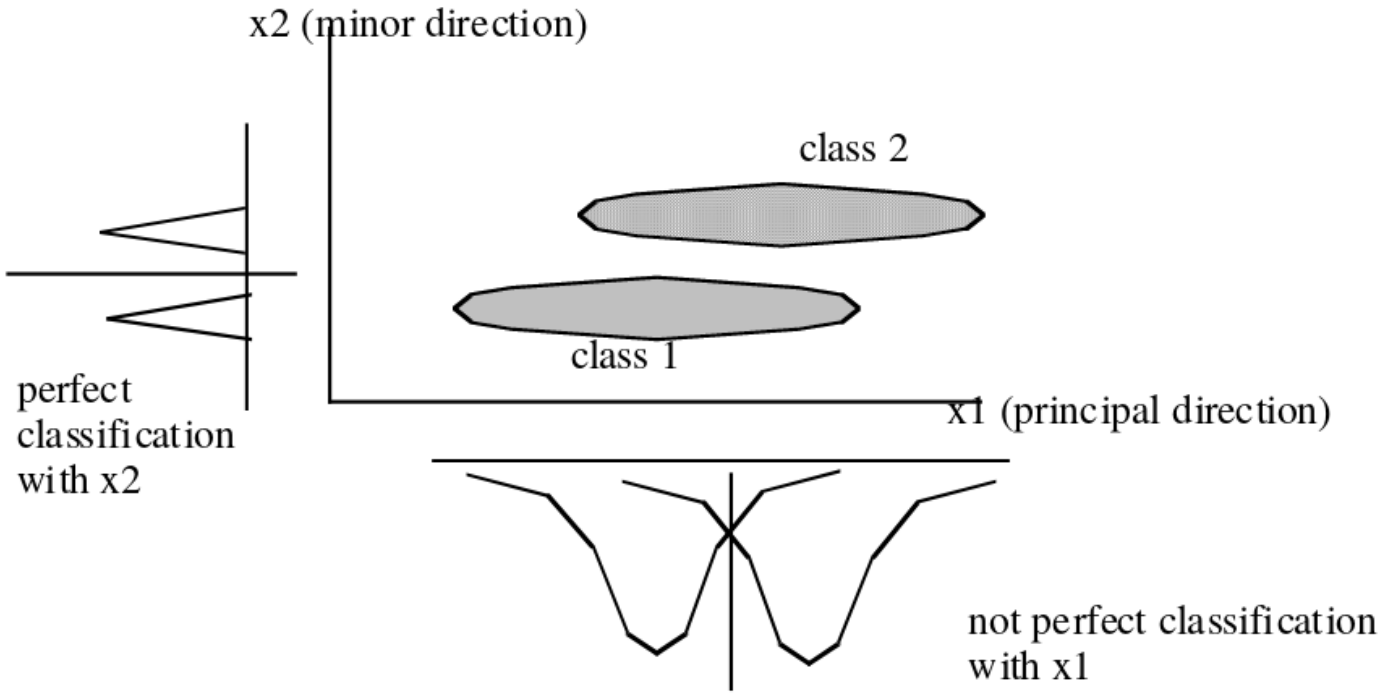


$$\lambda_1 = 2.8914 \quad \lambda_2 = 0.9151 \quad \lambda_3 = 0.1464 \quad \lambda_4 = 0.0205$$

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

MDS i PCA

- MDS wynik zależny od punktu startowego, stosowany wielokrotny start
- PCA może zostać użyte jako punkt startowy dla MDS
- PCA jest linowe, nie zachowuje odległości pomiędzy obiektami
- PCA wymaga utworzenia macierzy korelacji $d \times d$, koszt $O(nd^2)$, diagonalizacja tej macierzy $O(d^3)$
- MDS wymaga obliczenia macierzy odległości $O(n^2 d)$ i minimalizacji $2n - 3$ parametrów

Trening pierwszego kierunku

Estymator pierwszego kierunku

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

za pomocą reguły 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,:

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

Uogólniony algorytm hebbowski (GHA)

Jednowarstwowa sieć liniowa (Sanger, Oja, 1989)

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

Reguła uczenia (Sanger)

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

oznaczając

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

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

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

Wartości wag $\mathbf{w}_i$ kolejnych neuronów uzyskują wartości kolejnych wartości własnych macierzy kowariancji $\mathbf{C}$

Uczenie anty-hebbowskie

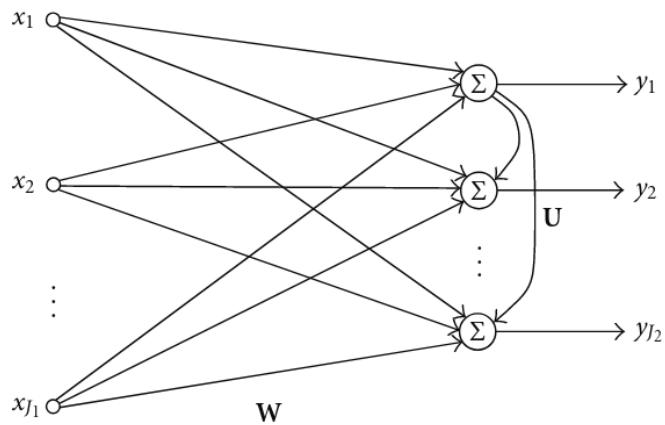
Reguła anty-Hebbowska

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

- 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 ma kierunku, wzdłuż którego wariancja wynosi 0 to wagi zbiegają do 0

APEX

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



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

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

gdzie
wyjścia $\mathbf{y} = (y_1, \ldots, y_{i-1})^T$
macierz wag pierwszych $i-1$ neuronów $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_{i-1}]^T$
wagi połączeń wewnątrz warstwy $\mathbf{u}_i = (u_{1i}, u_{2i}, \ldots, u_{(i-1)i})^T$

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

Aktualizacja wag APEX

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 międzywarstwowe c_{ij} uczone reguła anty-hebbowską odpowiedzialną za ortogonalizację y_i względem poprzednich składowych

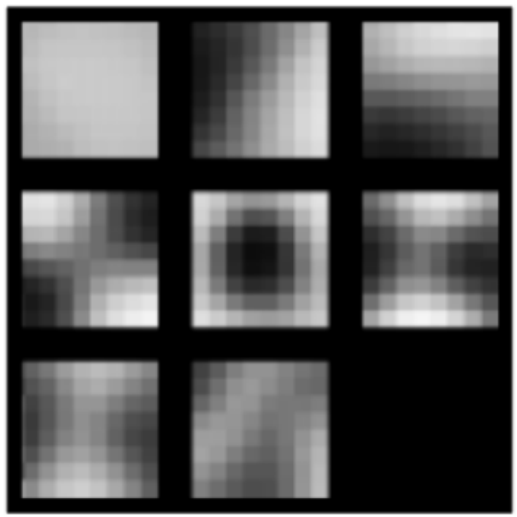
$$\Delta \mathbf{u}_i = -\eta(n) y_i(n) \left[\mathbf{y}(n) + y_i(n) \mathbf{u}_i \right]$$

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 (np. 8x8)
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

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

- neurony 3-5: pochodna drugiego rzędu obrazu

Przykład: Kompresja obrazu za pomocą APEX

a) oryginał, b) 4PC, c) 8PC, d) 12PC

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

Sieci dynamiczne

Pamięć asocjacyjna

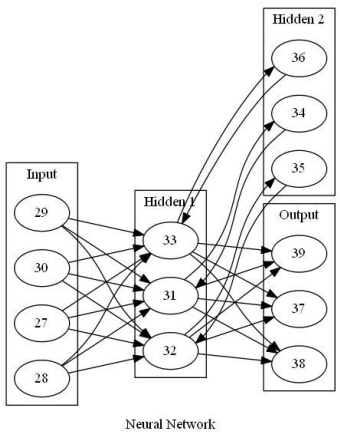
Plan

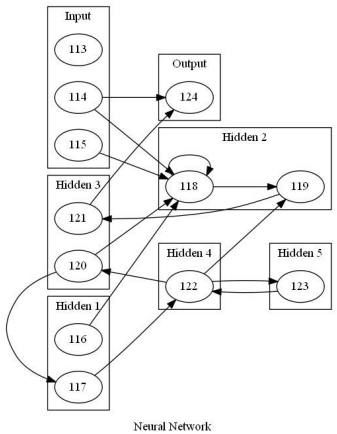
- Sieci dynamiczne: sieci ze sprzężeniami zwrotnymi
- Modele pamięci asocjacyjnej
- Model Hopfielda, siec Hamminga, BAM

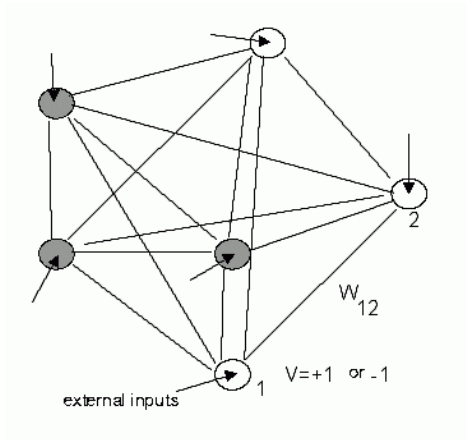
Sieci ze sprzężeniami zwrotnymi

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







Sieci ze sprzężeniami zwrotnymi

Najprostsze modele sieci z rekurencją:

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

Modele złożone:

- RTRN - Real Time Recurrent Network, przetwarzająca sygnały w czasie rzeczywistym;
- sieć Elmana i inne o uproszczonej strukturze rekurencji
- RCC - Recurrent Cascade Correlation
- LSTM, Long Short Term Memory

Model Hopfielda

John Hopfield (1982, 1984), model pamięci autoasocjacyjnej.
Założenia:

- wszystkie neurony są ze sobą połączone z wagami synaps w_{ij}, brak warstwy ukrytej
- macierz wag połączeń jest symetryczna, $w_{ij} = w_{ji}$
- brak połączeń zwrotnych do tego samego neuronu $w_{ii} = 0$

Symetria wag jest wygodna, upraszcza analizę sieci i pozwala wprowadzić funkcję energii; jednak jest nierealistyczna z biologicznego punktu widzenia.
Dyskretny stan neuronu - potencjał, zmiany następują w dyskretnych jednostkach czasu

$$y_i = \pm 1 = \mathrm{sgn}(\mathbf{w}^T \mathbf{x})$$

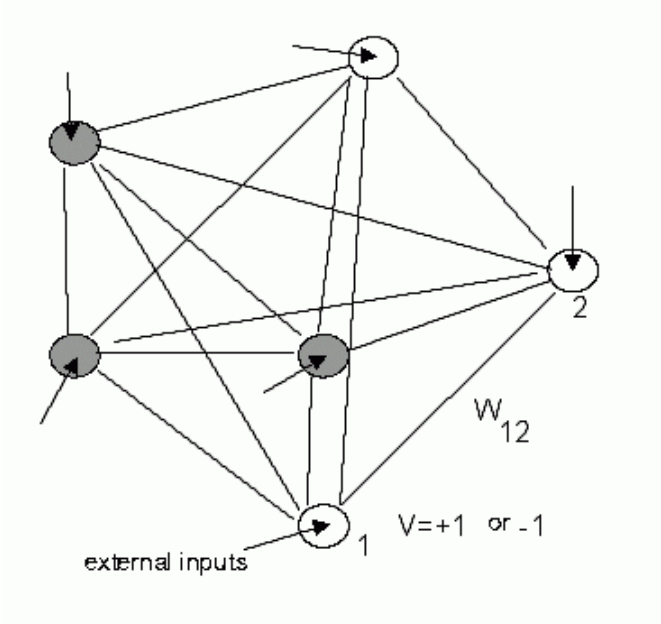
Sieci ciągłe - stany rzeczywiste, ciągłe funkcje aktywacji, zmiany aktywacji w sposób ciągły

Model Hopfielda

- każdy neuron jest wejściem i wyjściem sieci

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

- brak sprzężenia własnego $w_{ii} = 0$
- wagi symetryczne $w_{ij} = w_{ji}$
- stany sieci $\mathbf{y}$ można utożsamić z wierzchołkami N wymiarowej kostki



Tryby pracy

- uczenie (zapamiętywanie)
- odtwarzanie wzorców

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

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

- proces asynchroniczny - aktualizowane jest wyjście jednego losowo wybranego neuronu, jego wyjście zależy od już zaktualizowanych neuronów,
- proces zatrzymuje się gdy osiągnięto stan stacjonarny (minimum lokalne), brak zmian wyjść przy pełnym cyklu

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

- asynchroniczność minimalizuje występowanie cykli i sieć zbiega do atraktora punktowego
- autoasocjacja - wyjście odtwarza $\mathbf{x}$

Minimalizacja energii

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

$$E(W) = -\frac{1}{2}\langle \mathbf{y}|\mathbf{W}|\mathbf{y}\rangle = -\frac{1}{2}\sum_{i\neq j} w_{ij} y_i y_j$$

Zmiana energii w czasie iteracji jest mniejsza od zera

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

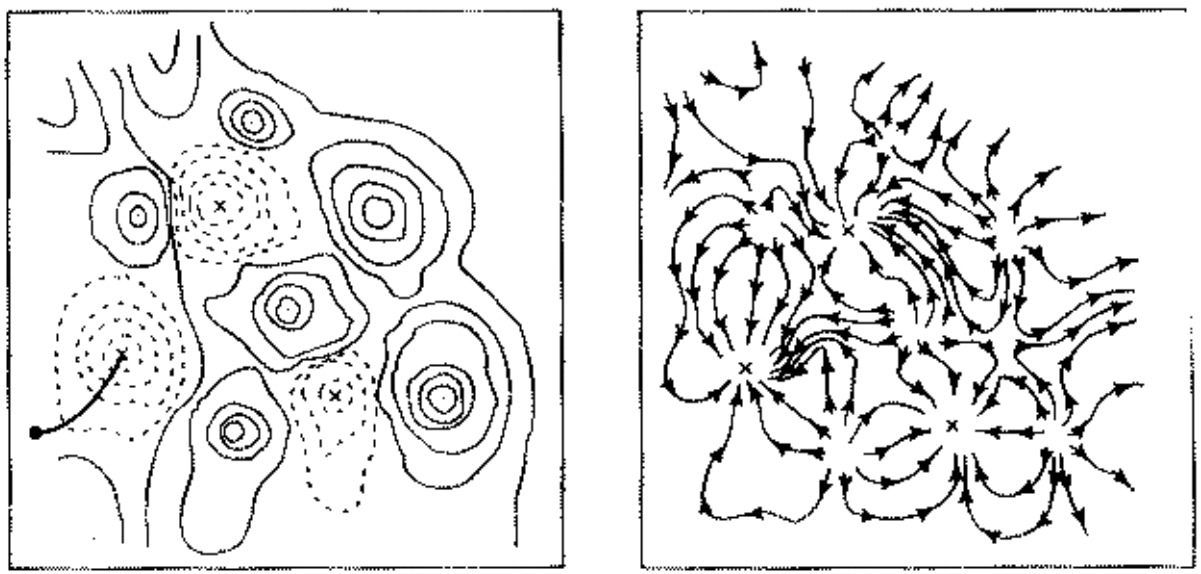
- jeżeli $I_i \geq 0$ to y_i nie może zmaleć
- jeśli $I_i < 0$ to $\Delta y_i < 0$

Każda zmiana stanu neuronu może tylko obniżyć energię sieci

Atraktory

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

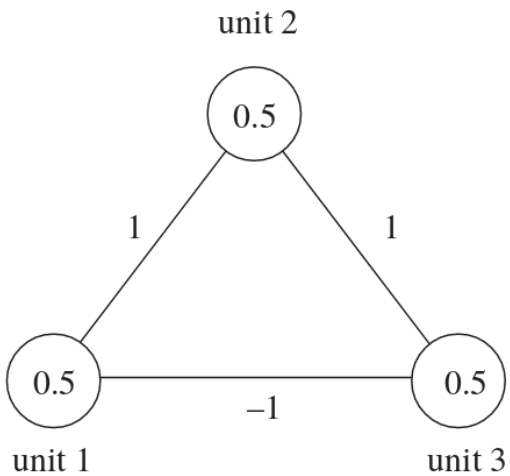
Jeśli y_i dyskretne $-1, +1$ to ruch po wierzchotkach hiperkostki

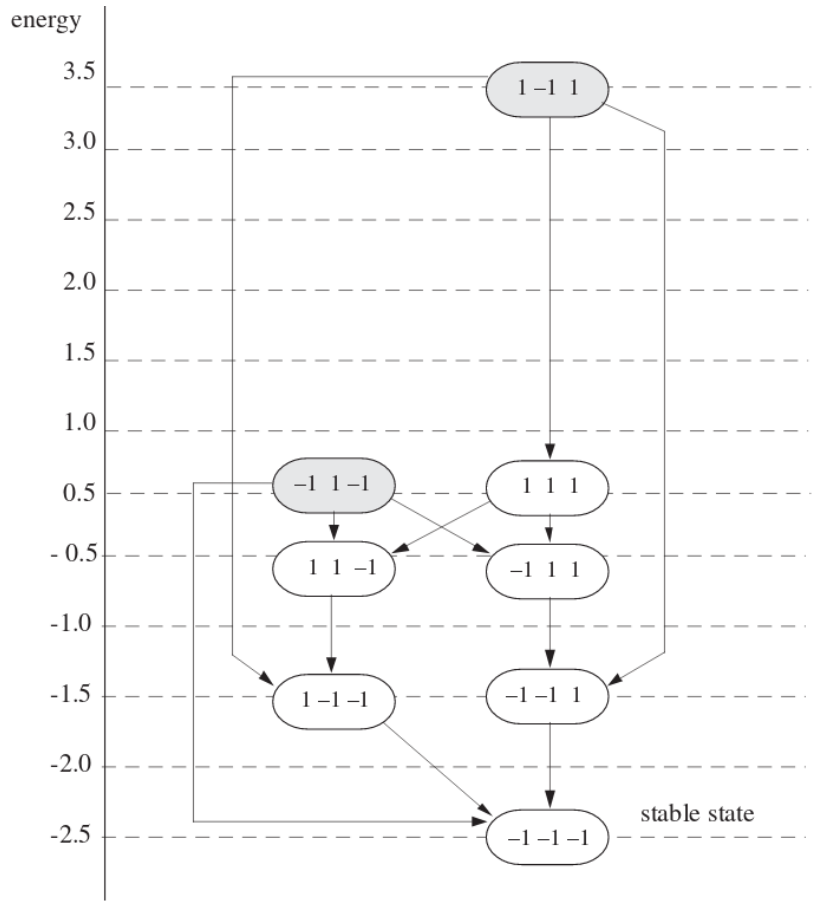


Obszary atrakcji sieci rekurencyjnej: linie ekwipotencjalne i kierunki zmian w trakcie uczenia

Rys: Osowski, Sieci Neuronowe

Przykład: 3 neurony

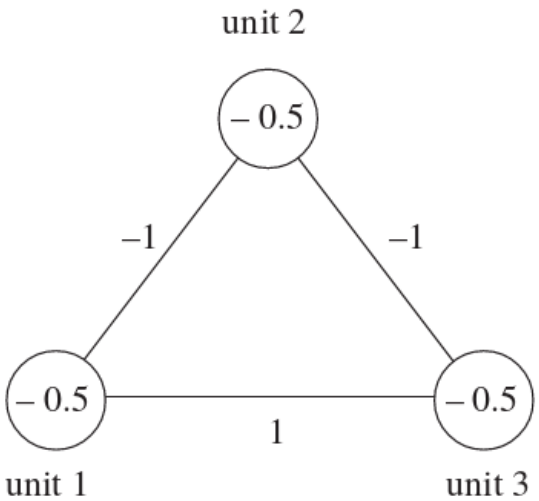


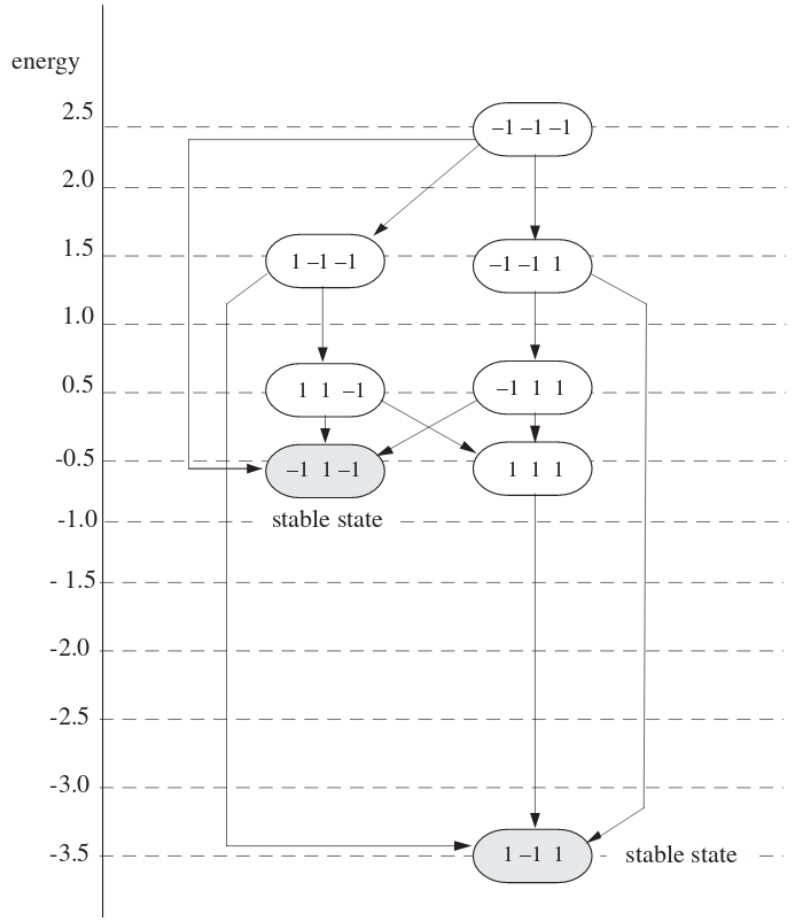


$$E = -y_1y_2 - y_2y_3 + y_1y_3 - 0.5y_1 - 0.5y_2 - 0.5y_3$$

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
  zapamiętuje wzorce $\mathbf{x}_1, \ldots, \mathbf{x}_p$, przy prezentacji wektora wejściowego $\mathbf{x}_i$ odpowiedzią sieci będzie jeden z zapamiętanych wzorców, najbardziej „podobny” do sygnału wejściowego.
  Każdy zapamiętany wzorzec odpowiada minimum lokalnemu energii
- problemy optymalizacyjne - odpowiedni konstrukcja sieci i funkcji energii pozwala znaleźć rozwiązania problemów optymalizacyjnych, także NP-trudnych
  - problem komiwojażera,
  - gospodarowanie zasobami,
  - optymalizacja połączeń w centralach telefonicznych,
  - rozmieszczanie układów scalonych,
  - rozwiązywanie problemów programowania liniowego i nieliniowego

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 - wzorzec na wyjściu uzyskiwane na podstawie odległości sygnału wejściowego od zapamiętanego wzorca
- pamięć rozproszona - brak wyraźnie określonego miejsca przechowywania wzorca
- lokalne uszkodzenie nie powinno całkowicie blokować możliwości odczytu

Odległość Hamminga

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

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

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

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

liczba różniących się bitów

Pamięć asocjacyjna będzie odtwarzać wektory zapamiętane najbliższe względem wektora wejściowego zgodnie z odległością Hamminga

Uczenie pamięci asocjacyjnej

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

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

wystarczy zażądać by:

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

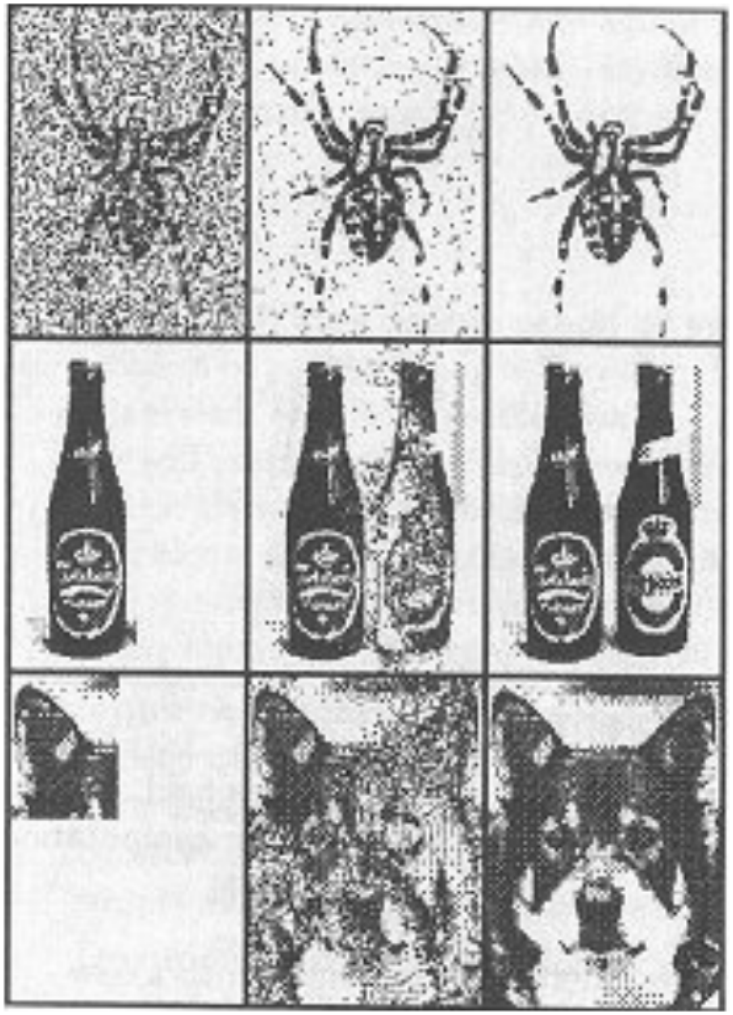
$$\frac{1}{N}\left(\sum_{j=1}^{N} x_i x_j x_j\right) = \frac{1}{N}\left(\sum_{j=1}^{N} x_i x_j^2\right) = \frac{1}{N}\left(\sum_{j=1}^{N} x_i\right) = x_i$$

Uczenie pamięci asocjacyjnej

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

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

Przykłady działania



odtwarzanie niekompletnych lub zaszumionych danych

Stabilność pamięci

Stabilność działania wymaga

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

co można zapisać

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

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

- Pojemnośc określa maksymalną liczbe odtwarzanych wzorców przy okreslonym poziomie błędu
- 2^N możliwych stanów sieci binarnej złożonej z N neuronów
- zbyt wiele wzorców $\Rightarrow$ chaos, zapominanie, zwiększając liczbę wzorców rośnie prawdopodobieństwo przekłamań
- dla uczenia regułą Hebba liczba poprawnie pamiętanych wzorców przy prawdopodobieństwie błędu 0.37% wynosi $0.138N$
- metoda rzutowania pozwala uzyskać pojemność maksymalną $N - 1$

Stany fałszywe

W sieci mogą pojawiać się stany fałszywe lub przekłamania pamięci

- funkcja energii jest symetryczna względem polaryzacji (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 wzorcowi zapamiętanemu

Metoda 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 $i = 1, \ldots, p$

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

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

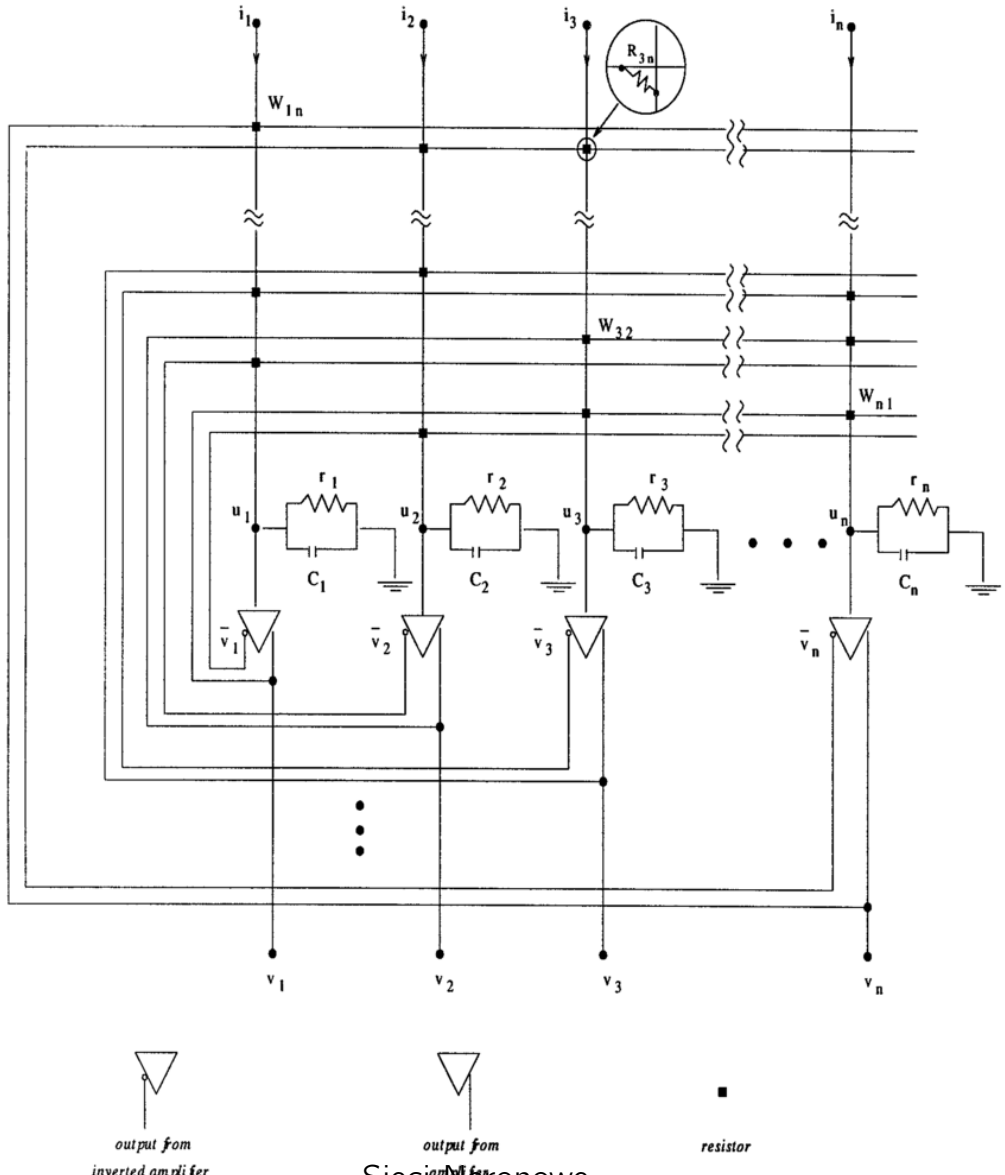
Metoda rzutowania Δ

Gradientowa odmiana metody rzutowania

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

- 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 stodunku do uczenia reguła Hebba, maksymalna liczba zapamiętanych wzorców $N - 1$

Realizacja sprzętowa



Realizacja sprzętowa - równania

Prosta realizacja sprzętowa, elektroniczna lub optyczna.
W stanie stacjonarnym wejście=wyjście.
Równania na sygnały wejściowe:

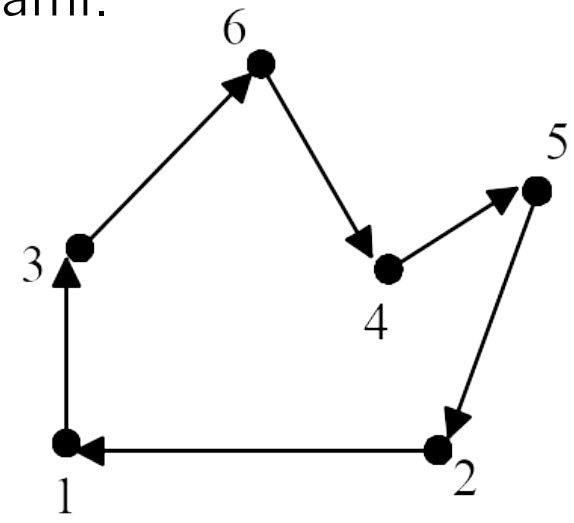
$$C_i \frac{dU_i(t)}{dt} = \sum_{j=1}^{N} R_{ij}^{-1} V_j(t) - R_i^{-1} U_i(t) + I_i$$

U_i - napięcie wejściowe i-tego wzmacniacza
V_i - napięcie wyjściowe i-tego wzmacniacza
C - pojemność wejściowa
I_i - zewnętrzny prąd i-tego wzmacniacza

$$V_i(t) = f\left(U_i(t)\right); \quad R_i^{-1} = \sum_{j=1}^{N} R_{ij}^{-1}$$

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

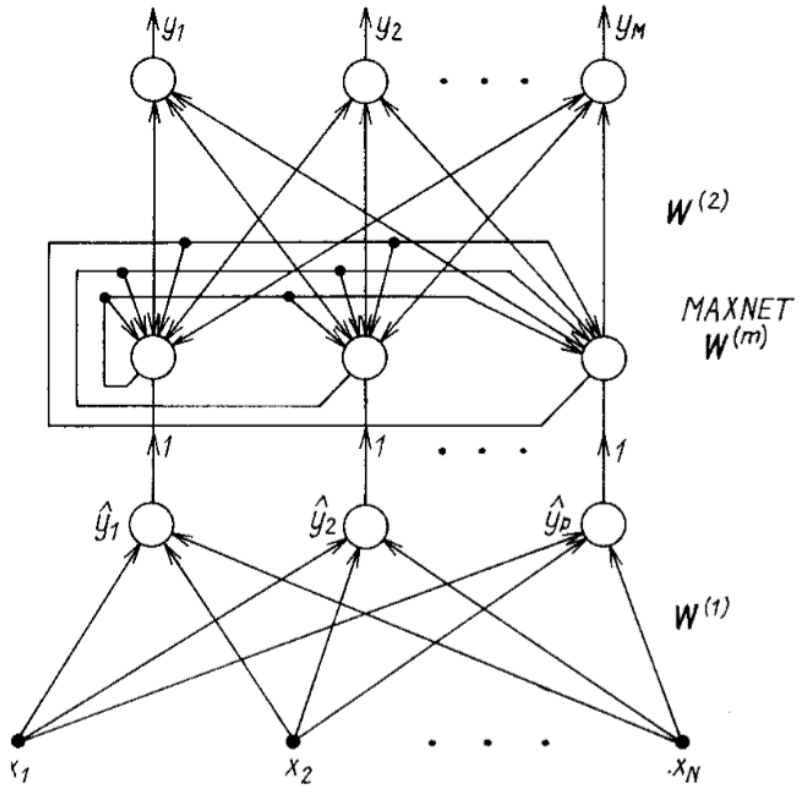
$$\begin{aligned} w_{i\alpha,k\beta} = \quad & -A\left(1-\delta_{\alpha\beta}\right)\delta_{ik} - B\left(1-\delta_{ik}\right)\delta_{\alpha\beta} - C \\ & -Dd_{ik}\left(1-\delta_{ik}\right)\left(\delta_{\alpha-1,\beta}+\delta_{\alpha+1,\beta}\right) \end{aligned}$$

oraz wartości progowe CN

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 Hamminga

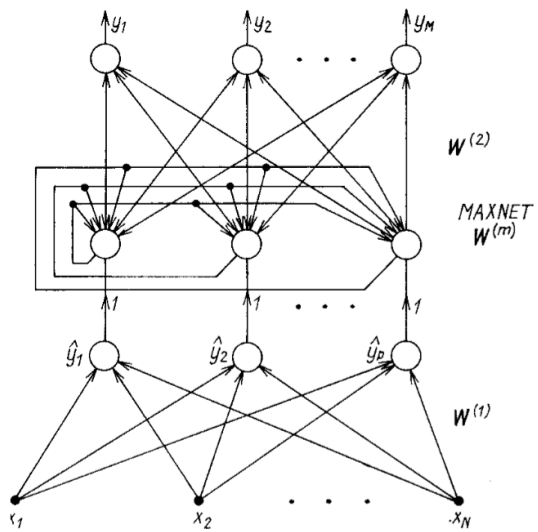


Sieć Hamminga

- 3 warstwowa sieć - rozszerzenie sieci Hopefielda
- pamięć heteroasocjacyjna: wejście $\mathbf{x} \in \{-1, +1\}^N$, wyjścia $\mathbf{y} \in \{-1, +1\}^M$
- klasyfikator wektorów - wyjściem jest zapamiętany wektor o najmniejsej odległości Hamminga
- warstwa pierwsza i trzecia - jednokierunkowe
- warstwa MAXNET ze sprzężeniami zwrotnymi, połączenia każdy z każdym
  - istnieją połączenia zwrotne do tych zamych neuronów $w_{ii} = +1$, mają charakter pobudzający
  - połączenia między różnymi neuronami inhibicyjne (hamujące) $w_{ij} = -\epsilon$, np. $\epsilon \approx \frac{1}{p}$, gdzie p - liczba wzorców
  - WTA - tylko jeden neuron w stanie ustalonym jest pobudzony, reszta w spoczynku

Działanie sieci Hamminga

1. podanie stanu **x** na wejście, zainicjowanie warstwy MAXNET
2. proces iteracyjny w warstwie MAXNET trwa dopóki wszystkie jednostki oprócz jednej osiągną stan 0. Neuron niezerowy reprezentuje klasę wektora
3. wytworzenie odpowiedzi sieci **y** na podstawie sygnału zwycięzcy



Dobór wag

1. wagi warstwy pierwszej odpowiadają sygnałom wejściowym

$$w_{ij}^{(1)} = x_j^{(i)}, \quad i = 1, \ldots, p$$

2. wagi warstwy wyjściowej kodują sygnały wyjściowe

$$w_{ij}^{(3)} = y_j^{(i)}, \quad i = 1, \ldots, p$$

3. warstwa MAXNET

$$w_{ii}^{(m)} = 1$$

$$-1\frac{1}{1-p} < w_{ij}^{(m)} < 0$$

Działanie sieci Hamminga

Pierwsza warstwa wyznacza odległość Hamminaga od podanego wzorca

$$\hat{y}_i = \frac{\sum_j w_{ij}^{(1)} x_j + n}{2n} = 1 - \frac{d_H\left(x^{(i)}, x\right)}{n}$$

Rekurencyjny proces wyłaniania zwycięzcy warstwy MAXNET

$$y_i(k) = f\left(\sum_j w_{ij}^{(m)} y_i(k-1)\right)$$

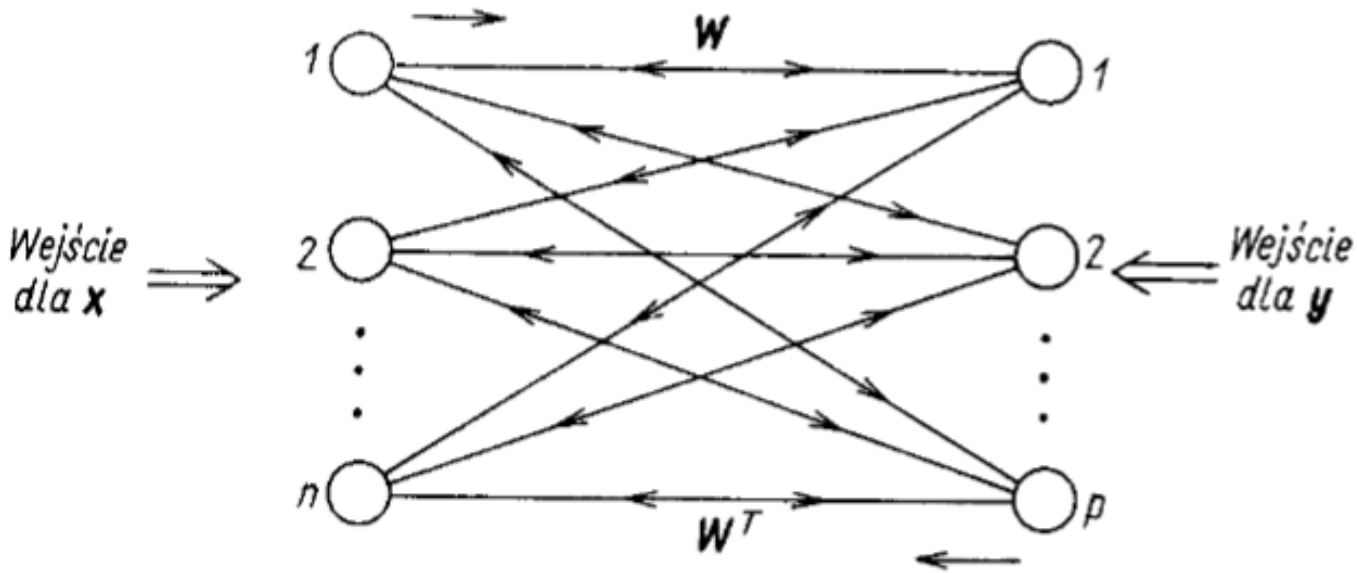
gdzie funkcja wyjściowa (ReLU)

$$f(y) = \begin{cases} y & \text{dla} \quad y \geq 0 \\ 0 & \text{dla} \quad y < 0 \end{cases}$$

Neuron zwycięzca przez wagi warstwy ukrytej odtwarza wektor **y** uznany przez MAXNET za najbliższy **x**

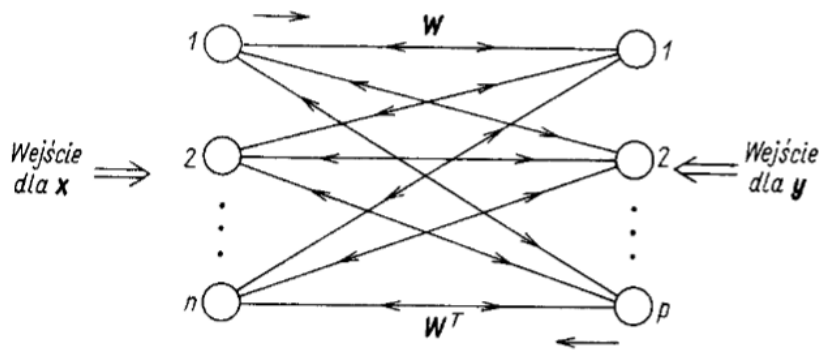
Sieć BAM

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



BAM

- przepływ sygnału w dwóch kierunkach
- praca synchroniczna
- aktywacje typu skokowego (binarne lub bipolarne)
- **W** rzeczywista, niesymetryczna
- koduje pary wektorów $\mathbf{x}_i, \mathbf{y}_i$ - pamięć heteroasocjacyjna



Działanie BAM

Trening - wagi (macierz korelacji)

$$w = \sum_{i=1}^{p} x_i^T y_i$$

Funkcja energii

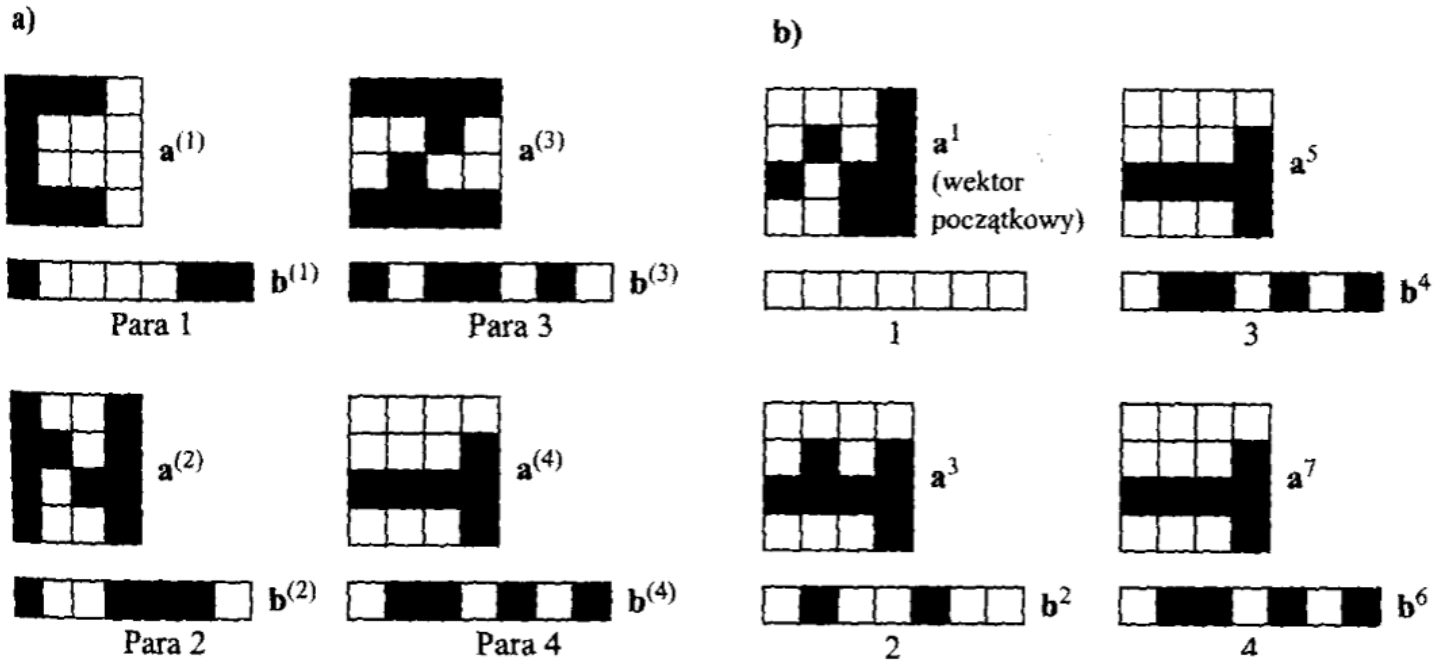
$$E_k = -\mathbf{x}_k \mathbf{W} \mathbf{y}_k^T$$

Odtwarzanie sygnału

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

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

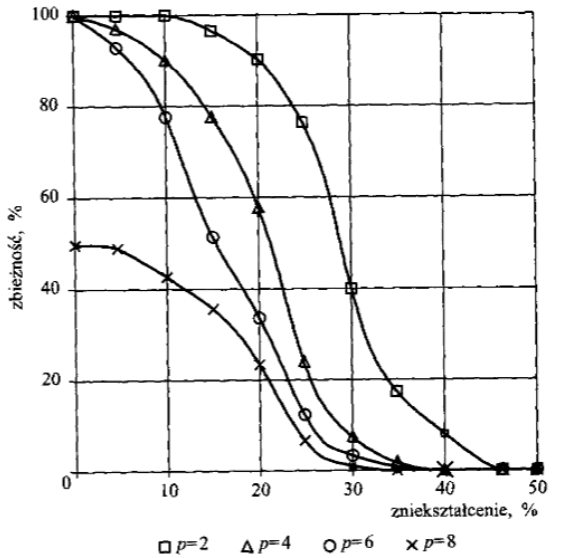
Przykład


a)
$\mathbf{a}^{(1)}$
$\mathbf{b}^{(1)}$
Para 1
$\mathbf{a}^{(3)}$
$\mathbf{b}^{(3)}$
Para 3
$\mathbf{a}^{(2)}$
$\mathbf{b}^{(2)}$
Para 2
$\mathbf{a}^{(4)}$
$\mathbf{b}^{(4)}$
Para 4
b)
$\mathbf{a}^1$
(wektor początkowy)
1
$\mathbf{a}^5$
$\mathbf{b}^4$
3
$\mathbf{a}^3$
$\mathbf{b}^2$
2
$\mathbf{a}^7$
$\mathbf{b}^6$
4

Pojemność BAM

Pojemność sieci

$$p = \sqrt{\min(n, m)}$$



Przykładowe wyniki dla $n = 25, m = 9$ i losowych wzorców ze średnią odległością 11

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. (?) Deep Reinforcement Learning (RL)

Literatura

[1] Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org

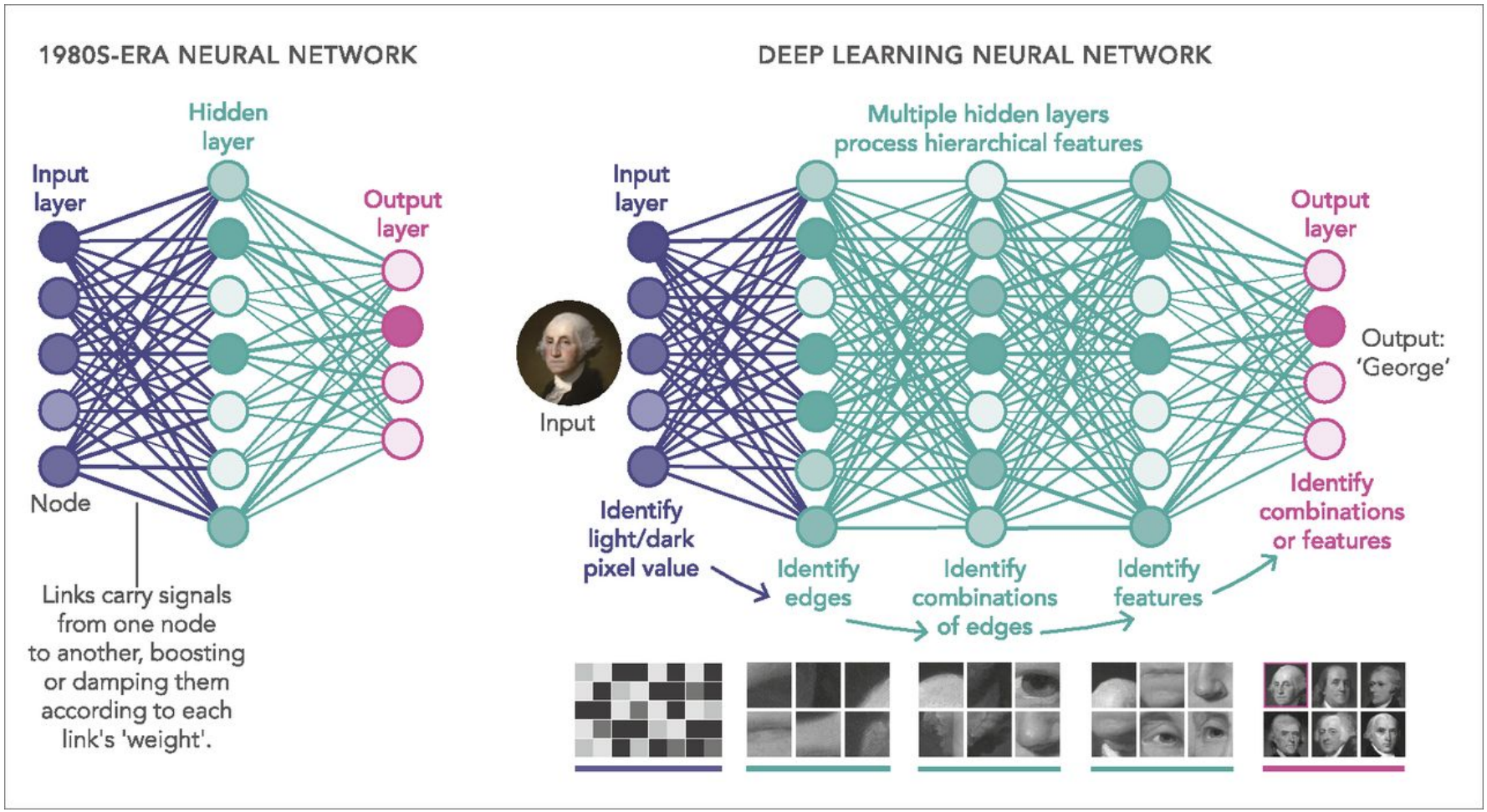
[2] Michael Nielsen, *Neural Networks and Deep Learning*, http://neuralnetworksanddeeplearning.com/

[3] Denny Britz, *Deep Learning Glossary*, http://www.wildml.com/deep-learning-glossary/

Głębokie uczenie - metody uczenia maszynowego, które

- zbudowane są z **wielu warstw realizujących nieliniowe transformacje**
- tworzą **model hierarchiczny** - warstwy odwzorowują kolejne poziomy abstrakcji, najniższe warstwy tworzą reprezentuję najprostszych cech z (surowego) sygnału wejściowego, wyższe warstwy - generują bardziej ogólne koncepty bazując na relacjach z poprzednich warstw.

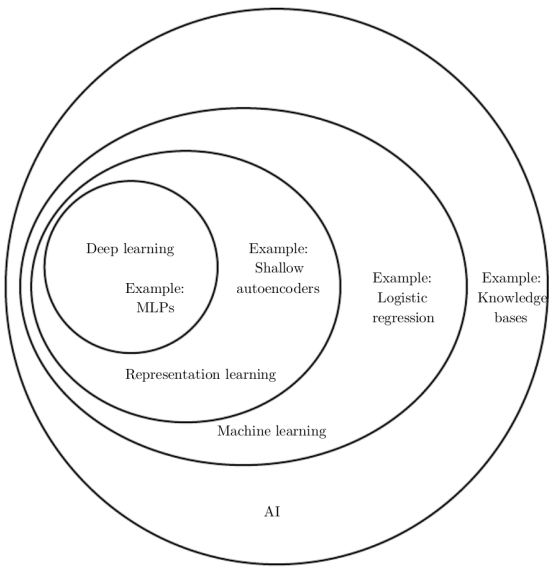
Hierarchia reprezentacji



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.



Gooffellow, 2016 [1]

Głębokie vs. płytkie uczenie

Krótka (nieformalna) definicja:
głębokie uczenie, gdy model ma więcej niż 2 nieliniowości

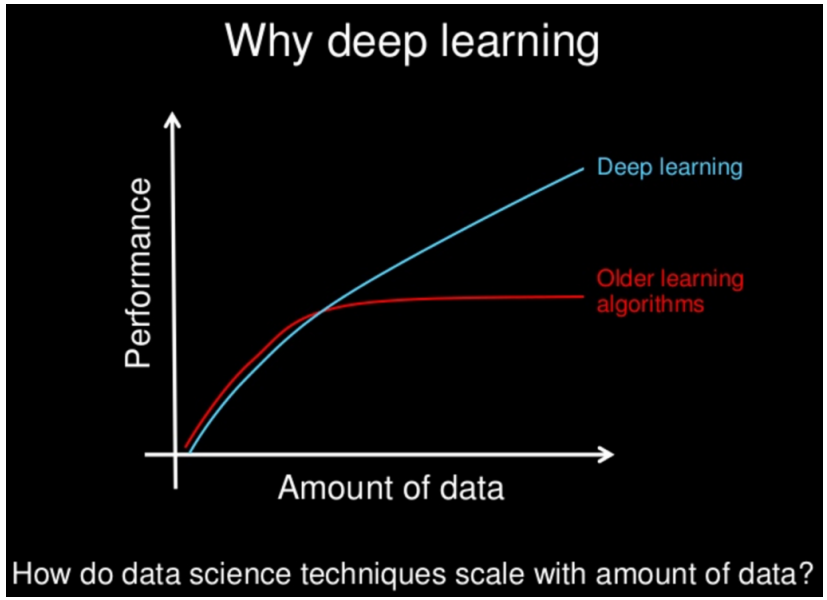
Płytkie modele

- 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

Skalowalność wyników



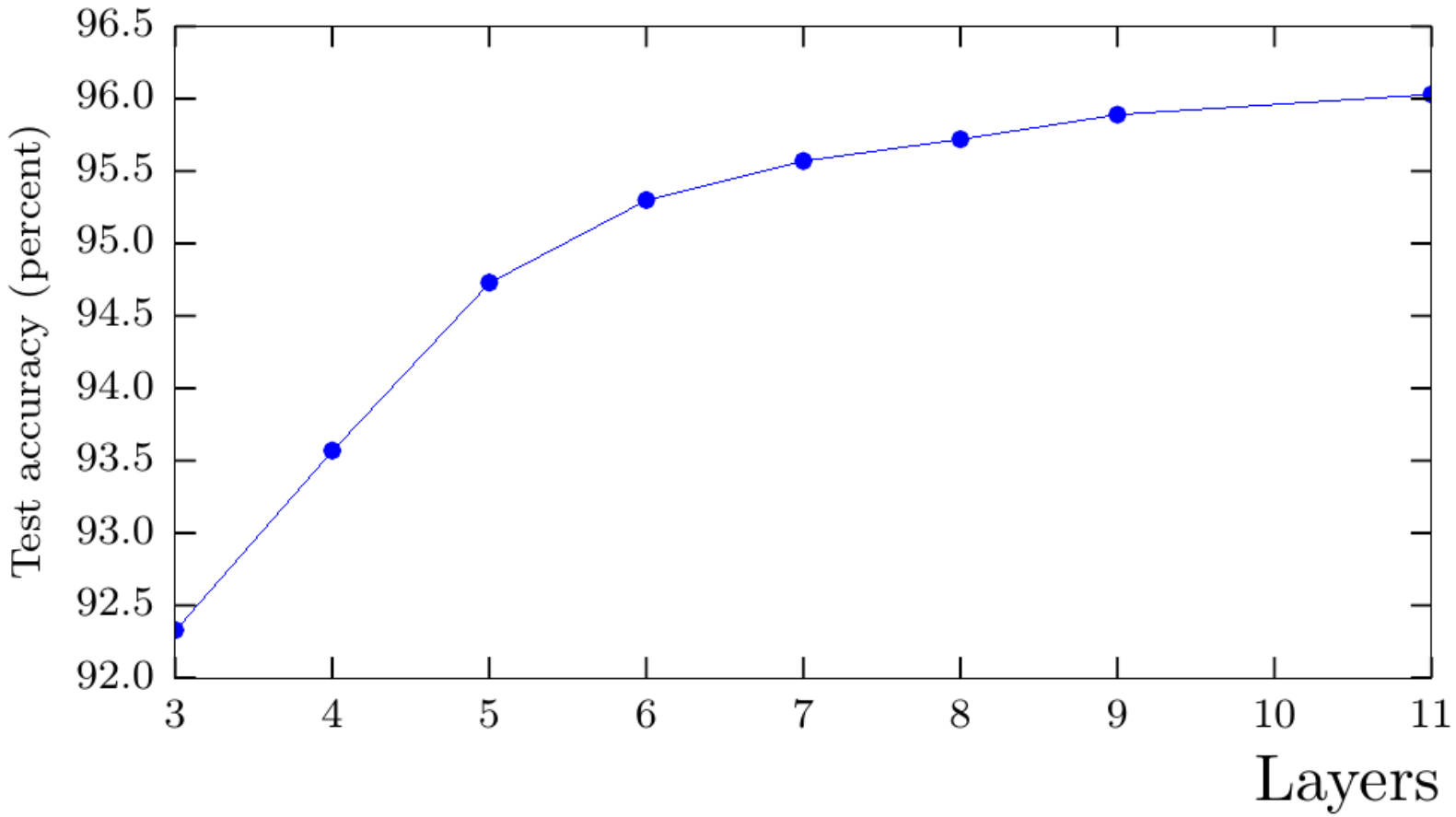
- duża ilość danych (z etykietami)
- duża moc obliczeniowa
- duże modele

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

Po co nurkować w głąb?

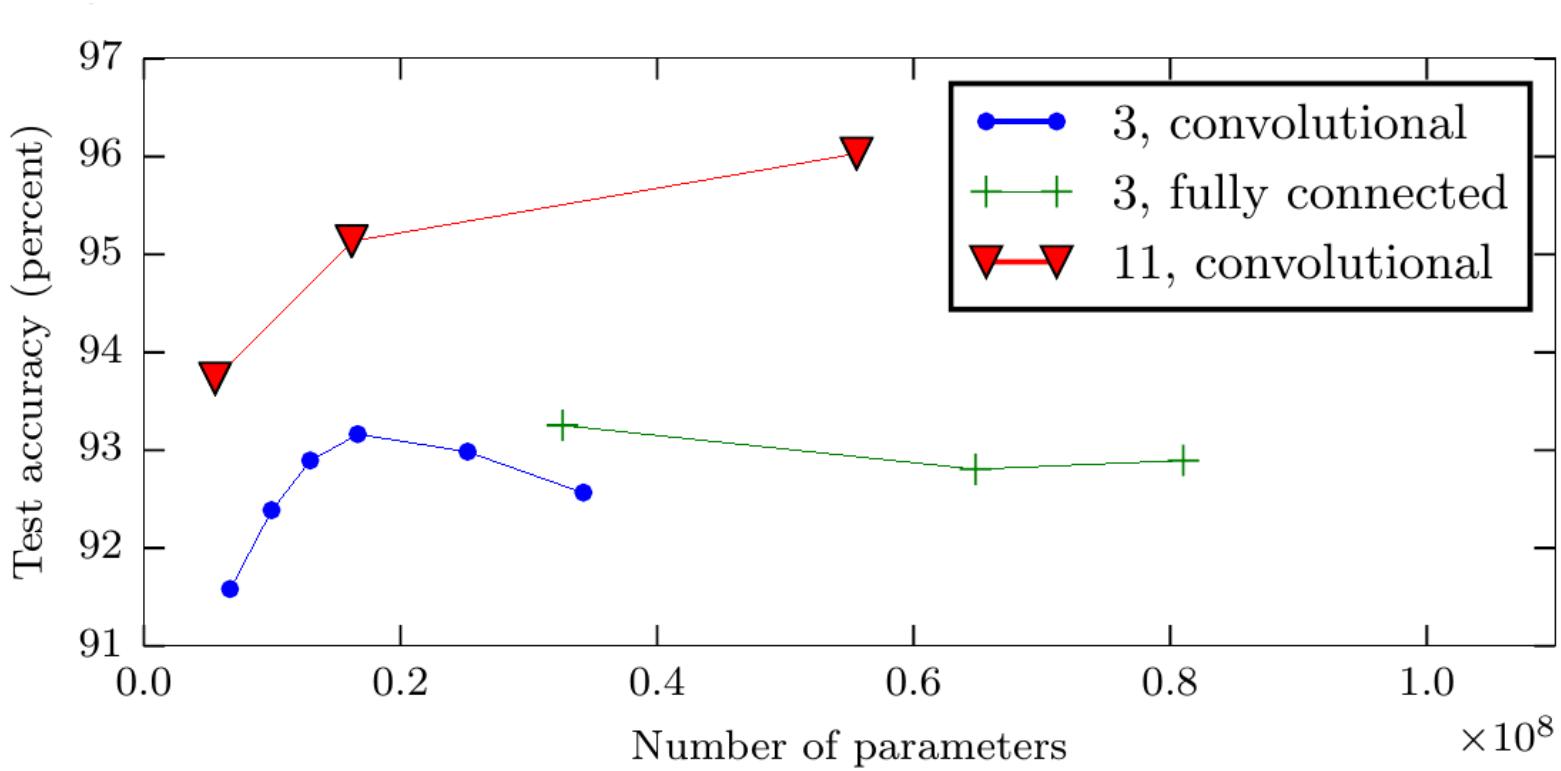
- **Uniwersalny aproksymator:** sieć MLP z jedną warstwą ukrytą jest w stanie aproksymować dowolną funkcję z dowolną dokładnością
- Deep learning też nie taki prosty, problemy nie tylko z treningiem ale z analizą działania
- Po co więc tworzyć głębsze sieci?
- „Płytkie" modele wymagają o wiele (wykładniczo) więcej neuronów
- „Płytkie" a szerokie modele łatwiej się przetrenowują, nie są w stanie odszukać lepszego rozwiązania
- Złożone problemy wymagają złożonych modeli

Głębokość a poprawność klasyfikacji

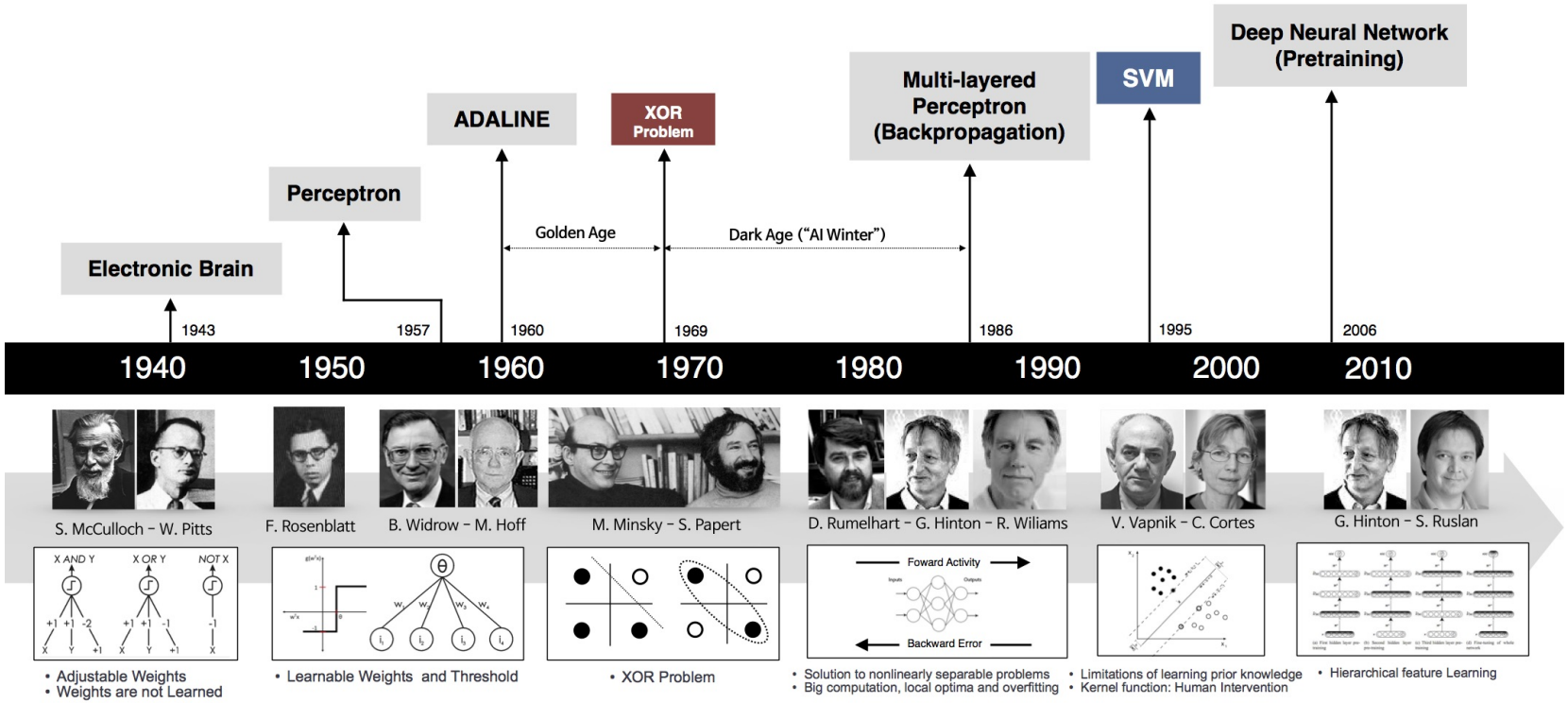


Goodfellow,et al. (2014), dokładność rośnie ze wzrostem głębokości

Głębokość ma znaczenie



Goodfellow,et al. (2014), głębokość może mieć większe znaczenie od liczby parametrów modelu

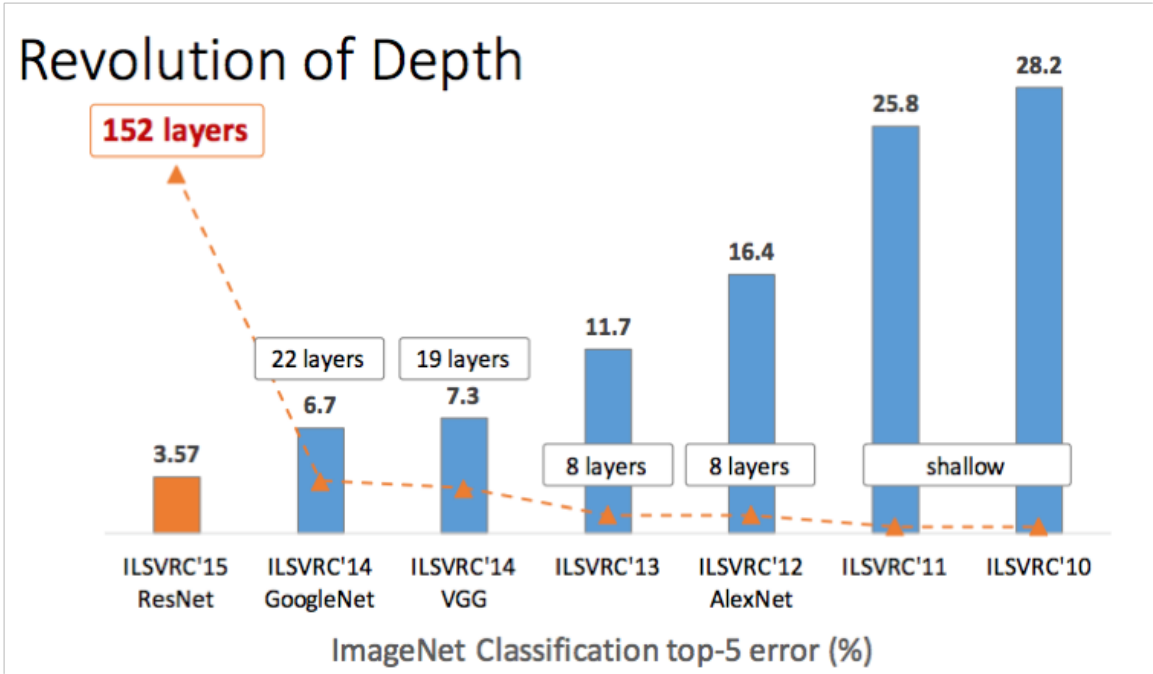


Geoffrey E. Hinton, et.al. „A fast learning algorithm for deep belief nets", 2006
R. Salakhutdinov, G. Hinton „Deep boltzmann machines", 2009

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

State of art results

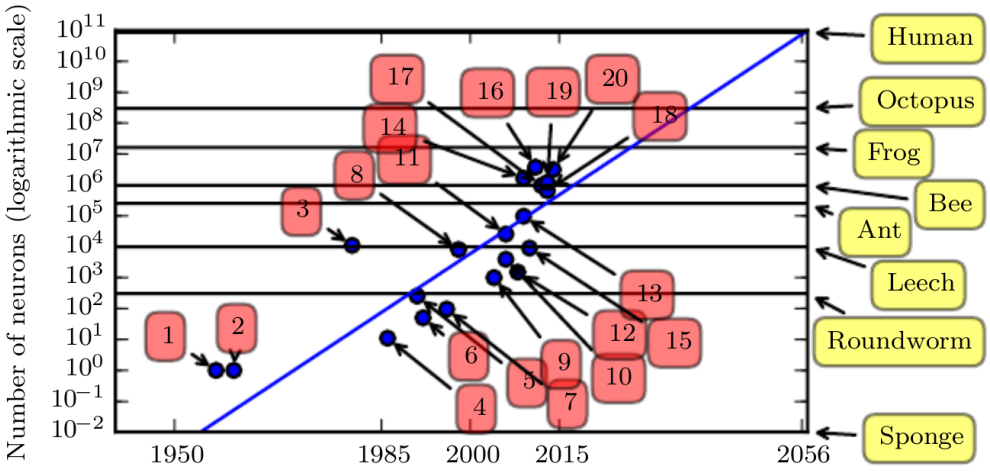
☞ ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



- klasyfikacja obrazów ☞ Are we there yet?
- rozpoznawanie mowy ☞ WER are we?

Rys: ☞ *The revolution of deepth, Armando Vieira*

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 Fallside, 1991
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)

Liczba neuronów podwaja się co 2.5 roku

Rys: Gooffellow, 2016 [1]

Wszędzie AI

- rozpoznawanie obiektów (twarzy, emocji), etykietowanie i lokalizacja obiektów
- automatyczne tłumaczenie, analiza tekstu, mowy, chatboty
- przewidywanie wyników wyborów, trzęsień ziemi, ...
- sterowanie robotami, samochodami, gry komputerowe (AlphaGo, AlphaStar)
- kompresja sygnałów, rekonstrukcja odszumianie sygnałów, odszumianie, kolorowanie czarno-białych filmów, ...
- generowanie i synteza sygnałów: muzyki, obrazów, tekstu, pisma ręcznego, mowy, kodu komputerowego, ...
- AI tworzące AI (Google AutoML), AI rozmiawiające z AI (Google Duplex)
- ☞ 30 amazing applications of deep learning

DNN

Głębokie MLP

MLP przypomnienie

Typy neuronów

- liniowe, sigmoidalne, tanh
- funkcje sigmoidalne nasycają się, co utrudnia optymalizację
- ReLU, Maxout

Optymalizacja metodami spadku gradientu lub innymi:

- wsteczna propagacja
- SGD, trening z momentem, RPROP
- metody 2 rzędu: Newtona, Levenberg-Marquarda, gradientów sprzężonych
  wymagają obliczenia Hessianu lub Jakobianu
- Adam, AdaGrad, moment Nesterova

Metody regularyzacji:

- L_1, L_2, wczesne zatrzymanie
- dropout, batch normaliation, 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)

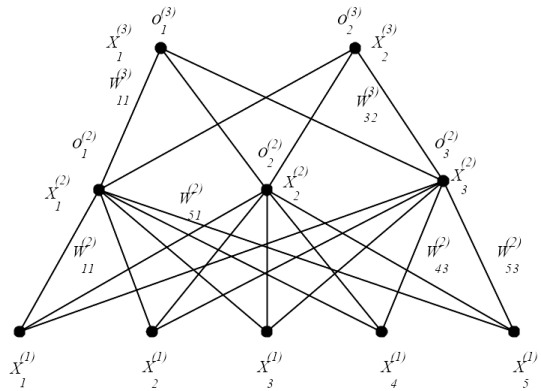
$$E = -\sum y \log f(\mathbf{x})$$

- klasyfikacja (wyjścia binarne)
- softmax lub sigmoidalne neurony wyjściowe

Wsteczna propagacja błędu

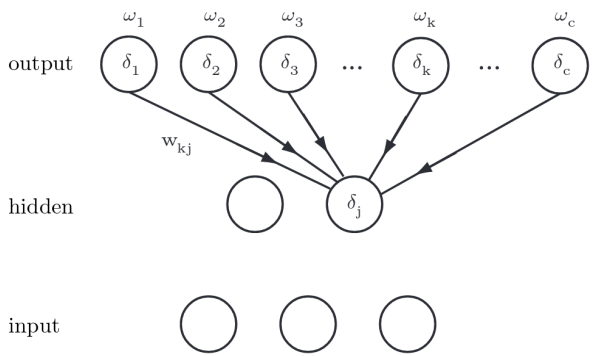
1. Zainicjuj wagi **W** małymi losowymi wartościami
2. Dopóki nie spłoniony warunek stopu wykonuj
   - oblicz sygnał od wejścia do wyjścia

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



   - oblicz sygnał błędu od warstwy wyjściowej do wejściowej

$$\delta_i^{(K)} = \sum_j \delta_j^{(K+1)} w_{ij}^{(K)}$$



   - aktualizuj wagi

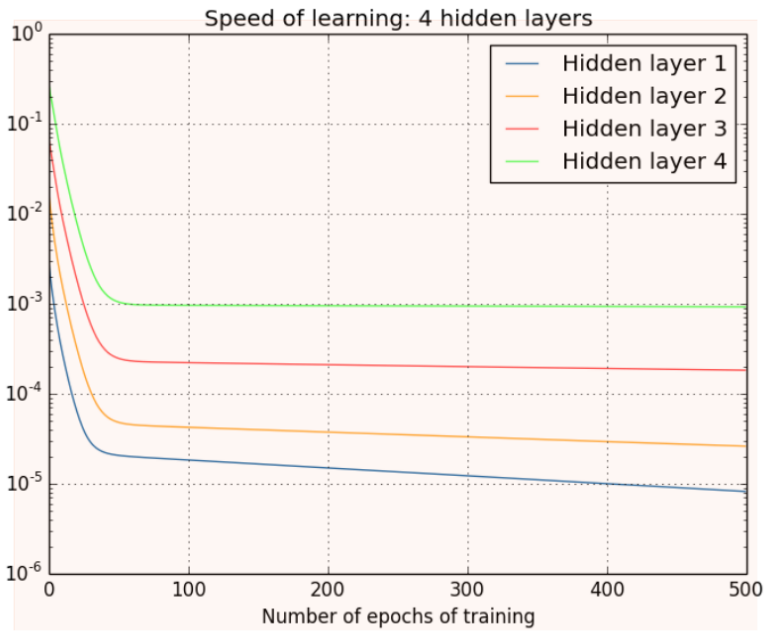
$$\Delta w_{ij}^{(K)} = \eta \delta_j^{(K)} o_i^{(K-1)}$$

Problemy związane z uczeniem (głębokich) sieci

- **zanikający gradient** - błąd propagowany od warstwy wyjściowej zanika przy niższych warstwach
- funkcje ograniczone (sigmoidalna, tangens hiperboliczny) „nasycają się" i posiadają niezerowy gradient tyko w wąskim przedziale aktywacji bliskim 0
- **przeuczenie** - większa liczba parametrów sprzyja przetrenowaniu, wymagane są techniki regularyzacyjne
- większa ilość parametrów + duże dane $\rightarrow$ długi czas uczenia i duże wymagania dotyczące pamięci

Zanikający gradient

Przykład:
MNIST data, MLP 784x30x30x30x30x10, regularyzacja $||\mathbf{w}||^2$



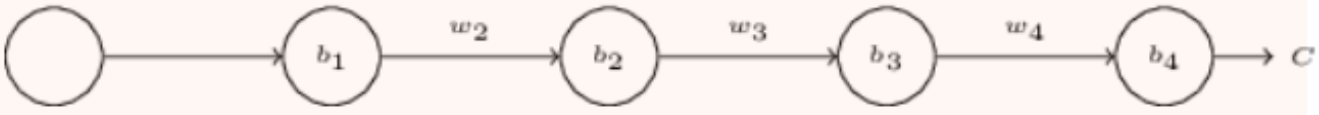
Szybkość uczenia $||\boldsymbol{\delta}||$ drastycznie maleje w głębszych warstwach

Nielsen, 2016 [2]

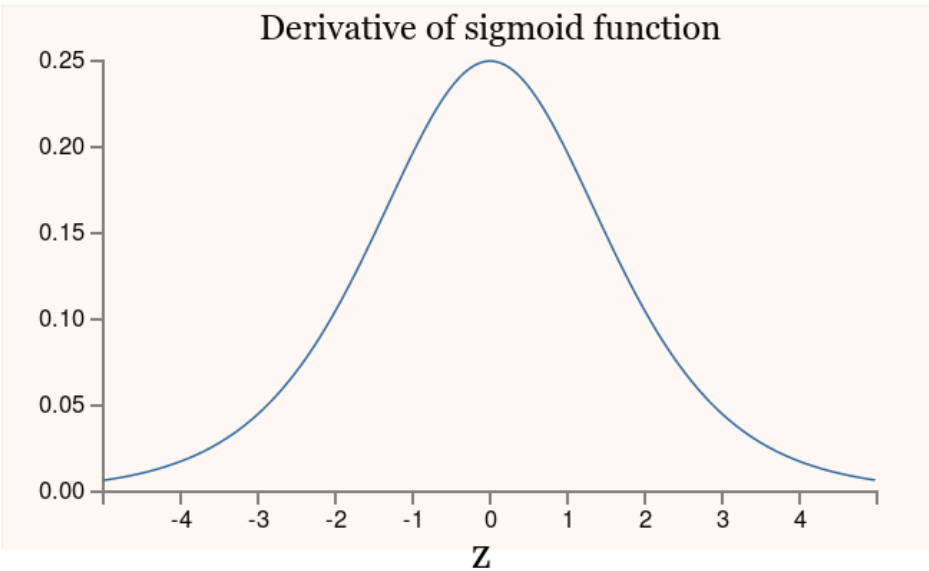
Znikający gradient

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



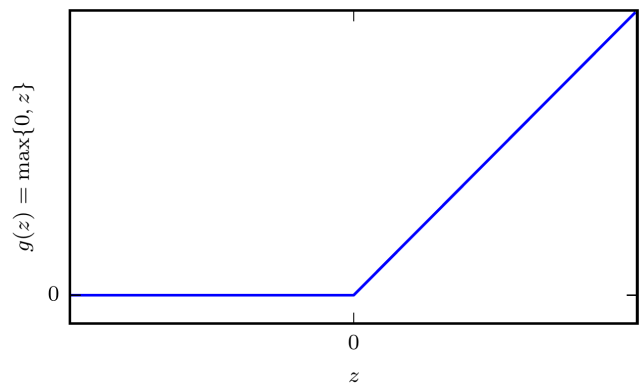
Nielsen, 2016 [2]

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 $|w_i \sigma'(z_i)| > 1$, 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

Rectified linear unit (ReLU)

$$g(z) = \max(0, z), \qquad g'(z) = \begin{cases} 1 & \text{dla} \quad z > 0 \\ 0 & \text{dla} \quad z < 0 \end{cases}$$



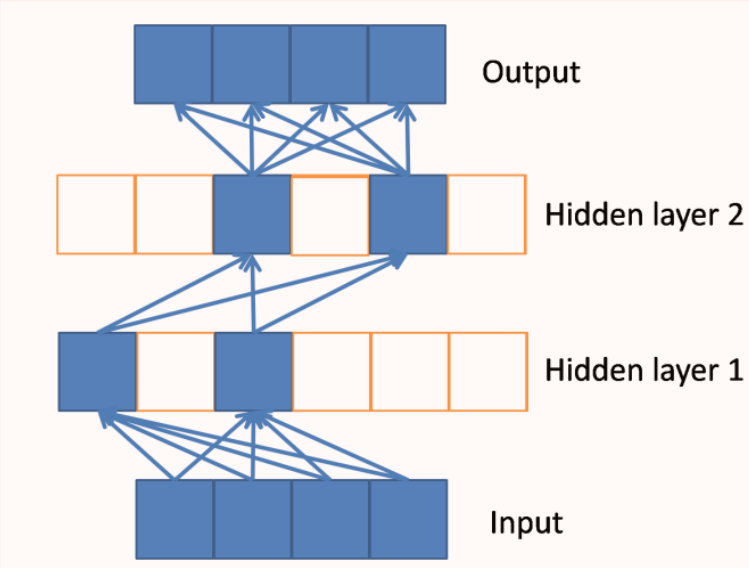
X. Glorot, A. Bordes and Y. Bengio (2011). Deep sparse rectifier neural networks

Korzyści ReLU

- gradient nie znika, gdy jednostka jest aktywna, ograniczenie problemu znikającego gradientu
- fragmentami liniowa, skuteczniejsza optymalizacja metodami gradientowymi
- efektywna obliczeniowo: mnożenie, dodawanie, warunek
- głębokie sieci ReLU nie wymagają pre-treningu (Galrot, 2011)
- rzadka reprezentacja (*sparse representation*)

X. Glorot, A. Bordes and Y. Bengio (2011). Deep sparse rectifier neural networks

Sparse representaion



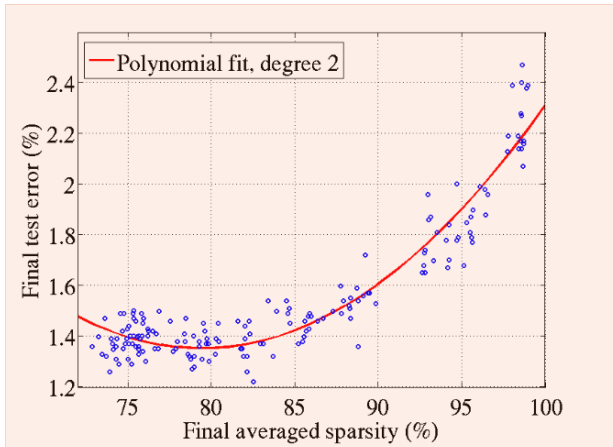


Figure 3: **Influence of final sparsity on accuracy.** 200 randomly initialized deep rectifier networks were trained on MNIST with various L_1 penalties (from 0 to 0.01) to obtain different sparsity levels. Results show that enforcing sparsity of the activation does not hurt final performance until around 85% of true zeros.

Rzadka reprezentacja dodaje nadmiarowość do modelu, zmiana wywołana przez dany wektor treningowy wprowadza zmiany tylko w podzbiorze neuronów

ReLU networks results

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%

Garlot, Deep Sparse Rectifier Neural Networks, 2011

Problemy ReLU

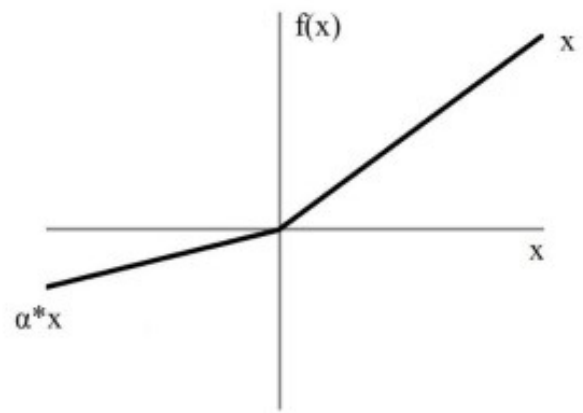
- nie jest różniczkowalne dla $z = 0$
- nie jest symetryczna i wycentrowana w 0
- nieograniczona, może powodować nieograniczony wzrost wartości aktywacji
- **problem umierających ReLU** - gdy neurony osiągają stan permanentnego braku aktywacji $g(z) = 0$ i ich wagi nie są modyfikowane w trakcie uczenia
- druga pochodna równa 0

Uogólnienia ReLU

- **Absolute ReLU** (Jarret, 2009), $g(z) = |z|$ skuteczna w szczególnych zastosowaniach dla obrazów z symetrią
- **Leaky ReLU** (Maas, 2013), wprowadza małą ($\alpha = 0.01$) wartość aktywacji dla $z < 0$

$$g(z) = \begin{cases} z & \text{dla} \quad z > 0 \\ \alpha z & \text{dla} \quad z < 0 \end{cases}$$

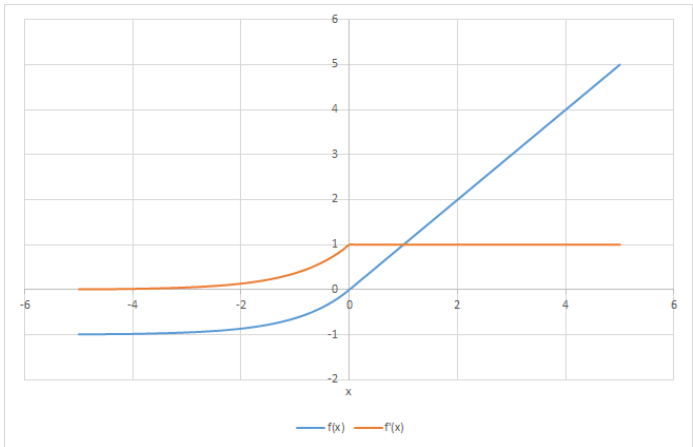
- **Parametric ReLU** α podlega optymalizacji w trakcie uczenia



Modyfikacje ReLU

- **Noisy ReLU** dodaje niewielki szum z rozkładu normalnego
- **Exponential linear unit (ELU)** (Clevert, 2015), średnia aktywacja bliższa 0, szybsza zbieżność w stosunku do ReLU

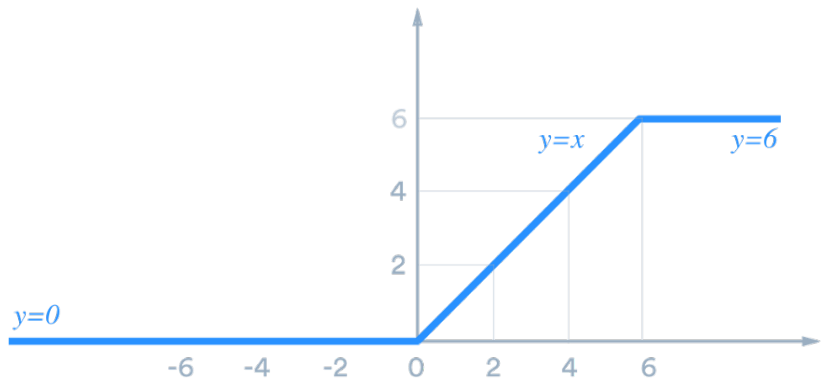
$$g(z) = \begin{cases} z & \text{dla} \quad z > 0 \\ \alpha(e^z - 1) & \text{dla} \quad z < 0 \end{cases} \qquad g'(z) = \begin{cases} 1 & z > 0 \\ \alpha e^z, & z < 0 \end{cases}$$



Rys: ☞ ELU as a Neural Networks Activation Function by Sefik

ReLU6

$$g(z) = \begin{cases} 6 & \text{dla} \quad z > 6 \\ z & \text{dla} \quad 0 < z < 6 \\ 0 & \text{dla} \quad z < 0 \end{cases}$$

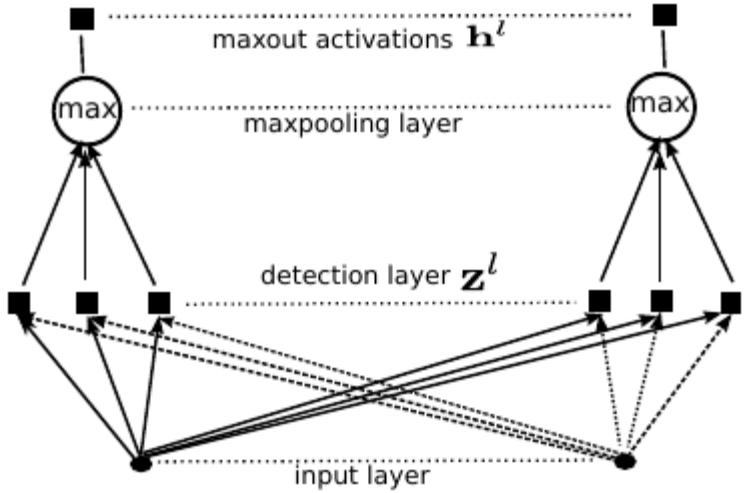


Górne ograniczenie przyspiesza tworzenie rzadkiej reprezentacji, dobre wyniki na CIFAR-10 (Krizhevsky)

Rys: ☞ A Practical Guide to ReLU by Danqing Liu

Maxout

$$g_i(\mathbf{z}) = \max_{j \in G_i} z_j$$



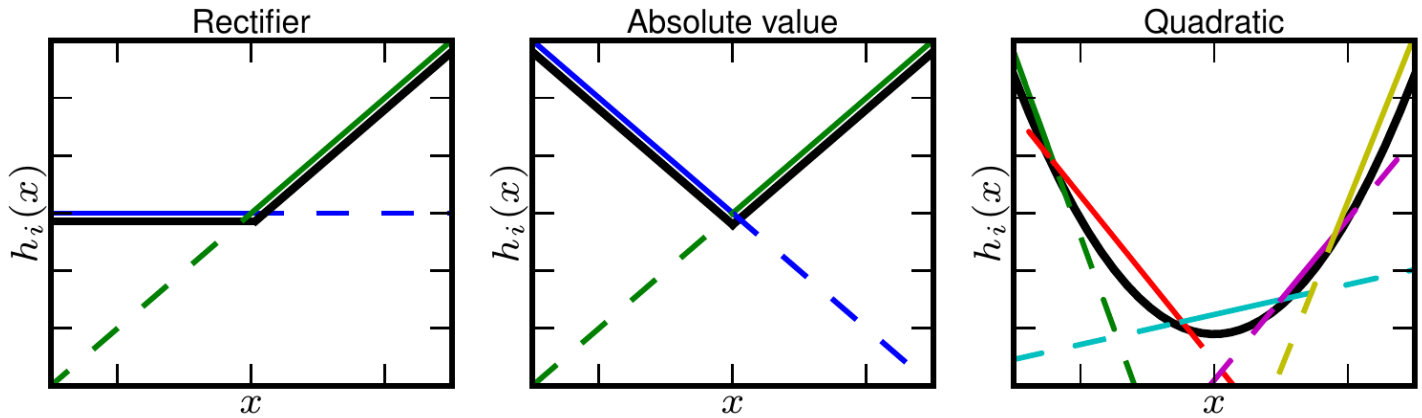
Zwraca wyjście najaktywniejszego neuronu z grupy k neuronów w warstwie, gdzie G_i to zbiór indeksów i-tej grupy

$$g'(z_j) = \begin{cases} 1 & \text{dla} \quad j = \arg\max_i z_i \\ 0 & \text{dla} \quad j \neq \arg\max_i z_i \end{cases}$$

Rys:Pawel Swietojanski, Investigation of maxout networks for speech recognition.
Goodfellow, et al, (2013). "Maxout Networks". JMLR WCP. 28 (3): 1319–1327.

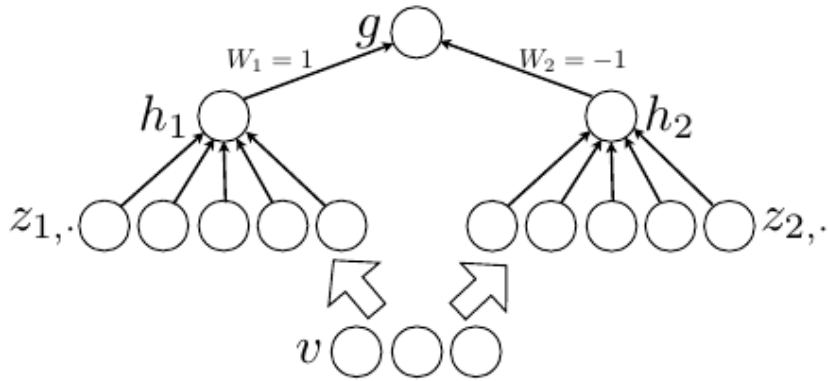
Maxout

- realizuje funkcję wypukła fragmentami liniowa złożoną z k fragmentów, kształt funkcji jest adaptowany w treningu



- ReLU i PReLU to szczególne przypadki Maxout
- fragmentami liniowa
- utrata rzadkiej reprezentacji (aktywacje nie tworzą żądkiej reprezentacji) - jednak gradient jest bardzo rzadki

Sieć Maxout



- złożenie funkcji wypukłych pozwaala zamodelowac dowolna funkcję ciągłą
- sieć z nauronami maxout jest uniwersalnym aproksymatorem
- aktywacja zależy od grypy wag, więc wprowadza do modelu redundancję, która pozwala niwelować fenomen „katastrofalnego zapominania”, gdy sieć zapomina wyuczone wcześniej wzorce

Rys: Goodfellow, et al, (2013). "Maxout Networks". JMLR WCP. 28 (3): 1319–1327.

Maxout netowrk on MNIST

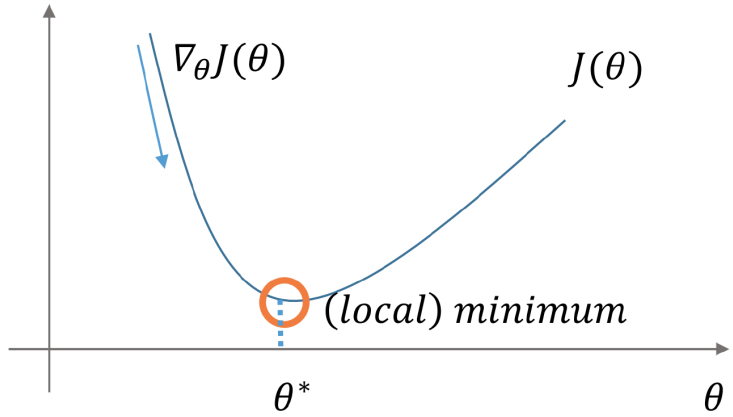
Method	Test error
Rectifier MLP + dropout (Srivastava, 2013)	1.05%
DBM (Salakhutdinov & Hinton, 2009)	0.95%
Maxout MLP + dropout	**0.94%**
MP-DBM (Goodfellow et al., 2013)	0.91%
Deep Convex Network (Yu & Deng, 2011)	0.83%
Manifold Tangent Classifier (Rifai et al., 2011)	0.81%
DBM + dropout (Hinton et 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 błedu na podstaie paczki zawierającej n losowych przypadków

Jak dobrać stałą uczenia η?

Rys: S. Ruder, „Optimising for Deep Learning"

SGD z momentem

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

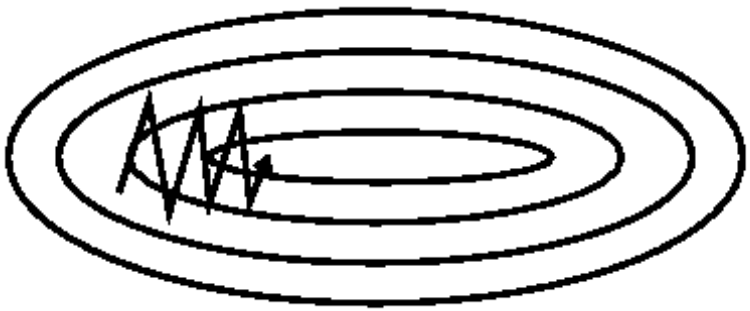
prędkość uczenia $v(t) = \Delta w(t)$ jest zależna od poprzednich wartości gradientów

$$v(t) = \gamma v(t-1) - \eta \nabla E(t)$$

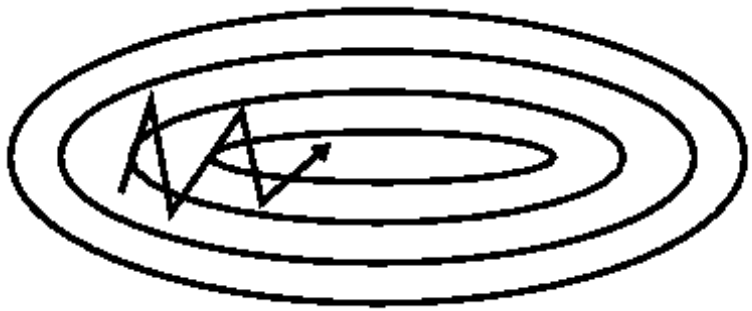
$$w(t+1) = w(t) + v(t)$$

- 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 $\gamma = 0.9$

SGD z momentem



SGD bez momentu



SGD z momentem

Nesterov accelerated gradient (NAG)

$$v(t) = \gamma v(t-1) - \eta \nabla E\left(w(t) + \gamma v(t-1)\right)$$

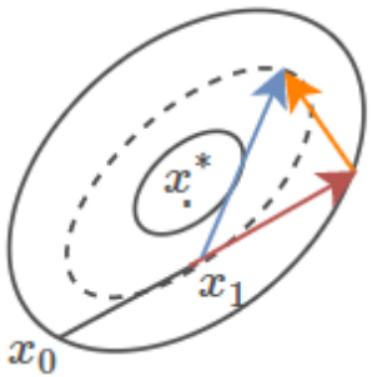
$$w(t+1) = w(t) + v(t)$$

- gradient liczony dla przybliżonej przyszłej pozycji $w(t) + \gamma v(t-1)$
- 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.

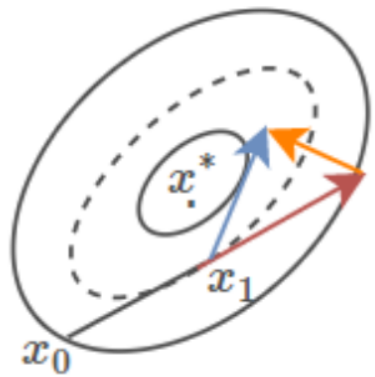
Polyak vs Nesterov

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 kwadraty gradientów

$$v(t) = v(t-1) + [\nabla E(t)]^2 = \sum_{t' < t} [\nabla E(t')]^2$$

- **efektywna stała uczenia** ustalana dla każdego optymalizowanego parametru, zależna od wszystkich poprzednich wartości gradientów
- domyślna wartość $\eta = 0.01$, zazwyczaj nie wymaga zmiany
- wada: mianownik może szybko przybrać duże wartości, co powoduje zanik uczenia (problem ten niwelują algorytmy Adadelta, RMSProp, Adam)

Duchi, J., Hazan, E., Singer, Y. (2011) „Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", JMLR

RMSProp

RMSProp (Hinton 2012) modyfikuje AdaGrad tak aby działał lepiejdla funkcji niewypukłych

$$w(t+1) = w(t) - \frac{\eta}{\sqrt{v(t)}} \nabla E(t)$$

gdzie $v(t)$ jest wykładniczo ważoną ruchomą średnią gradientów

$$v(t) = \beta v(t-1) + (1-\beta) \left[\nabla E(t)\right]^2$$

- największy wpływ na wielkość $v(t)$ mają ostatnie iteracje
- akumulowanie wszystkich gradientów w AdaGrad może spowodować zbyt duże zmniejszenie szybkości uczenia, przed osiągnięcicem wypukłej niecki
- β kontroluje skalę długości ruchomej średniej
- dore wyniki w połaczeniu z momentem Nesterova

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

$$m(t) = \beta_1 m(t-1) + (1-\beta_1)\nabla E(t)$$

$$v(t) = \beta_2 v(t-1) + (1-\beta_2)[\nabla E(t)]^2$$

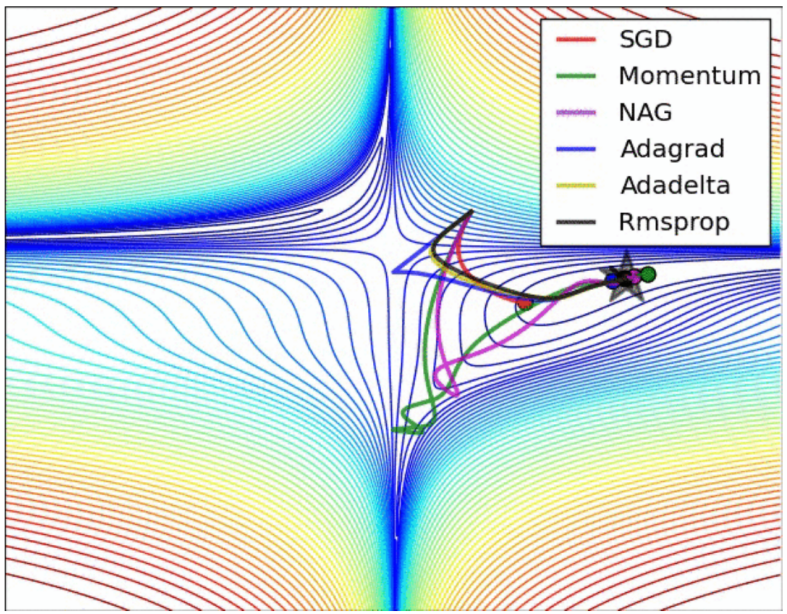
Poprawka na odchylenie w stronę wartości 0 (z powodu zerowych początkowych wartości estymaty)

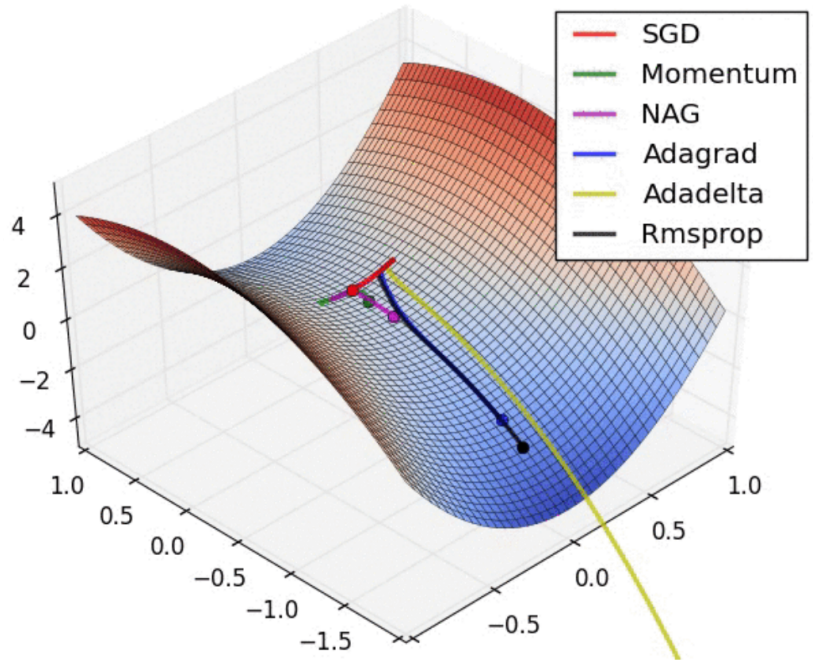
$$\hat{m}(t) = \frac{m(t)}{1-\beta_1}, \qquad \hat{v}(t) = \frac{v(t)}{1-\beta_2}$$

typowo $\beta_1 = 0.9$, $\beta_2 = 0.999$ współczynniki zanikania, $\eta = 0.01$

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 dla danych bez struktury (przestrzennej, czasowej)
- w większości zastosowań ReLU sprawdza się bardzo dobrze
- Adam w wielu przypadkach lepszy od innych algorytmów z domyślnymi parametrami η, β_1, β_2
- SGD z zanikającą wykładniczo wartością stałej uczenia może dorównać algorytmom takim jak Adam

CNN
Convolutional Neural Networks

CNN

- **Convolutional Neural Networks (CNN, ConvNet)** wariant MLP inspirowany biologicznie, gdzie mnożenie macierzy wag i sygnału wejściowego zastąpione jest operacją splotu
- rzadka reprezentacja, współdzielone wagi, pooling
- zdolne do generalizacji sygnału posiadającego relacje przestrzenne, odporne na przestrzenne transformacje sygnału (skalowanie, obrót, przesunięcie):
  - 1D sygnały czasowe
  - 2D obrazy
  - 3D fMRI, video, obrazy RGB
  - sygnały wielokanałowe

Splot

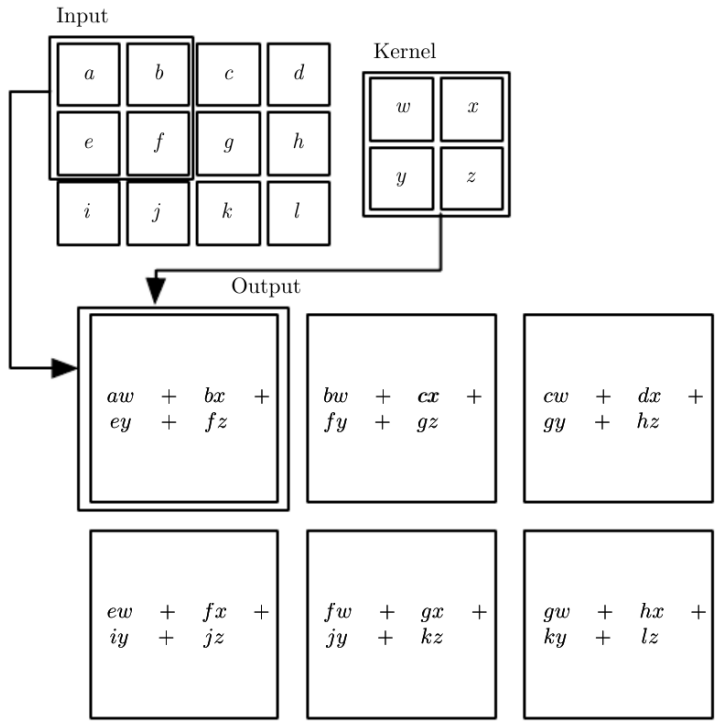
$$(x \star w)(t) = \int x(a)w(t-a)da$$

- wynikiem jest funkcja, np. uśredniona wartość $x(x)$ względem wszystkich pozycji $w(a)$ jeśli$w$ spełnia wymagania gęstości prawdopodobieństwa
- W terminologii CNN:
  x - sygnał wejściowy
  w - kernel (filtr), wagi połączeń neuronu
  wartości wyjściowe tworzą mapę cech (*feature map*)

$$s(i) = \sum_m x(m)w(i-m)$$

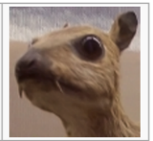
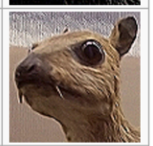
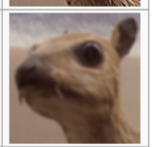
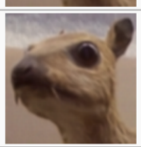
- w sieciach CNN w niezerowe tylko w organicznym obszarze (pole recepcyjne)

Splot 2D



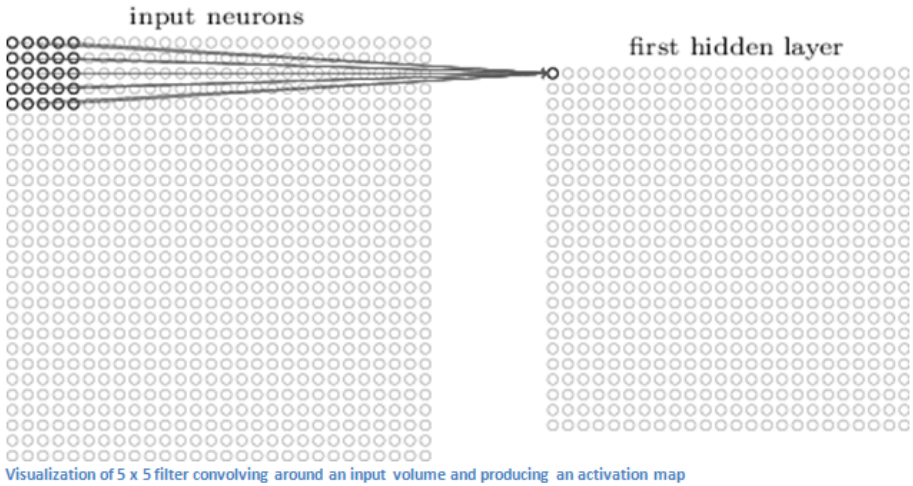
$$s(i,j) = \sum_l \sum_m w(l,m) \cdot x(i+l, j+m)$$

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



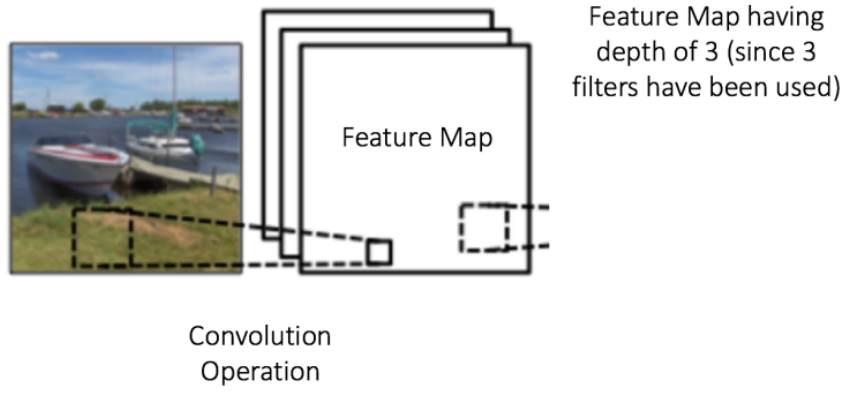
Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

- **Pole recepcyjne** - obszar „widziany” przez neuron (rozmiar filtra)
- wyjście neuronu: splot sygnału i liniowego filtra, ewentualny wyraz wolny (bias) i nieliniowa funkca wyjściowa (ReLU, tanh), generują **mapę cech** (*feature map)*

$$s(i,j) = \sigma\left(b + \sum_l \sum_m w(l,m) \cdot x(i+l, j+m)\right)$$

Pola recepcyjne i mapy cech

- Warstwa zawierająca N neuronów (filtrów) tworzy N map (objętość, tensor n wymiarowy)

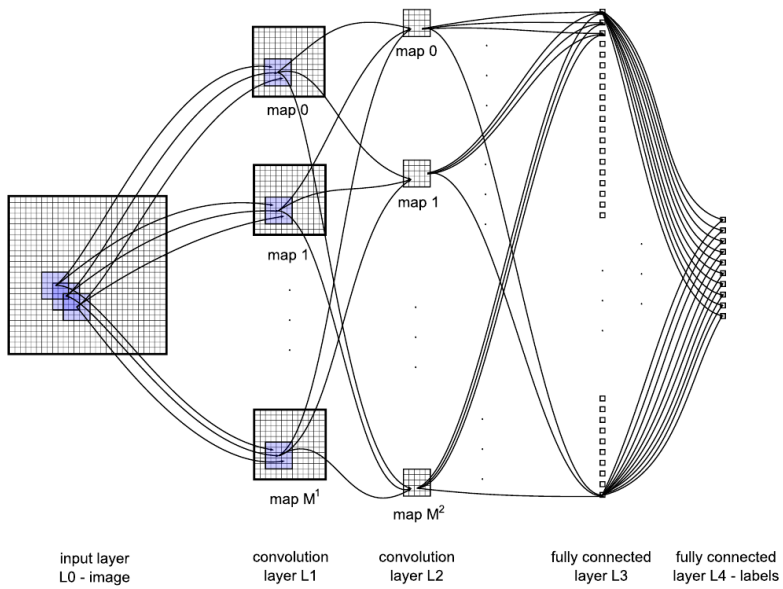


- Przy przetwarzaniu obrazów sygnał wejściowy to tensor 3D (kanał, szerokość, wysokość), w zastosowaniu rozszerzony do 4D o wymiar związany z rozmiarem mini-batcha

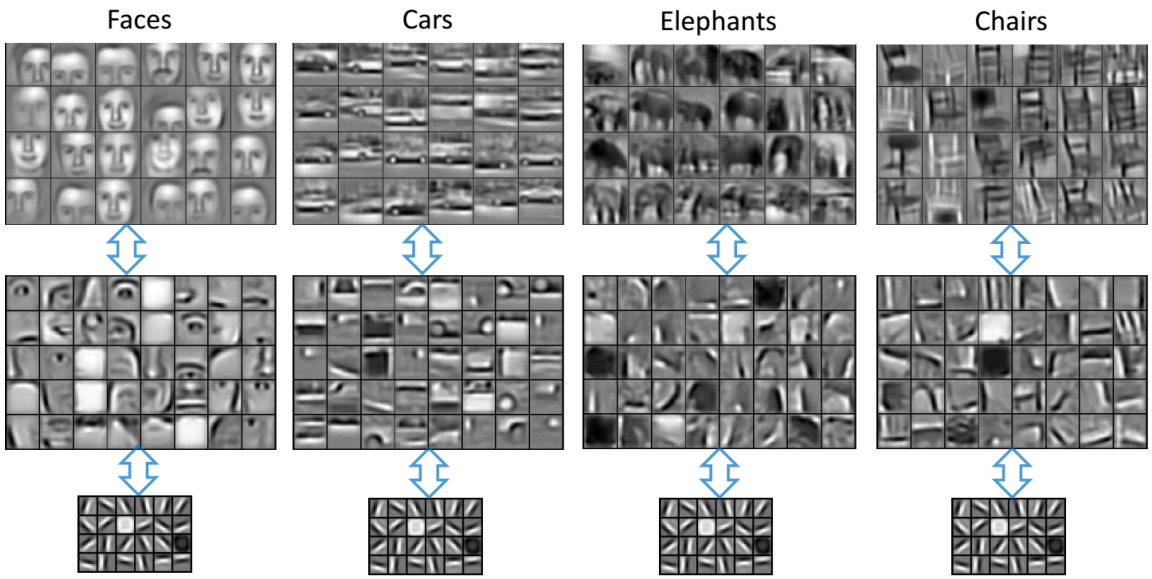
$$s(i,j,k) = \sum_{lmn} x(l, j+m, k+n) w(i,l,m,n)$$

- tensor wyjściowy warstwy zawierającej N filtrów (N, M_x, M_y), gdzie M_x, M_y - rozmiary mapy wyjściowej

Typowa architektura CNN



Przykłady filtrów



Gooffellow, 2016 [1]

Rozmiar mapy cech

- Warstwa zawierająca N neuronów (filtrów) tworzy N map (objętość, tensor n wymiarowy)
- Rozmiar mapy wyjściowej (M_x, M_y) zależy od:

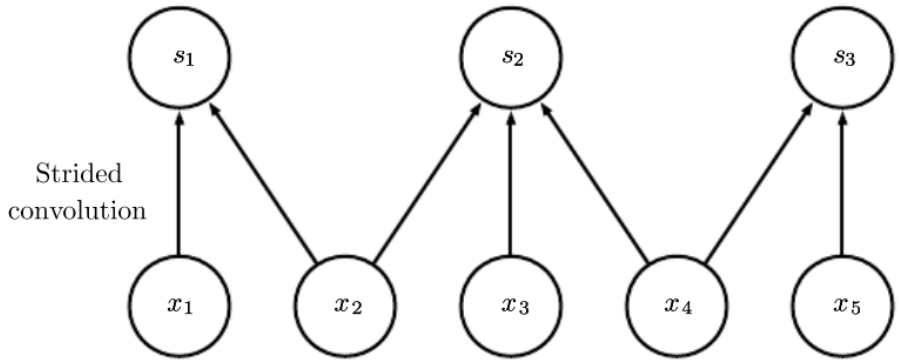
$$M_i^n = \frac{M_i^{n-1} - K_i^n + 2P_i^n}{S_i^n} + 1$$

  - rozmiar filtra (K_x, k_y) (szerokość i długość)
  - przesunięcie (*stride*) (S_x, S_y) w każdym z wymiarów, np. splot 2D

$$s(i,j,k) = \sum_{lmn} x(l, j \times S + m, k \times S + n) w(i,l,m,n)$$

  - sposób uwzględnienia wartości brzegowych (*zero padding*), P_i - wielkość rozszerzenia brzegu w wymiarze i

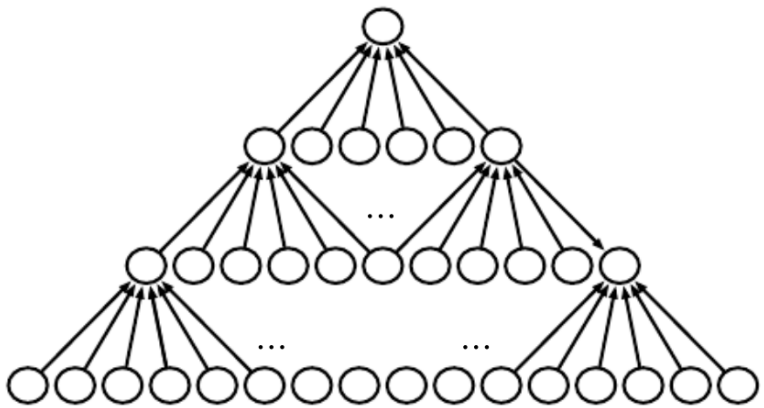
Stride



- redukcja wymiaru - zmniejszenie wymogów obliczeniowych i pamięciowych
- zmniejszenie rozdzielczości sygnału
- kosztem mniej dokładnej reprezentacji

Zero padding

valid zero padding - rozmiar kolejnych map maleje w kolejnych warstwach



- ogranicza to możliwość budowania głębokich sieci i wymusza stosowanie małych filtrów
- sygnał wejściowy na brzegach ma mniejszy wpływ na sygnał wyjściowy

Zero padding

same zero padding - brzegi wypełnione dodatkowymi wartościami (zerami) aby zapewniać odtworzenie wymiaru sygnału wejściowego

- pozwala budować bardzo głębokie sieci o dowolnych wielkościach filtrów $M_i^n = \lceil \frac{M_i^{n-1}}{S_i} \rceil$
- sygnał wejściowy na brzegach ma mniejszy wpływ na sygnał wyjściowy

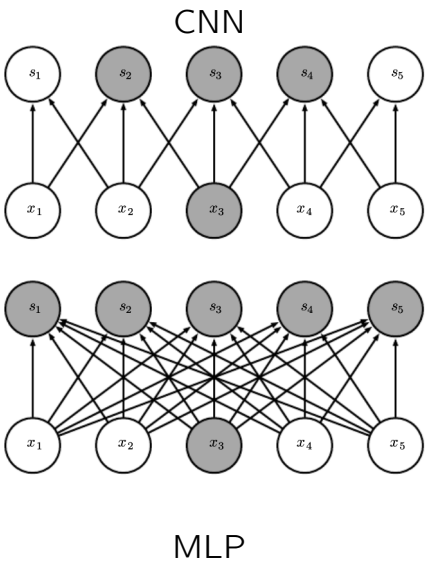
Właściwości CNN

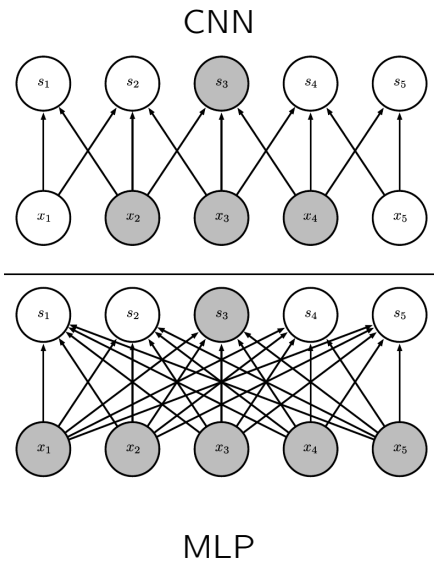
- **rzadka reprezentacja** - rozmiar filtra jest dużo mniejszy od rozmiaru sygnału wejściowego, pojedyncze wejście oddziałuje tylko na grupę neuronów
- **współdzielenie parametrów** - warstwa splotowa może być widziana jako w pełni połączona warstwa ze współdzielonymi wagami (dużo mniejsza liczba parametrów w stosunku do MLP)
- **równoważność względem przesunięcia** sygnału - ta sama cecha znajdująca się w różnych miejscach obrazu będzie aktywowała ten sam filtr
- możliwość użycia sygnału wejściowego o zmiennym 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

wyjście neuronu zależne od małego obszaru sygnału wejściowego





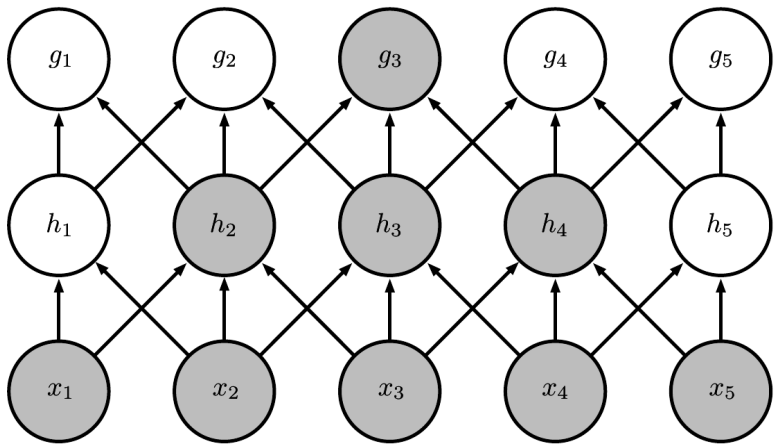
MLP: mnożenie macierzy $O(m \times n)$ parametrów

CNN: warstwa splotowa $O(k \times n)$, gdzie $k \ll m$

Źródło grafiki: Gooffellow, 2016 [1]

Efektywne pole recepcyjne

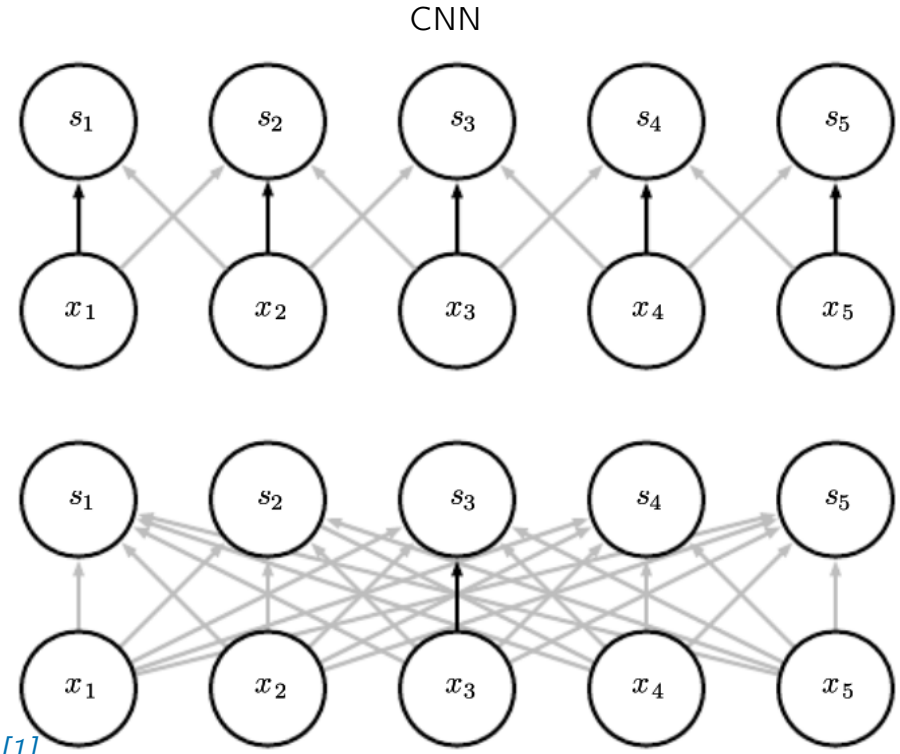
- **Efektywne pole recepcyjne** - obszar sygnału wejściowego pokryty przez neurony w wyższych warstwach, rośnie z głębokością



- stride, pooling i dilation (rozrzedzony splot) dodatkowo zwiększają efektywne pole recepcyjne
- pomimo rzadkich połączeń sieć jest w stanie w ten sposób modelować złożone zależności

Współdzielenie wag

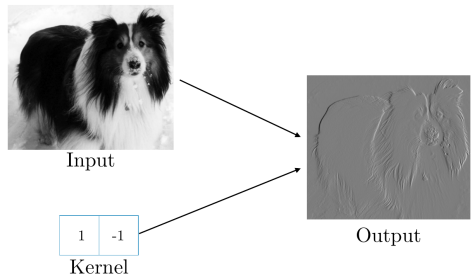
- ta sama waga jest używana przy przetwarzaniu każdego punktu wejściowego
- pojedynczy filtr pozwala wykryć tę samą cechę w różnych położeniach obrazu wejściowego (*equivariance to translation*)



Źródło grafiki: Gooffellow, 2016 [1]

Rzadkie połączenia i współdzielenie wag

Przykład: wykrywanie krawędzi dla obrazu $n \times n$ w poziomie



- MLP: mnożenie pełnej macierzy $n^2 \times n^2 = n^4$
- CNN: splot filtrem 1x2 (2 wagi) wymaga $n^2 \times 3$ operacji (2 mnożenia i dodawanie)
- splot drastycznie bardziej wydajny w mapowaniu powtarzających się relacji w małych, lokalnych rejonach sygnału wejściowego

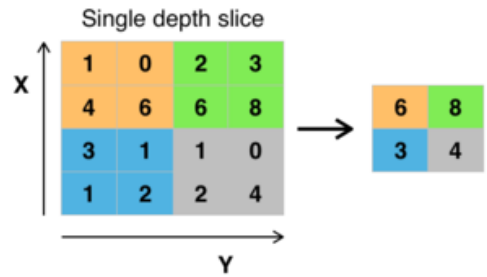
Źródło grafiki: Gooffellow, 2016 [1]

Równoważność przy przesunięciu

- przesunięcie sygnału wejściowego powoduje identyczne przesunięcie sygnału wyjściowego na mapie cech
- własność pożądana w rozpoznawaniu obrazów, gdzie lokalne cechy (np. krawędzie) mogą wystąpić w każdym miejscu na obrazie
- współdzielenie wag dla całego wejścia nie zawsze jest pożądane (różne filtry w różnych obszarach obrazu), np. rozpoznawanie twarzy ze zdjęć paszportowych, posiadają charakterystyczne cechy występujące wyłącznie w określonych rejonach sygnału wejściowego
- splot nie jest niezmienniczy względem zmiany skali lub obrotu obrazu, osiąga się to poprzez rozszerzenie zbioru treningowego o przypadki zdeformowane (szum, skalowanie, obrót, zmiana kontrastu, etc..)

Pooling

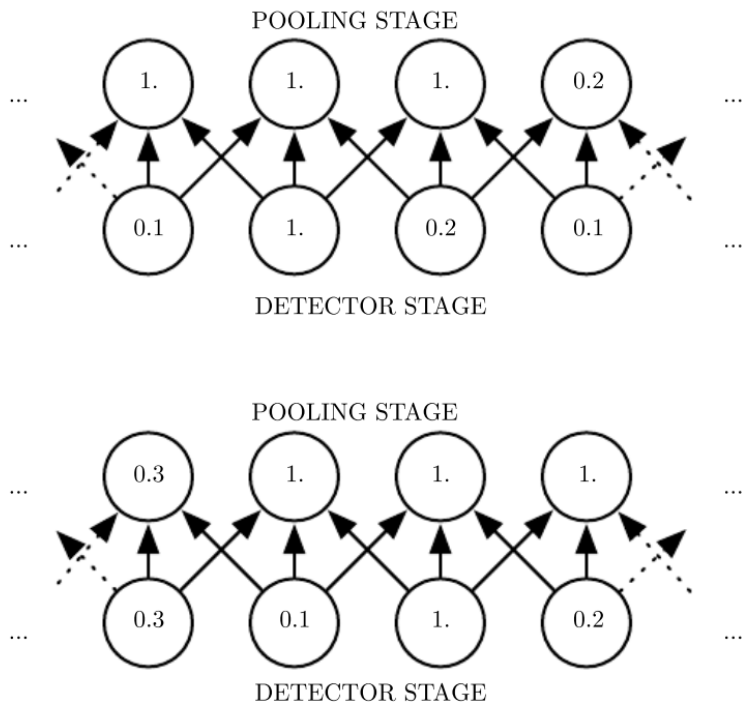
- typowa warstwa w sieciach CNN:
  - liniowa aktywacja (splot)
  - detekcja (nieliniowość np. ReLU)
  - funkcja redukcji (pooling) - „uogólnienie” wartości sąsiadujących wyjść
- **Max pooling** - maksimum z pewnego podobszaru (*winner takes all*)



- redukcja wymiarowości - przesunięcie filtra typowo równe jego wielkości (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

Pooling względem obrazu wejściowego

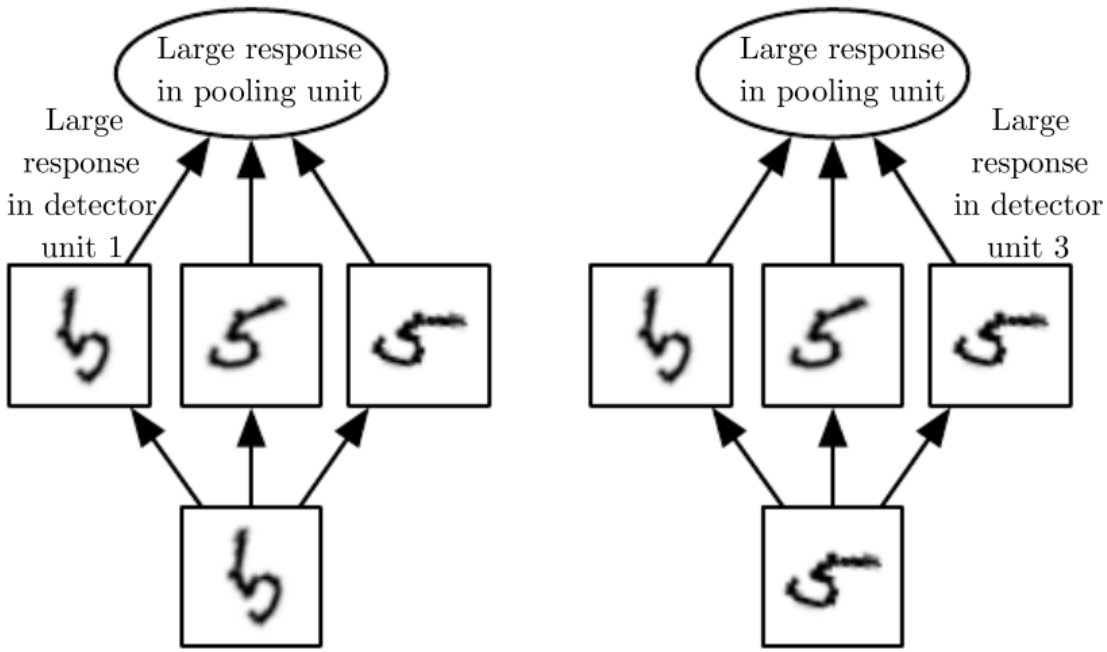
Pooling zapewnia niezmienniczość względem drobnego przesunięcia obrazu wejściowego



Źródło grafiki: Gooffellow, 2016 [1]

Pooling względem map cech

Redukcja względem wyjść różnych splotów umożliwia wprowadzenie do modelu niezmienniczości względem pewnych transformacji sygnału wejściowego (np. obrotu)



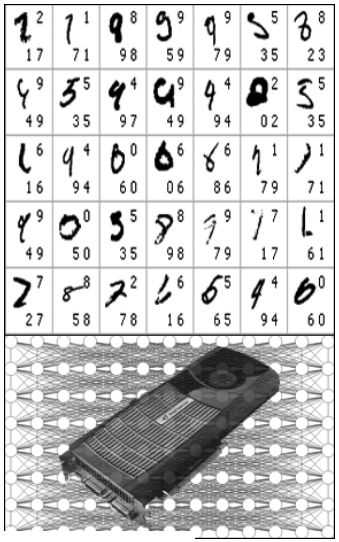
Źródło grafiki: Gooffellow, 2016 [1]

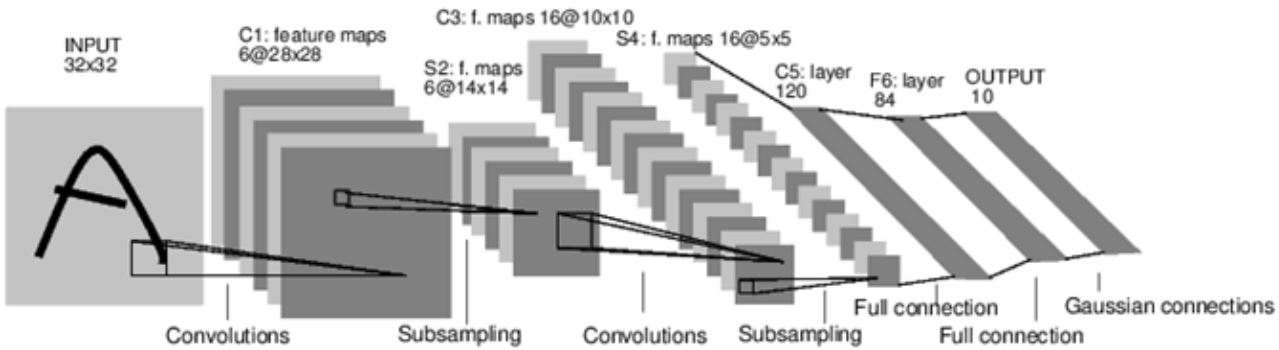
LeNet5 (LeCun 1998)

klasyfikacja cyfr pisanych ręcznie
(odczytywanie czeków)

☞ MNIST

trening: 60k cyfr, 250 osób,
test: 10k cyfr



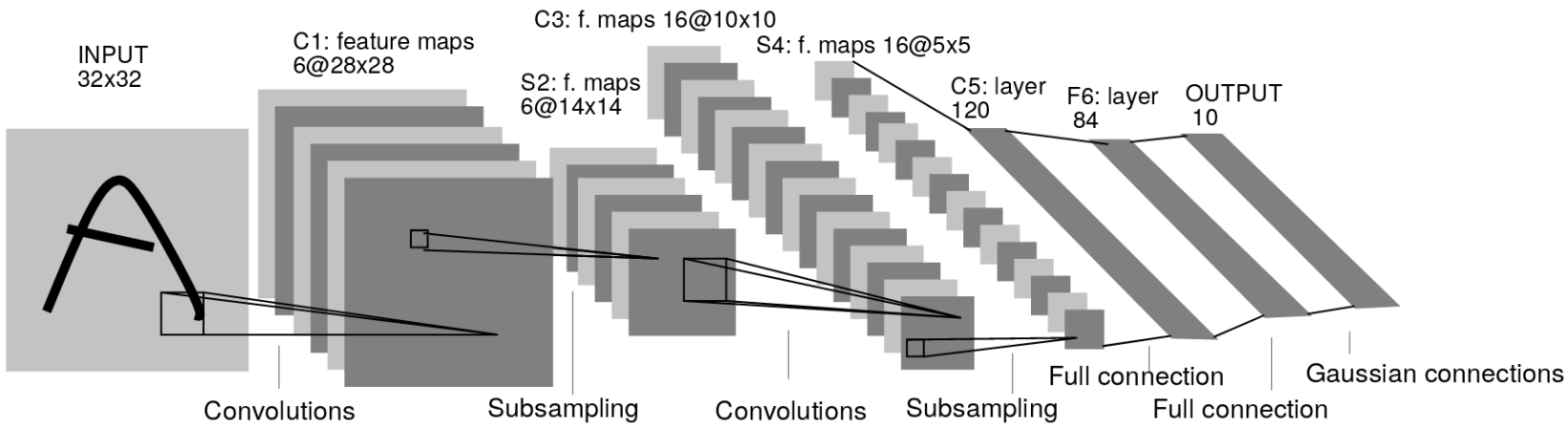


A Full Convolutional Neural Network (LeNet)

☞ 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
F6	84		84x1x1		
Output	10		10x1x1		

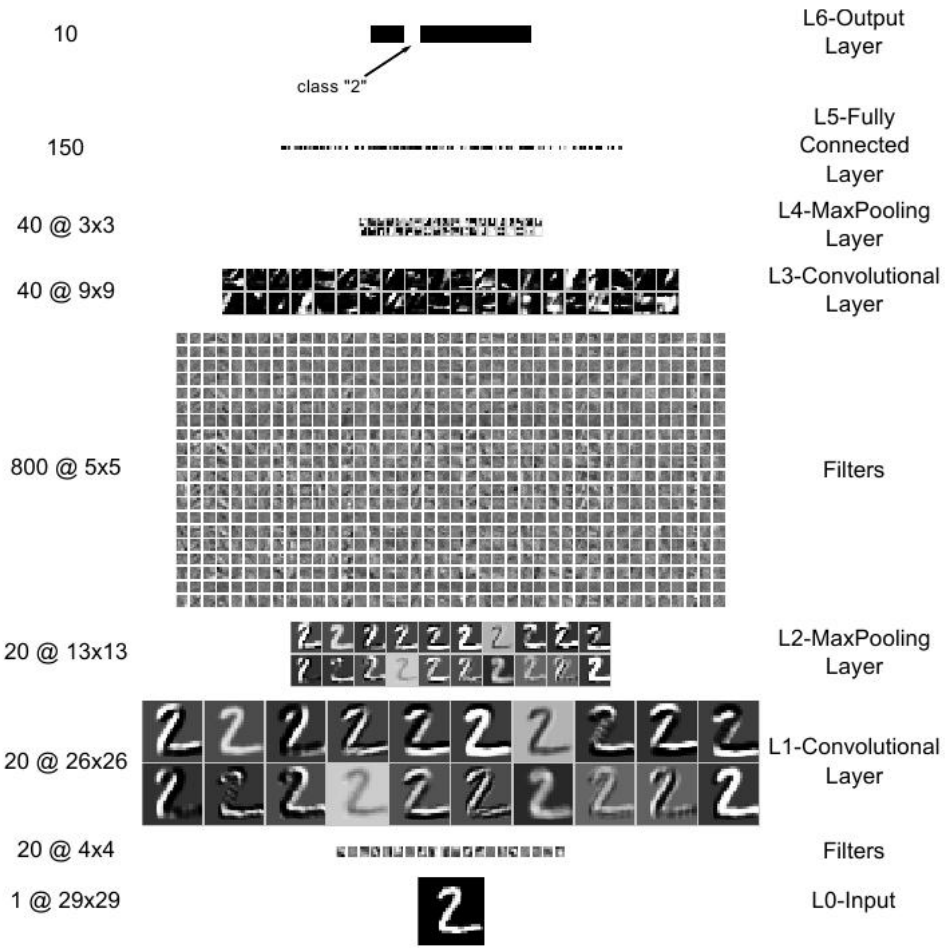
LeNet-5 połaczenia S2-C3

Mapy C3 zależą wyłącznie od wybranych map 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

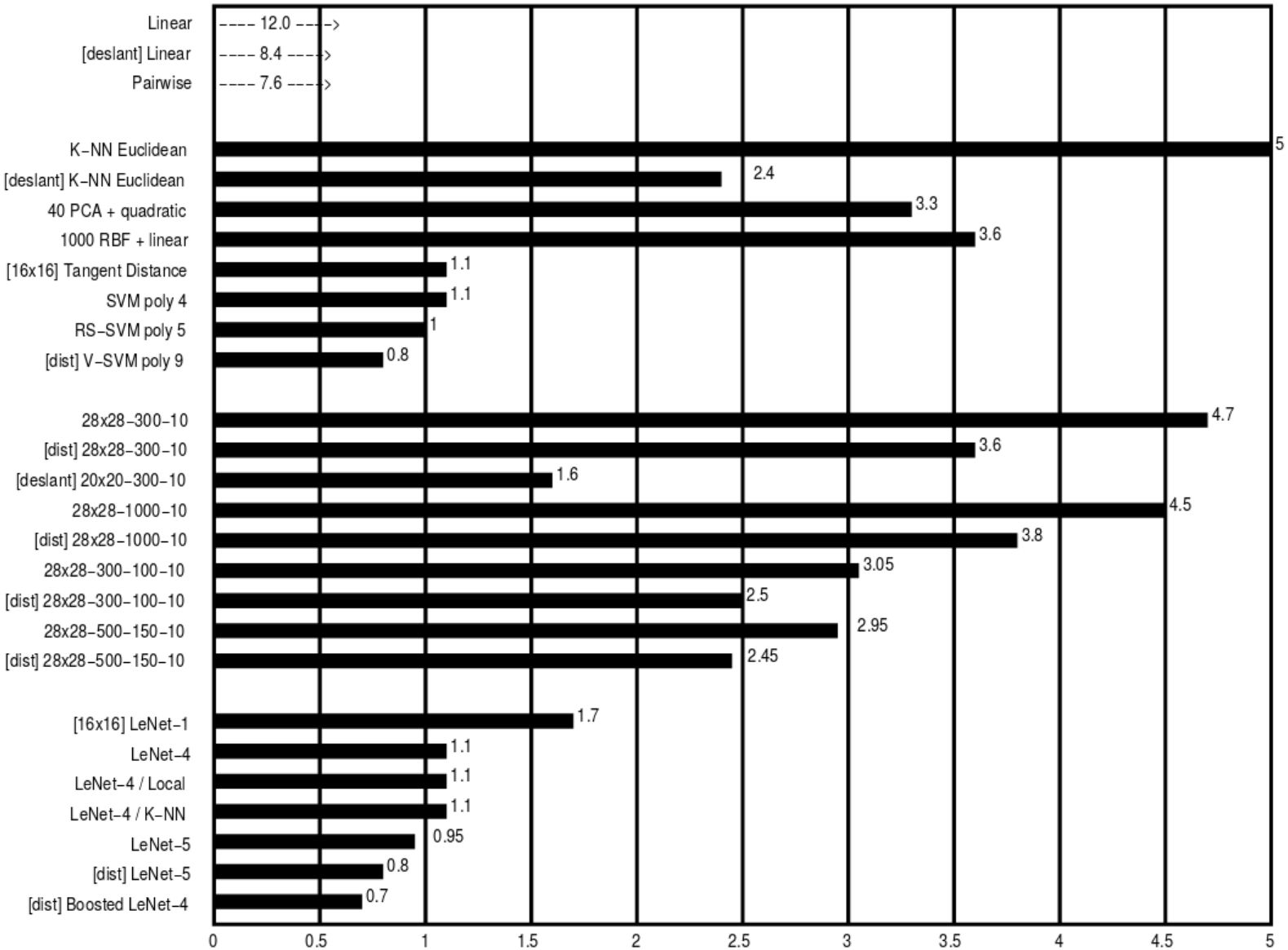
Przykład wizualizacji aktywacji sieci



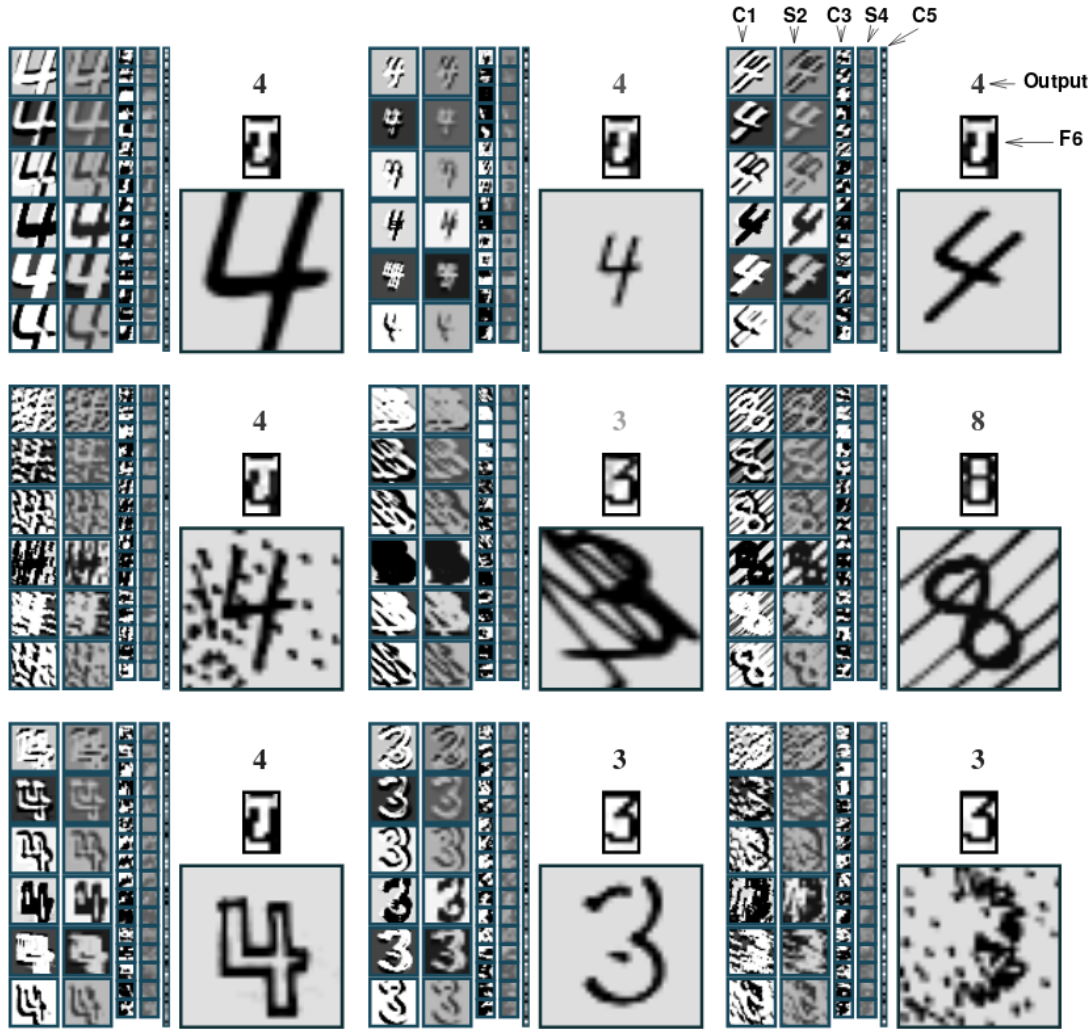
☞ Wizualizacja sieci splotowej

LeNet-5 błędy klasyfikacji

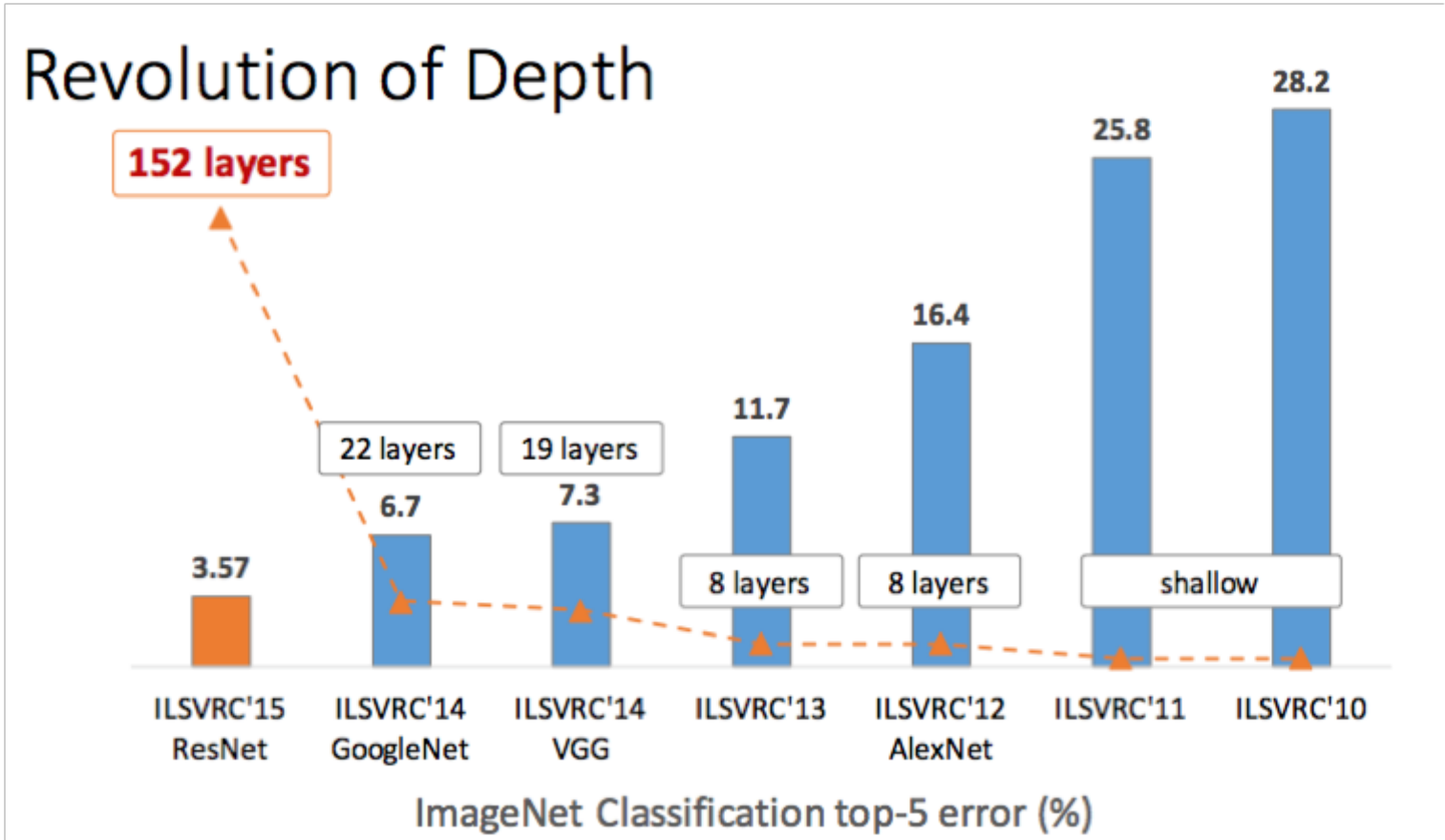
LeNet-5 porównanie z innymi metodami



LeNet-5 odporność na szum, zniekształcenia i nietypowe przypadki



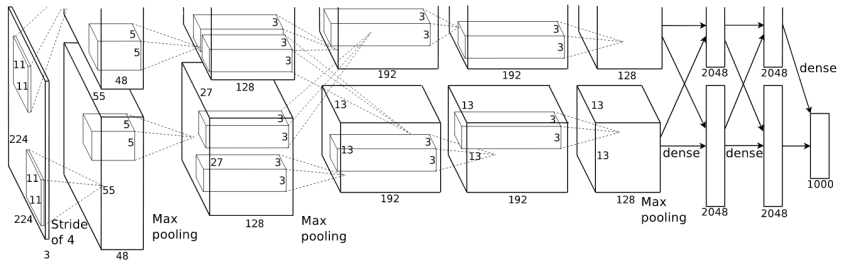
ImageNet



- ☞ CNNs Architectures used on ImageNet

AlexNet (A. Krizhevsky, 2012)

- Zwycięzca ☞ImageNet w 2012 z poprawnością 15.3% (poprawa z 26%)
- 1.2M obrazów, 1000 klas

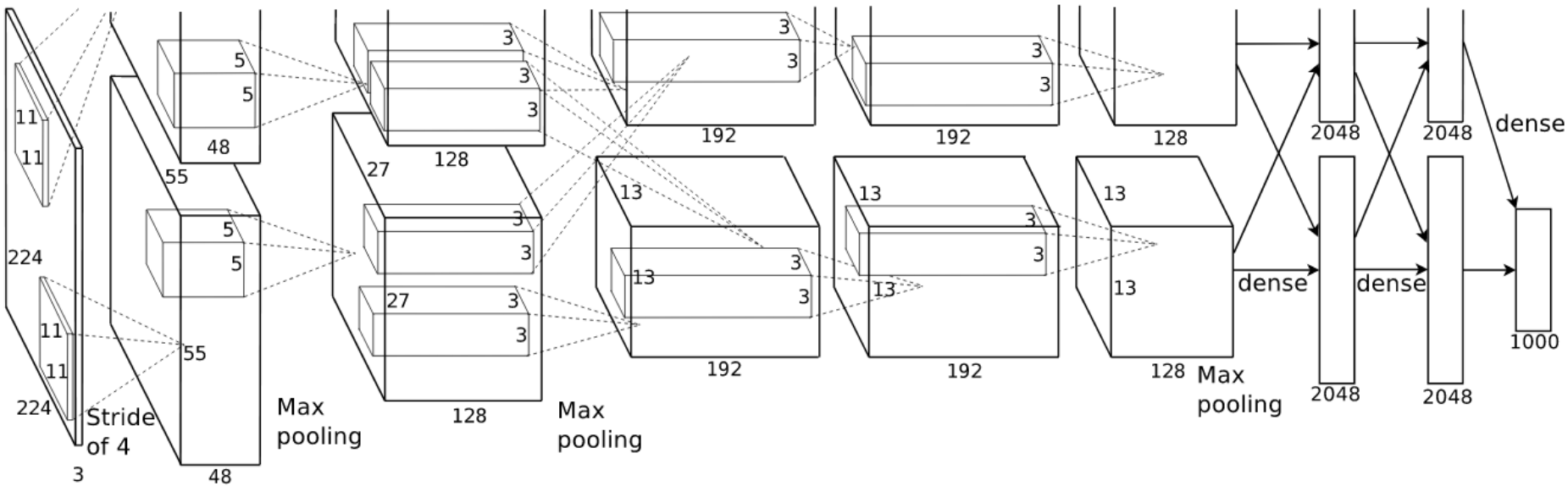


- architektura wzorowana LeNet5 ale głębsza (8 warstw) i więcej filtrów na warstwę (11x11, 5x5, 3x3)
- sekwencje warstw splotowych z jednostkami ReLU
- regularyzacja: dropout, L2, rozszerzanie danych, normalizacja wyjść wybranych warstsw
- SGD z momentem, 20 epok, 6 dni treningu (2x NVIDIA GTX 580 3GB GPUs), 60M parametrów

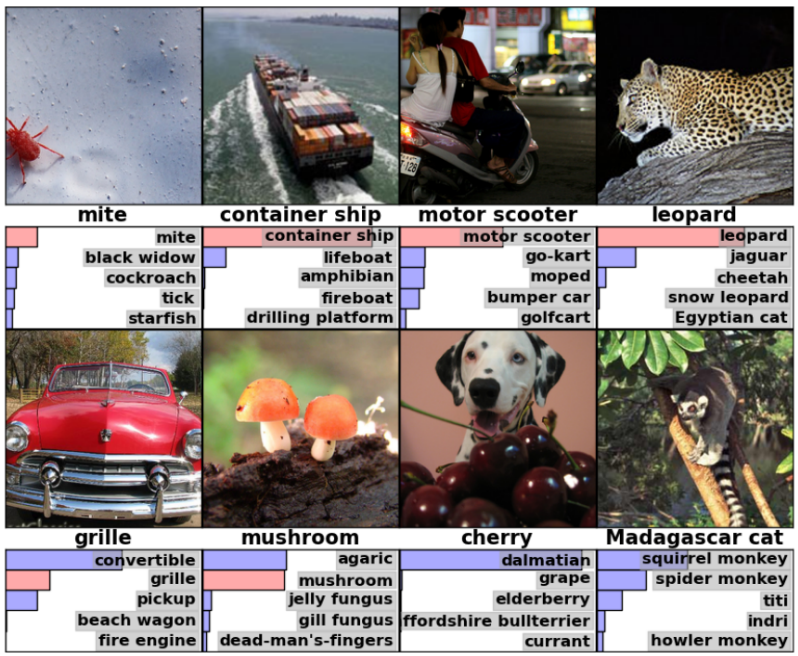
A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2012

AlexNet

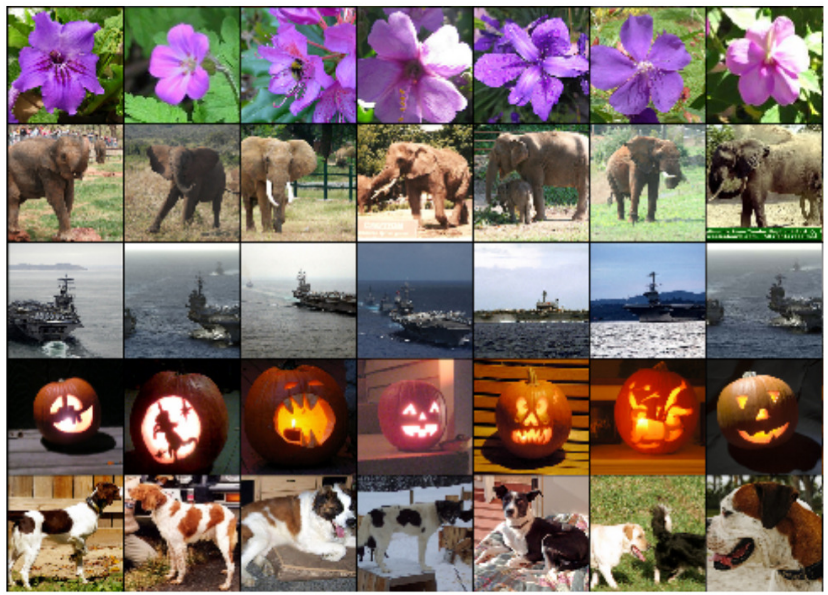
Layer	kernels	size	stride	output
Input				224x224x3
C1 + LRN	96	11x11	4x4	
MaxPooling		3x3	2x2	55x55x96
C2 + LRN	256	5x5		
MaxPooling		3x3	2x2	27x27x256
C3	384	3x3		13x13x384
C4	384	3x3		13x13x384
C5	256	3x3		
MaxPooling		3x3	2x2	13x13x256
F6 + dropout($p = 0.5$)	4096			
F7 + dropout($p = 0.5$)	4096			
Output softmax	1000			



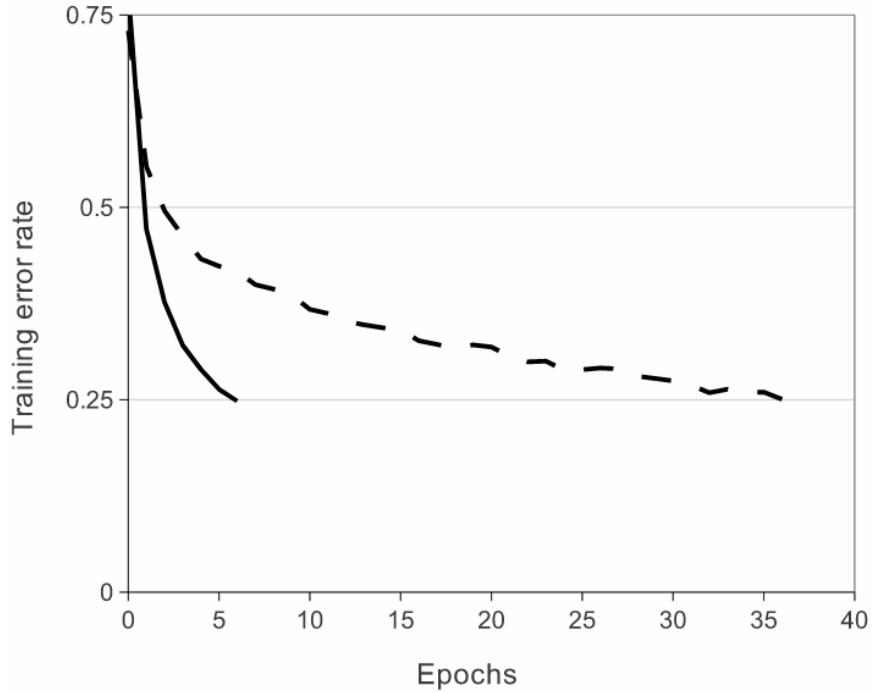
AlexNet



5 najsilniejszych odpowiedzi



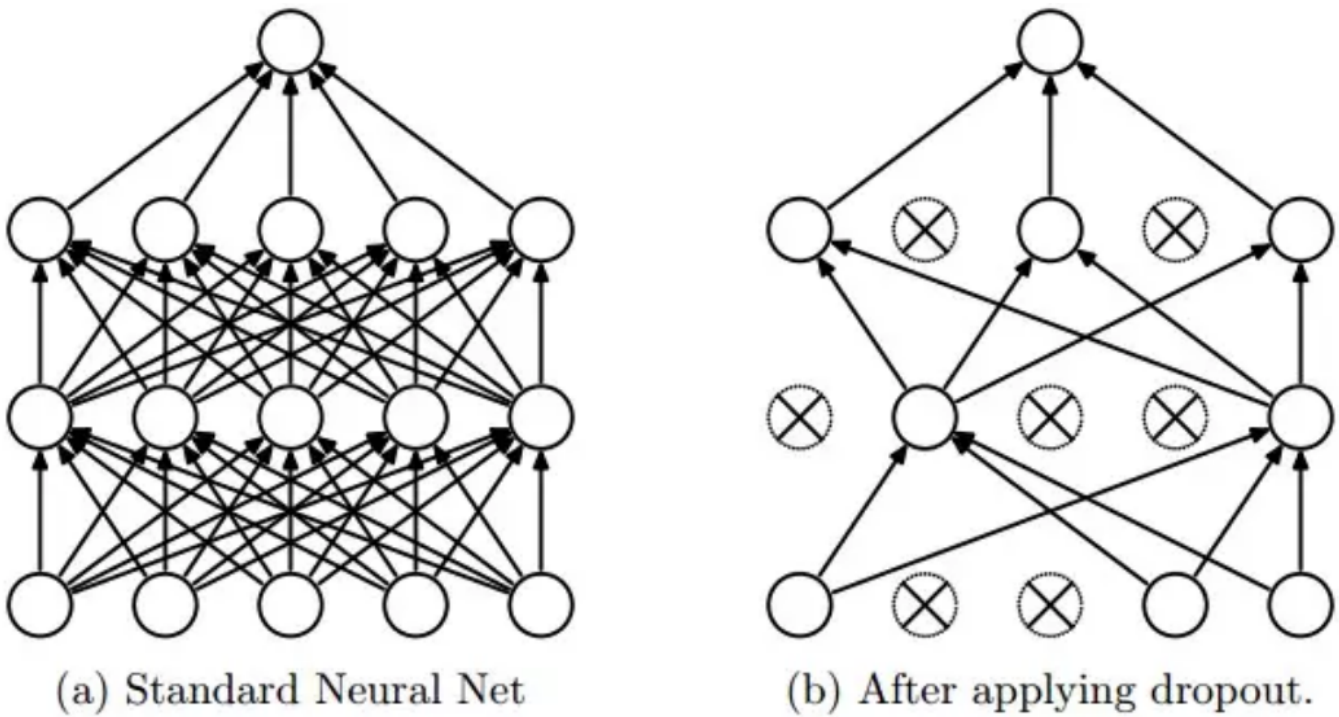
obraz wejściowy i 6 najbliższych wzorców względem odległości Euklidesowej wektora pobudzeń ostatniej warstwy ukrytej



- 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

Droopout (Srivastava et al., 2014) dla każdego wzorca treningowego z prawdopodobieństwem p „wyrzuca” jednostki (zeruje ich aktywacje), odrzycoone 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

- W czasie ewaluacji dropout jest wyłączany ($p = 1$)
- Każdy krok uczenia odbywa się ze zmienioną losowo architekturą sieci ale wagi są współdzielone,
- 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 (AlexNet 2 x wolniej) ale silnie zapobiega przeuczeniu

Local response normalization

$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} \left(a^j_{x,y}\right)^2 \right)^\beta$$

dla kanału i oraz pikseli w pozycji x, y

- normalizacja wartość wyjściowych ReLU
- $k = 2, n = 5, \alpha = 10^{-4}, \beta = 0.75$ dobrane heurystycznie ze zbioru walidacyjnego
- realizuje normalizację jasności dla n sąsiadujących kolejnych
- w AlexNet poprawa o 1.4% (top 1) oraz 1.2% (top 5)

Dodatkowa regularyzacja

Data augumentation

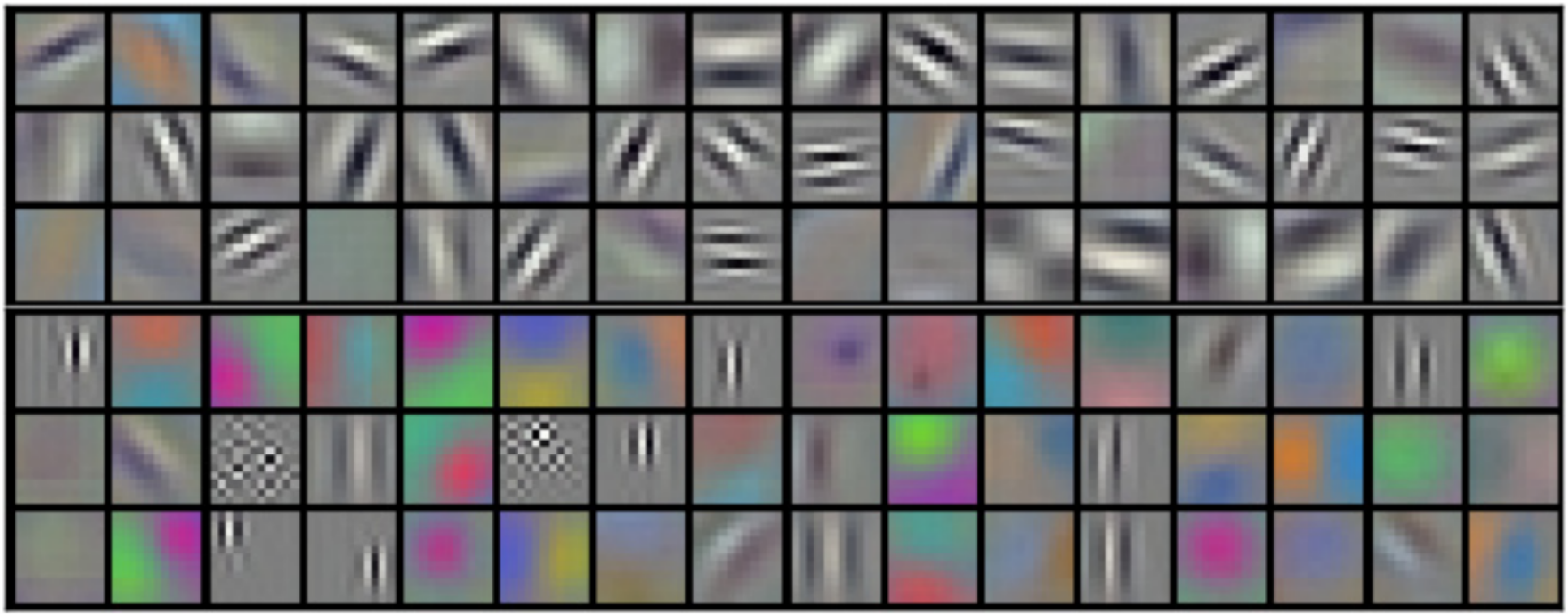
- przesunięcie i odbicia: losowanie obrazków 224x224 z 256x256, zwiększenie zbioru 2048 razy niezbędne do uniknięcia przeuczenia przy tak dużej sieci
- losowa modyfikacja intensywności kanałów RGB
- porawa ponad 1% (top 1, top 5)

MaxPooling z nakrywaniem

- kernel 3x3, stride 2x2
- poprawa 0.4% (top 1) i 0.3% (top 5) względem próbkowania bez nakrywania (kernel 2x2)
- obserwacja: AlexNet rzadziej ulegał przeuczeniu

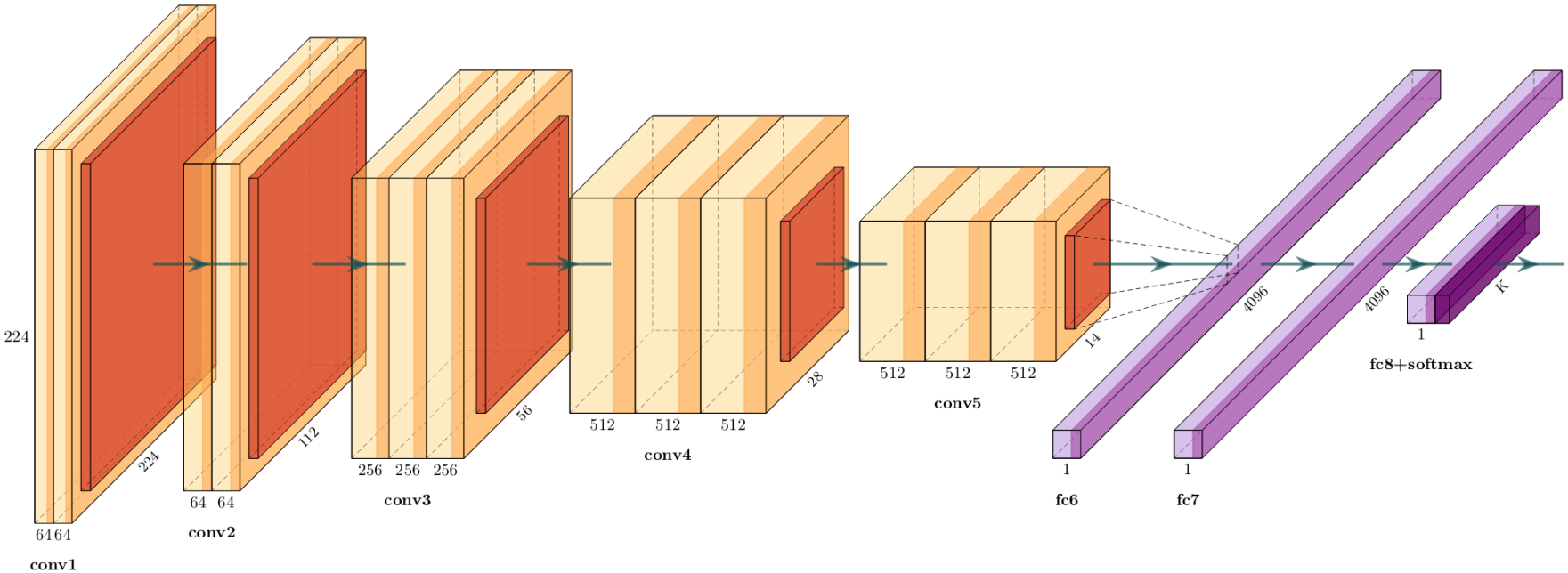
Zastosowanie 2 rdzeni GPU pozwoliło na szkolenie sieci o większej liczbie filtrów co zaowocowało poprawą 1.7% (top-1) oraz 1.2% (top-5) w porównaniu z mniejsza sieci na 1 rdzeniu

Filtry w pierwszej warstwie



96 filtrów pierwszej warstwy splotowej, osobne kolumny warstw splotowych uczą się wykrywania innego typu relacji, obserwowane przy każdym treningu

VGGNet (Simonyan and Zisserman, 2014)



- bloki warstw splotowych + max pooling
- wielkość map zmniejszana o połowę po każdym bloku
- ilość filtrów zwiększana 2 krotnie po każdym bloku

ConvNet Configuration					
A 11 weight layers	A-LRN 11 weight layers	B 13 weight layers	C 16 weight layers	D 16 weight layers	E 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					

VGGNet

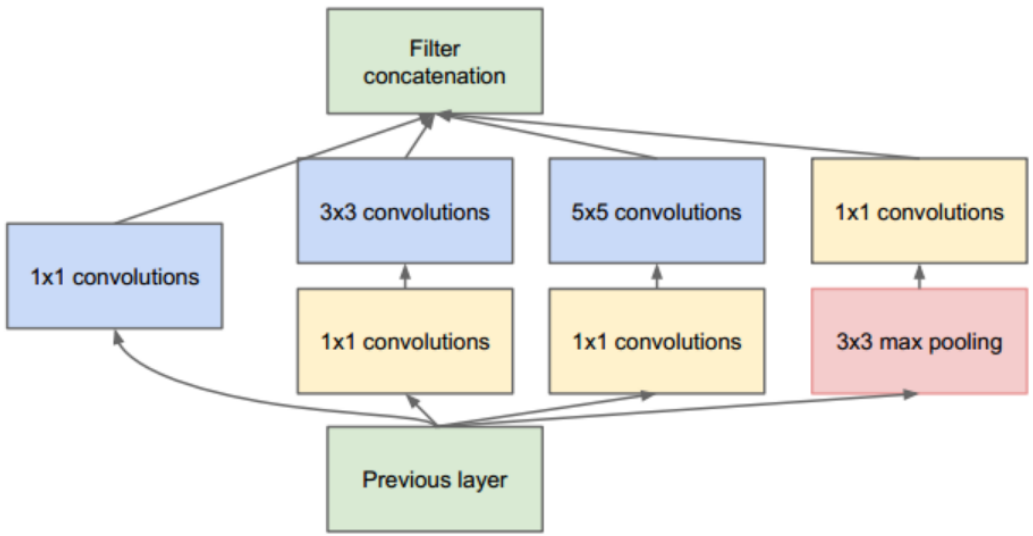
- zamiast dużych filtrów (11x11, 7x7, 5x5) stosuje we wszystkich warstwach filtry 3x3 zwiększając ich efektywne pola recepcyjne sekwencjami warstw splotowych
- małe filtry 3x3 zdolne do wykrycia bardziej subtelnych relacji w obrazach
- zwiększenie głębokości pozwala trenować bardziej złożone cechy
- stride=1, brak utraty informacji, gęsta konwolucja
- kilka architektur od 11 do 19 warstw (VGG16, VGG19)
- 138M 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

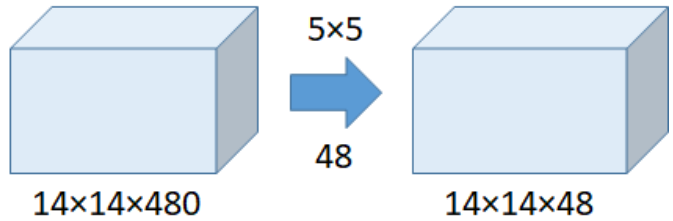
GoogLeNet/Inception

Inception module

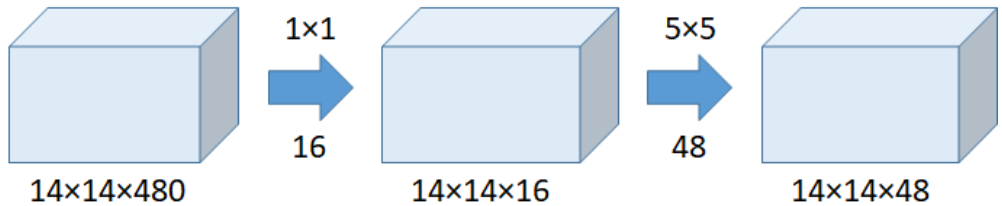
- mały stosunek rozmiaru mapy do ilości filtrów, np. 128 filtrów 3x3 i 32 filtry 5x5
- równoległe sploty o różnej wielkości filtrów 1x1, 3x3, 5x5 - detektory cech o różnej skali
- każdy większy filtr poprzedzony splotem 1x1 w celu redukcji liczby parametrów (redukcja liczby kanałów do 1)



Redukcja złożoności splotem 1x1



liczba operacji $(14 \times 14 \times 48) \times (5 \times 5 \times 480) = 112.9\text{M}$

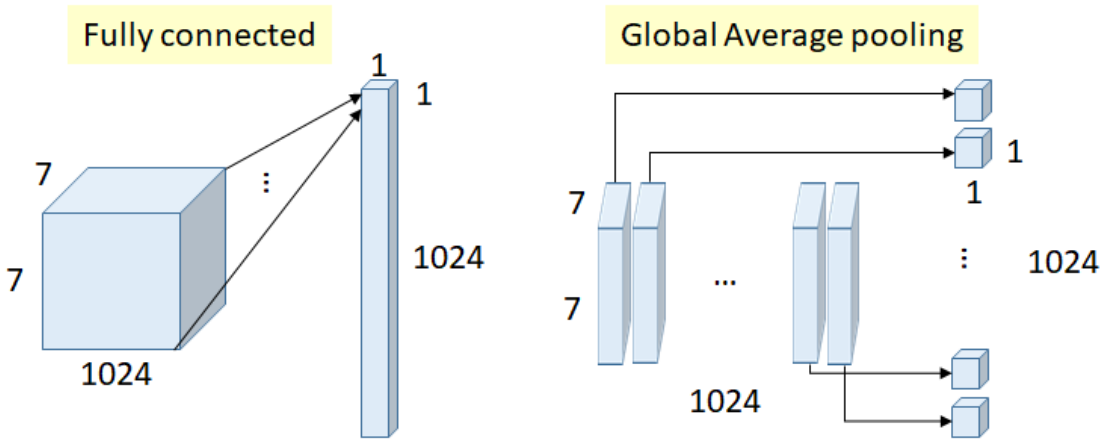


liczba operacji
$(14 \times 14 \times 16) \times (1 \times 1 \times 480) + (14 \times 14 \times 48) \times (5 \times 5 \times 16) =$
$1.5\text{M} + 3.8\text{M} = 5.3\text{M}$

Rys: ☞ Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification) by Sik-Ho Tsang

Global average pooling

Globalne uśrednienie względem kanałów (global average pooling) zamiast pełnej połączonej warstwy za ostatnią warstwą splotową



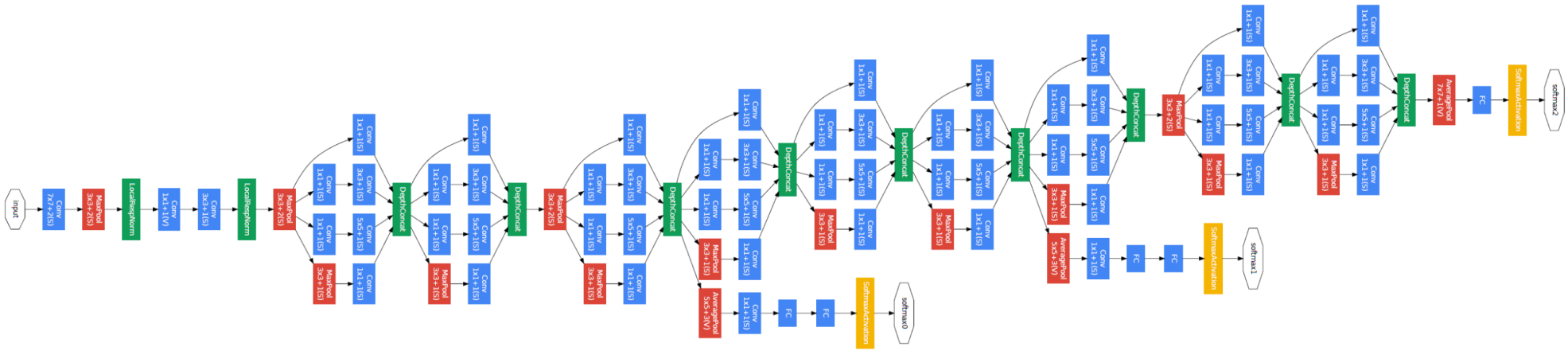
$7 \times 7 \times 1024 \times 1024 = 51.3\text{M}$

- Warstwy w pełni połączone zawierają najwięcej parametrów (w AlexNet 90% parametrów) w sieci
- global average pooling: 0 wag, uśrednienie wartości dla poszczególnych kanałów

Rys: ☞ Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification) by Sik-Ho Tsang

- GoogleNet poprawa poprawności top-1 o 0.6%

GoogLeNet/Inception



- 22 warstwy
- warstwy przewężające (bottlenetk) - dodatkowo redukują złożoność sieci
- kilka wyjść softmax
- poprawność 93.3% top-5 na ImageNet, dużo szybszy w treningu of VGG

ResNet (Residual Neural Network, Kaiming He et. al 2015)

Bloki ze skrótami (residual module)

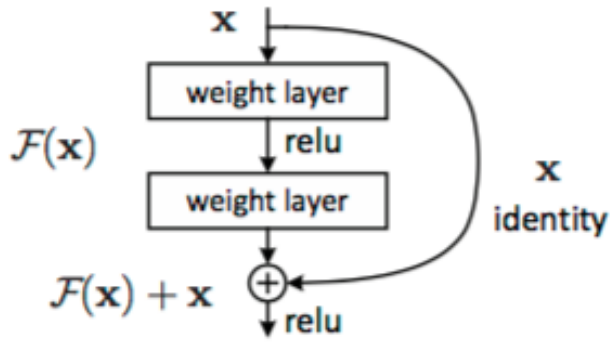
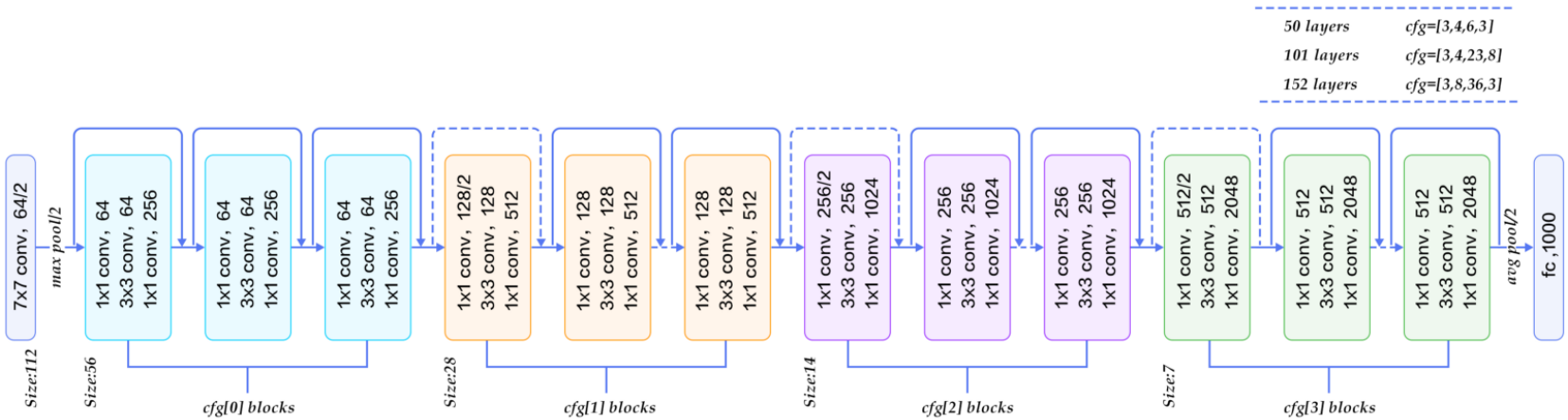


Figure 2. Residual learning: a building block.

- „sktóry” 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 mini pakietu

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}$$

gdzie γ i β podlegają adaptacji w czasie treningu
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

- przyśpiesza trening, poprawia stabilność treningu oraz polepsza generalizację
- rozwiązuje problem przesunięcia kowariancji - gdy zmienia się wyjście warstwy poprzedniej wówczas warstwa następna musi dopasować się do nowego rozkładu danych
- można stosować większe kroki uczenia bez obawy znikającego lub eksplodującego gradientu
- większa odporność na wpływ różnorakiej specjalizacji i metod treningu
- wygładza powierzchnię błędu (Santurkar, 2018)

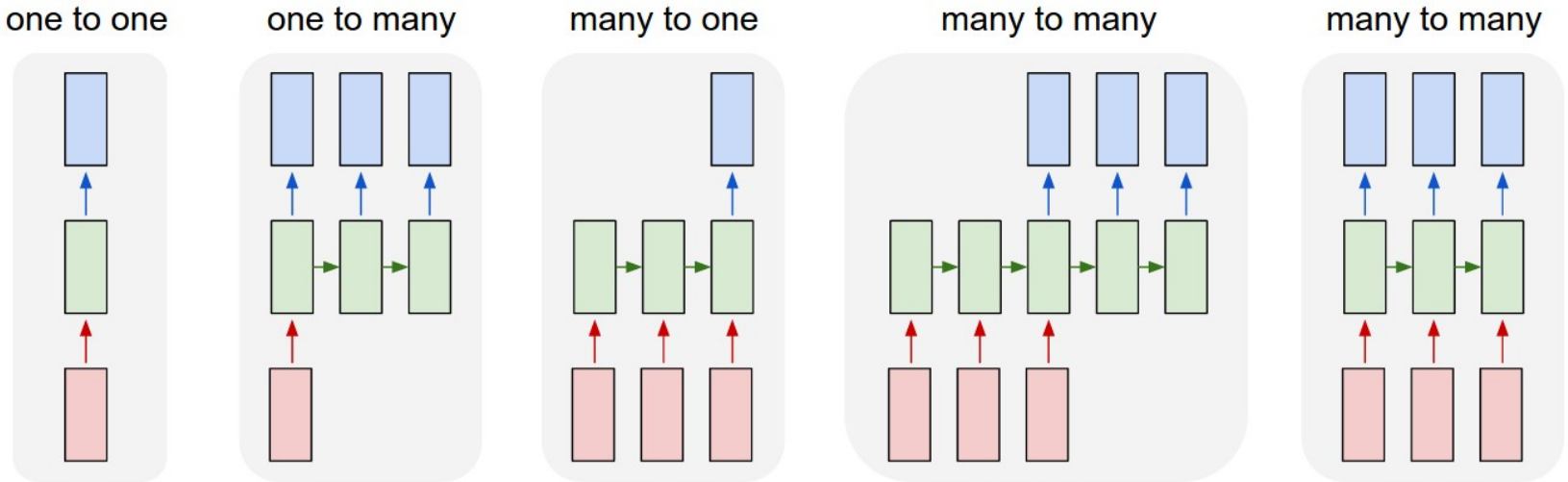
RNN

Sieci rekurencyjne i modelowanie sekwencji

Modelowanie sekwencji

- Dane sekwencyjne lub zależne od czasu: sygnał audio, tekst, ruch obiektów, EEG, ...
- Modelowanie sekwencji: generowanie tekstu (chatboty), generowanie muzyki, etykietowanie obrazów, sterowanie ruchem robota, ...
- Przetwarzanie sekwencji: maszynowe tłumaczenie, rozpoznawanie mowy, rozpoznawanie pisma, przetwarzanie strumieni wideo, ...
- Klasyfikacja sygnałów czasowych, wykrywanie wzorców w sekwencjach, predykcja (np. przewidywanie notowań giełdowych)

Modelowanie sekwencji



- sieci jednokierunkowe: wejście i wyjście o stałym rozmiarze
- generowanie sekwencji dla pojedynczego wejścia, np. opis obrazów
- sekwencyjne wejście: np. klasyfikacja tekstów
- wejście i wyjście sekwencyjne:
  - bez dopasowania ramek, np. tłumaczenie maszynowe, rozpoznawanie mowy
  - z dopasowaniem ramek (*alignment*), np: etykietowanie ramek wideo

☞ *Andrej Karpathy, Hacker's guide to Neural Networks*

Modelowanie sekwencji

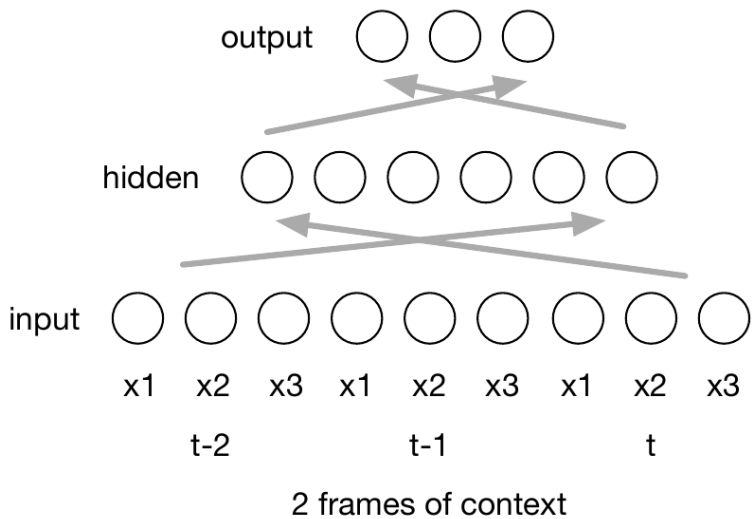
- MLP: dodanie **stałego kontekstu** do wejścia: np. 2 ramki z lewej i prawej (poprzednie i przyszłe wektory wejściowe)

$$\hat{\mathbf{x}}^t \leftarrow [\mathbf{x}^{t-2}, \mathbf{x}^{t-1}, \mathbf{x}^t, \mathbf{x}^{t+1}, \mathbf{x}^{t+2}]$$

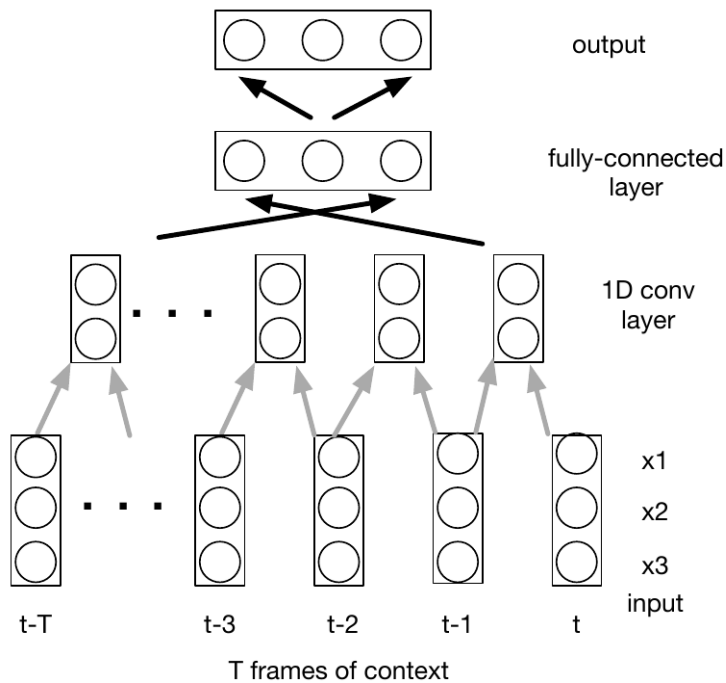
- CNN: splot względem wymiaru czasu, stały kontekst odpowiada wielkości efektywnego pola recepcyjnego w wymiarze czasu.
  TDNN (*time-delay neural network*) model ze splotem 1D w domenie czasu.

Uczenie sekwencji z MLP i CNN

MLP

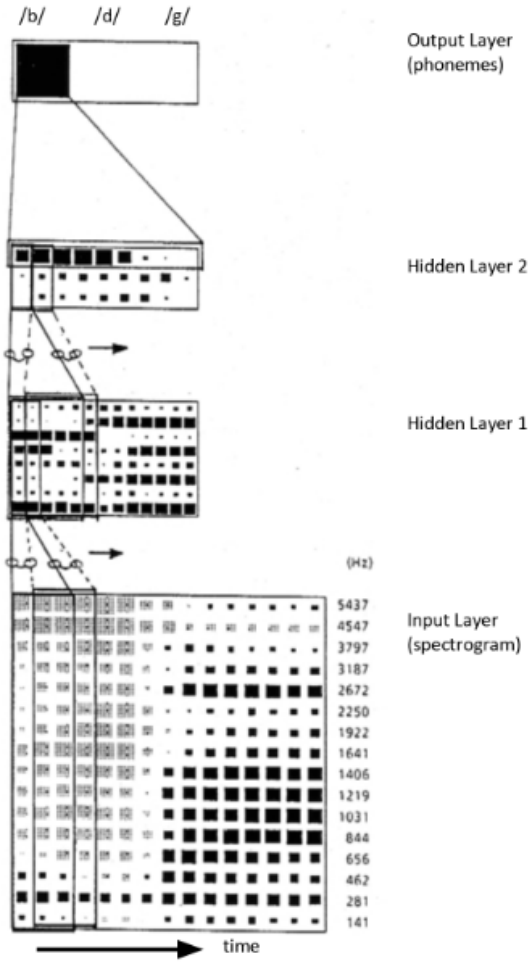


TDNN



Źródło grafik: Steve Renals, Machine Learning Practical — MLP Lecture 9

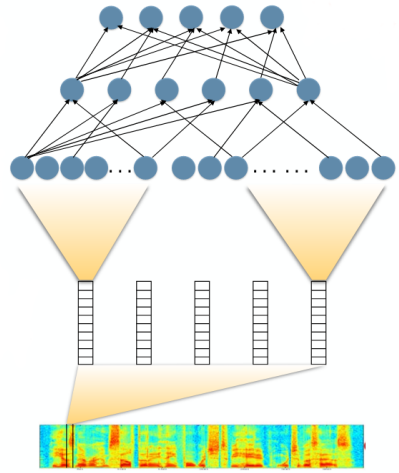
TDNN (Waibel, 1987)



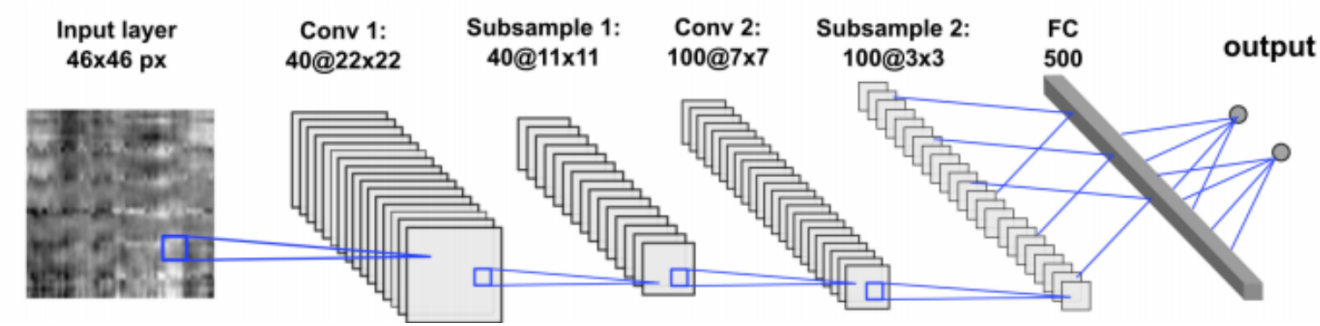
Alexander Waibel, Phoneme Recognition Using Time-Delay Neural Networks, 1987.

Przykłady

Model akustyczny, rozpoznawanie fonemów z nagrań audio



Analiza sygnałów EEG, CNN + widma EEG

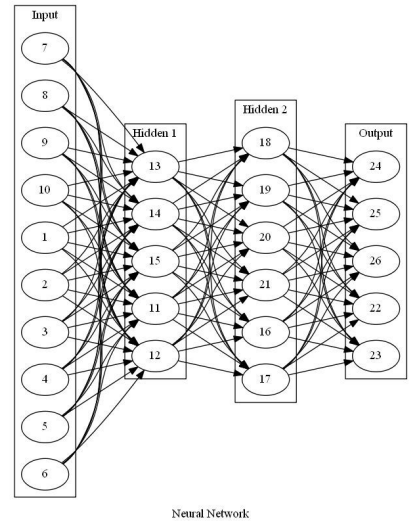


Nurse, E., Mashford, B. S., Yepes, A. J., Kiral-Kornek, I., Harrer, S., & Freestone, D. R. (2016). Decoding EEG and LFP Signals using Deep Learning: Heading TrueNorth

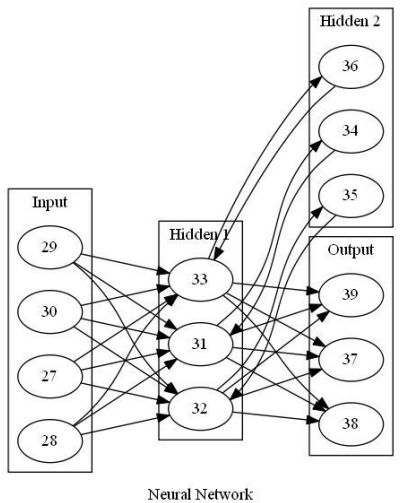
RNN

- **Sieci 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 $t-1$

Sieć jednokierunkowa

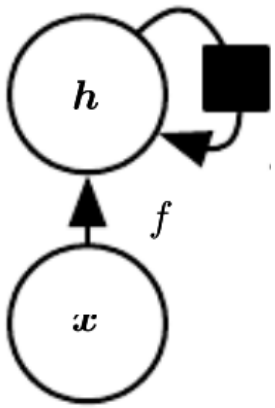


Sieć rekurencyjna Elmana



Stan ukryty jednostek RNN

$$\mathbf{h}^t = f(\mathbf{h}^{t-1}, \mathbf{x}^t; \boldsymbol{\theta})$$

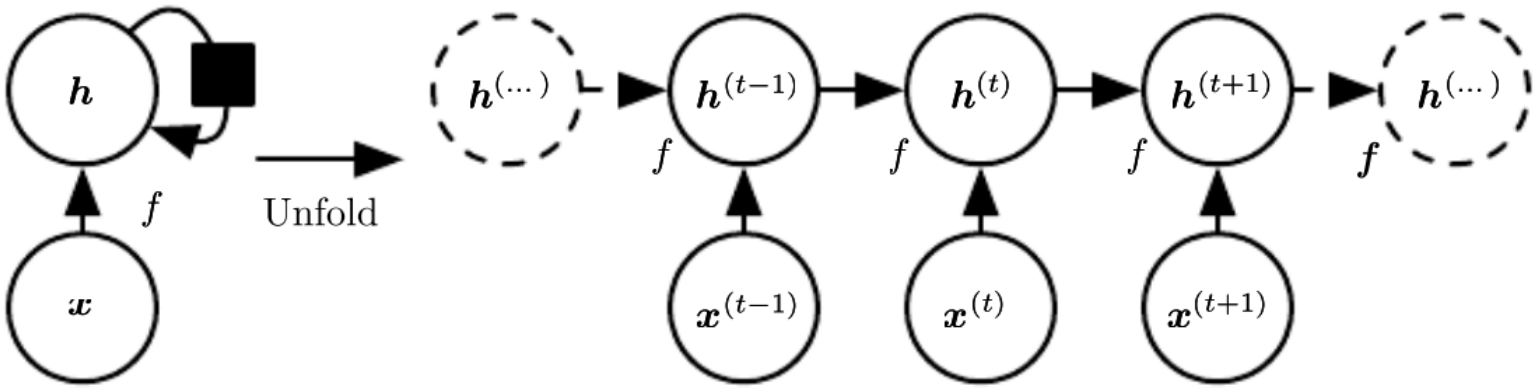


- neurony z połączeniami rekurencyjnymi działają jak pamięć - potencjalnie są w stanie modelować relacje występujące z dowolnie długim odstępem czasowym (nieskończony kontekst)

$$\mathbf{h}^t = g(\mathbf{x}^t, \mathbf{x}^{t-1}, \ldots, \mathbf{x}^1)$$

- wektor $\mathbf{h}^t$ tworzy skompresowaną reprezentację sekwencji, mapuje dowolnie długą sekwencję na wektor o skończonej długości
- stan początkowy $\mathbf{h}^0$ może być interpretowany jako dodatkowy sygnał wejściowy, typowo inicjowany zerami

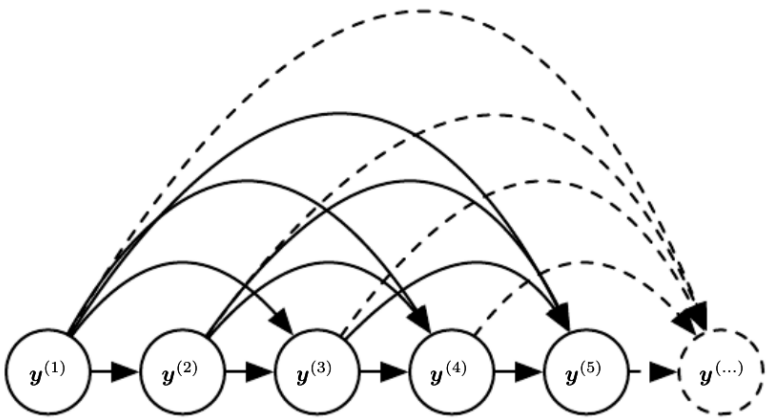
Rozwinięcie rekurencji w czasie



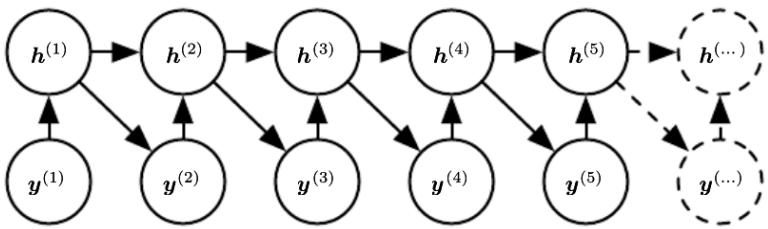
Rozwinięcie sieci RNN do postaci sieci jednokierunkowej, gdzie wagi połączeń są współdzielone

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

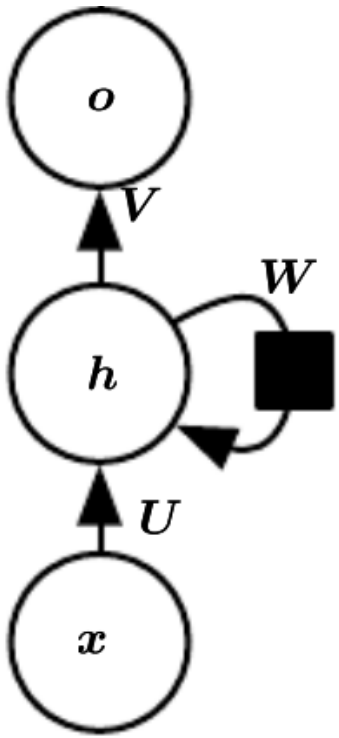


pełny model sekwencji, ilość parametrów $O(k^\tau)$



RNN, stan obecny zależy pośrednio od stanów z przeszłości, ilość parametrów stała względem dł. sekwencji $O(1)$

Sieć z jedną warstwą rekurencyjną

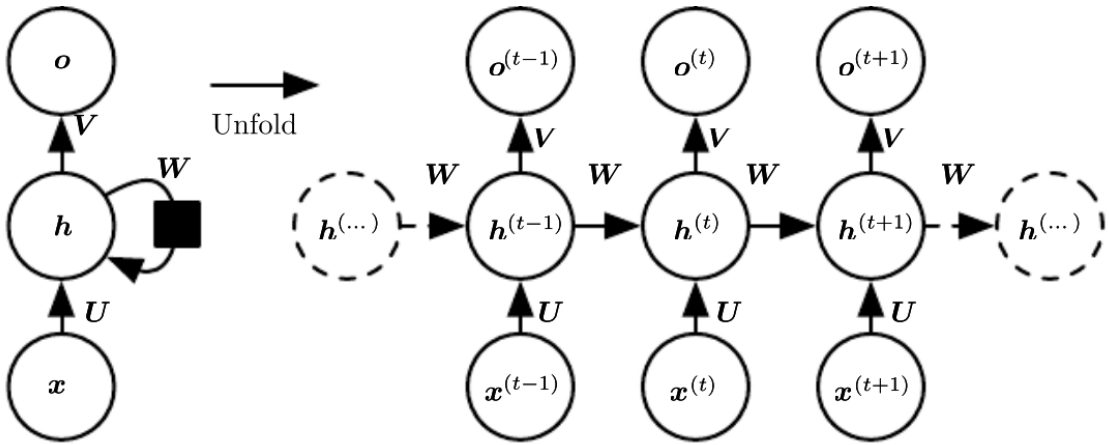


$$o_k^t = softmax_k\left(\sum_r v_{kr} h_r^t + b_k\right)$$

$$h_i^t = \sigma\left(\sum_j u_{ij} x_j^t + \sum_r w_{ir} h_r^{t-1} + b_j\right)$$

Źródło grafiki: Goodfelow, Deep Learning, 2016

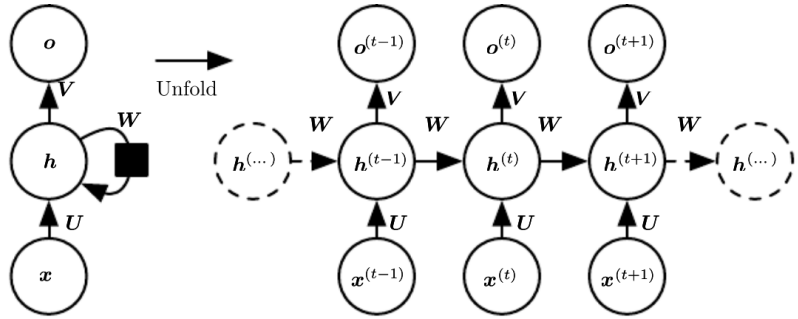
Wsteczna propagacja w czasie



- Rozwinięta sieć odpowiada głębokiej sieci jednokierunkowej ze współdzielonymi wagami $\mathbf{W}, \mathbf{V}, \mathbf{U}$
- BPTT *backpropagation through time* - trening za pomocą wstecznej propagacji odbywa się analogicznie jak w jednokierunkowych sieciach.

Źródło grafiki: Goodfelow, Deep Learning, 2016

Wsteczna propagacja w czasie



Funkcja kosztu dla całej sekwencji:

$$L\left(\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^\tau\}, \{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^\tau\}\right) = \frac{1}{\tau} \sum_t L^t$$

L^t - koszt w kroku t (np. cross-entropy) zależny od $\mathbf{x}^1, \ldots, \mathbf{x}^\tau$

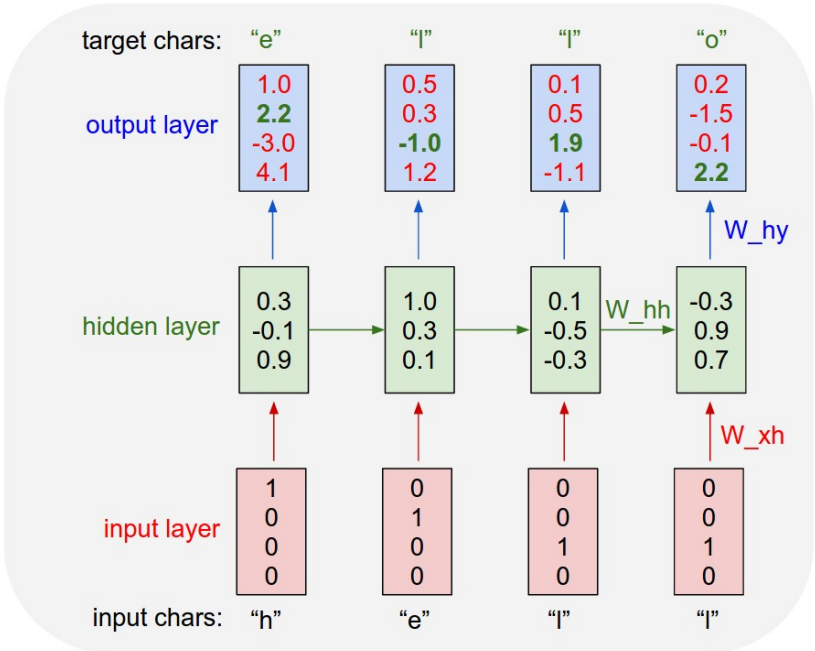
$$\partial_w L = \sum_{t=1}^{\tau} \partial_w L\left(y^t, o^t\right)$$

Aktualizacje wag są akumulowane dla sygnału błędu propagowanego w kolejnych krokach czasu

Źródło grafiki: Goodfelow, Deep Learning, 2016

Przykład: CharRNN

- model języka oparty na znakach (alfabet: h, e, l, o)
- RNN modeluje rozkład prawdopodobieństwa wystąpienia następnego znaku w sekwencji

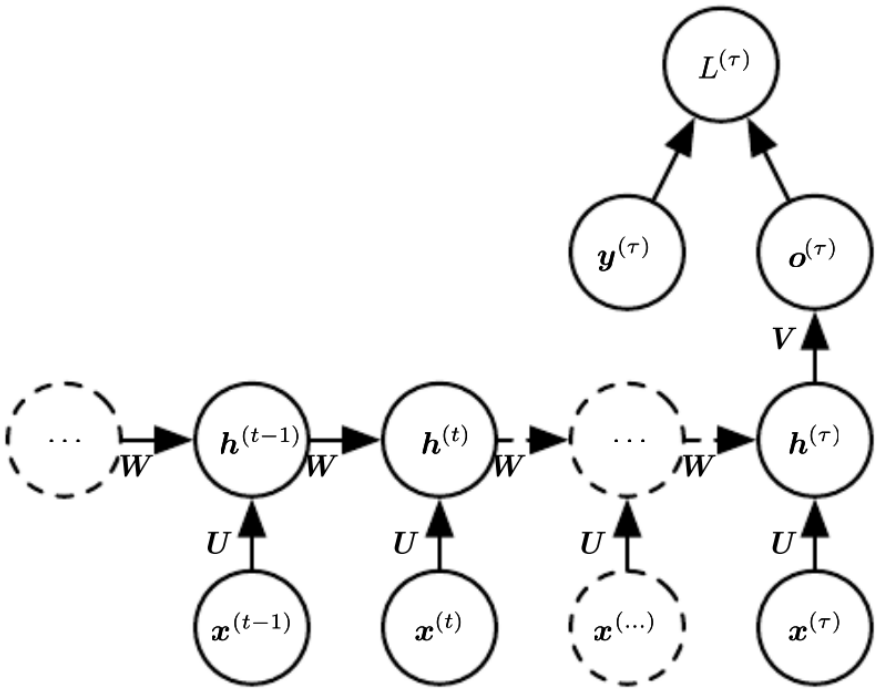


☞ The Unreasonable Effectiveness of Recurrent Neural Networks

Wsteczna propagacja w czasie

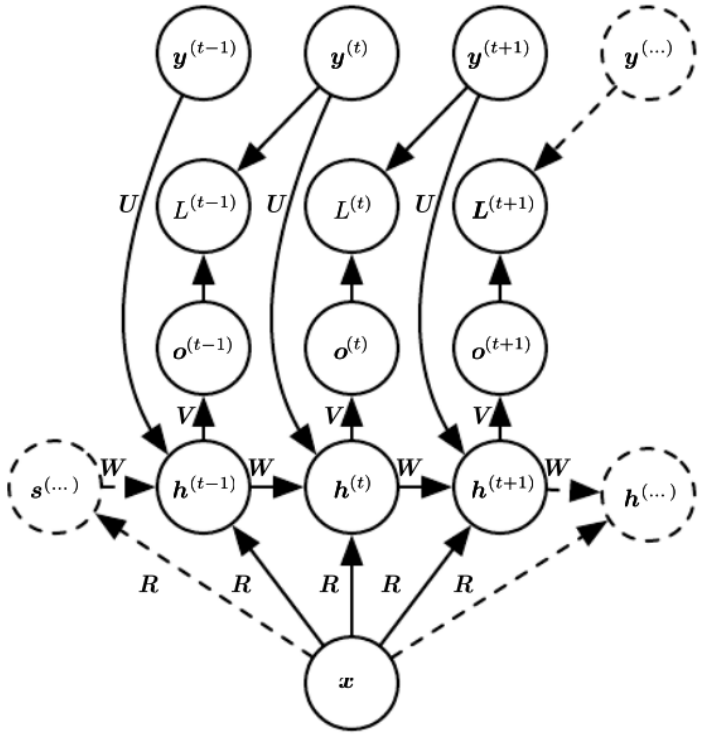
- Rozwinięta w czasie sieć tworzy głęboką sieć neuronową ze wszystkimi tego konsekwencjami (zanikający gradient dla relacji odległych w czasie, rzadziej - wybuchający gradient)
- Obliczenia dla długich sekwencji są kosztowne
- **Truncated BPTT** - ograniczenie długości sekwencji do stałej wartości (np. 20)

Pojedyncze wyjście na końcu sekwencji



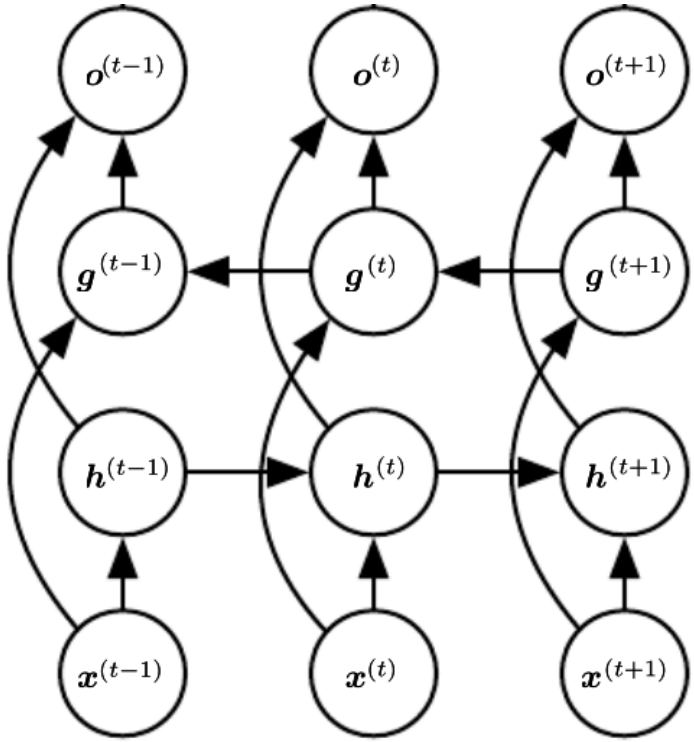
Sieć produkuje wyjście po obejrzeniu całej sekwencji, np. klasyfikacja wpisów w portalach społecznościowych pod względem emocji

Pojedyncze wejście generujące sekwencję



- Sieć produkuje sekwencję wyjść przy prezentacji stałego wzorca (np. etykietowanie zdjęć).
- Sygnał wejściowy pełni rolę kontekstu

Dwukierunkowa sieć RNN



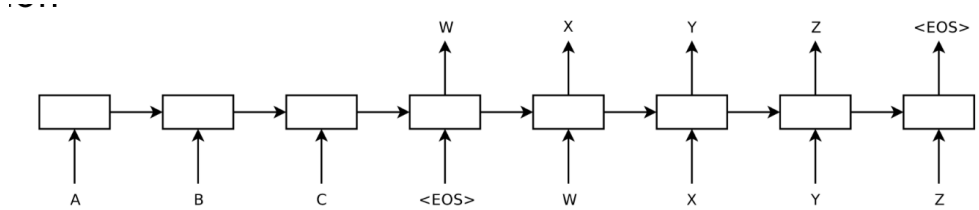
Sieci dwukierunkowe RNN łączą warstwy z rekurencją zależna od sygnału przeszłego **h** z rekurencją zależną od przyszłego sygnału **g**

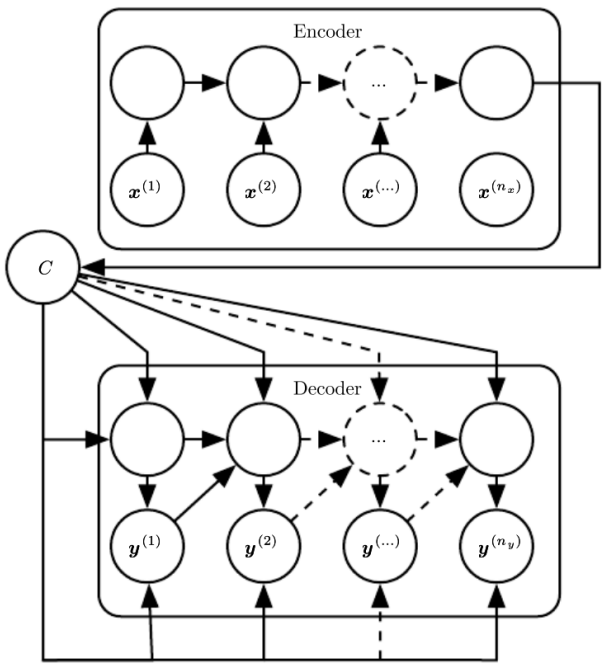
Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks.Śignal Processing, 1997

Dwukierunkowa sieć RNN

- informacja z przyszłości może zawierać istotną informację, np. dzwięki kolejnych fonemów (koartykulacja), znaczenie słów w zdaniu
- wyjścia sieci w dowolnym momencie zależą od całej sekwencji wejściowej
- zastosowanie: rozpoznawanie mowy (Graves, 2008, 2009), bioinformaryka (Baldi, 1999), rozpoznawanie pisma (Liwicki, Marcus, 2007)
- dla obrazów 2D możliwe zastosowanie sieci 4 kierunkowych, co pozwala modelować odległe relacje, jednak nie są one tak wydajne w tych zastosowaniach jak CNN

Model Encoder-Decoder





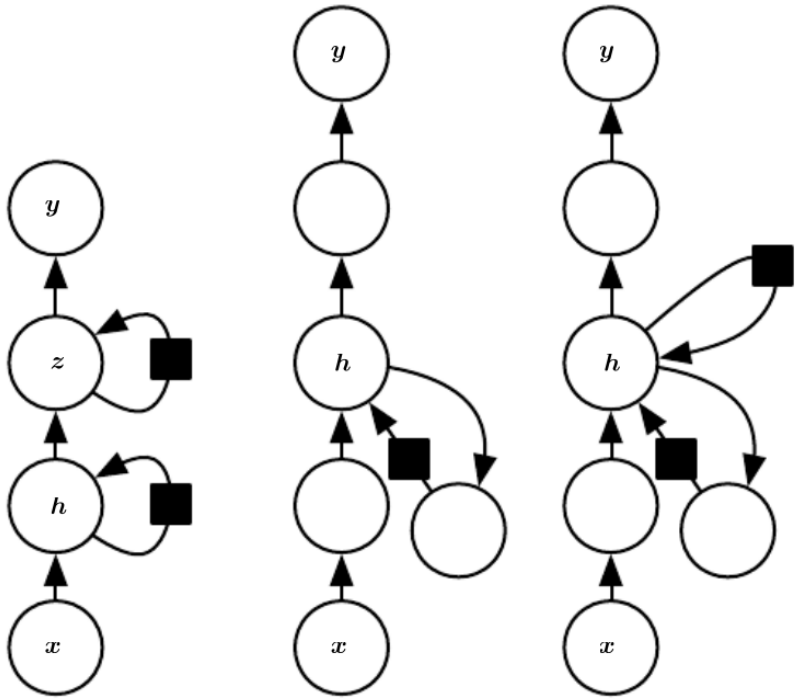
Sutskever et al (2014). "Sequence to Sequence Learning with Neural Networks", NIPS.
K Cho et al (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", EMNLP.

Model Encoder-Decoder

seq2seq (Sutskever, 2014) to dwie sieci RNN uczone jednocześnie:

- **enkoder** (reader, input) przetwarza sekwencję wejściową generując kontekst C
- **dekoder** (writer, output) zależnie od kontekstu C generuje sekwencję wyjściową
- wielkość kontekstu C może ograniczyć zdolność sieci do reprezentacji dalekich w czasie relacji
- dodanie uwagi (*attention model*, Bahdanan, 2015) - uwzględnienie w C więcej informacji z enkodera, które skalują wyjścia dekodera

Deep RNN

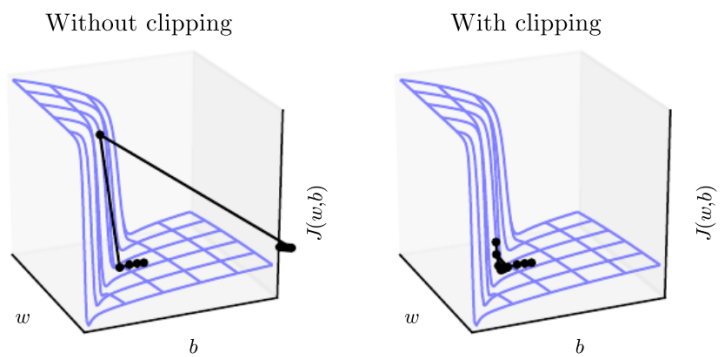


- warstwowe RNN - kolejne warstwy tworzą hierarchię reprezentacji (Graves, 2013)

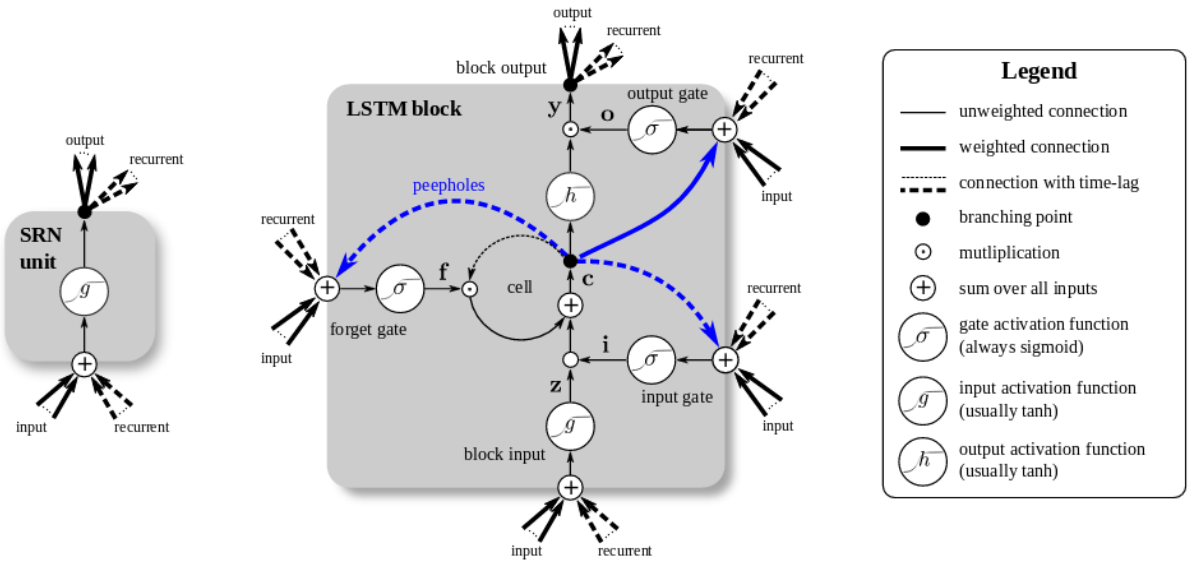
Goodfelow. Deep Learning, 2016

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



$$\mathbf{z}^t = g(\mathbf{W}_z \mathbf{X}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z)$$
$$\mathbf{i}^t = \sigma(\mathbf{W}_i \mathbf{X}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i)$$
$$\mathbf{f}^t = \sigma(\mathbf{W}_f \mathbf{X}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f)$$
$$\mathbf{c}^t = \mathbf{i}^t \odot \mathbf{z}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1}$$
$$\mathbf{o}^t = \sigma(\mathbf{W}_o \mathbf{X}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o)$$
$$\mathbf{y}^t = \mathbf{o}^t \odot h(\mathbf{c}^t)$$

Sepp Hochreiter, Jürgen Schmidhuber, „Long short-term memory", 1997

LSTM

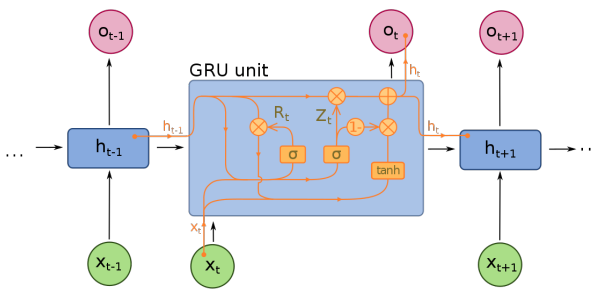
- LSTM niweluje wpływ zanikania gradientu, dzięki czemu pozwala mapować odległe w czasie relacje
- Wprowadza dodatkową wewnętrzną jednostkę stanu $\mathbf{c}$
- Stan $\mathbf{c}^t$ zależy w sposób liniowy od stanu poprzedniego $\mathbf{c}^{t-1}$ (liniowa rekurencja), więc nawet przy bardzo długich rekurencjach gradient ma szanse nie zaniknąć (*long-term memory*)
- 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 (*forgate gate*) steruje wpływem poprzedniego stanu $\mathbf{c}^{t-1}$ na obecny stan $\mathbf{c}^{t}$
- bramka wejściowa i zapominania wpływają na to co umieszczane jest 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

Gated Reccurent Unit

- GRU (Kyunghyun Cho et al., 2014) brak jednooki wewnętrznej **c** i bramki wyjściowej
- bramka resetu R_t i bramka aktualizacji Z_t
- mniej parametrów wzgl. LSTM, porównywalna jakość wyników dla wielu problemów



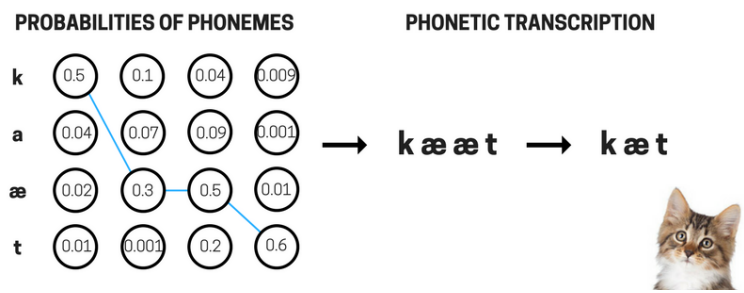
$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

Źródło: https://en.wikipedia.org/wiki/Recurrent_neural_network

Connectionist Temporal Classification

- metoda treningu sekwencji, gdy nie ma dostępnego dopasowania ramek (*aligement*), długość sekwencji wejściowej jest dłuższa od sekwencji wyjściowej, np. transkrypcja tekstu na podstawie nagrań audio
- wyjście softmax uzupełnione o dodatkową etykietę 'blank' $\mathcal{L}' = \mathcal{L} \cup \{blank\}$ generuje rozkład prawdopodobieństw zaobserwowania sekwencji etykiet (ścieżki) $\mathbf{y}'$ o długości T

$$P(\mathbf{y}'|\mathbf{x}) = \prod_{t=1}^{T} P(y'_t|\mathbf{x})$$



Connectionist Temporal Classification

CTC poszukuje sekwencji etykiet o największej wiarygodności $P(\mathbf{y}|\mathbf{x})$ marginalizując po wszystkich możliwych ścieżkach $\mathbf{y}'$ z blankiem

$$p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{y}' \in \mathcal{M}^{-}1(\mathbf{y})} P(\mathbf{y}'|\mathbf{x})$$

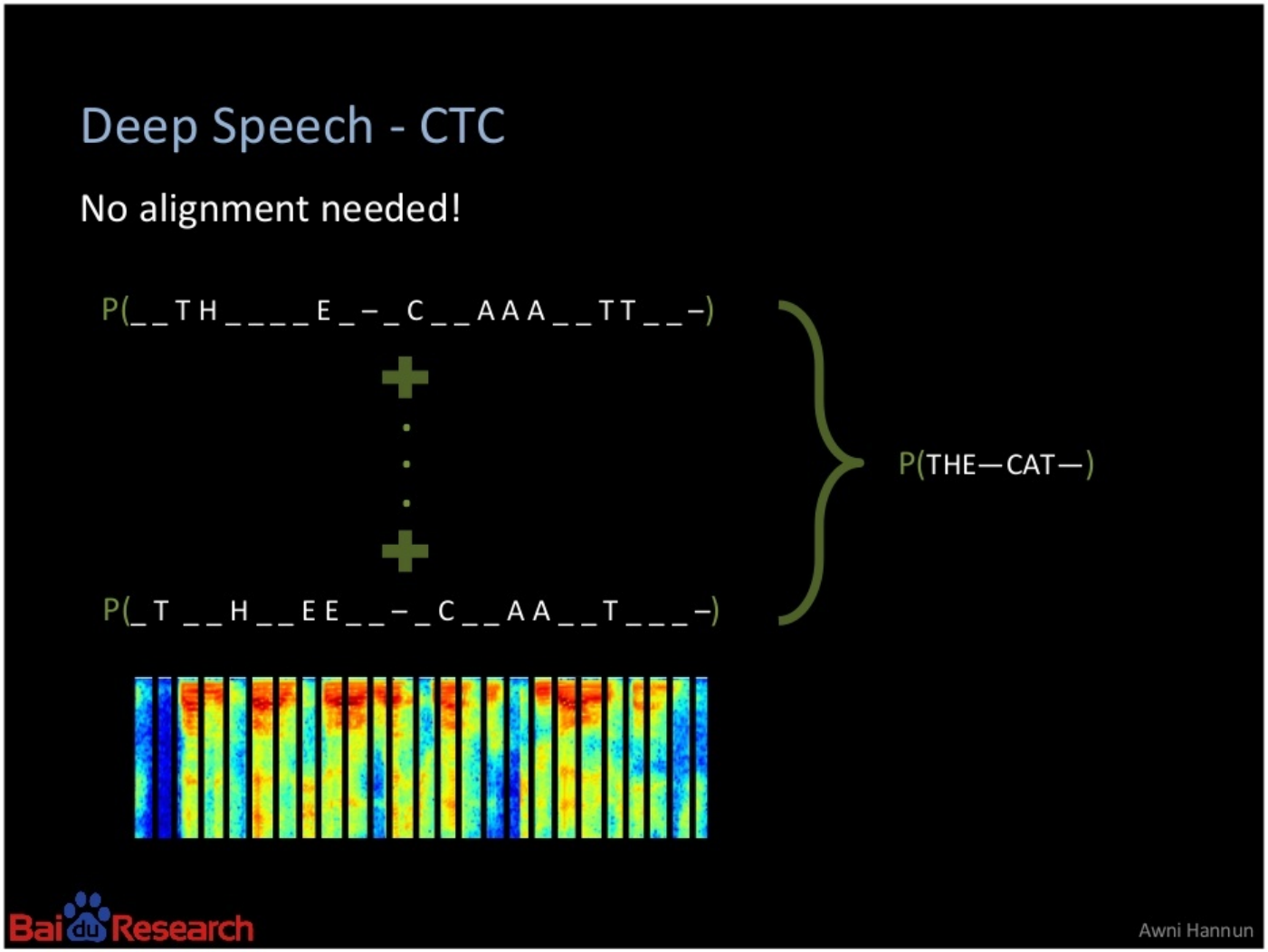
gdzie $\mathcal{M}$ mapuje sekwencję etykiet o długości T do sekwencji krótszej poprzez usunięcie wszystkich blanków i powtarzających się etykiet $\mathcal{M}(a - ab-) = \mathcal{M}(-aa - -abb) = aab$

Funkcja kosztu

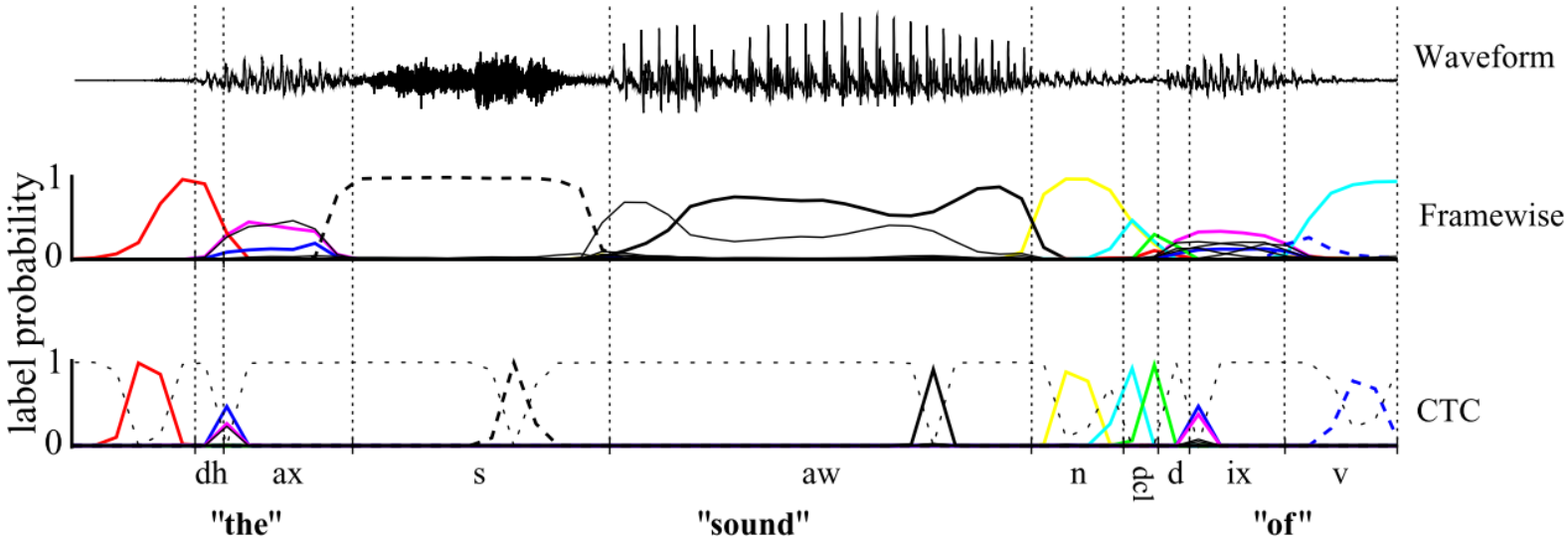
$$L = -\sum \log p(\mathbf{y}|\mathbf{x})$$

Graves, A., Fernández, S., Gomez, F., Schmidhuber, J., Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. 2006.

Connectionist Temporal Classification (CTC)

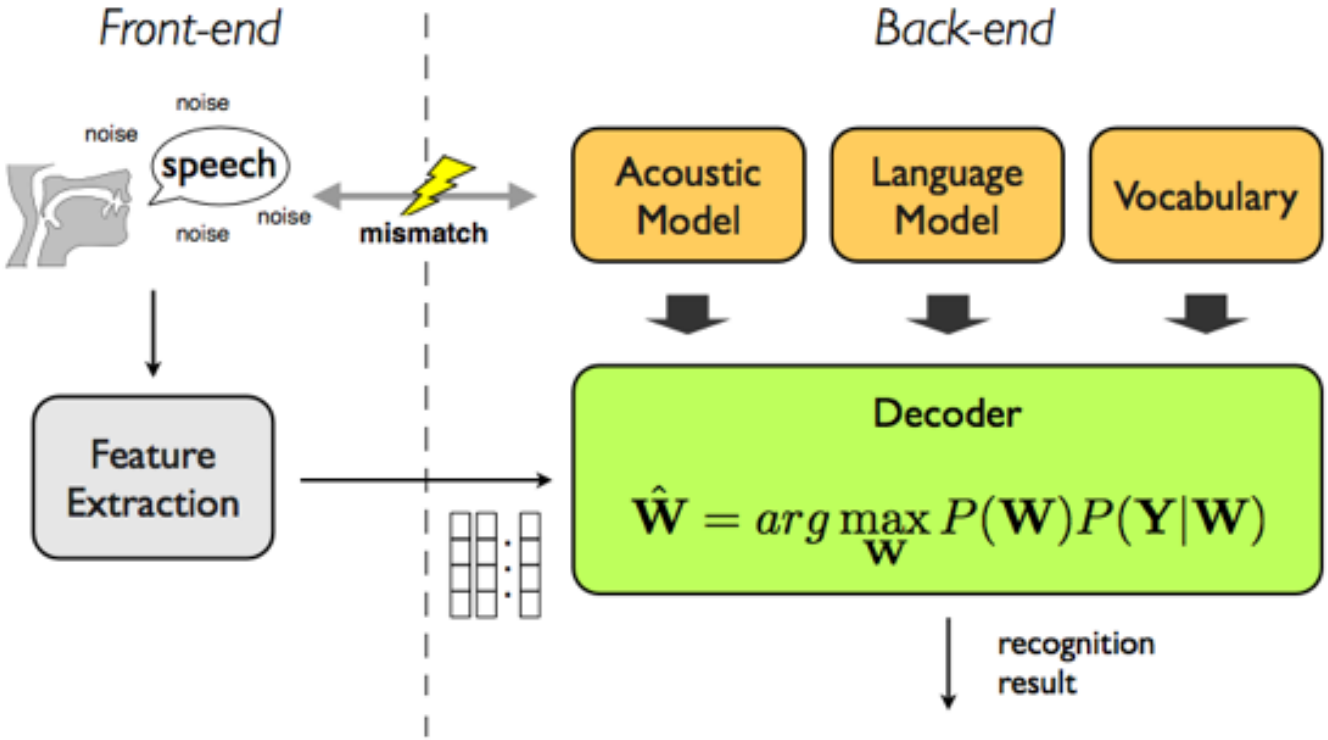


Predykcja CTC



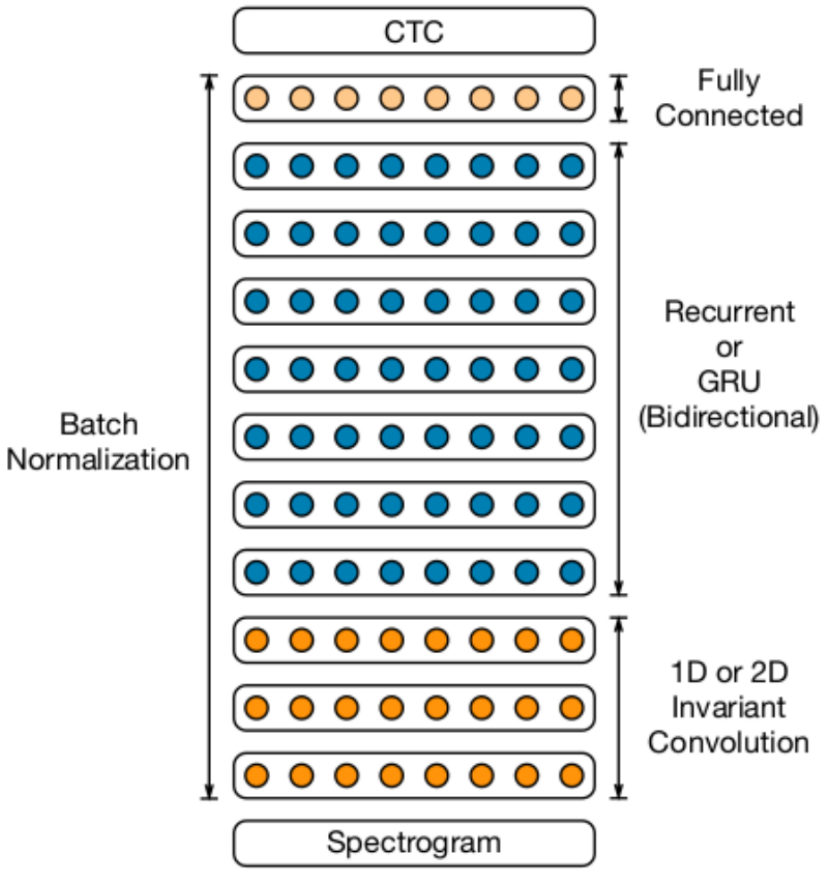
- CTC generuje sekwencję pików etykiet (fonemów) przedzielonych etykietą 'blank'
- w sieci z kosztem CE przesunięcie granic fonemu względem poprawnego dopasowania owocuje błędem nawet gdy fonem jest poprawnie przewidziany
- wyjście CTC może być wykorzystane bezpośrednio (bez konieczności dalszego przetwarzania) do transkrypcji

Automatyczne rozpoznawanie mowy



Źródło grafiki: `https://www.esat.kuleuven.be/psi/spraak/theses/08-09-en/MDT.php`

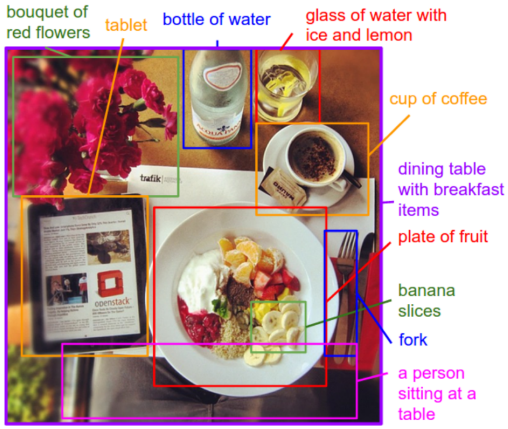
Przykład: Deep Speach 2

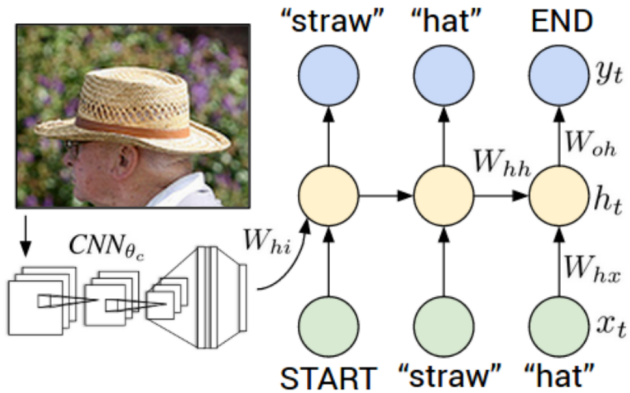


Read Speech			
Test set	DS1	DS2	Human
WSJ eval'92	4.94	3.60	5.03
WSJ eval'93	6.94	4.98	8.08
LibriSpeech test-clean	7.89	5.33	5.83
LibriSpeech test-other	21.74	13.25	12.69

Amodei, Dario, et al. "Deep speech 2: End-to-end speech recognition in english and mandarin."(2015).

NeuralTalk - generator opisów zdjęć





Multimodal Recurrent Neural Network ☞ NeuralTalk2

Karpathy A., Fei-Fei L. „Deep Visual-Semantic Alignments for Generating Image Descriptions", 2015

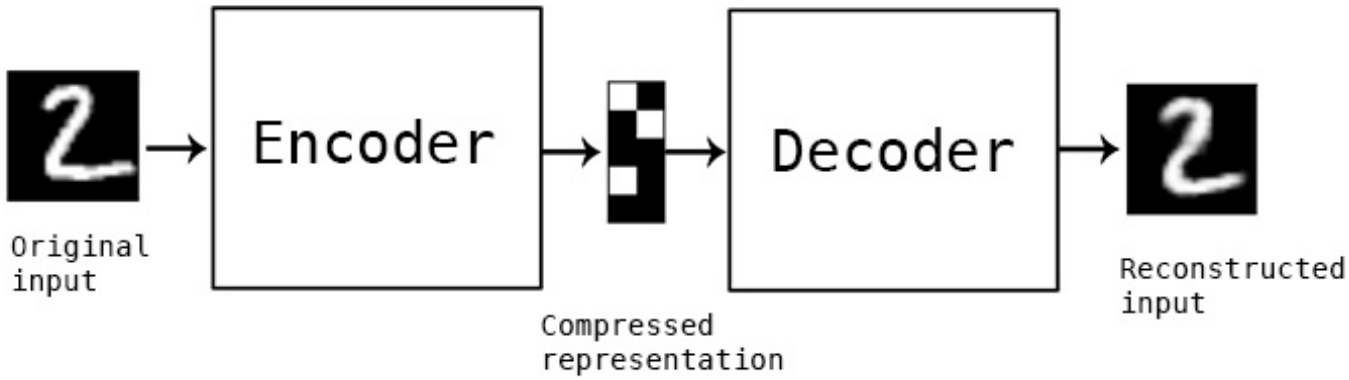
Uczenie nienadzorowane

Autoenkodery i RBM

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 pr. danych
- Modele sieci:
  - autoenkodery (AE)
  - Restricted Boltzman Machines (RBM)
  - GAN
- Zastosowanie:
  - kodowanie sygnału, wykrywanie istotnych cech, usuwanie szumu i rekonstrukcja sygnału
  - inicjowanie głębokich sieci DNN
  - generowanie sygnałów

Autoenkodery



- kodują cechy specyficzne dla danych treningowych, nie generalizują
- kompresja stratna - wymagamy by sygnał nie był jednostkowym odwzorowaniem
- w sposób automatyczny tworzą reprezentację danych, modelują rozkład prawdopodobieństwa sygnału
- koder i dekoder może być siecią neuronową, uczenie wsteczną propagacją błędu

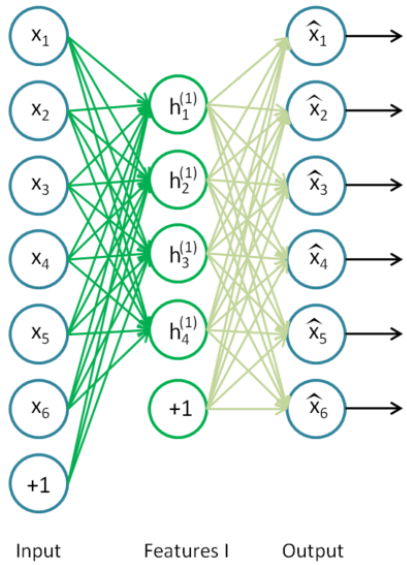
Rys: ☞ *Building Autoencoders in Keras*

Autoenkodery

Encoder:

$$\mathbf{h} = f(\mathbf{x})$$

$$= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$



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 wejściowym, detektor cech, skompresowana reprezentacja danych
- często $\mathbf{W}' = \mathbf{W}^T$ współdzielone wagi

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 $\hat{\mathbf{x}} = \mathbf{W}'\mathbf{h} + \mathbf{b}'$

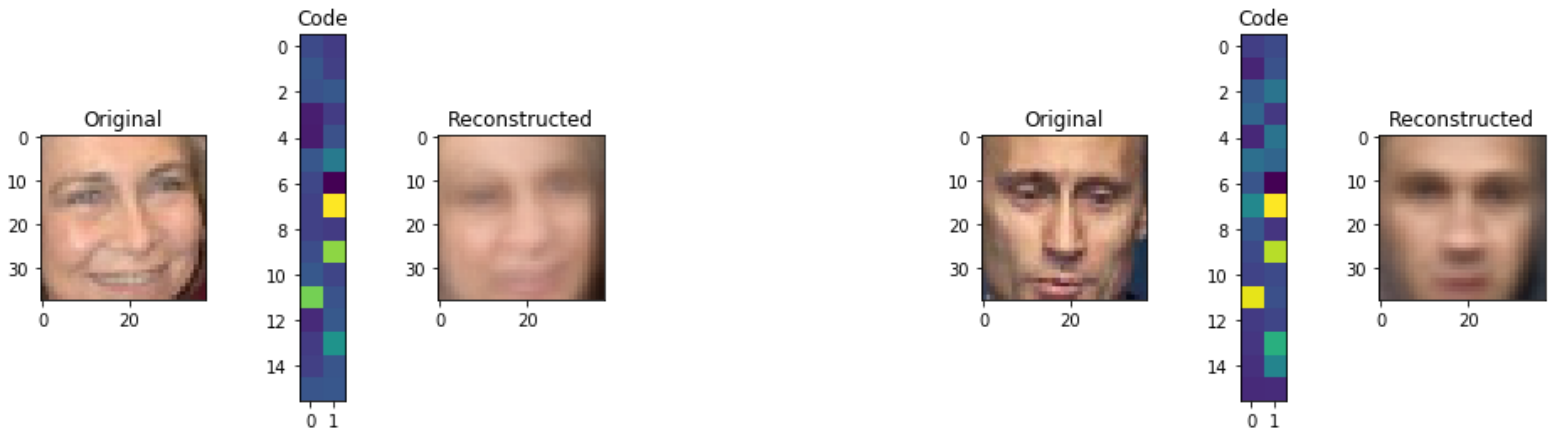
- Cross Entropy dla danych wejściowych binarnych

$$L(\mathbf{x}, \hat{\mathbf{x}}) = -\sum x_k \log \hat{x}_k + (1 - x_k) \log (1 - \hat{x}_k)$$

- trening wsteczną propagacją błędu
- odwzorowanie jednostkowe nie jest pożądane, dlatego trening posiada różne ograniczenia zapobiegające utworzeniu takiego odwzorowania

Niekompletny autokoder

- **Niekompletny AE** (*undercomplete*) gdy warstwa **h** mniejsza od rozmiaru wejścia,
- kompresja sygnału do rozmiaru **h**
- dla liniowego dekodera wynik jest równoważny z PCA
- Przykład: input-output: 32x32 , hidden (code size): 32



src: ☞ *Manash's blog*

Regularyzowany AE

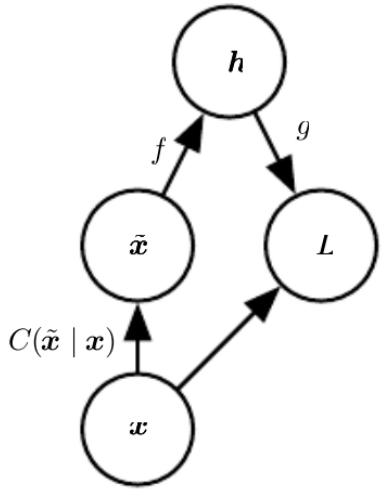
- wielkość **h** decyduje o pojemności ale za duża pojemność sprawia, że AE nie jest w stanie wyekstrahować wartościowej informacji o rozkładnie sygnału wejściowego (wejścia mogą być kopiowane na wyjście)
- **Nadkompletny** AE (*overcomplete*) gdy rozmiar **h** większy od **x**,
  - brak kompresji
  - nie gwarantują uzyskania wartościowych reprezentacji bez dodania odpowiedniej regularyzacji wymuszającej odpowiednią reprezentację w **h** (np. rzadkość)
- **Rzadki autoencoder** z karą za nie rzadką reprezentację

$$L(\mathbf{x}, \hat{\mathbf{x}}) + \lambda \sum |h_i|$$

  - w celu uzyskania rzadkich aktywacji można też użyć ReLU (Glorat, 2011)

Autoenkoder z odszumianiem (DAE)

- **Denoise autoencoder (DAE)** usuwa szum (lub inne zniekształcenia) sygnału wejściowego.
- Sygnał wejściowy $\tilde{\mathbf{x}}$ jest zniekształconą kopią $\mathbf{x}$ a AE uczy się odtwarzać oryginalny sygnał bez deformacji



src: ☞ *Building Autoencoders in Keras*

Kurczliwy AE

- **Contractivw autoenkoder (CAE)**

$$L(\mathbf{x}, \hat{\mathbf{x}}) + \lambda ||\nabla_{\mathbf{x}} \mathbf{h}(\mathbf{x})||_F^2$$

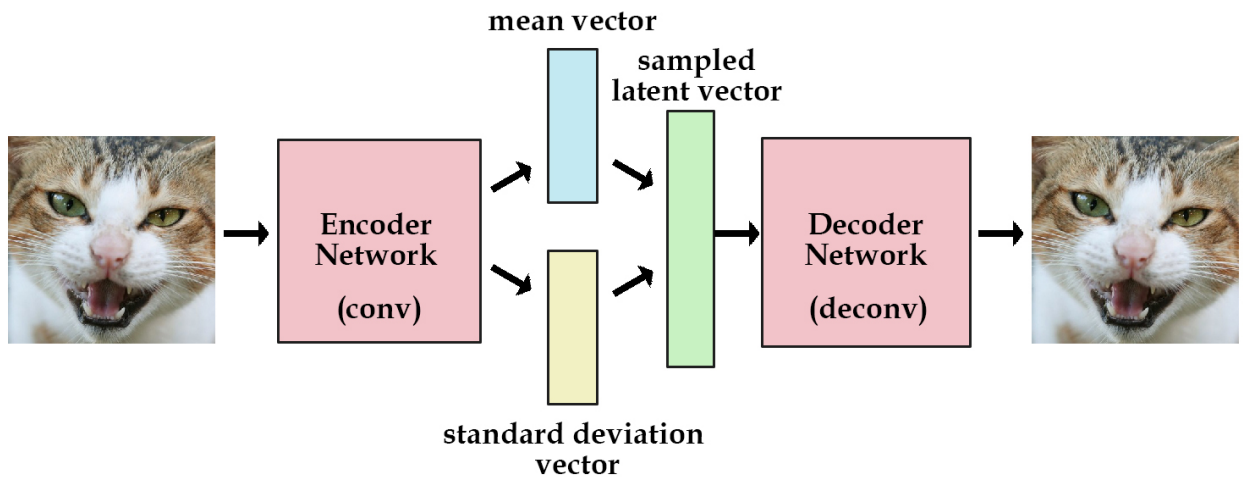
- kara za dużą zmienność reprezentacji przy zmianie sygnału (norma Frobeniusa)

$$||\nabla_{\mathbf{x}} \mathbf{h}(\mathbf{x})||_F^2 = \sum_i \sum_j \left(\frac{\partial h_i(x)}{\partial x_j} \right)^2$$

S. Rifai, et. al, (2011) Contractive Auto-Encoders:Explicit Invariance During Feature Extraction

Wariacyjny AE

- **Variational autoencoders (VAE)** jawnie modeluje rozkład gęstości kodowanej reprezentacji (np. σ_i, μ_i rozkładu $N(\mu_i, \sigma_i)$)



- sygnał dekodera jest próbowany z nauczonego rozkładu
- model generatywny - możliwe próbkowanie danych wyjściowych

Kullback–Leibler divergence

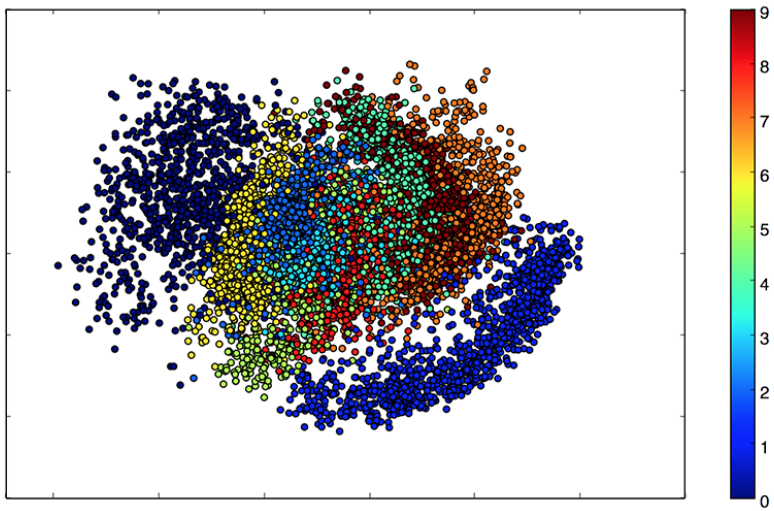
- do kosztu rekonstrukcji (np .MSE) minimalizuje człon regularyzujący wymuszający odpowiedni rozkład danych w warstwie kodującej (KL divergence), np. preferujący rozkład $N(0, 1)$

$$\sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1$$

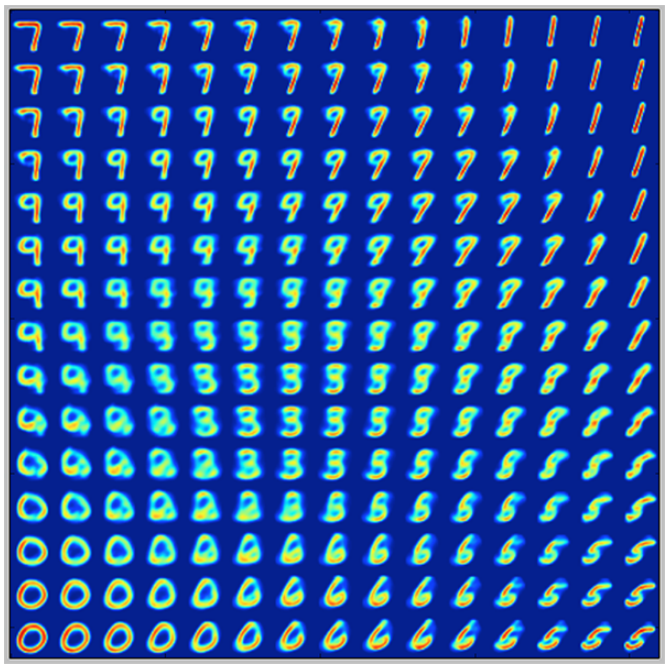
- minimum gdy $\sigma_i = 1, \mu_i = 0$
- wymusza rozkład zakodowanych danych w centrum przestrzeni ukrytej, minimalizacja kosztu rekonstrukcji konkuruje z regularyzajcą próbując rozdzielić gduły różnych danych

VAE na MNIST

wizualizacja danych treningowych w 2 wymiarowej warstwie ukrytej kodera



generowanie cyfr dla siatki 15x15 wartości 2 wymiarowej warstwy ukrytej dekodera

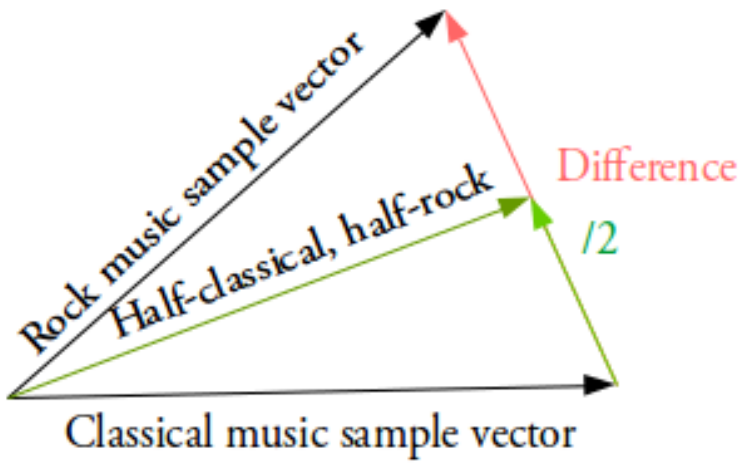


Architektura: Input-512-2-(μ, σ)-2-512-Output

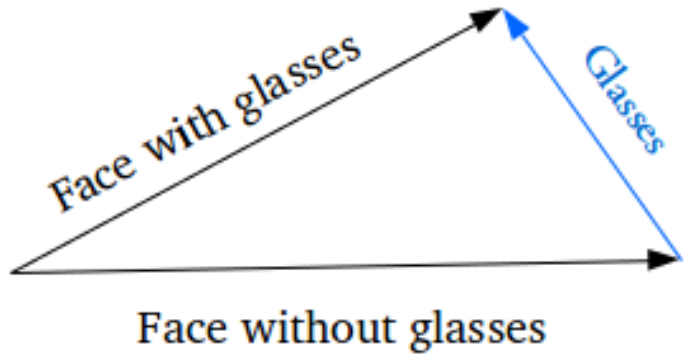
src: ☞ *Building Autoencoders in Keras*

Interpolowanie i generowanie pojęć

Arytmetyka w przestrzeni zmiennych utajonych



Środek między wektorami μ kodującymi pojęcia

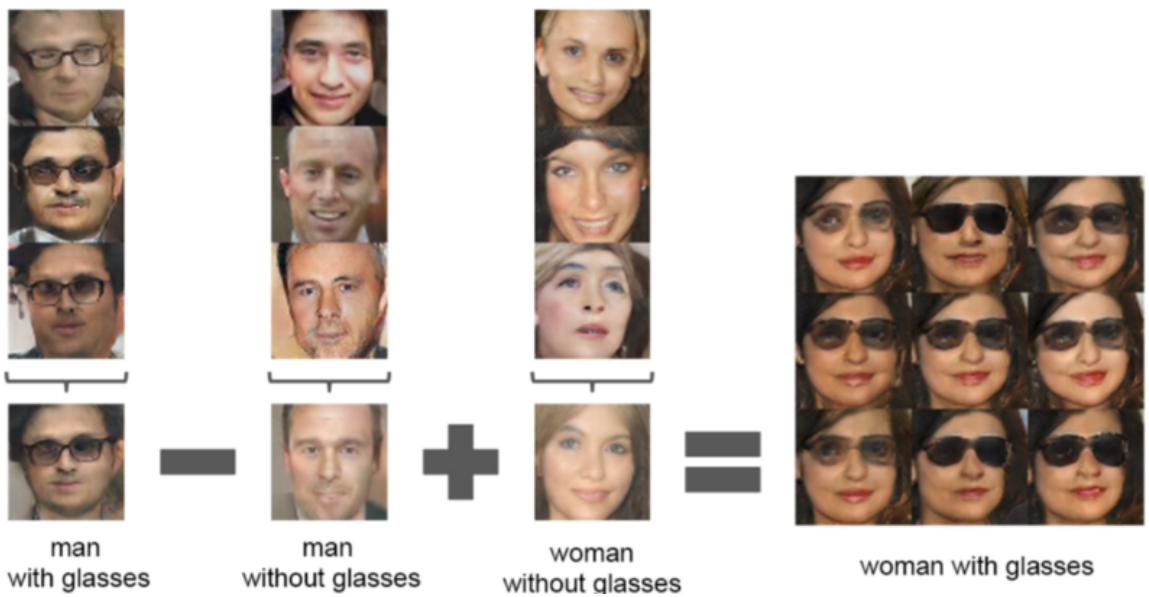


Róznica pomiędzy pojęciami może być dodana do innego pojęcia

☞ *Irhum Shafkat, Intuitively Understanding Variational Autoencoders*

Ukryta reprezentacja pojęć

- utworzona ukryta reprezentacja koduje pojęcia
- arytmetyczne operacja na wektorach kodujących odpowiadają relacjom semantycznym pomiędzy pojęciami



Przykład: generowanie twarzy

Wejście

Rekonstrukcja

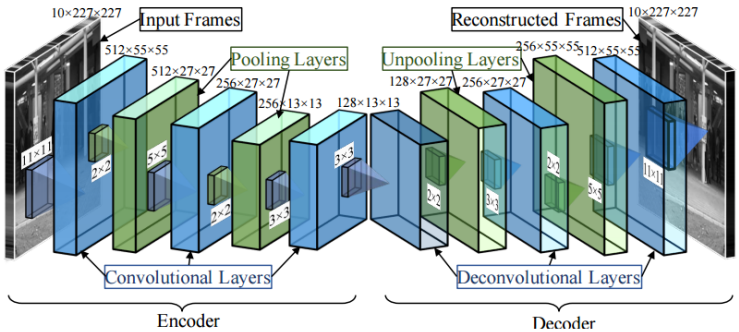
Losowe

☞ Deep Feature Consistent Variational Autoencoder

☞ *Variational auto-encoder trained on celebA*

Inne typy autoenkoderów

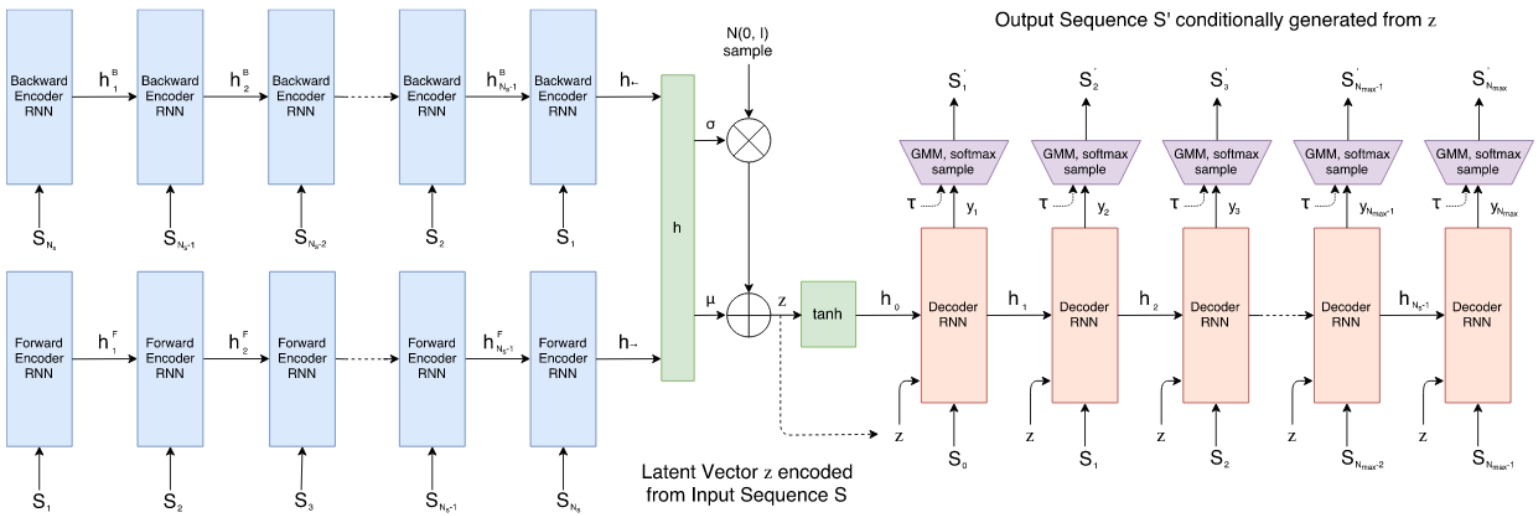
- Convolutional AE, dekoder jest symetryczną do enkodera siecią dekonwolucyną



- sequence-to-sequence AU (z użyciem np. LSTM)
- głębokie autoenkodery - więcej warstw sprzyja kompresji danych o złożonych rozkładach
- stacked autoenkoders - tworzenie głębokich sieci z uczonych sekwencyjnie autoenkoderów (metoda inicjalizacji)

Przykład: Sketch-RNN

Sequence-to-Sequence Variational Autoencoder do trenowania reprezentacji rysowania odręcznego

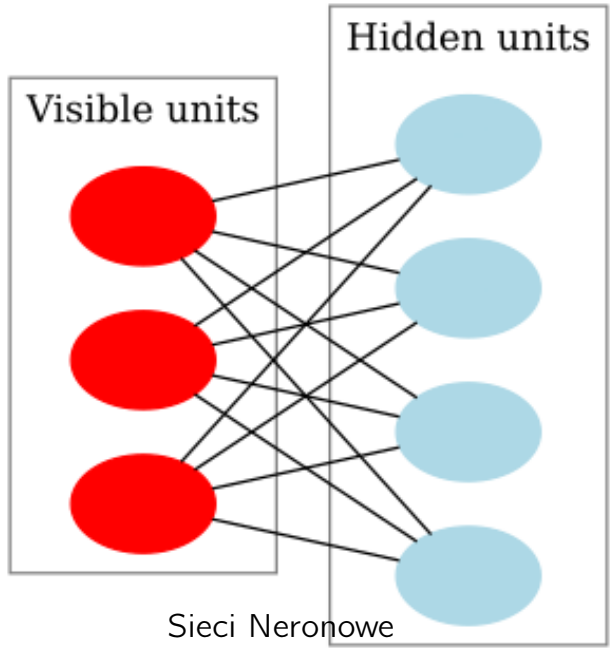


☞ Draw Together with a Neural Network

David Ha, Douglas Eck, A Neural Representation of Sketch Drawings

Restricted Boltzman Machine

- **Restricted Boltzman Machines (RBM)** (Hinton 2000), wcześniej Harmonium (P. Smolensky, 1986), to szczególny przypadek Maszyny Boltzmana
- stochastyczna sieć neuronowa modelująca rozkład prawdopodobienstw wejść
- *restrictive* - brak połączeń wewnątrz warstwy
- algorytm uczenia: *contrastive divergence* (Hinton)



Źródło grafiki: Wikipedia

Budowa RBM

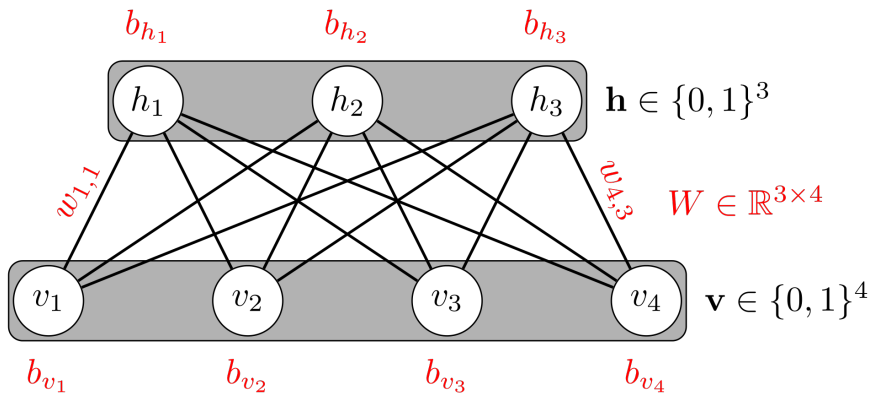
- binarne jednostki umieszczone w dwóch warstwach: widzialnej $\mathbf{v}$ i ukrytej $\mathbf{h}$
- sygnał wejściowy podawany do jednostki widzialnej $\mathbf{v} = \mathbf{x}$
- energia układu

$$E(\mathbf{x}, \mathbf{h}) = -\sum a_i x_i - \sum b_i h_i - \sum \sum x_i w_{ij} h_j$$

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

- prawdopodobieństwo marginalne

$$P(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}$$

- brak połączeń wewnątrz warstw, więc stany w warstwach są niezależne

$$P(\mathbf{h}|\mathbf{x}) = \prod_{j=1}^{n} P(h_j|\mathbf{x}), \qquad P(\mathbf{x}|\mathbf{h}) = \prod_{i=1}^{m} P(v_i|\mathbf{h})$$

$$P(h_i = 1|\mathbf{x}) = \sigma(\mathbf{w}_i\mathbf{x} + b), \qquad P(x_k = 1|\mathbf{h}) = \sigma(\mathbf{h}^T\mathbf{w}_k + c)$$

Contrastive divergence CD

- maksymalizacja prawdopodobieństwa $P(\mathbf{x})$

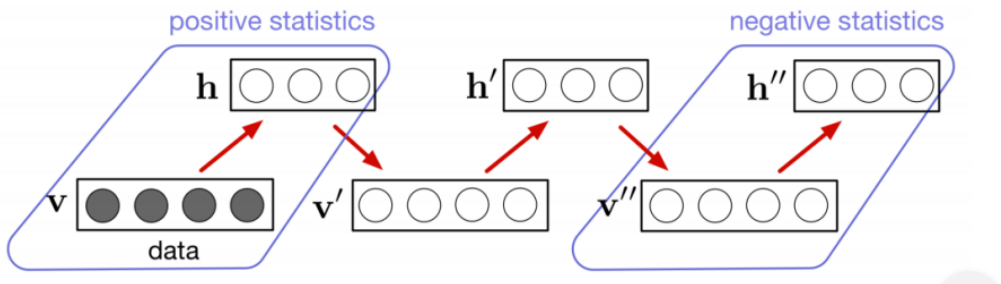
$$L = \frac{1}{N} \sum \log P(\mathbf{x})$$

- *Contrastive divergence* (Hinton) metoda przybliżona korzystająca z samplowania Gibbsa
  - dla każdego $\mathbf{x}$ wejściowego wygeneruj $\mathbf{x}'$ z k krotnego samplowania Gibbsa
  - aktualizacja

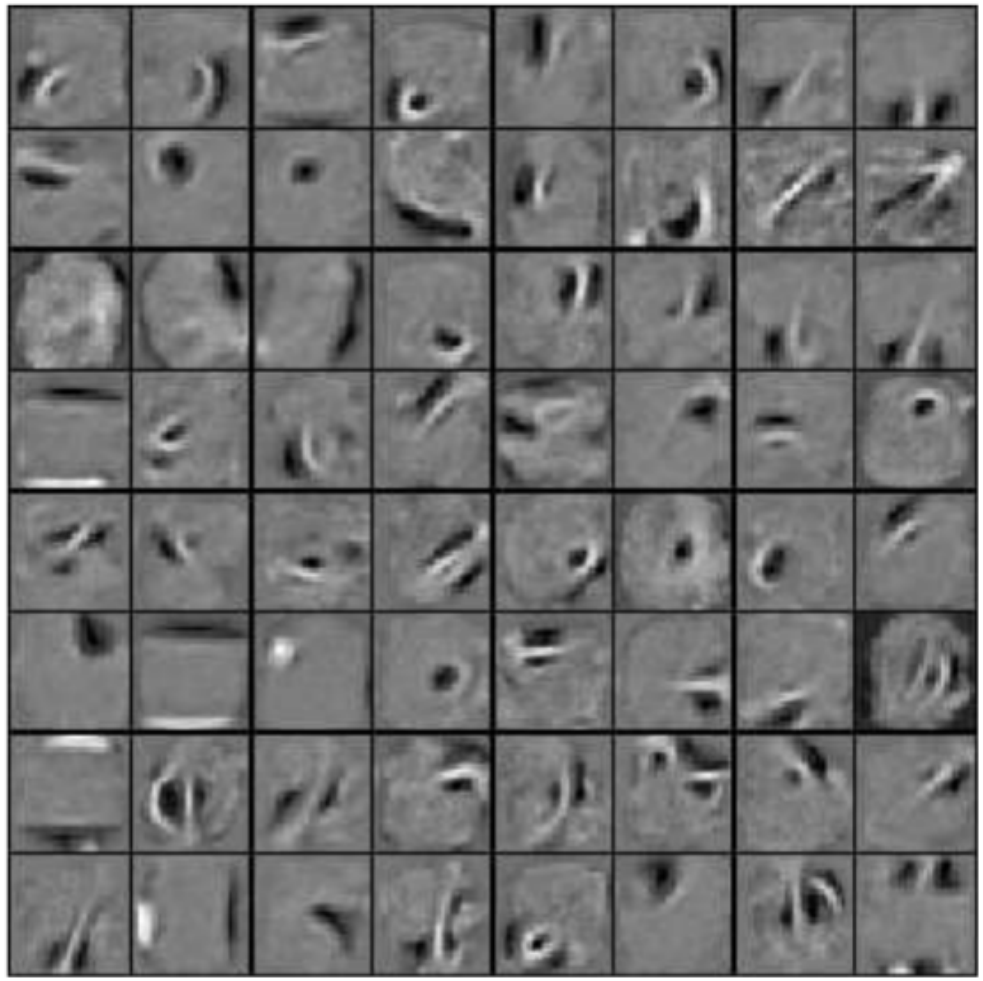
$$\Delta \mathbf{W} = \epsilon(\mathbf{x}\mathbf{h}^T - \mathbf{x}'\mathbf{h}'^T)$$

$$\Delta \mathbf{a} = \epsilon(\mathbf{x} - \mathbf{x}'), \qquad \Delta \mathbf{b} = \epsilon(\mathbf{h} - \mathbf{h}')$$

- w praktyce $k = 1$ daje dobre cechy

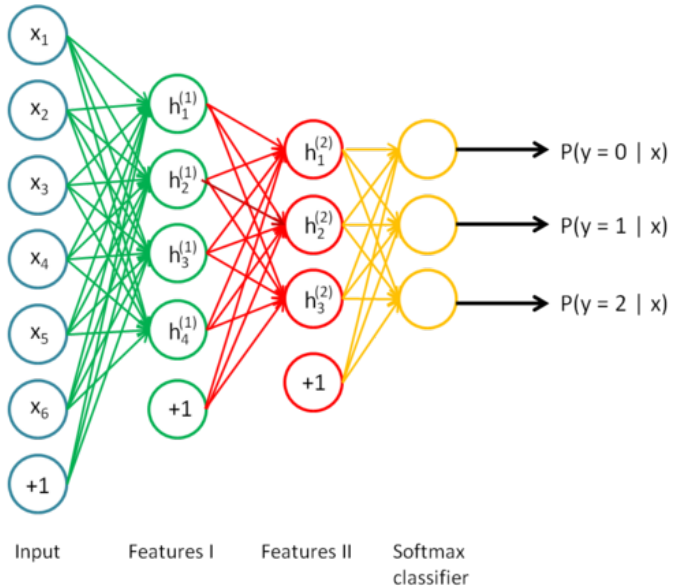


Filtry RBM na MNIST



Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

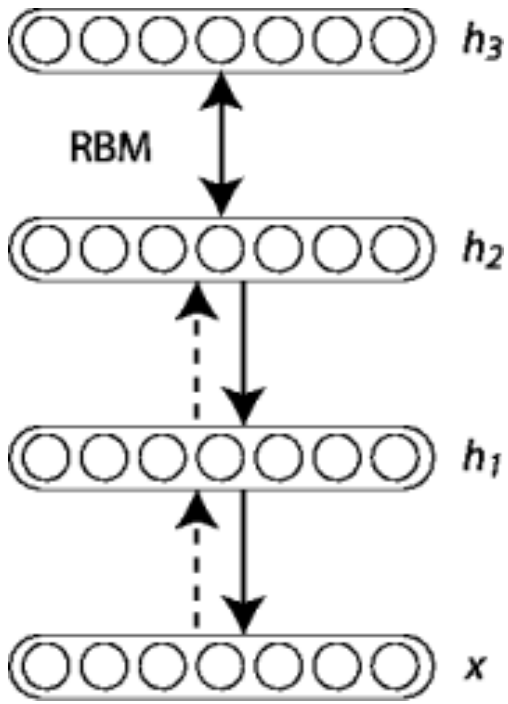
Stacked autoencoders



- Sygnał na wyjściu warstwy ukrytej jest wejściem kolejnego autoenkodera
- Hierarchia reprezentacji: pierwsza warstwa koduje najprostsze warstwy, kolejne bardziej ogólne relacje, itd.
- Podobna architektura: wielowarstwowy RBM to Deep Belief Networks

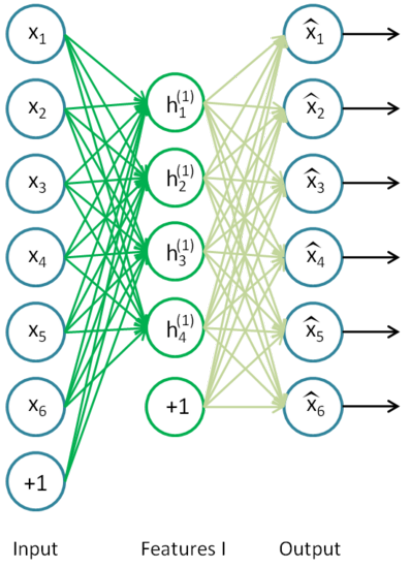
Deep Belief Networks (DBN)

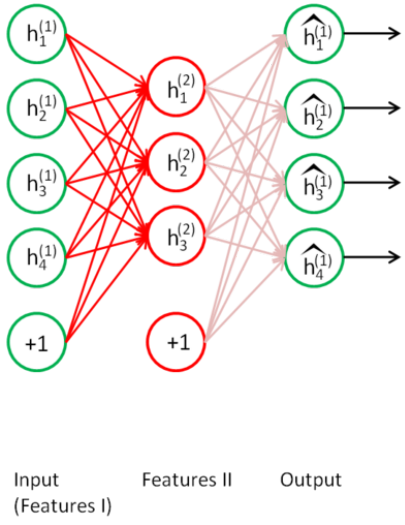
- DBN (Teh, Hinton 2006) wielowarstwowy RBM,
- uczy się głębokiej, hierarchicznej reprezentacji danych, każda kolejna warstwa analizuje sygnał widoczny w warstwie ukrytej poprzedniego RBM
- uczenie zachłanne: kolejne warstwy dodawane i uczone pojedynczo
- pierwszy efektywny model głęboki.
- to nie jest sieć jednokierunkowa

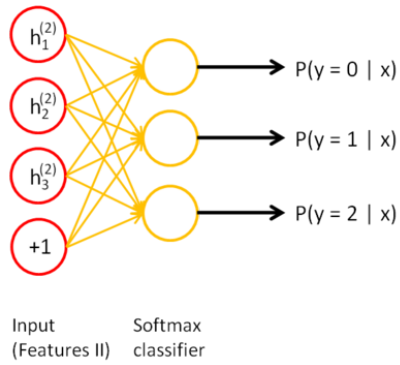


src: *http://deeplearning.net/tutorial/DBN.html*

Inicjalizacja sieci DNN



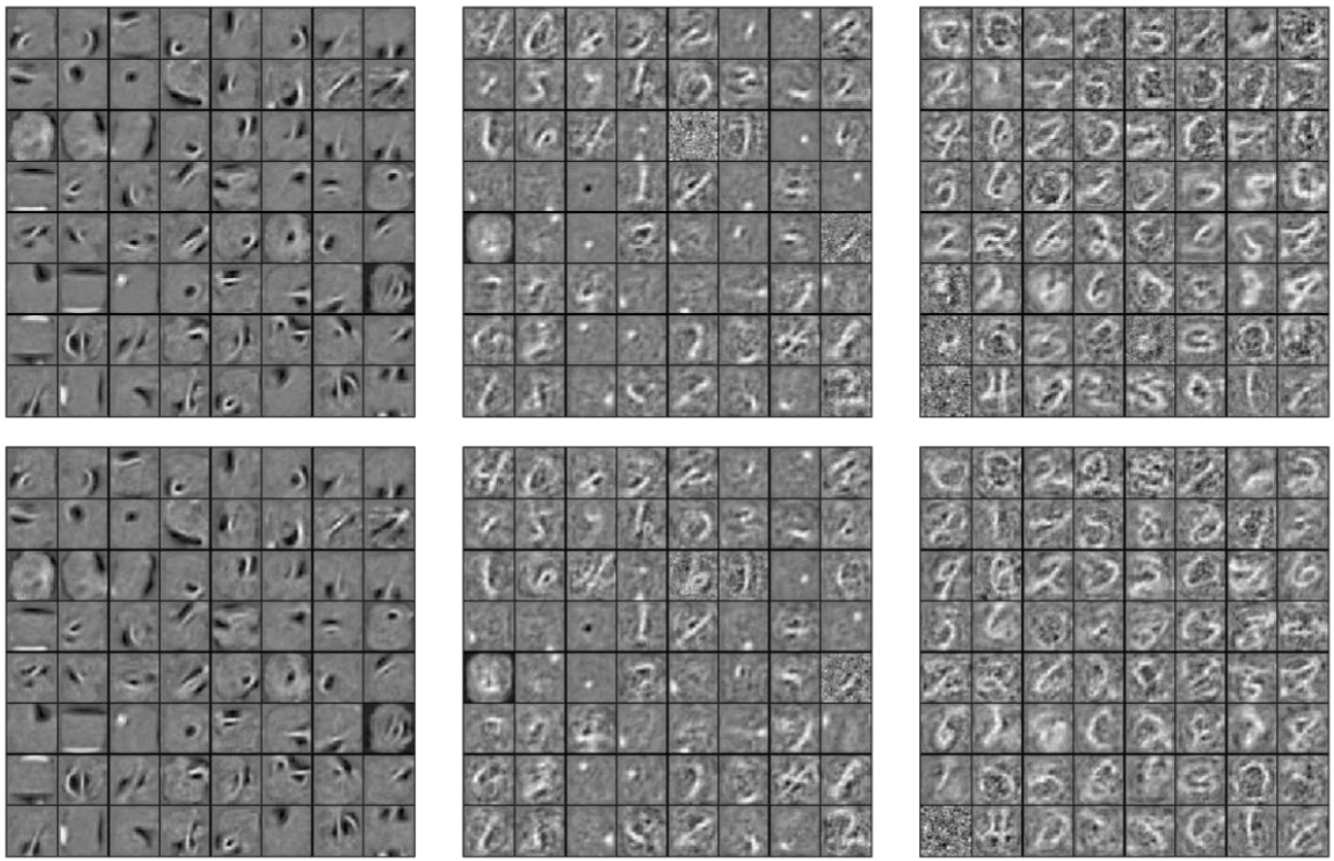




1. *Greedy layer-wise pre-training*:
   - trening nienadzorowany autoenkodera wsteczną propagacją
   - usuwamy warstwę wyjściową (dekoder) i uczymy kolejny AE, którego wejściem jest sygnał z warstwy ukrytej poprzedniego AE przy niezmiennych wartościach wag w poprzednich warstwach
   - powtarzamy procedurę dla każdej kolejnej warstwy
2. *fine tuning*: dodajemy w pełni połączoną warstwę wyjściową (lub kilka warstw) i uczymy całą sieć w sposób nadzorowany

Filtry DBN na MNIST

pre-training

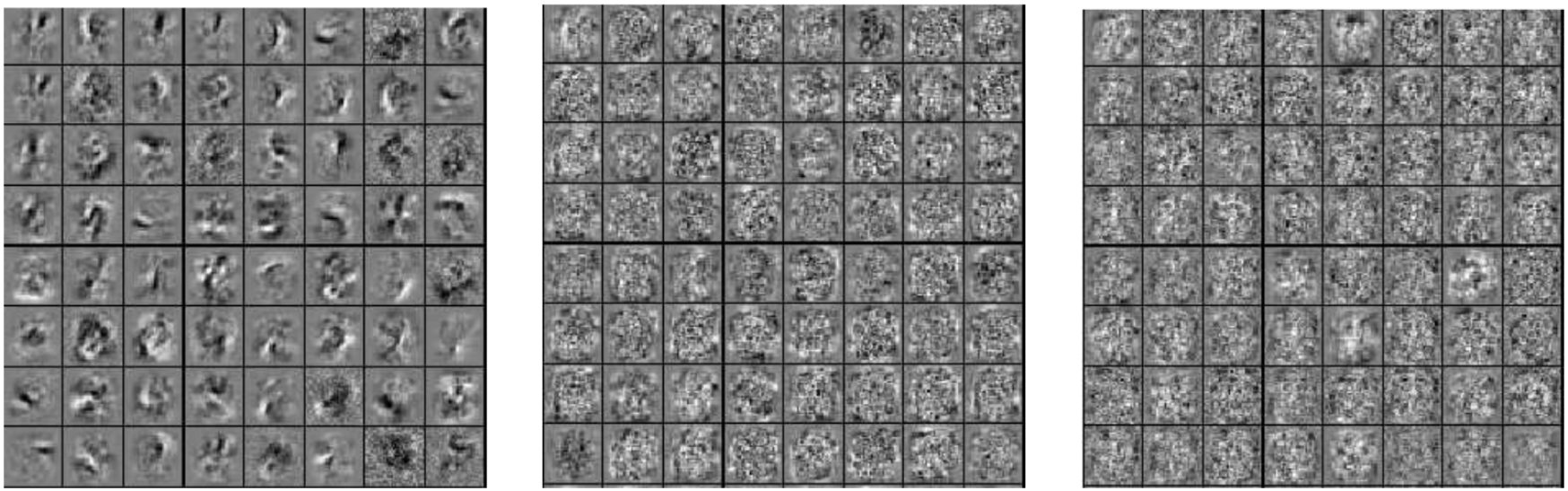


Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

fine tune

Filtry DBN na MNIST

without pre-training



Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

Wyniki MNIST, Bengio 2007

	train.	valid.	test
DBN, unsupervised pre-training	0%	1.3%	1.4%
Deep net, auto-associator pre-training	0%	1.4%	1.4%
Deep net, supervised pre-training	0%	1.75%	2.0%
Deep net, no pre-training	.004%	2.1%	2.4%
Shallow net, no pre-training	.004%	1.8%	1.9%

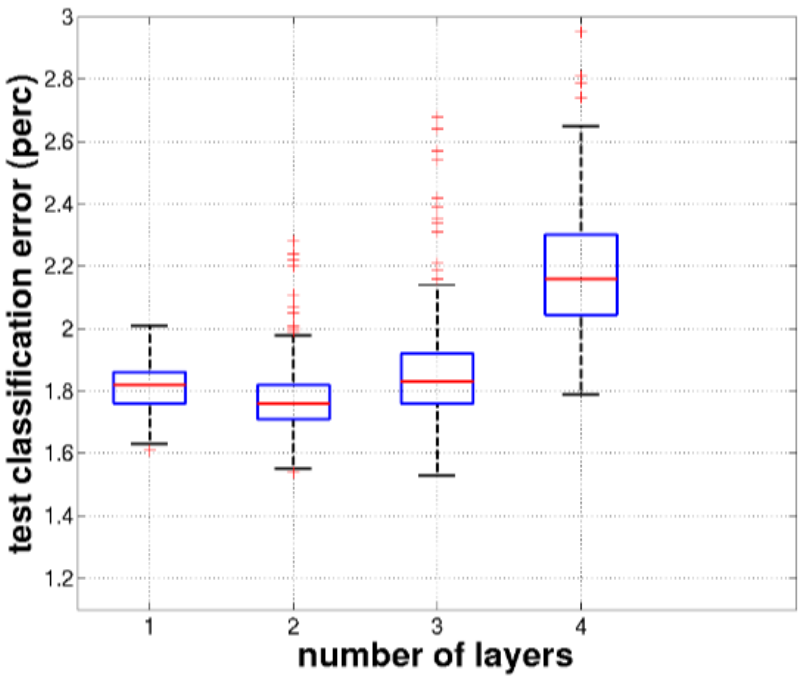
Table 2: *Classification error on MNIST training, validation, and test sets, with the best hyper-parameters according to validation error, with and without pre-training, using purely supervised or purely unsupervised pre-training.*

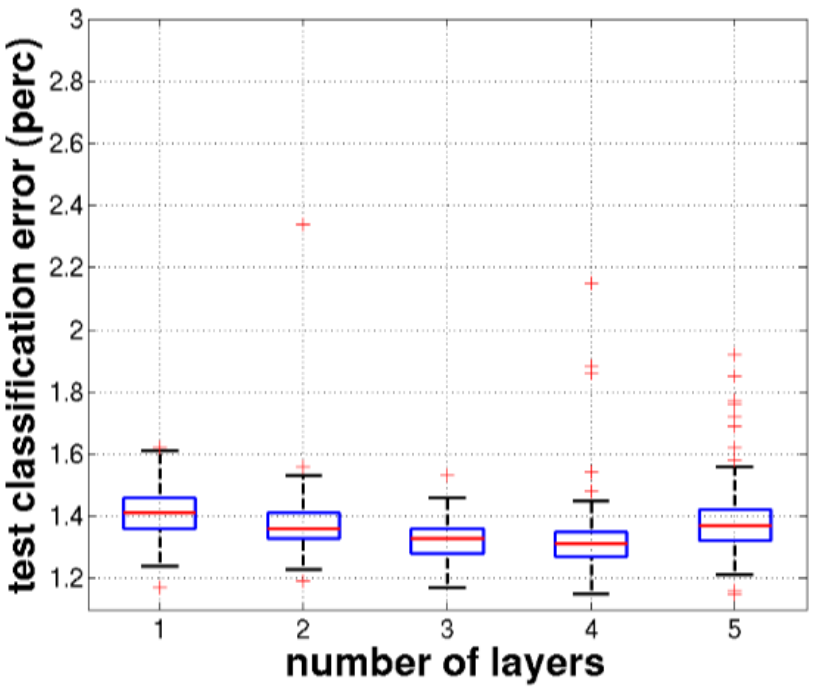
- architektura: 784 wejść, 10 wyjść, 3 warstwy ukryte
- na MNIST 0.1% statystycznie istotna różnica

Bengio, et al., Greedy layer-wise training of deep networks, 2007.

Pre-trening DNN

Pre-trening poprawia generalizację, działa jak regularyzacja

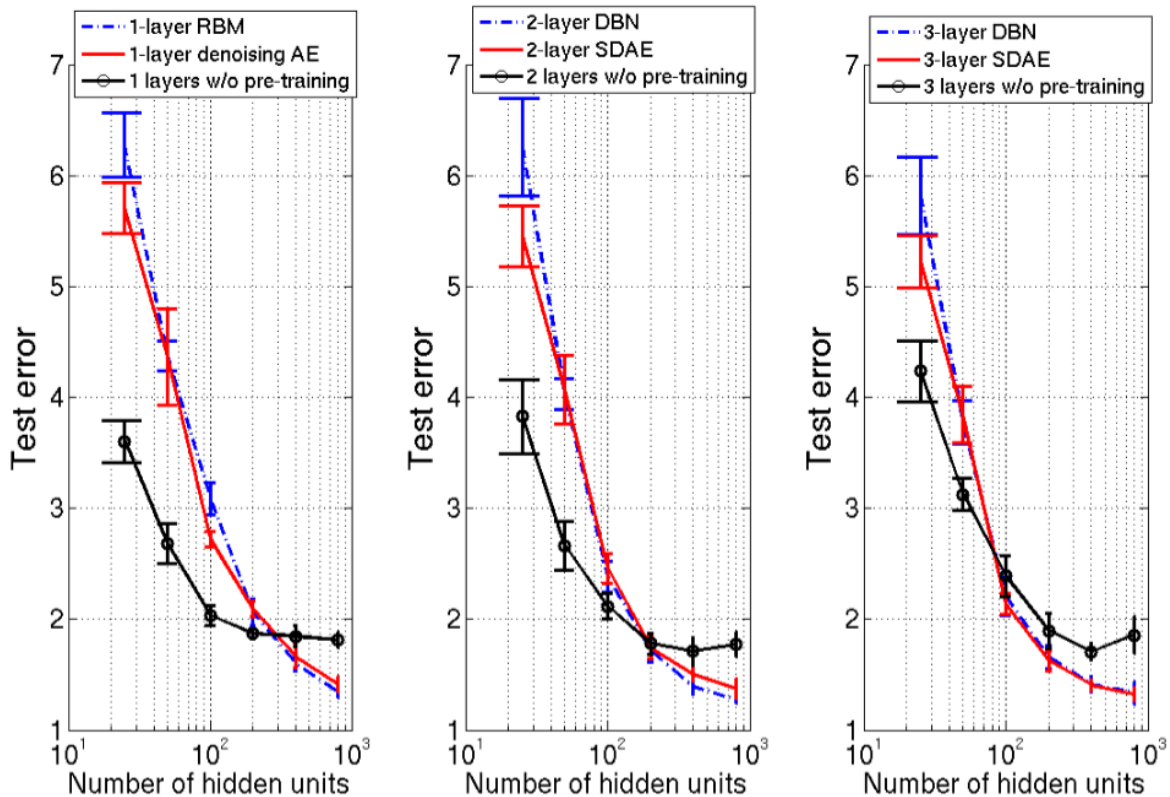




Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

Pre-trening DNN

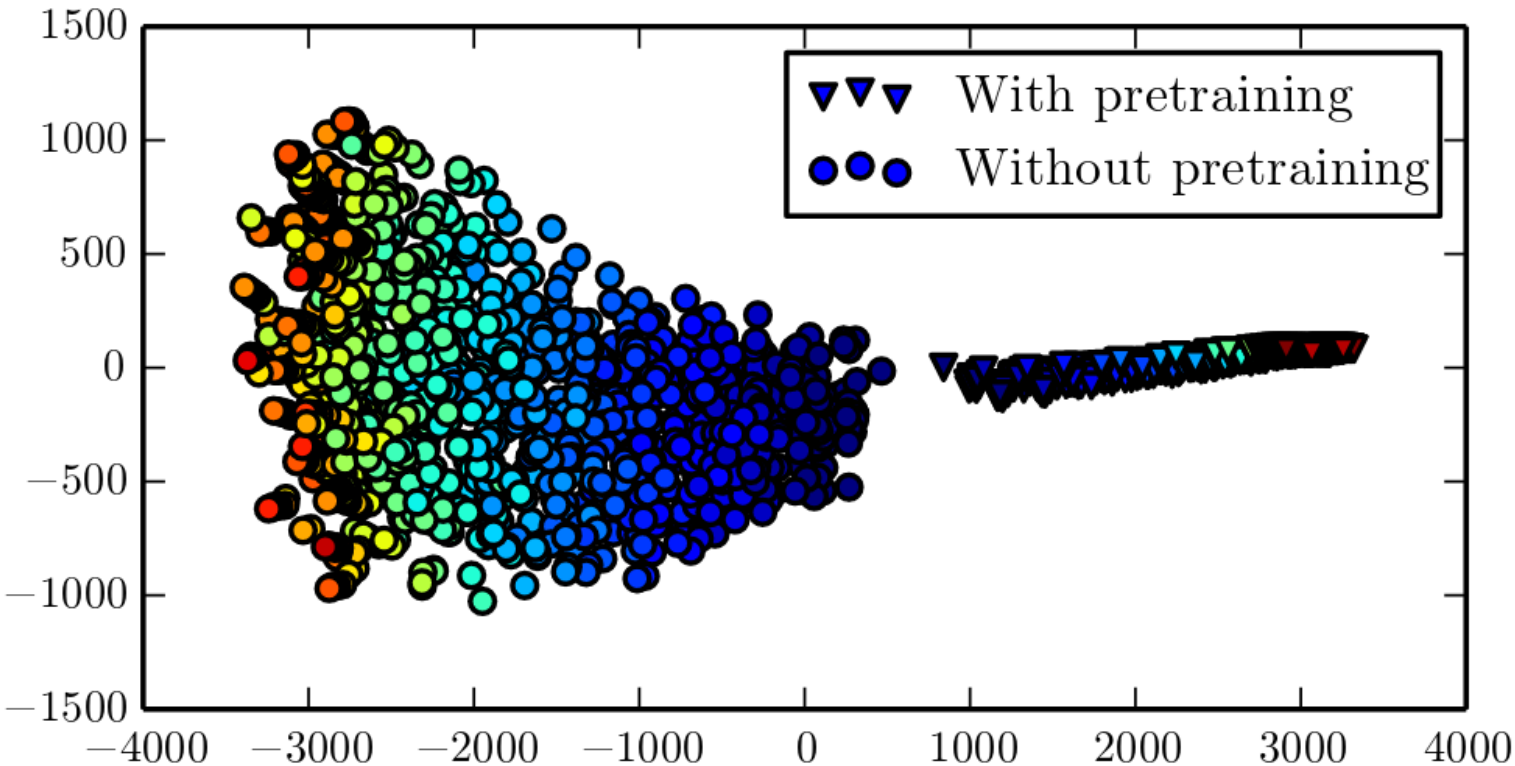
Wartość pre-treningu wzrasta z liczbą warstw



Erhan, Bengio, (2010) Why Does Unsupervised Pre-training Help Deep Learning?

Pre-training DNN

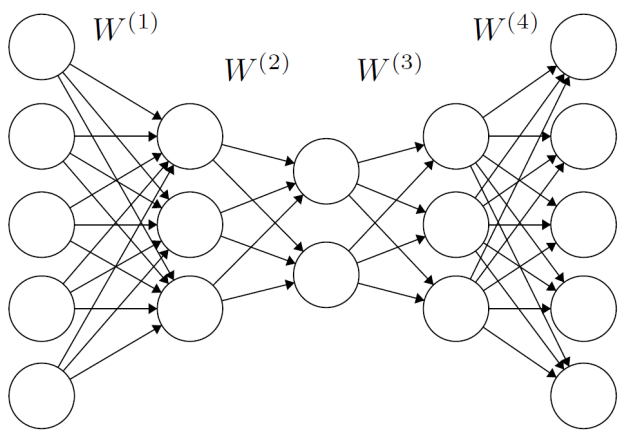
Pre-training ma wpływ na trajektorię uczenia



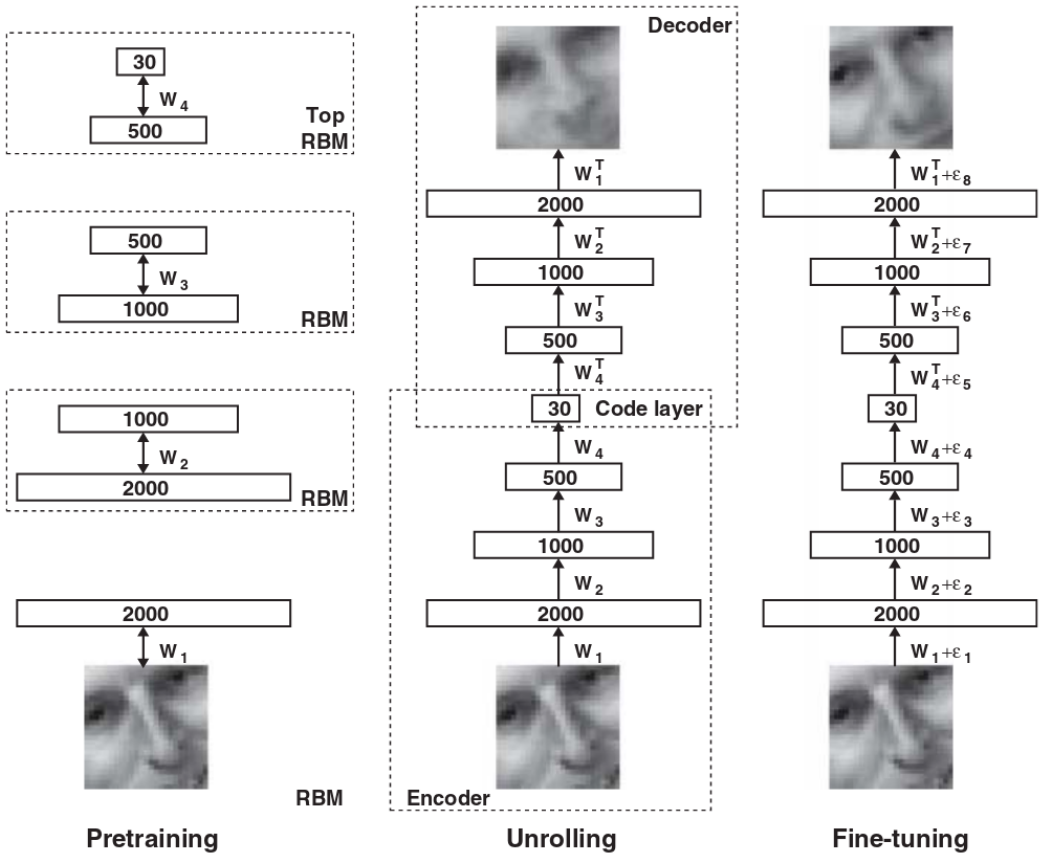
Goodfelow, 2016

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



Pretraining dla głębokich AE



- RBM używany do inicjacji kodera i dekodera

G. Hinton, R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science 2006

Kompresja obrazów: Deep AE vs. PCA

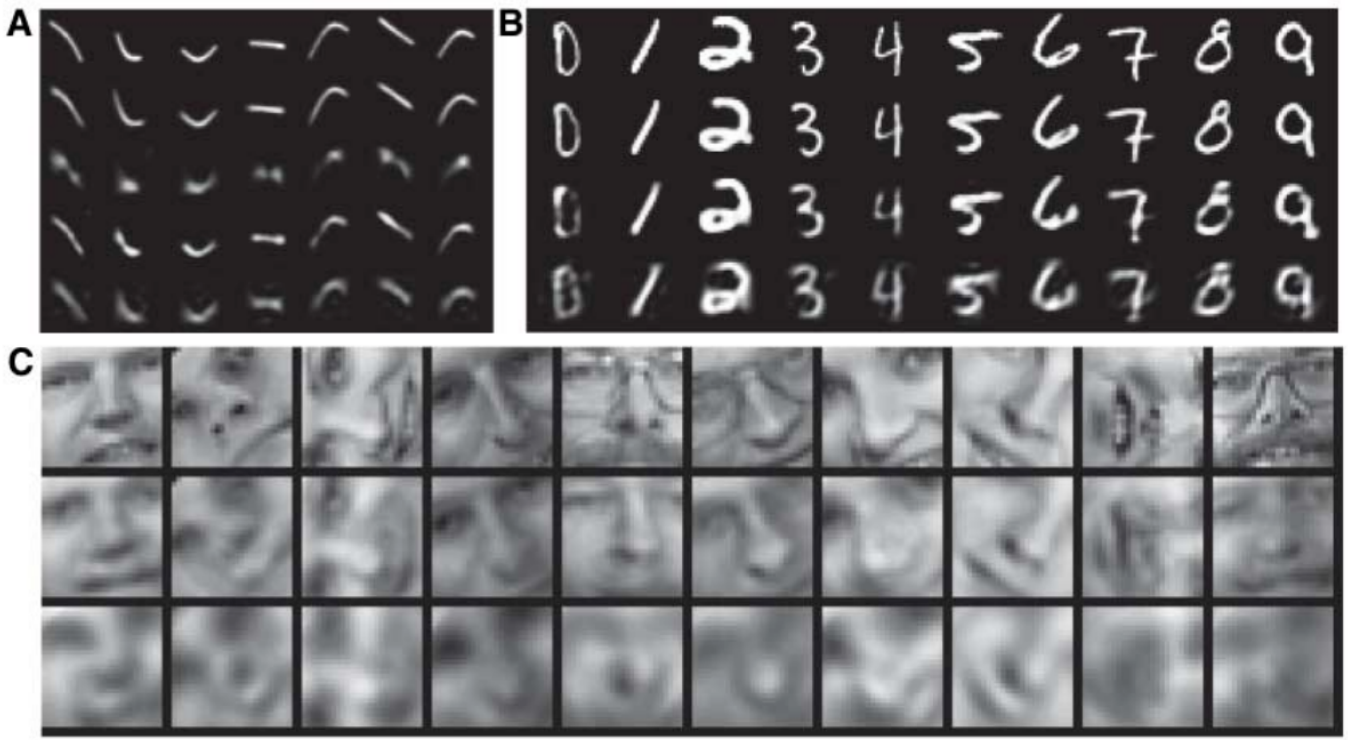
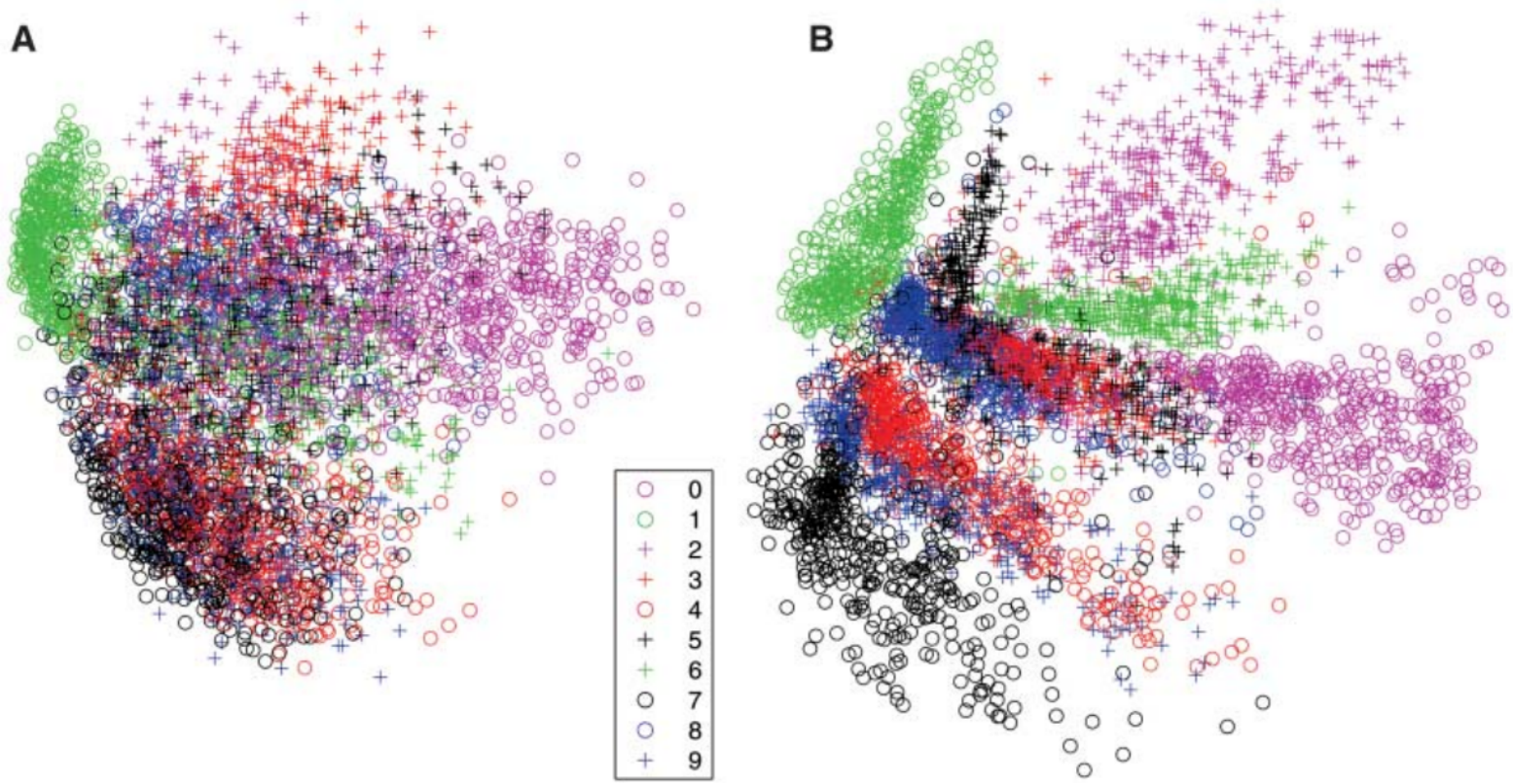


Fig. 2. (**A**) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by "logistic PCA" (*8*) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error per image for the last four rows is 1.44, 7.64, 2.45, 5.90. (**B**) Top to bottom: A random test image from each class; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional logistic PCA and standard PCA. The average squared errors for the last three rows are 3.00, 8.01, and 13.87. (**C**) Top to bottom: Random samples from the test data set; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.

G. Hinton, R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science 2006

Projekcja do 2D (dwa neurony kodujące)

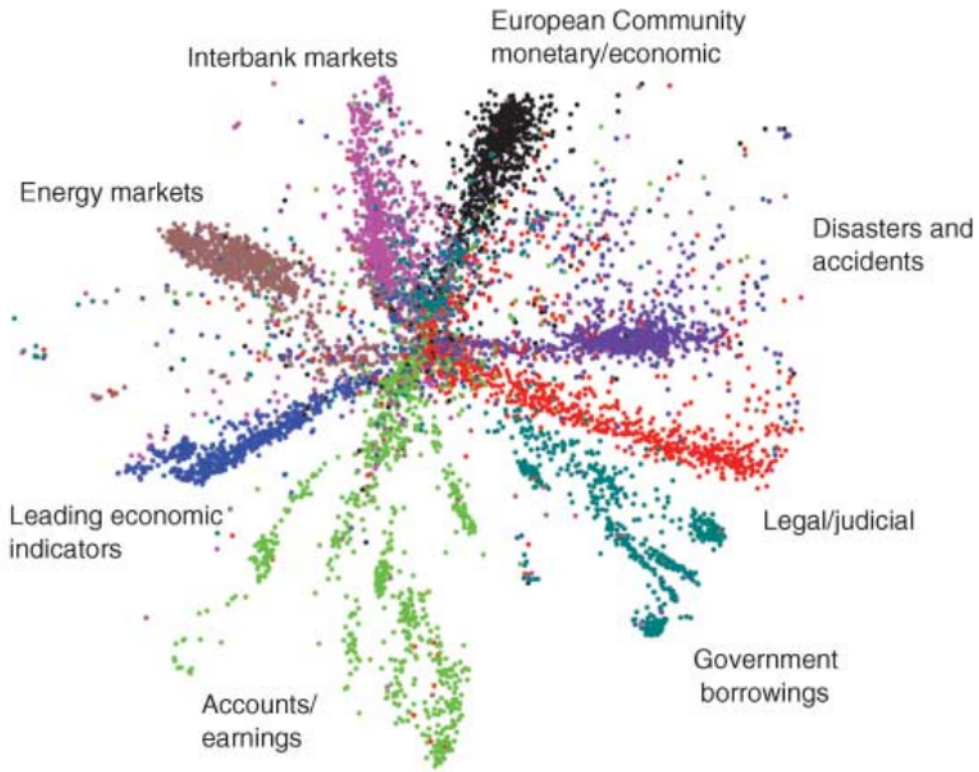
MNIST: PCA vs. Deep AE (784-1000-500-250-2)



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

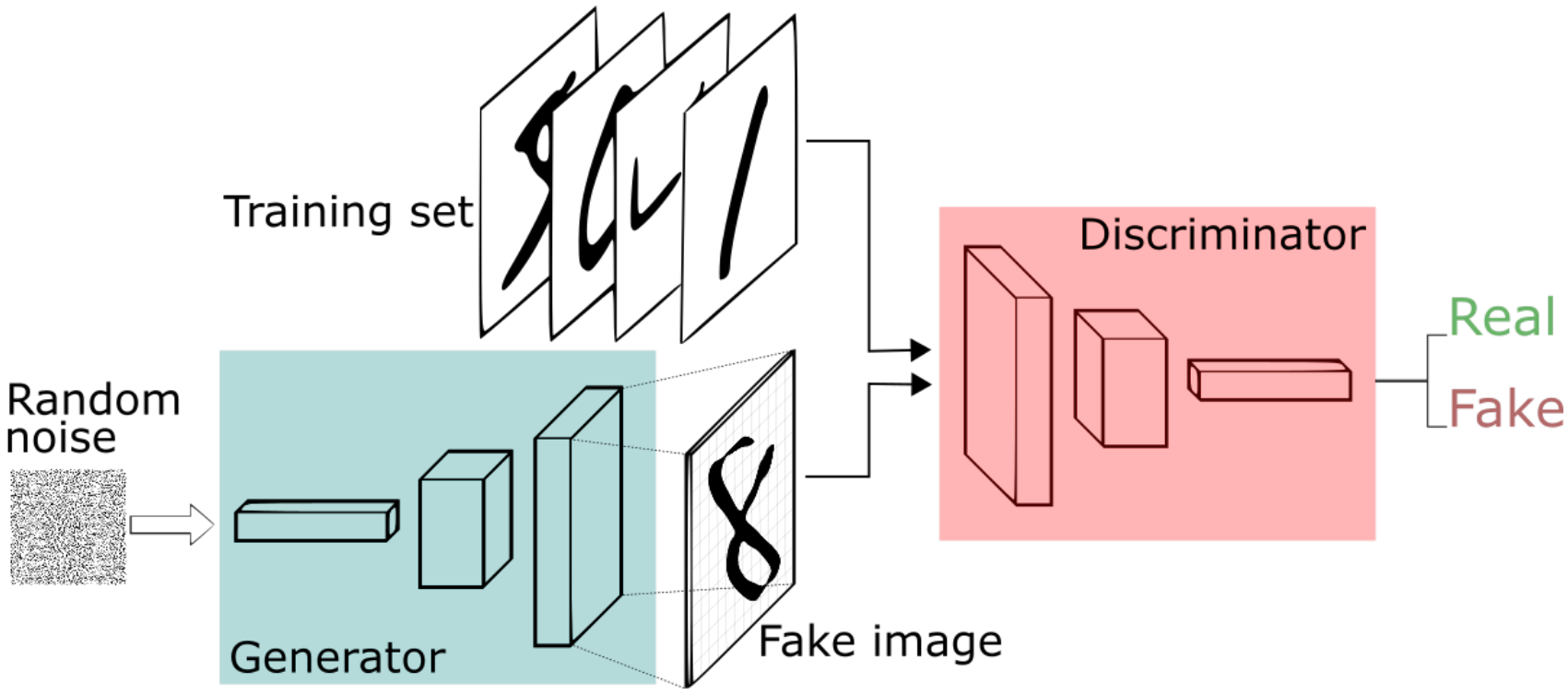


G. Hinton, R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science 2006

Generative Adversal Network (GAN)

- GAN (Goodfelow 2014) - dwie sieci (generująca i oceniająca) rywalizujące ze sobą w grze o sumie zerowej
- model uczy się generować dane o właściwościach zbliżonych do zbioru treningowego, np. generowanie zdjęć o realistycznych cechach
- sieć oceniająca (dyskryminująca, np. CNN) stara się odróżnić prawdziwy sygnał od wygenerowanego przez siec generującą
- sieć generująca (np. sieć dekonwolucyjna) tworzy sygnał z pewnego rozkładu starając się „oszukać" sieć oceniającą, dąży do maksymalizacji błędu dyskryminacji
- obie sieci uczone wsteczną propagacją, sieć generująca twory coraz bardziej realistyczne obrazy, sieć oceniająca specjalizuje się w rozróżnianiu coraz subtelniejszych różnic pomiędzy obrazami prawdziwymi i wygenerowanymi

GAN



src: *https://deeplearning4j.org/generative-adversarial-network*

Trening GAN

Prosta funkcja kosztu GAN - Cross Entropy 2 klas dla dyskryminatora

$$L^{(D)} = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{ data (x)}}} \log D(\mathbf{x}) - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})} \log(1 - D(G(\mathbf{z})))$$

gdzie $p_{data}(\mathbf{x})$ rozkład danych prawdziwych, $p_z(\mathbf{z})$ rozkład danych zaszumionych

Funkcja kosztu generatora

$$L^{(G)} = -\frac{1}{2}\mathbb{E}_z \log D(G(\mathbf{z}))$$

maksymalizuje prawdopodobieństwo popełnienia błedu przezez dyskryminator

Przykłady

DCGAN (Denton et al., 2015) generowanie obrazów wysokiej rozdzielczości

Goodfwlow, Deep Learning, 2016

Zwiększanie rozdzielczości obrazów

Rys: ☞ *Understanding and building Generative Adversarial Networks(GANs)*

StyleGAN

Generowanie twarzy

Rys: ☞ *StyleGAN - Official TensorFlow Implementation*

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 cybenetycznych, 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