



Wrocław  
University  
of Science  
and Technology



Zintegrowany  
Program Rozwoju  
Politechniki Wrocławskiej

# SIECI NEURONOWE



HR EXCELLENCE IN RESEARCH



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Politechnika Wrocławskiego

Unia Europejska  
Europejski Fundusz Społeczny





# Sieci neuronowe

## Wstęp

**URSZULA MARKOWSKA-KACZMAR**  
Katedra Inteligencji Obliczeniowej  
Wydział Informatyki i Zarządzania



# Opis wymagań formalnych

## WYKŁAD

- Kurs składa się z :
  - 30 godzin wykładu,
  - 30 godzin laboratoriów.Zorganizowanych w ciągu 10 tygodni,
- Wykład kończy się zaliczeniem na ocenę
- kolokwium 24 listopada 2022, kolokwium poprawkowe 14 grudnia 2022 ).
- Kolokwium w postaci pytań testowych i 1-2 pytania otwarte. Mogą być także zadania obliczeniowe.
- Nie przewiduje się kolejnych terminów poprawkowych.
- Premiowanie aktywności na wykładzie
- Zajęcia laboratoryjne obowiązkowe
- Godziny konsultacji:
  - poniedziałki 12-13
  - czwartki 14-15



# Cele kursu

- Dobre **rozumienie modeli** prezentowanych na wykładzie.
- Praktyka w implementacji i trenowaniu podstawowych modeli sieci neuronowych
- Praktyczna znajomość technik użytecznych w trenowaniu modeli,
- Poznanie interesujących zastosowań i dedykowanych do ich realizacji typów sieci.



# Zawartość merytoryczna wykładu

**Wykład 1.** Wstęp, pierwsze proste modele.

**Wykład 2.** Sieć wielowarstwowa, metoda propagacji wstecznej w ujęciu klasycznym.

**Wykład 3.** Metoda propagacji wstecznej w ujęciu macierzowym, projektowanie aplikacji bazujących na sieci neuronowej.

**Wykład 4.** Techniki użyteczne w uczeniu sieci neuronowych

- **Wykład 5.** Dobre praktyki w projektowaniu sieci
- **Wykład 6.** Przykłady zastosowań płytkich sieci neuronowych
- **Wykład 7.** Sieć konwolucyjna i jej uczenie
- **Wykład 8.** Sieci rekurencyjne
- **Wykład 9.** Sieci samoorganizujące się+kolokwium
- **Wykład 10.** Interesujące zastosowania sieci głębokich +kolokwium poprawkowe

Proszę sobie przypomnieć zasady

różniczkowania. Będą potrzebne na kolejnym wykładzie.

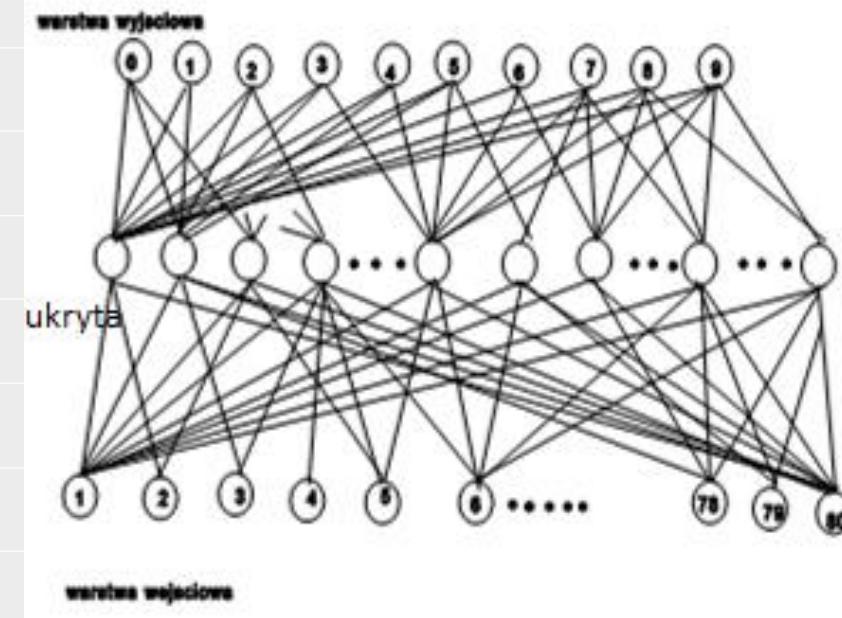
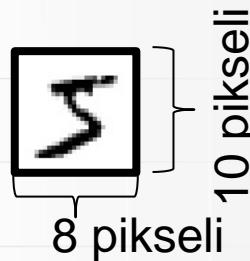
# Laboratorium

## Proponowane ćwiczenia

- **Ćwiczenie 1.** – implementacja i badania prostych sieci – 2 terminy laboratorium
- **Ćwiczenie 2.** – Sieć MLP do klasyfikacji cyfr ze zbioru MNIST, własna implementacja, poszukiwanie najlepszych hiperparametrów sieci.
- **Ćwiczenie 3.** – sieć konwolucyjna do rozpoznawania cyfr ze zbioru MNIST – implementacja, badania eksperymentalne w KERAS
- **Ćwiczenie 4.** – Sieć rekurencyjna KERAS

# Przykład przetwarzania cyfr

## INTUICJE



## WZORCE UCZĄCE



“5”



“0”



“4”



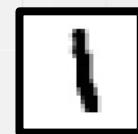
“1”



“9”



“2”



“1”



“3”



# Zalety sieci neuronowych

## ZALETY

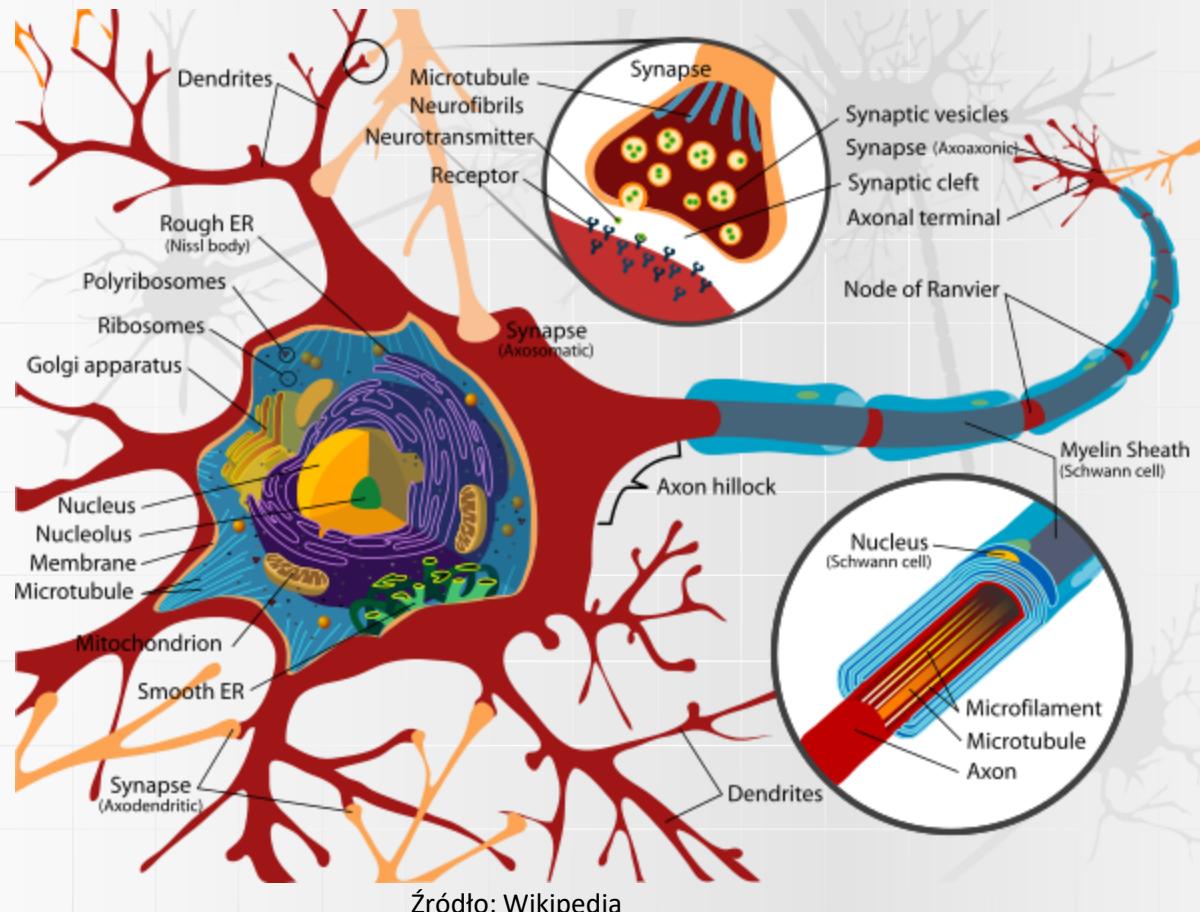
- Nie musimy mieć dobrze zdefiniowanego algorytmu – potrzebne są jedynie reprezentatywne przykłady przetwarzania wejścia w wyjście.
- Oddzielenie dwóch trybów pracy (uczenia i przetwarzania). Sieć przez większość czasu jest wykorzystywana w trybie przetwarzania danych.
- Zdolność do radzenia sobie z zakłóconymi wzorcami – zdolność generalizacji
- Bez przeprogramowywania jest możliwy rozwój sieci.



# Wady sieci neuronowych

- Brak możliwości objaśniania wypracowywanych decyzji – modele *black box*
- Brak dobrych teorii matematycznych opisujących ich działanie i osiąganie zbieżności
- Brak sposobu douczania modeli, jeśli pojawiły się nowe dane odbiegające od danych uczących (ang. *concept drift*)

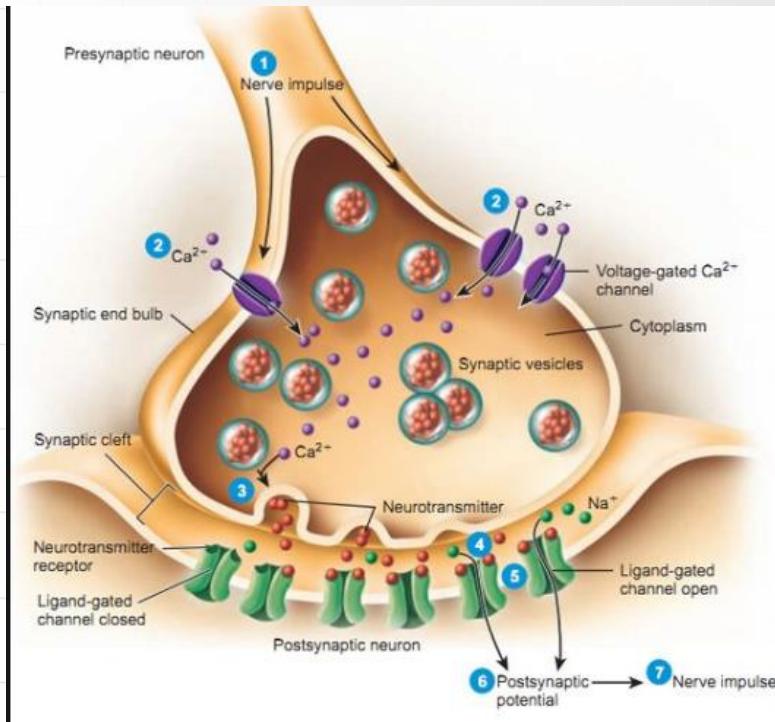
# Biologiczne inspiracje



Źródło: Wikipedia

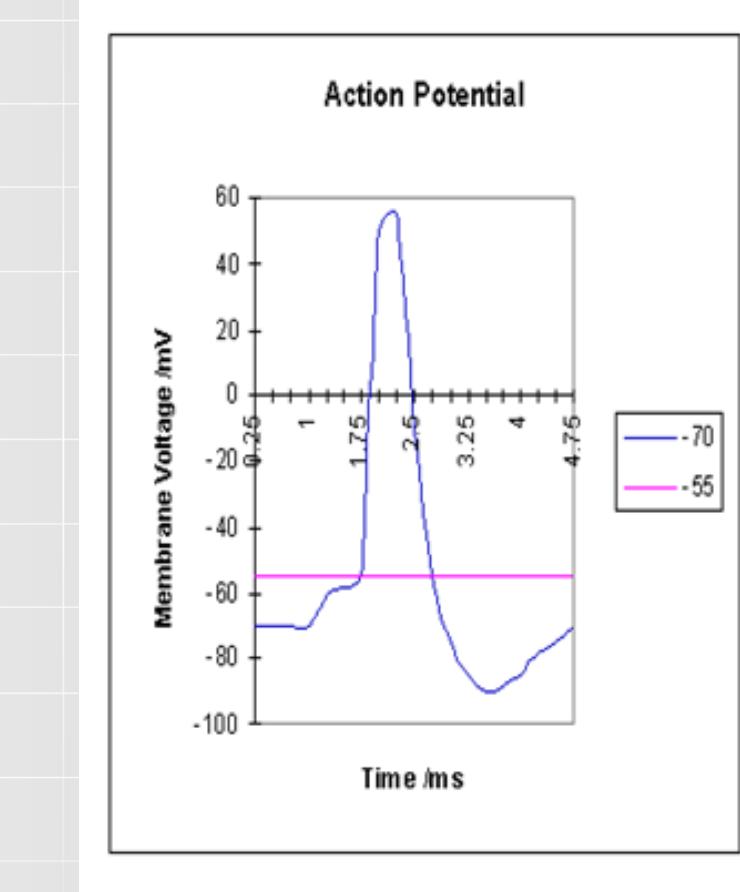
Pojedynczy neuron z wieloma dendrytami zakończonymi synapsami służącymi do połączenia z innymi neuronami.

# Sygnal w neuronie biologicznym



Źródło: <https://www.pinterest.co.kr/pin/615656211535528857/>

Połączenie synaptyczne

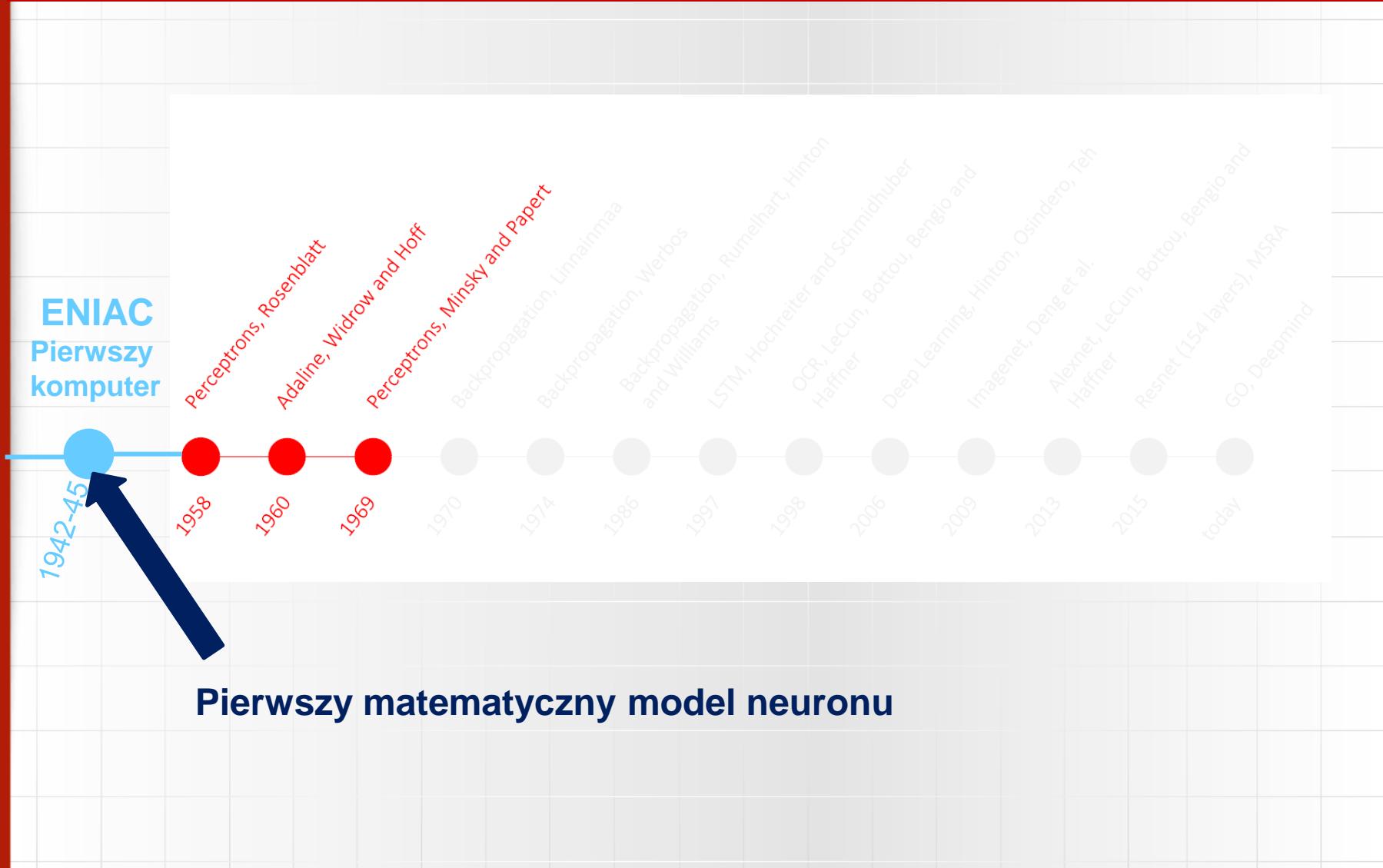


Kształt impulsu w  
biologicznym neuronie

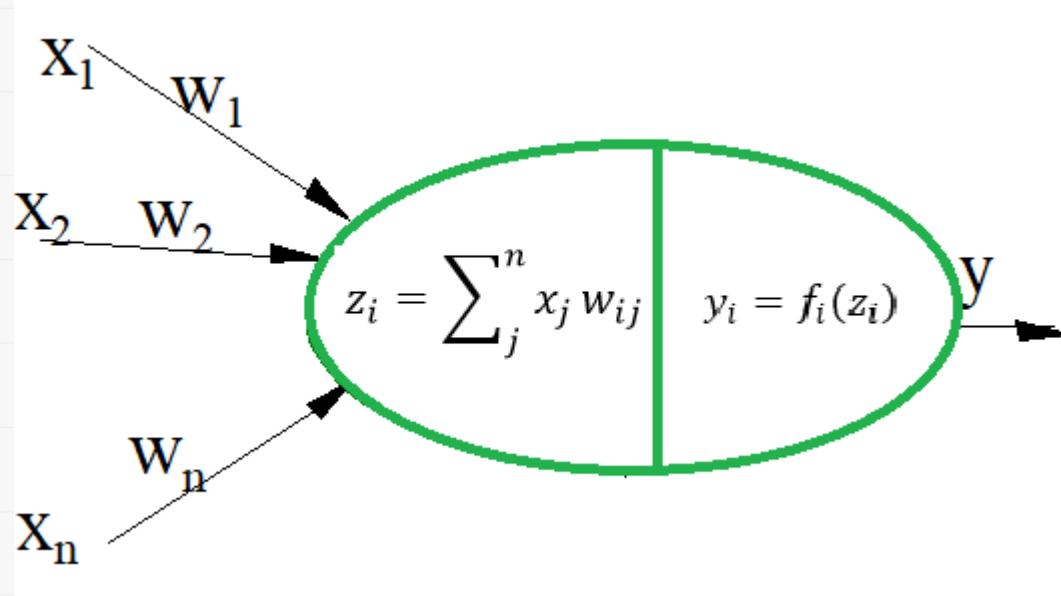


# Historia

## Początki



# Pierwszy model neuronu



## Model Pittsa Mc Cullocha

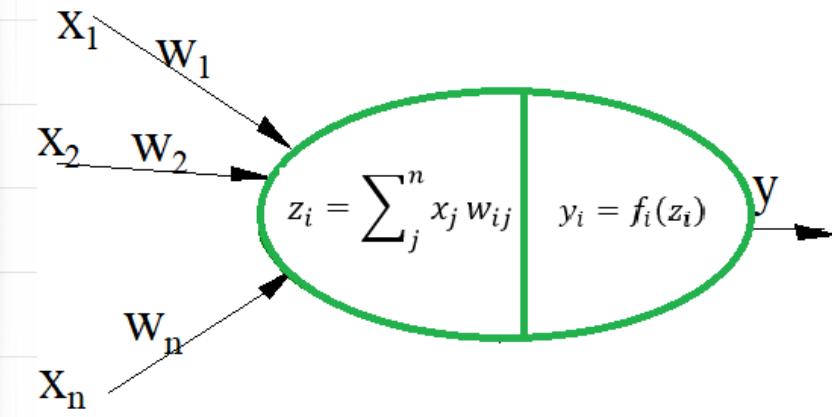
- $x_i$  – sygnał wejściowy
- $w_i$  – waga
- $y$  – wyjście
- $z_i$  – sieciowe wejście, całkowite pobudzenie
- $f$  – funkcja aktywacji



# Opis wektorowy neuronu

- We współczesnych zastosowaniach korzystny jest opis wektorowy neuronu
- Wejścia do neuronu tworzą wektor ( $t$ - oznacza transpozycję)  $\mathbf{x} = [x_1, x_2, \dots, x_n]^t$
- $\mathbf{x}$  to wektor kolumna
- $\mathbf{x}^t$  to wektor wiersz
- Wektor wag  $i$ -tego neuronu:  $\mathbf{w}_i = [w_{i1}, w_{i2}, \dots, w_{in}]^t$
- Całkowite pobudzenie jako iloczyn skalarny dwóch wektorów:  $z_i = \mathbf{x} * \mathbf{w}_i$  (\* operator iloczynu skalarnego)
- Czasem wygodniej zapisać to jako zwykły iloczyn wektorowy:  $z_i = \mathbf{x}^t \mathbf{w}_i$

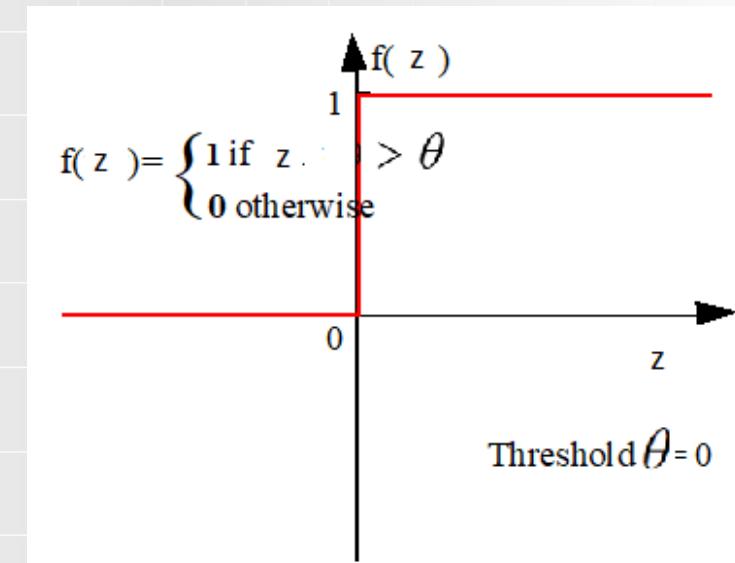
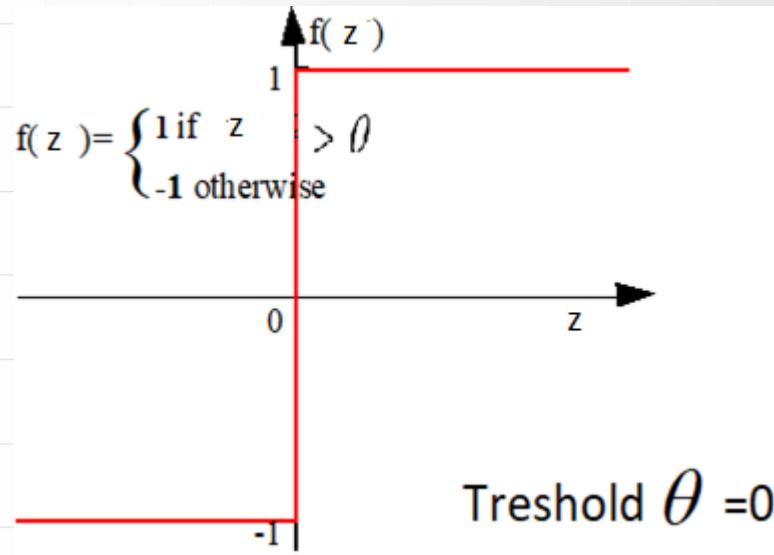
# Perceptron prosty



- Funkcja aktywacji jest funkcją skokową

- Struktura perceptronu prostego jest identyczna jak w modelu neuronu Pittsa McCullocha.
- $x_i = 1$ , jeśli i-ty punkt na siatkówce jest zapalony i 0 w przeciwnym przypadku.
- Do obliczenia całkowitego pobudzenia  $z_i$ , każda wartość wejścia  $x_j$  jest ważona przy użyciu wagi  $w_j$ ,
- Wyjście  $y = 1$  gdy
$$\sum_n w_i x_i > \theta$$
- gdzie  $\theta$  jest wartością progową

# Funkcje aktywacji dla perceptronu prostego



- Funkcja bipolararna

- Funkcja unipolarna



# Uczenie – znalezienie wag

- Uczenie nadzorowane
- Zbiór uczący w postaci par:
  - $\langle [x_i] - \text{wektor wejściowy}, y_i^d - \text{żądane wyjście} \rangle$   
( $d$  – *desired*, czasem nazywany ang. *target pattern*)



# Reguła uczenia

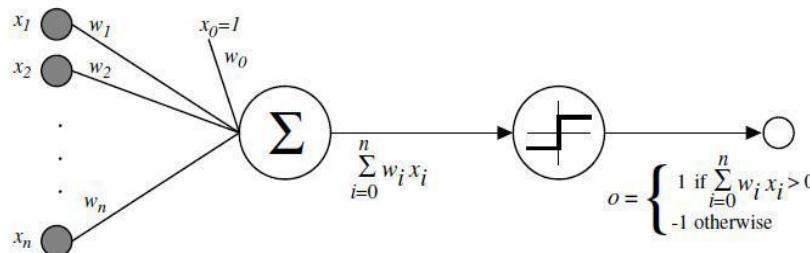
Procedura uczenia jest prosta polega na tym, że po każdym wzorcu sprawdzamy czy otrzymane wyjście jest równe zadanemu ( $y_i^d = y_i$ ).

Jeśli tak, nie wykonujemy zmian wag, jeśli nie, dodajemy przyrost proporcjonalny do wejścia i różnicy między zadanym wyjściem i wyjściem otrzymanym.

- Wagi uaktualniamy następująco:  $w_{inew} = w_{iold} + \Delta w_i$
- gdzie  $\Delta w_i = \alpha x_i (y^d - y)$
- $\alpha$  -współczynnik uczenia
- Czasem ten przyrost jest proporcjonalny do iloczynu wejścia i zadanego wyjścia :  $\Delta w_i = \alpha x_i y^d$

# Perceptron prosty

Realizuje binarną klasyfikację



Architektura prostego perceptronu

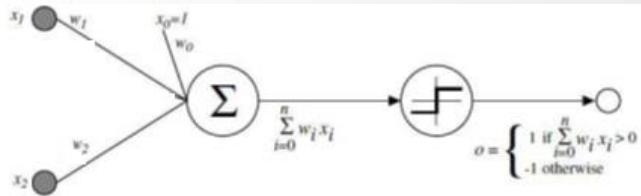
Rozwiążanie znajdowane przez perceptron (dla dwóch wag) jest linią prostą o równaniu:

$$x_1 w_1 + x_2 w_2 = -w_0$$

- Waga  $w_i$  przypisana do i-tego połączenia
- Zamiast progu  $\theta$ , dodatkowe połączenie (bias)  $w_0$  na którym płynie sygnał  $x_0$  zawsze równy 1. Dwie funkcje aktywacji: bipolarna  $\{-1,1\}$  lub unipolarna  $\{0,1\}$  do wyboru
- Procedura uczenia
  - 1.Zainicjuj losowo wagи
  - 2.Weź jeden wzorzec wejściowy  $x$  predykuj  $y$ )
  - 3.Uaktualnij wagи jeśli pojawił się błąd
    - Jeśli wyjście zadane  $y^d=-1$  and  $y_i = 1$  , zwiększ wagи
    - Jeśli wyjście zadane było  $y^d=1$  a  $y_i = -1$  , zmniejsz wagи
  - 4.Powtarzaj od punktu 2. dla wzorców ze zb. uczącego (1 epoka)
  - 5. Powtarzaj 4., dopóki w epoce nie będzie nie będzie błęduś

# Perceptron prosty

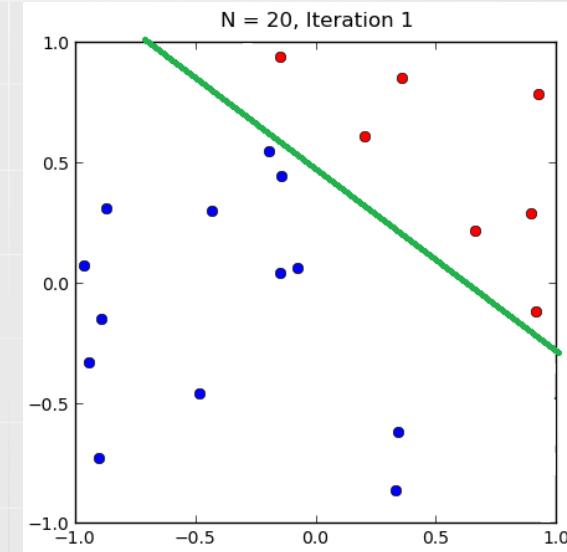
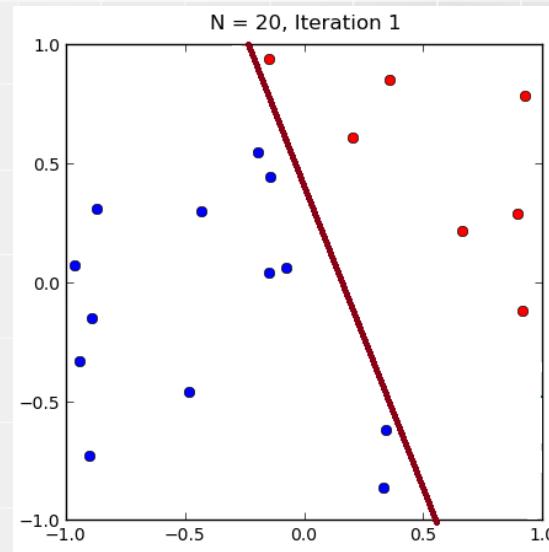
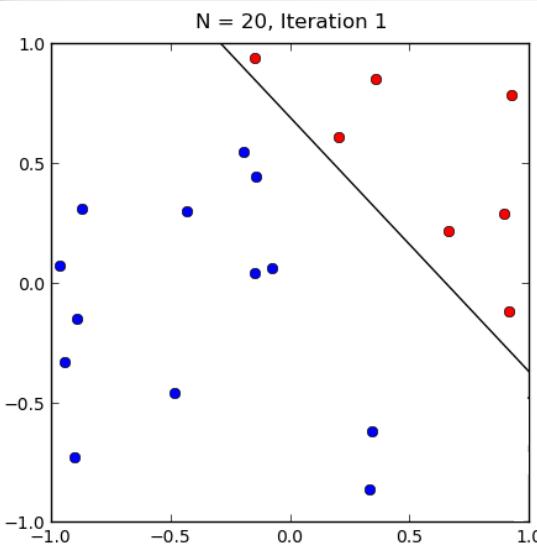
Wykonuje klasyfikację binarną



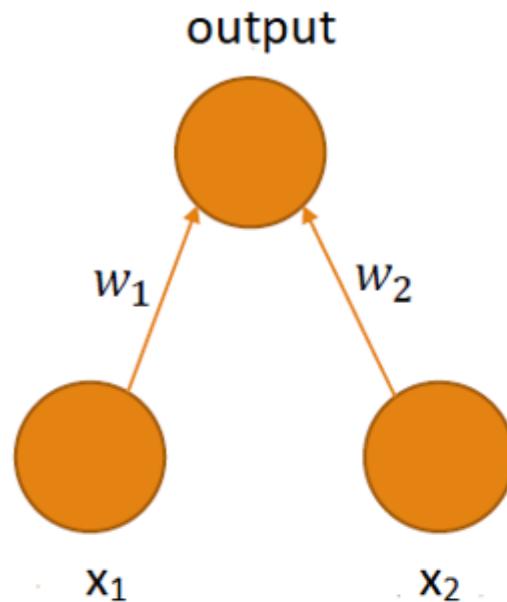
Problemy separowalne liniowo

- Chcemy zaklasyfikować, czy dane opisują dżokeja czy koszykarza?
- Cechy:  $x_1$ =waga and  $x_2$ = wzrost (na osiach)
- Perceptron szuka linii dzielącej przestrzeń na dwie części

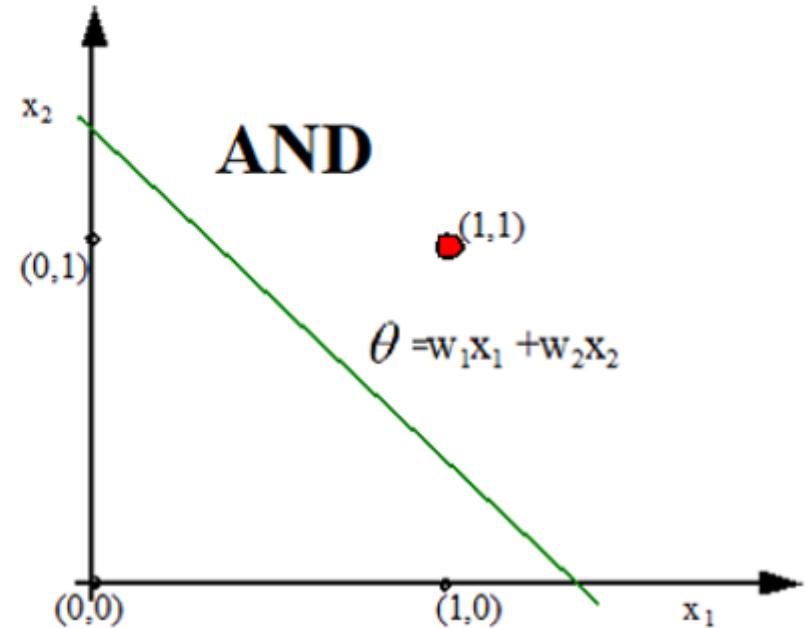
$$x_1 w_1 + x_2 w_2 = -w_0$$



# AND – liniowo separowalny problem

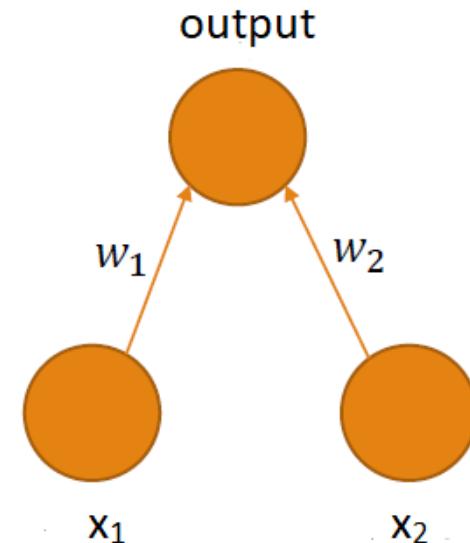
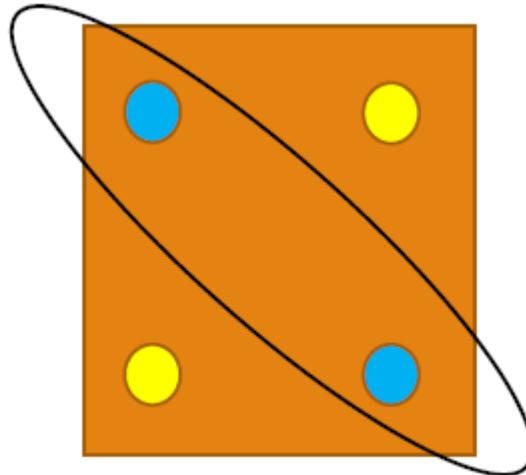


$x_1$	$x_2$	output
1	1	1
1	0	0
0	1	0
0	0	0



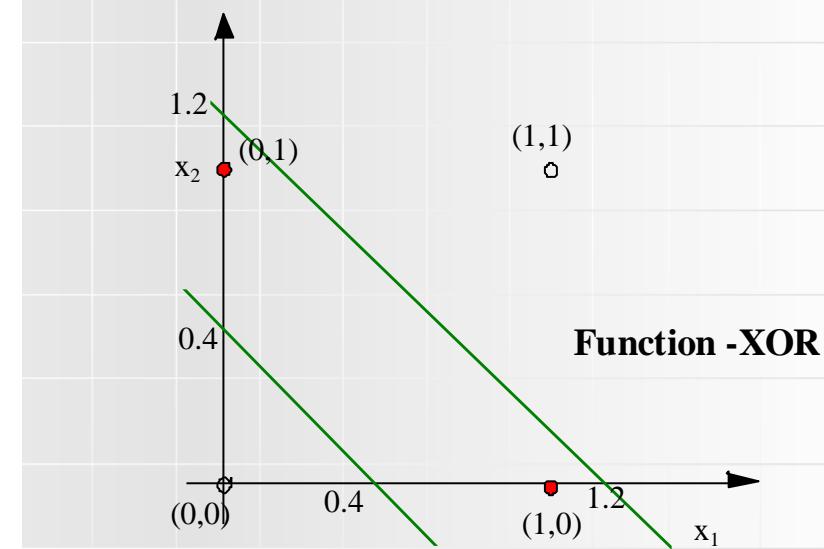
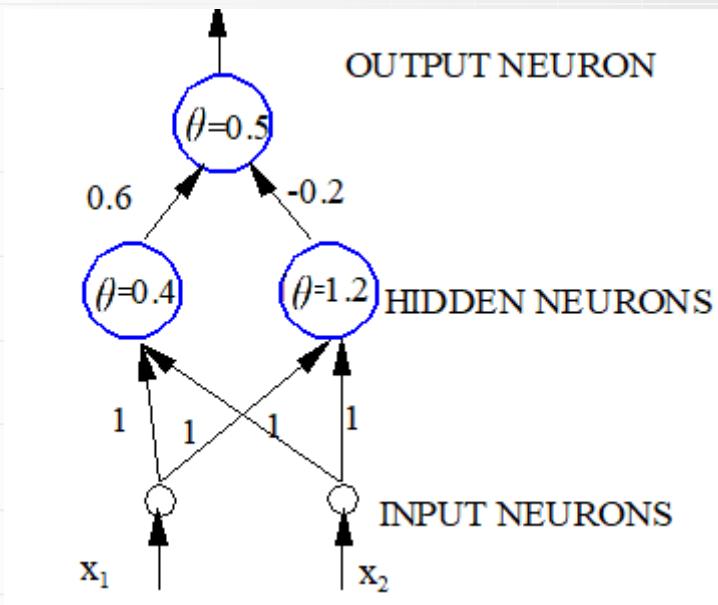
# XOR liniowo nieseparowalny

- $0 \ w_1 + 0 \ w_2 < \theta \rightarrow 0 < \theta$
- $0 \ w_1 + 1 \ w_2 > \theta \rightarrow w_2 > \theta$
- $1 \ w_1 + 0 \ w_2 > \theta \rightarrow w_1 > \theta$
- $1 \ w_1 + 1 \ w_2 < \theta \rightarrow w_1 + w_2 < \theta$

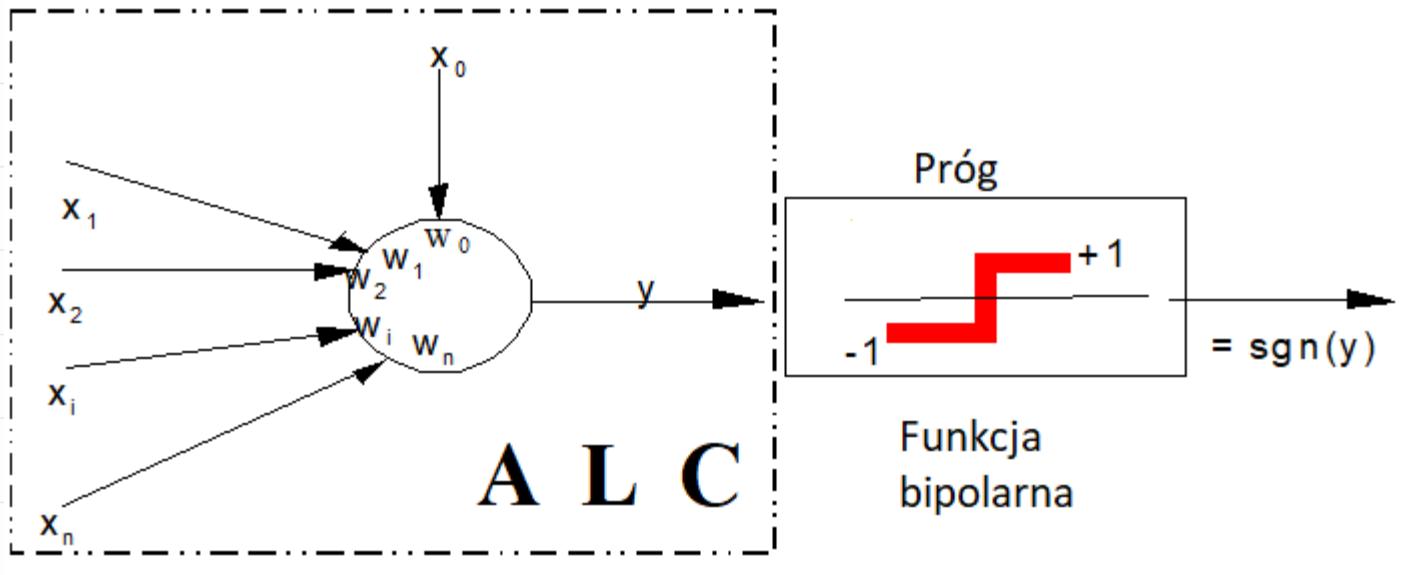


$x_1$	$x_2$	output
1	1	0
1	0	1
0	1	1
0	0	0

# XOR- rozwiązanie z warstwą ukrytą



# Adaline



- Adaline jest akronimem od ADaptive Linear Combiner
- Struktura neuronu jest prawie identyczna jak perceptronu prostego. Dwie podstawowe różnice:
  - Bias (waga na dodatkowym połączeniu, na którym wartość wejścia jest zawsze ustalona na 1).
  - Bipolarna funkcja jako funkcja aktywacji (czyli wyjście ma wartość +1 lub -1).



# Reguła uczenia LMS

- Założymy, że mamy wektory wejściowe  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L\}$  mające zadane unikalne, poprawne wyjście  $\{d_1, d_2, \dots, d_L\}$  dla każdego wzorca wejściowego.
- Reguła Least Mean Square (LMS) polega na znalezieniu wag poprzez przyrostowe zmiany ich wartości wag po przetworzeniu każdego wzorca uczącego, tak aby zmniejszał się błąd pomiędzy poprawnym wyjściem a wartością otrzymaną na wyjściu.
- Proces adaptacji wag będziemy nazywać uczeniem.



# LMS

- Przyjmijmy, że mamy zbiór par (zbiór trenujący)

$$\{\langle \mathbf{x}_1, d_1 \rangle, \langle \mathbf{x}_2, d_2 \rangle, \dots \langle \mathbf{x}_L, d_L \rangle\}$$

- Próbowejmy znaleźć najlepszy wektor wag  $\mathbf{w}^*$ , który odwzorowuje wektor wejściowy  $\mathbf{x}_k$  w zadane wyjście  $d_k$ .
- Co oznacza najlepszy?



# LMS c.d.

- Oznaczmy  $\varepsilon_k = d_k - y_k$  jako błąd występujący po zastosowaniu  $k$ -tego wektora wejściowego.

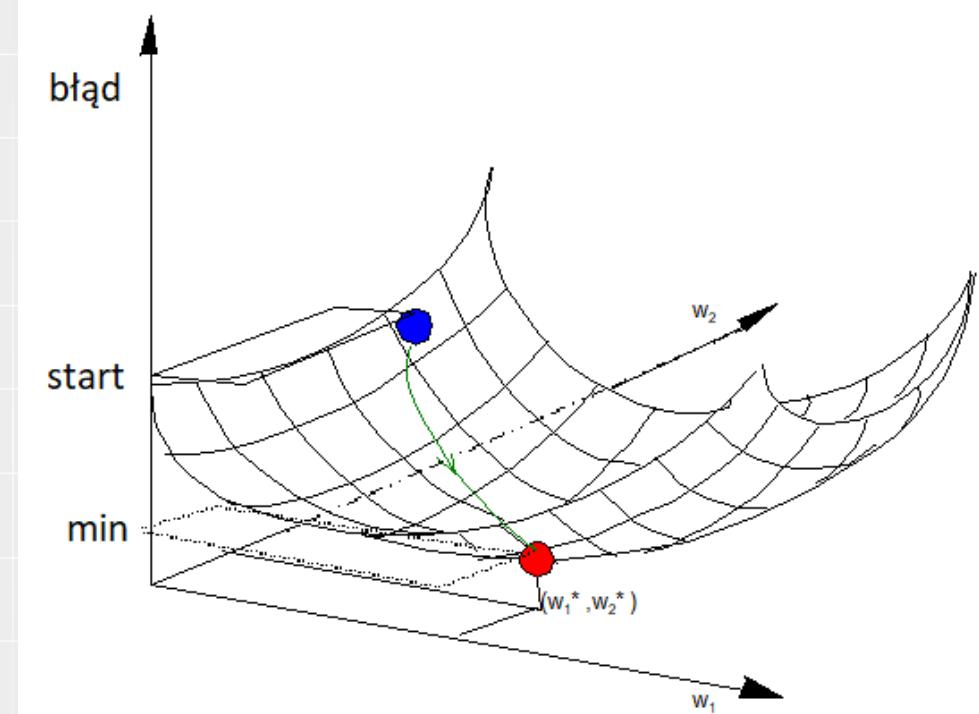
$$\varepsilon_k^2 = (d_k - w^t x_k)^2$$

- Chcemy zminimalizować błąd średniokwadratowy dla zbioru wektorów wejściowych :

$$\langle \varepsilon^2 \rangle = 1/L \sum_{k=1}^L \varepsilon_k^2$$

# LMS c.d.

- Obliczenie najlepszych wag  $w^*$ , które spełniają warunek nie jest łatwe i dlatego używamy procedury iteracyjnej.



Błąd zależy w kwadracie od wag, dlatego wykres jest paraboloidą lub hiperparaboloidą dla wyższych wymiarów.



# LMS- ogólna idea

- Poszukujemy metody znajdowania optymalnego wektora wag  $\mathbf{w}^*$  w iteracyjnej procedurze (ang. Steepest Descent Method).
- Zaczynamy od arbitralnych wartości wag.
- Od tego punktu startowego wyznaczamy kierunek, najszybszego spadku błędu.
- Zmieniamy wagi o niewielką wartość, tak aby nowy wektor leżał niżej na wykresie.
- Powtarzamy, dopóki minimum błędu nie będzie osiągnięte.

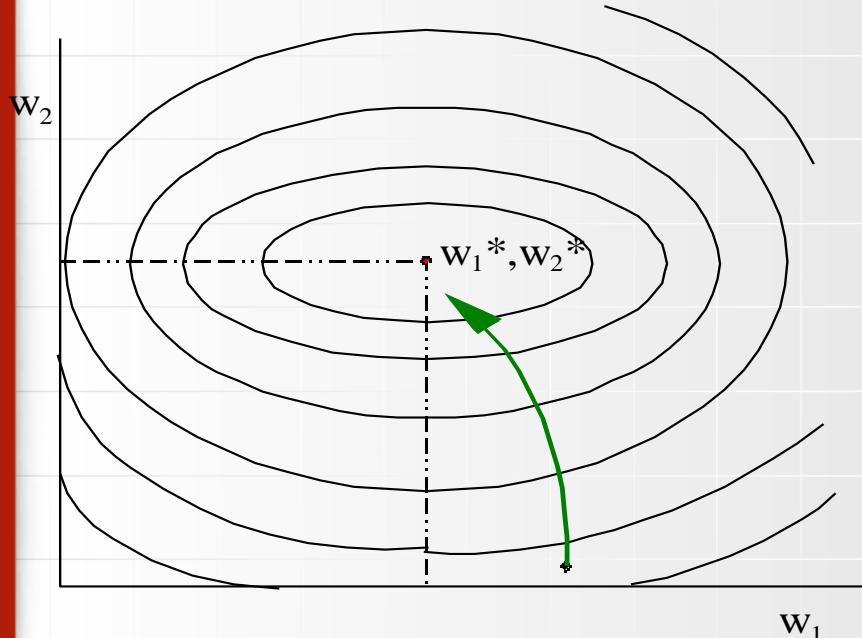


# LMS - metoda największego spadku

- Wektor wag nie przesuwa się bezpośrednio w kierunku minimum. Przecięcie paraboloidy jest zazwyczaj elipsą a więc ujemny gradient nie wskazuje dokładnie minimum.
- Wektor wag jest zmienną w czasie. Oznaczymy go  $w(t)$ .
- W każdym kroku obliczamy go zgodnie ze wzorem :

$$w(t + 1) = w(t) + \alpha \Delta w(t)$$

# LMS- steepest Descent Method



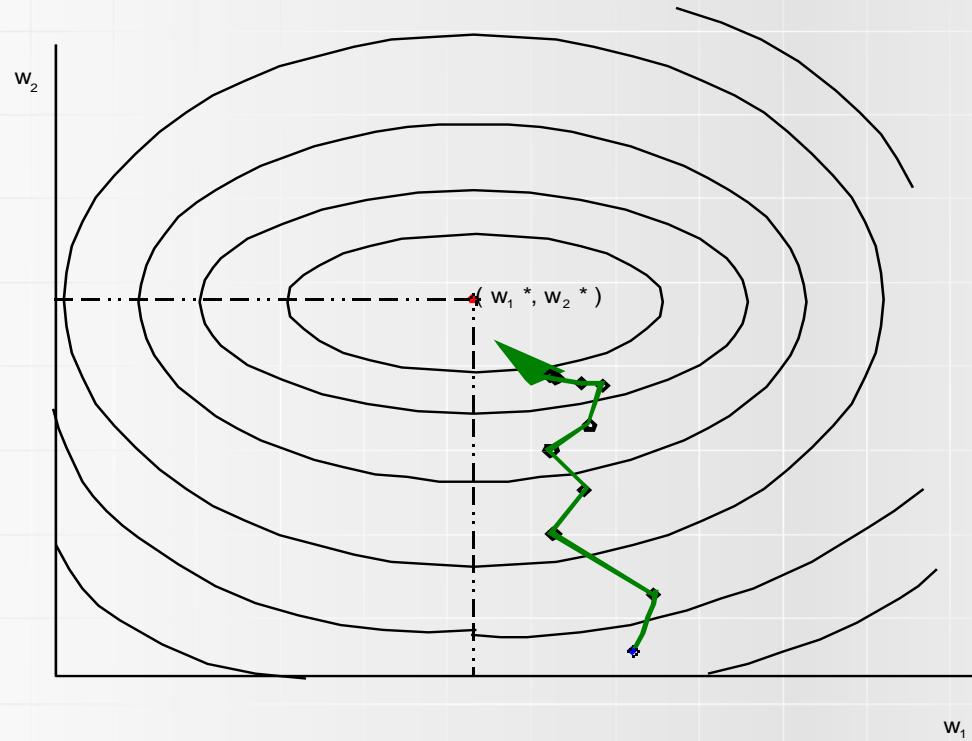
- Dla każdego konturu (przecięcie wykresu błędu) kierunek wyznaczany przez najszybszy spadek błędu jest prostopadły do konturu w tym punkcie. Kierunek nie zawsze wskazuje na punkt odpowiadający minimum

- Poszukujemy kierunku najszybszego spadku a więc musimy policzyć gradient na tej powierzchni.
- Ujemny gradient będzie wskazywał kierunek do dołu.

- Obliczamy gradient w każdej iteracji

$$w(t+1) = w(t) - \mu \nabla \xi(w(t))$$

- $\mu$  jest współczynnikiem uczenia



- W każdym punkcie gradient jest aproksymowany. W wyniku otrzymujemy taką drogę w kierunku do minimum funkcji błędu.



# LMS algorytm uczenia

- 1.Zastosować wektor  $x_k$  do wejść Adaline
- 2.Określić wartość błędu kwadratowego

$$\langle \varepsilon_k^2 \rangle = \langle (d_k - w^t x_k)^2 \rangle$$

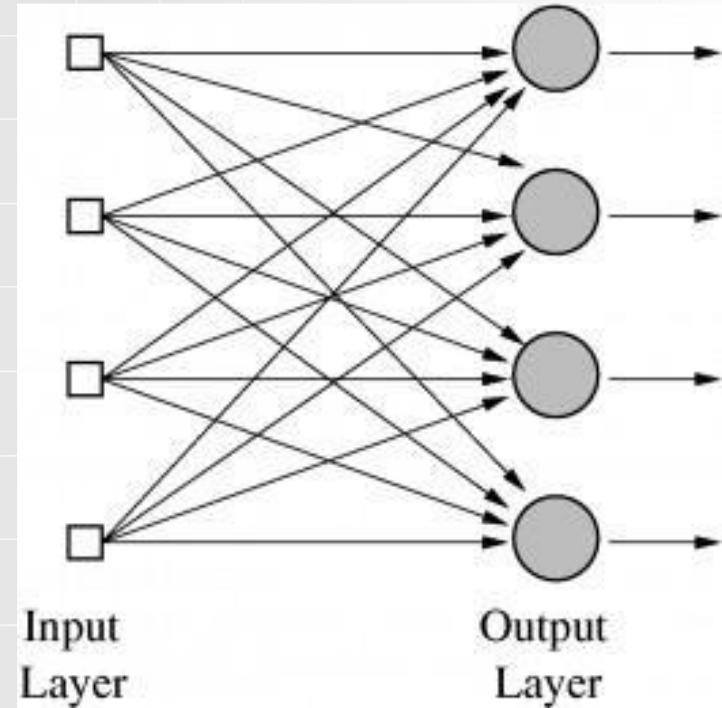
- Przybliżyć  $\nabla \xi(w(t))$  przez  $\nabla \varepsilon_k^2 = -2\varepsilon_k(t)x_k$
- 3.Uaktualnić wagę następująco:

$$w(t+1) = w(t) + 2\mu \varepsilon_k x_k$$

- 4. Powtarzać kroki 1 do 4 dla następnego wektora wejściowego, dopóki błąd nie osiągnie wystarczająco małej wartości.

# Od neuronu do sieci

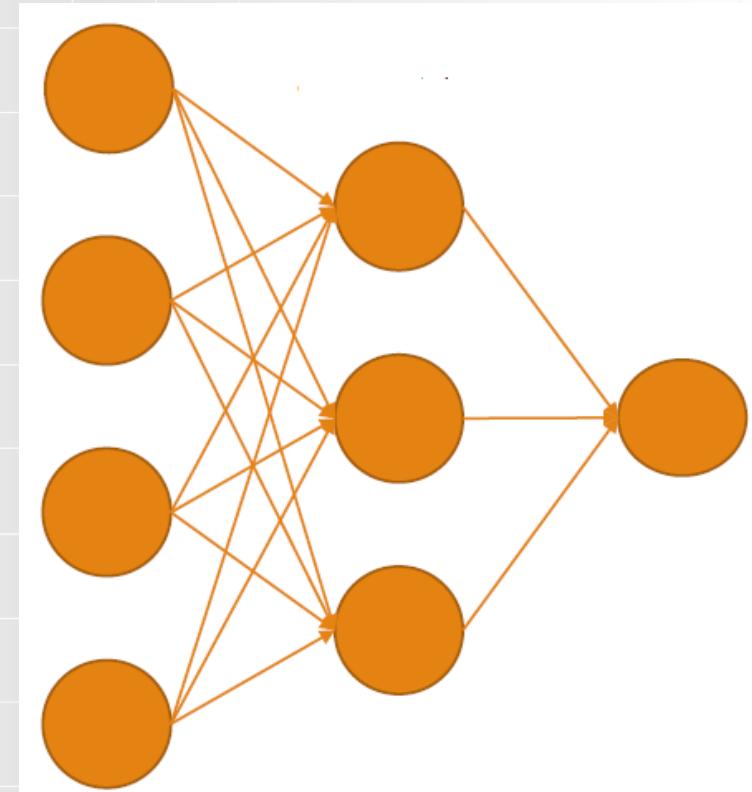
- Prosty perceptron (adaline)= jedna decyzja
- Wiele decyzji = wiele wyjść
- Dla problemów nieseparowalnych liniowe => MultiLayer Perceptron (MLP) – sieć wielowarstwowa (co najmniej jedna warstwa ukryta).





# MLP

- Jak je trenować?
- Nie wiemy, jakie powinno być wyjście z warstwy ukrytej
- Algorytm  
**BACKPROPAGATION!!**





# Podsumowanie

- Poznaliśmy 2 pierwsze proste modeli sieci:
  - Perceptron prosty i Adaline
- 
- Mają różne reguły uczenia
  - Oba modele rozwiązują jedynie problemy separowalne liniowo (binarna klasyfikacja).



Wrocław  
University  
of Science  
and Technology



# SIECI NEURONOWE



HR EXCELLENCE IN RESEARCH



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Politechnika Wrocławskiego

Unia Europejska  
Europejski Fundusz Społeczny





# Sieci neuronowe

## Sieć wielowarstwowa MLP

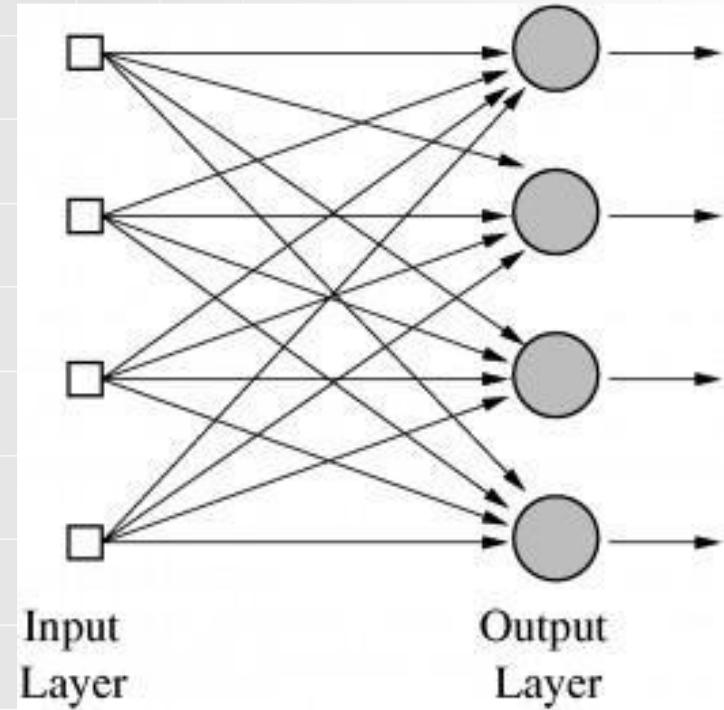
**URSZULA MARKOWSKA-KACZMAR**  
Katedra Inteligencji Obliczeniowej  
Wydział Informatyki i Zarządzania



Politechnika Wrocławskiego

# Poprzedni wykład

- Model pierwszego neuronu
- Elementy proste: Adaline i Perceptron prosty (jedna decyzja jedno wyjście)
- Różne reguły uczenia
- Umożliwiają rozwiązywanie problemów separowalnych liniowo

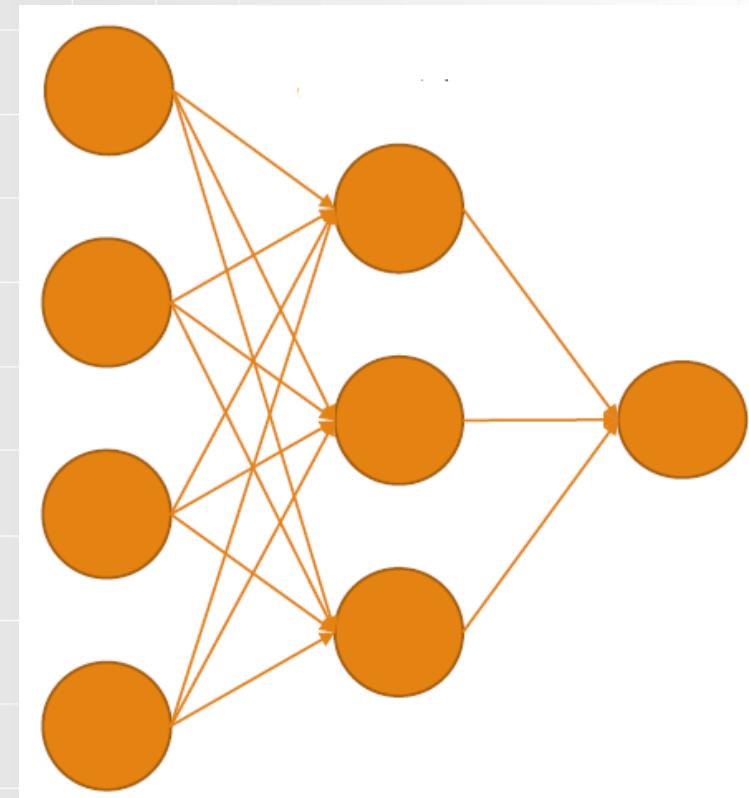


Wiele decyzji – wiele wyjść



# MLP

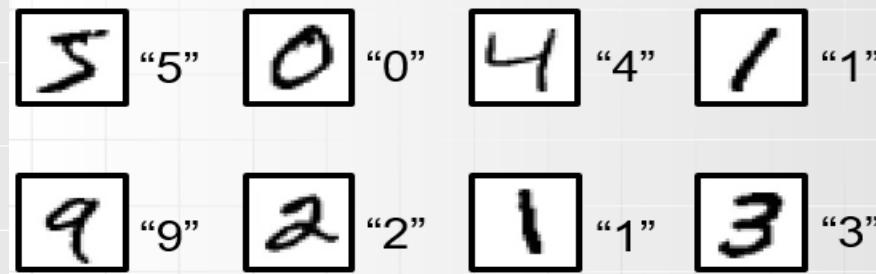
- Dla problemów nieseparowalnych liniowo => MultiLayer Perceptron (MLP)
- Jak je trenować?
- Nie wiemy jakie powinno być wyjście z warstwy ukrytej
- Algorytm BACKPROPAGATION!!



# Rozważania wstępne

- Mamy zbiór P par wektorów  $\{<\mathbf{x}_1, \hat{\mathbf{y}}_1>, <\mathbf{x}_2, \hat{\mathbf{y}}_2>, \dots, <\mathbf{x}_p, \hat{\mathbf{y}}_p>\}$ .
- Są przykładami odwzorowania:  
$$\mathbf{y} = \varphi(\mathbf{x}): \mathbf{x} \in R^N, \mathbf{y} \in R^M$$
- Chcemy wyuczyć sieć, która będzie aproksymować tę funkcję:  $\mathbf{y}' = \varphi'(\mathbf{x})$
- Uczenie nadzorowane
- Będziemy stosować uogólnienie reguły LMS znanej z zastosowania w Adaline

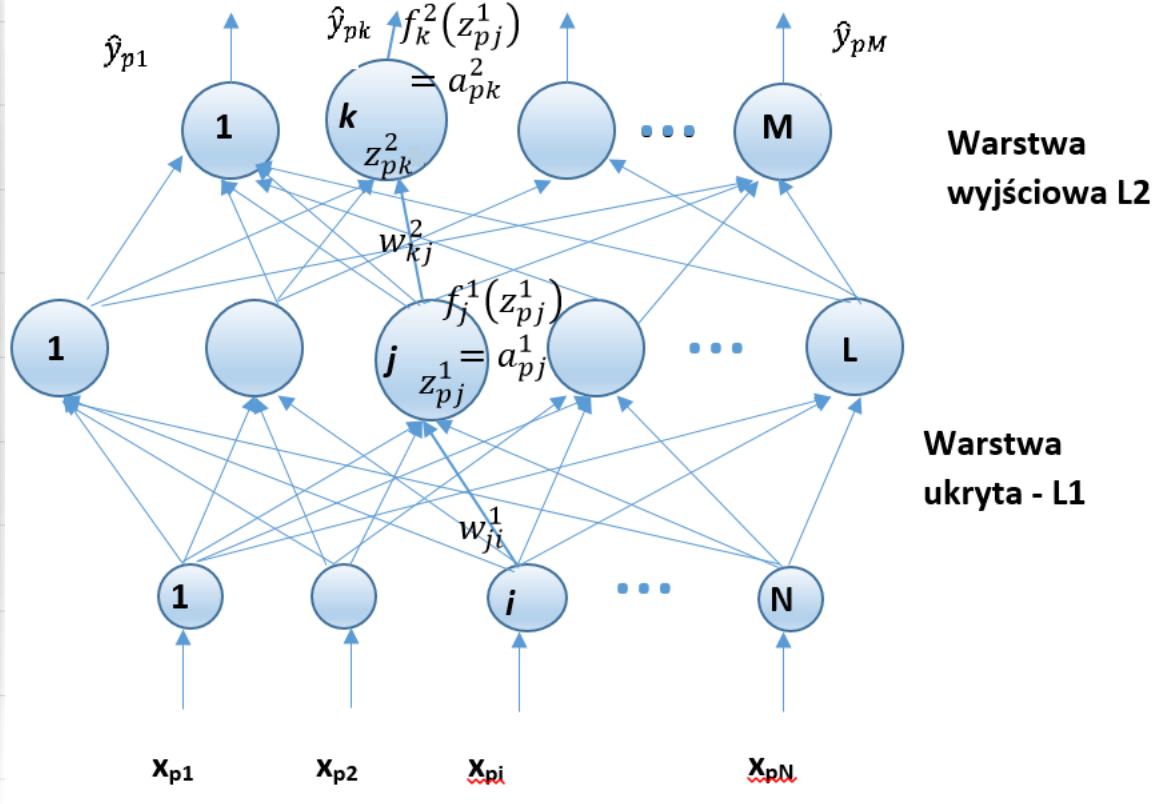
**WZORCE UCZĄCE**



- **Problemy:**

- Mamy wiele wyjść z sieci – musimy zmodyfikować funkcję błędu,
- Nie wiemy ile wynosi i jak policzyć błąd w warstwie ukrytej.

# Architektura sieci MLP



W oznaczeniach na rysunku:

- górny indeks oznacza numer warstwy: 1 – warstwa ukryta , 2 - warstwa wyjściowa
- $p$  – oznacza  $p$ -ty wzorzec podawany na wejście
- $z$  - całkowite pobudzenie neuronu
- $a$  – wyjście z neuronu po przetworzeniu przez funkcję aktywacji
- $x$ - wejście do sieci
- $\hat{y}_{pi}$ -produkowane przez neuron i wyjście

Sieć warstwowa, jednokierunkowa

## Przetwarzanie pojedynczego wzorca wejściowego w sieci

- Weźmy  $p$ -ty wzorzec ze zbioru uczącego:

$$\mathbf{x}_p = (x_{p1}, x_{p2}, \dots, x_{pN})^t$$

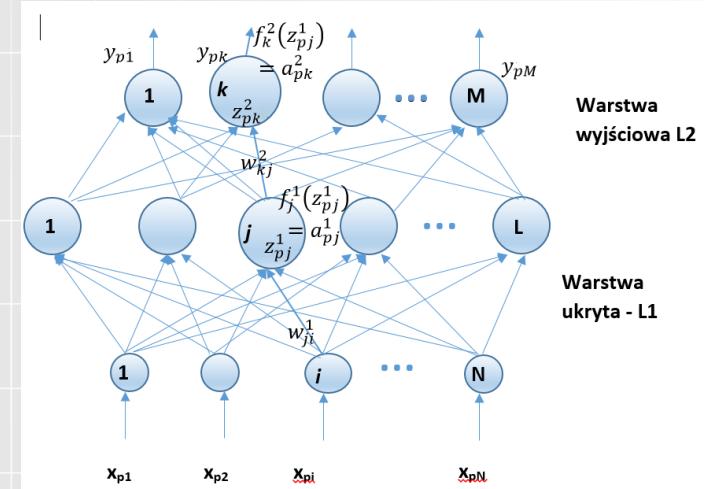
## warstwa ukryta

- Całkowite pobudzenie do  $j$ -tego neuronu w warstwie ukrytej jest równe:

$$z_{pj}^1 = \sum_{i=1}^N w_{ji}^1 x_{pi}$$

- Wyjście

$$a_{pj}^1 = f_j^1(z_{pj}^1)$$



## warstwa wyjściowa

- wejście do  $k$ -tego neuronu

$$z_{pk}^2 = \sum_{j=0}^N w_{kj}^2 a_{pj}^1$$

- wyjście

$$\hat{y}_{pk} = f_k^2(z_{pk}^2)$$



# Ogólny zarys procedury uczenia

1. Wprowadź wektor wejściowy do sieci i oblicz odpowiadające wartości wyjściowe
2. Porównaj aktualną wartość wyjścia z właściwą wartością wyjściową i określ miarę błędu.
3. Określ, w którym kierunku (+ lub -) zmienić każdą z wag w celu zredukowania błędu.
4. Określ wielkość, o jaką należy zmienić każdą z wag.
5. Powtarzaj kroki od 1 do 5 dla wszystkich wektorów uczących, aż błąd dla wszystkich wektorów w zbiorze uczącym zostanie zredukowany do akceptowalnej wielkości.

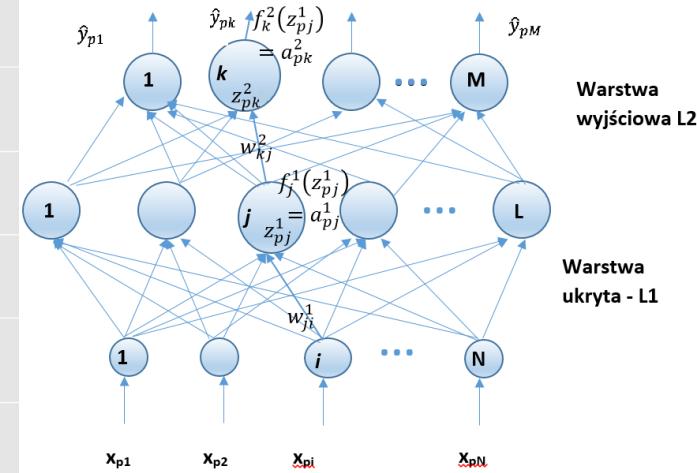
# Obliczanie wag w warstwie wyjściowej

- Zdefiniujmy błąd w  $k$ -tym neuronie warstwy wyjściowej w odpowiedzi na  $p$ -ty wzorzec ze zbioru uczącego:  $\delta_{pk} = (y_{pk} - \hat{y}_{pk})$ ,
- Będziemy minimalizować sumę kwadratów tych błędów po wszystkich wyjściach:

$$E_p = 1/2 \sum_{k=1}^M (\delta_{pk})^2$$

- M- liczba neuronów w warstwie wyjściowej
- Uwzględniając definicję błędu:

$$E_p = 1/2 \sum_{k=0}^M (y_{pk} - \hat{y}_{pk})^2$$



- Pojedyncze wagi będą aktualizowane w kierunku przeciwnym do gradientu błędu:
- $$w_{kj}^2(t+1) = w_{kj}^2(t) + \Delta_p w_{kj}^2(t)$$
- przyrost wagi jest proporcjonalny do gradientu błędu (w tym przypadku jego składowej):

$$\frac{\partial E_p}{\partial w_{kj}^2}$$



## Obliczanie wag w warstwie wyjściowej

- Dla  $j$ -tego połączenia w  $k$ -tym neuronie wyjściowym (reguła łańcuchowa)

$$\frac{\partial E_p}{\partial w_{kj}^2} = -(y_{pk} - \hat{y}_{pk}) \frac{\partial f_k^2}{\partial z_{pk}^2} \frac{\partial z_{pk}^2}{\partial w_{kj}^2}$$

- Obliczmy poszczególne pochodne cząstkowe

$$\frac{\partial z_{pk}^2}{\partial w_{kj}^2} = \frac{\partial}{\partial w_{kj}^2} \sum_{j=0}^L w_{kj}^2 a_{pj}^1 = a_{pj}^1$$

po tym różniczkujemy, więc pozostałe składowe nie wchodzą do pochodnej

Pochodna funkcji aktywacji w warstwie wyjściowej



## Obliczanie wag w warstwie wyjściowej

- Wielkość zmian wag jest proporcjonalna do ujemnego gradientu

$$w_{kj}^2(t+1) = w_{kj}^2(t) + \Delta_p w_{kj}^2(t)$$

- gdzie  $\Delta_p w_{kj}^2(t)$  obliczamy następująco:

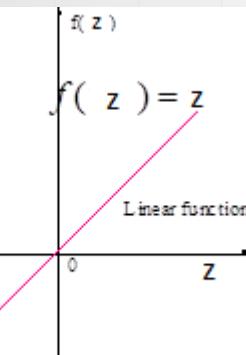
$$\Delta_p w_{kj}^2(t) = \mu(y_{pk} - \hat{y}_{pk}) {f_k^2}'(z_{pk}^2) a_{pj}^1$$

- $\mu$  jest nazywany współczynnikiem szybkości uczenia. Zawsze jest dodatni i mniejszy od 1
- We wzorze występuje pochodna funkcji aktywacji=> funkcja aktywacji musi być **różniczkowalna**

## Funkcje aktywacji

- Funkcja liniowa:  $f_k^2(z_{jk}^2) = z_{jk}^2$

Pochodna:  $f_k^{2'} = 1$



- Funkcja logistyczna (sigmoidalna):

$$f_k^2(z_{jk}^2) = (1 + e^{-\beta z_{jk}^2})^{-1}$$

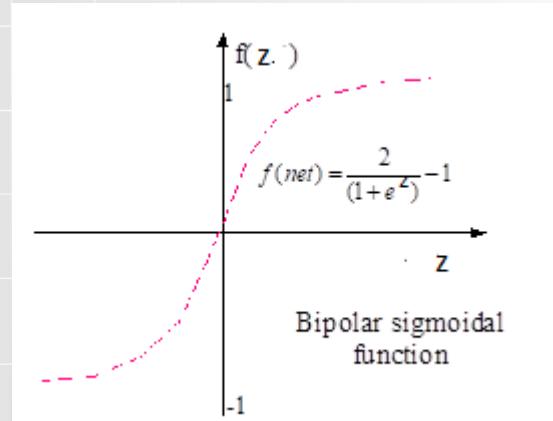
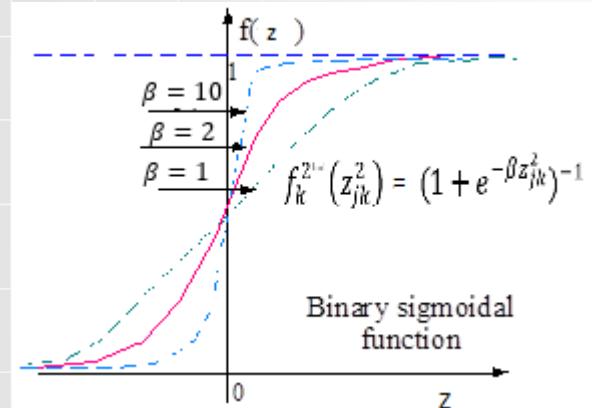
Pochodna

$$\begin{aligned} f_k^{2'}(z_{jk}^2) &= \beta f_k^2 (1 - f_k^2) \\ &= \beta y_{pk}(1 - y_{pk}) \end{aligned}$$

- Funkcja tangensa hiperbolicznego:

$$f_k^2(z_{jk}^2) = \frac{e^{\beta z_{jk}^2} - e^{-\beta z_{jk}^2}}{(e^{\beta z_{jk}^2} + e^{-\beta z_{jk}^2})}$$

Pochodna:  $f_k^{2'} = \beta(1 - (f_k^2)^2)$





## Zmiana wag w warstwie wyjściowej

- Oznaczmy błąd w warstwie drugiej jako:

$$\delta_{pk}^2 = (y_{pk} - \hat{y}_{pk}) {f_k^2}'(z_{pk}^2) = \delta_{pk} f_k^{2'}(z_{pk}^2)$$

Wówczas zmianę w warstwie wyjściowej możemy zapisać jako:

$$w_{kj}^2(t+1) = w_{kj}^2(t) + \mu \delta_{pk}^2 a_{pj}^1$$



- Poprzednie obliczenia aktualizowały wagi po wzorcu.
- Możemy aktualizować po całej epoce.  
Wówczas minimalizowany błąd:

- $E = \sum_{p=1}^P E_p$
- $p$  jest indeksem wzorca
- A przyrost wag:  $\Delta w_{kj}^2 = -\mu \sum_{p=1}^P a_{pj}^1 \delta_{pk}^2$
- Taki sposób nazywany uczeniem po epoce (wymaga pamiętania dużej ilości informacji)



## Aktualizacja wag w warstwie ukrytej

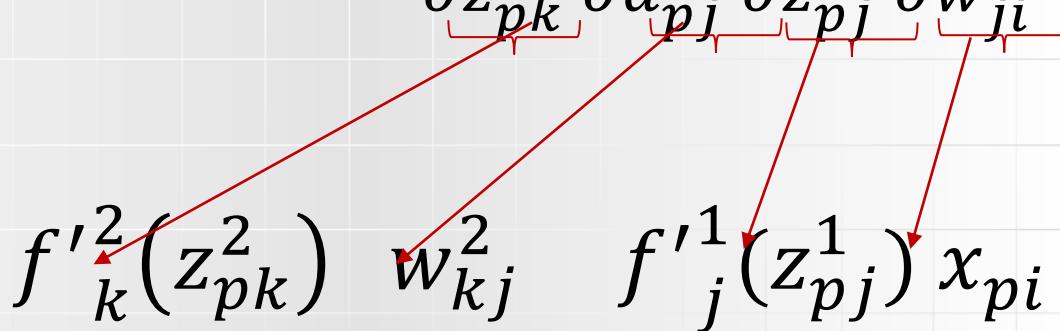
- Jak określić miarę błędu dla neuronów w warstwie ukrytej?
- Rozpatrzmy jak błąd zależy od wag z warstwy ukrytej

$$\begin{aligned} E_p &= 1/2 \sum_{k=1}^M (y_{pk} - \hat{y}_{pk})^2 \\ &= 1/2 \sum_{k=1}^M (y_{pk} - f_k^2(z_{pk}^2))^2 \\ &= 1/2 \sum_{k=1}^M (y_{pk} - f_k^2 \left( \sum_{j=1}^L w_{kj}^2 a_{pj}^1 \right))^2 \end{aligned}$$

Tutaj ukryte są wagi  
z warstwy ukrytej

## Aktualizacja wag w warstwie ukrytej

- Na tej podstawie obliczymy gradient  $\frac{\partial E_p}{\partial w_{ji}^1}$

$$\begin{aligned}\frac{\partial E_p}{\partial w_{ji}^1} &= 1/2 \sum_{k=1}^M \frac{\partial}{\partial w_{ji}^1} (y_{pk} - f_k^2(z_{pk}^2))^2 \\ &= \sum_{k=1}^M (y_{pk} - \hat{y}_{pk}) \frac{\partial y_{pk}}{\partial z_{pk}^2} \frac{\partial z_{pk}^2}{\partial a_{pj}^1} \frac{\partial a_{pj}^1}{\partial z_{pj}^1} \frac{\partial z_{pj}^1}{\partial w_{ji}^1}\end{aligned}$$


-



- Gradient wyraża się następująco:

$$\frac{\partial E_p}{\partial w_{ji}^1}$$

$$= - \sum_{k=1}^M (y_{pk} - \hat{y}_{pk}) f'^2_k(z_{pk}^2) w_{kj}^2 f'^1_j(z_{pj}^1) x_{pi}$$

- Stąd przyrost wag:

$$\bullet \Delta_p w_{ji}^1 =$$

$$\mu f'^1_j(z_{pj}^1) x_{pi} \sum_{k=1}^M (y_{pk} - \hat{y}_{pk}) f'^2_k(z_{pk}^2) w_{kj}^2$$

- $\mu$  – współczynnik uczenia



- Wykorzystując definicję  $\delta_{pk}^2$  możemy zapisać:

$$\Delta_p w_{ji}^1 = \mu f_j'^1(z_{pj}^1) x_{pi} \sum_{k=1}^M \delta_{pk}^2 w_{kj}^2$$

Definiując błąd w warstwie ukrytej jako:

$$\delta_{pj}^1 = f_j'^1(z_{pj}^1) \sum_{k=1}^M \delta_{pk}^2 w_{kj}^2$$

Możemy zapisać:

$$w_{ji}^1(t+1) = w_{ji}^1(t) + \mu \delta_{pj}^1 x_{pi}$$

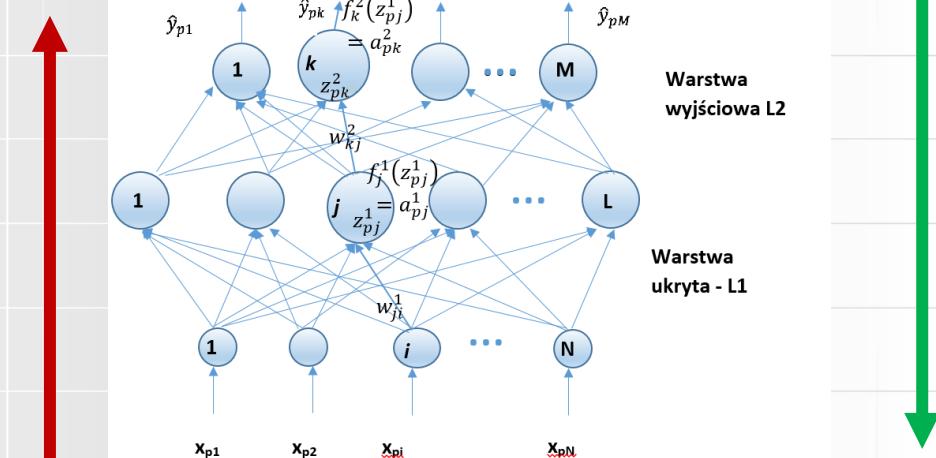
# Algorytm uczenia – dwie fazy

- Algorytm zawsze wykonywany jest w dwóch fazach
- **Przesłanie w przód (ang. forward)**

przesyłamy wektor wejściowy aż do obliczenia wyjścia

- **Przesłanie błędów wstecz (ang. backward)**

rzutowanie błędów od warstwy wyjściowej do wejściowej





## Algorytm – dla pojedynczego wzorca wejściowego

### Faza przesłania w przód

- Zastosować wektor wejściowy  $\mathbf{x}_p = [x_{p1}, x_{p2} \dots, x_{pN}]^T$
- Obliczyć wartości całkowitego pobudzenia dla jednostek warstw ukrytej:  $z_{pj}^1 = \sum_{i=0}^N w_{ji}^1 x_{pi}$
- Obliczyć wyjście z warstwy ukrytej:  $a_{pj}^1 = f_j^1(z_{pj}^1)$
- Przejść do warstwy wyjściowej. Obliczyć wartość sieciowego wejście dla każdej jednostki

$$z_{pk}^2 = \sum_{j=0}^L w_{kj}^2 a_{pj}^1$$

- Obliczyć wyjścia:  $y_{pk} = f_k^2(z_{pk}^2)$
-



## Algorytm – dla pojedynczego wzorca wejściowego

### Przesłanie błędów wstecz

Obliczyć błąd dla jednostek wyjściowych

$$\delta_{pk}^2 = (y_{pk} - \hat{y}_{pk}) f_k^{2'}(z_{pk}^2)$$

Obliczyć błędy dla jednostek ukrytych:

$$\delta_{pj}^1 = f_j'^{-1}(z_{pj}^1) \sum_{k=1}^M \delta_{pk}^2 w_{kj}^2$$

- Uaktualnić wagę w warstwie wyjściowej:

$$w_{kj}^2(t+1) = w_{kj}^2(t) + \mu \delta_{pk}^2 a_{pj}^1$$

- Uaktualnić wagę w warstwie ukrytej

$$w_{ji}^1(t+1) = w_{ji}^1(t) + \mu \delta_{pj}^1 x_{pi}$$

Porządek uaktualniania wag w ramach jednej warstwy jest nieistotny.



- Działanie algorytmu wymaga:
  - Zainicjowania wag (losowo)
  - Określenia wartości współczynnika uczenia
  - Określenia dopuszczalnej wartości błędu (Jest to wielkość określająca jak dobrze sieć jest wyuczona)
- Warunek zatrzymania algorytmu:
  - Proces uczenia można zakończyć, jeżeli błąd jest akceptowalnie mały dla każdej pary wektorów uczących.



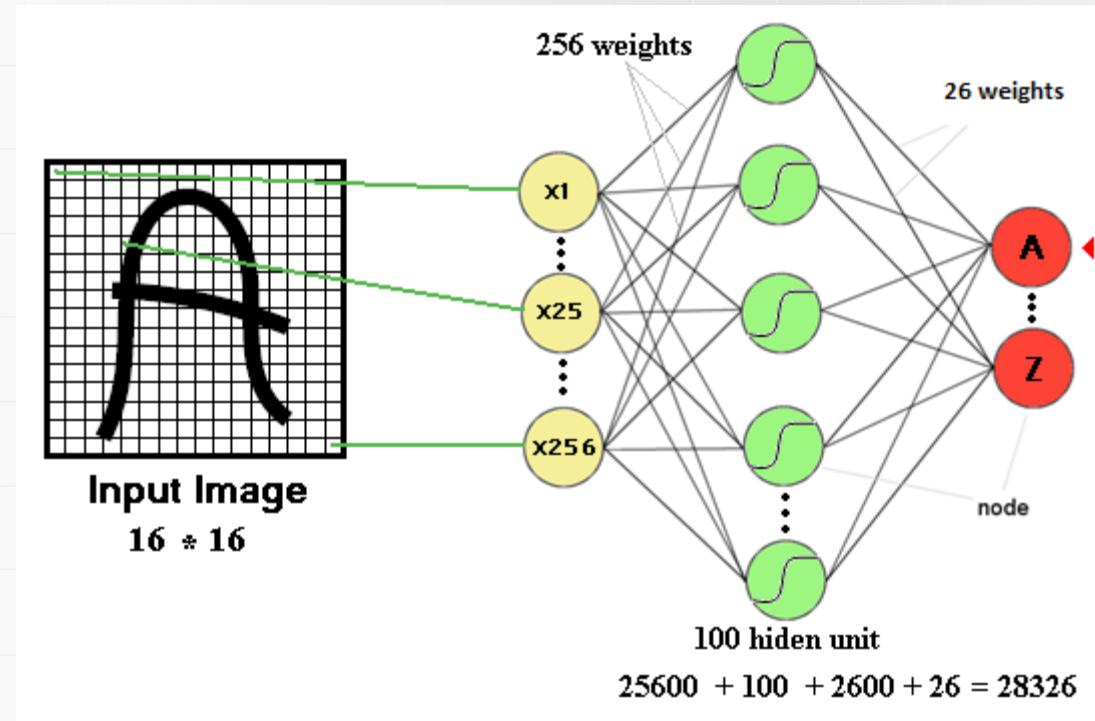
- **Uczenie po wzorcu (online)** – korekcja parametrów po każdym wzorcu
- **Uczenie po epoce (offline)** – korekcja wag po przejściu całej epoki uczenia ( podaniu wszystkich wzorców uczących ) – sumuje się gradienty pochodzące od wszystkich wzorców
- **Uczenie paczkami** (ang. mini batch), określa się liczbę wzorców stanowiących paczkę (od 50 do 300) i po każdej paczce następuje korekcja wag – najbardziej popularne podejście obecnie

# Charakterystyka sieci warstwowych

## Typy regionów decyzyjnych

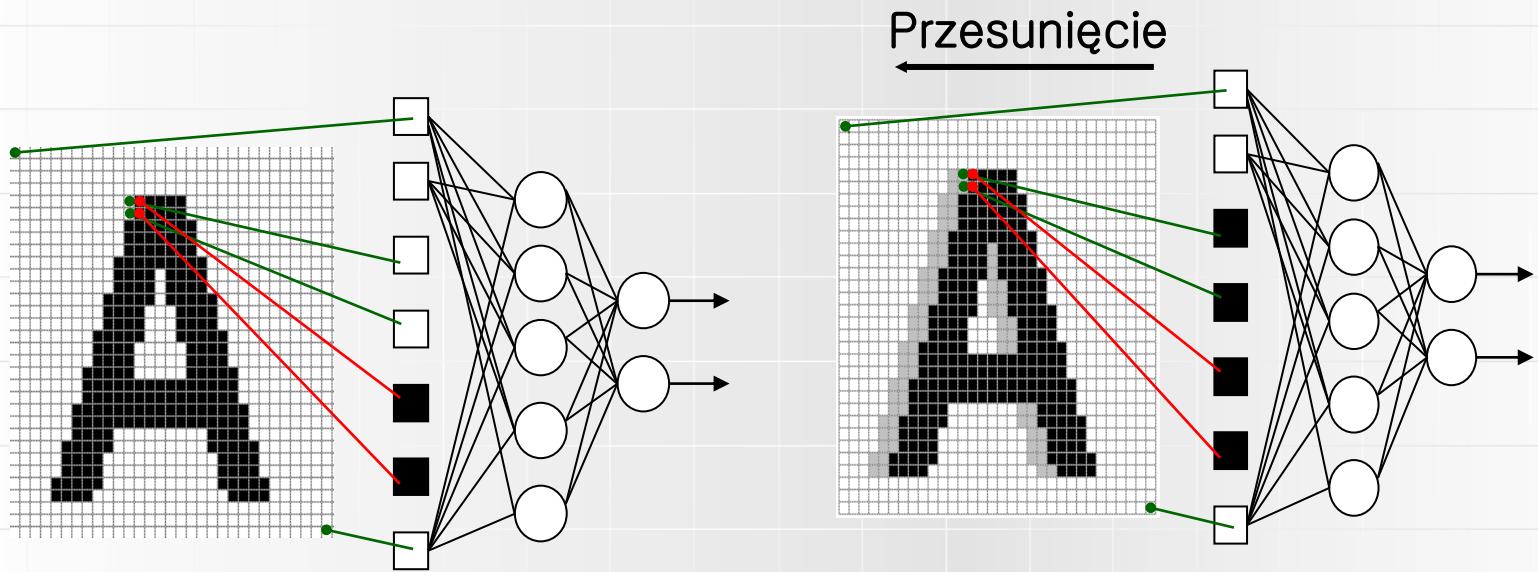
Struktura	Typy regionów decyzyjnych	XOR	Klasy ze splecionymi regionami	Najczęstsze kształty
Pojedyncza warstwa 	Półpłaszczyzna ograniczona hiperpłaszczyzną			
dwuwarstwowa 	otwarty wypukły lub zamknięty			
Trzywarstwowa 	dowolnie złożony			

# MLP w przetwarzaniu obrazów



- Wejście jest płaską reprezentacją obrazu wektorem
- Tracimy informację o sąsiedztwie pikseli

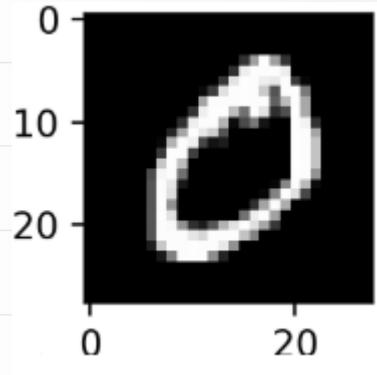
# MLP w przetwarzaniu obrazów



Wynik jest bardzo wrażliwy na przesunięcia, obroty i inne formy zakłóceń.



## Przetwarzanie zbioru MNIST



Przykład obrazu cyfry ze zbioru MNIST

- Cyfry zapisane są w formacie 28x28
- Ile będzie neuronów wyjściowych?
- Rozpoznajemy 10 cyfr=> na wyjściu 10 neuronów, z których każdy rozpoznaje inną cyfrę.
- Kodowanie 1 z n, co oznacza, że tylko jeden neuron będzie aktywny, pozostałe powinny mieć aktywności bliskie 0.



# Podsumowanie

- Poznaliśmy architekturę i sposób uczenia sieci warstwowej.
- Uczenie metodą propagacji wstecznej błędu=> stąd nazwa metody Backpropagation
- Przedstawiona wersja metody uczenia traktuje każdy neuron oddzielnie
- Na następnym wykładzie przedstawiona będzie metoda w ujęciu macierzowym.



Wrocław  
University  
of Science  
and Technology



Zintegrowany  
Program Rozwoju  
Politechniki Wrocławskiej

# SIECI NEURONOWE



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Politechnika Wrocławskiego

Unia Europejska  
Europejski Fundusz Społeczny



HR EXCELLENCE IN RESEARCH



# Sieci neuronowe

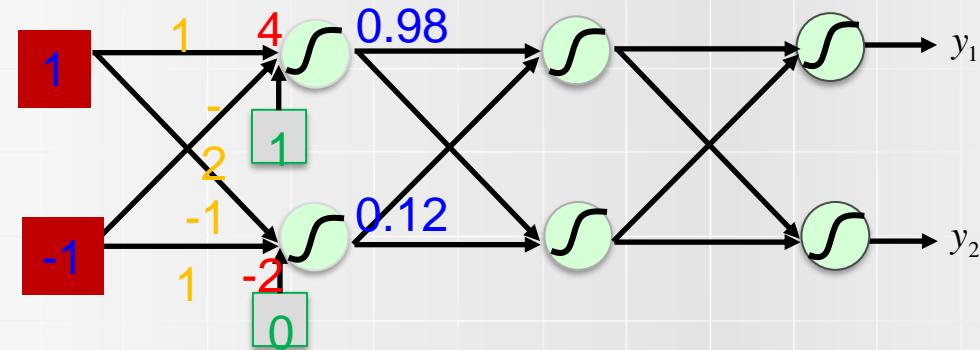
## Uczenie metodą propagacji wstecznej w ujęciu macierzowym

**URSZULA MARKOWSKA-KACZMAR**  
Katedra Inteligencji Obliczeniowej  
Wydział Informatyki i Zarządzania



Politechnika Wrocławskiego

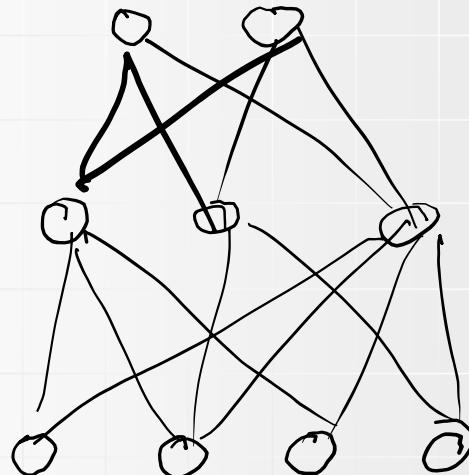
# Spróbujmy zapisać wektorowo



$$\sigma \left( \underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}} \right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$



$$w^2 = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

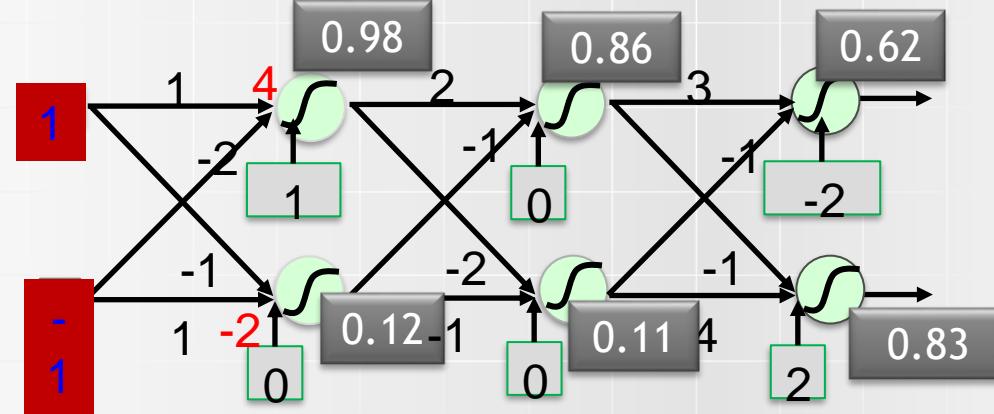


$$w^1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \quad \begin{bmatrix} x_1 \\ \vdots \\ x_u \end{bmatrix}$$

neurony w warstwie  
wysyłają

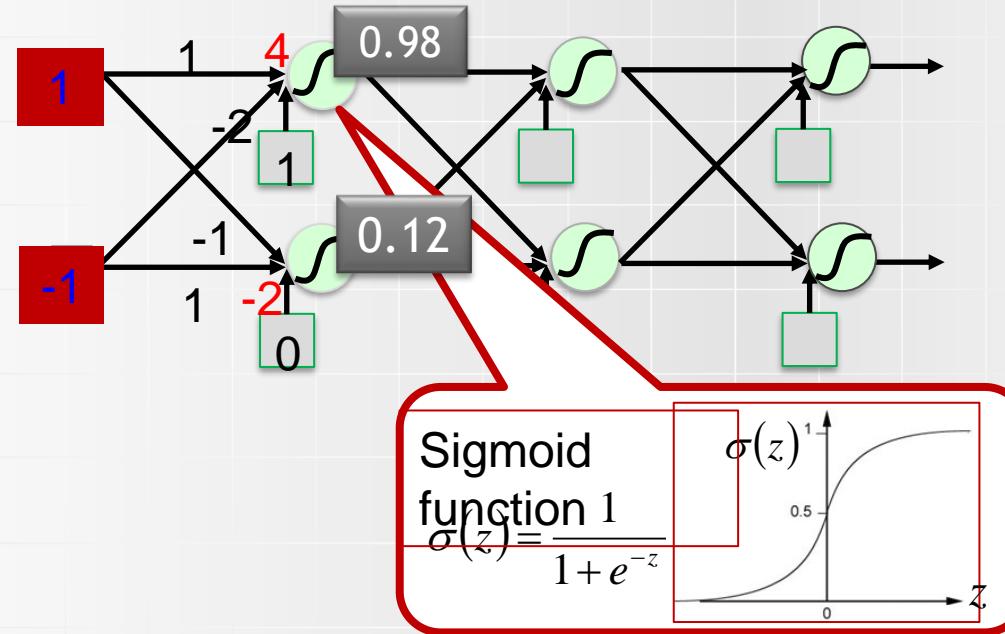
# Przykład przetwarzania w sieci

Tutaj mamy przykład głębszej sieci (2 warstwy ukryte)



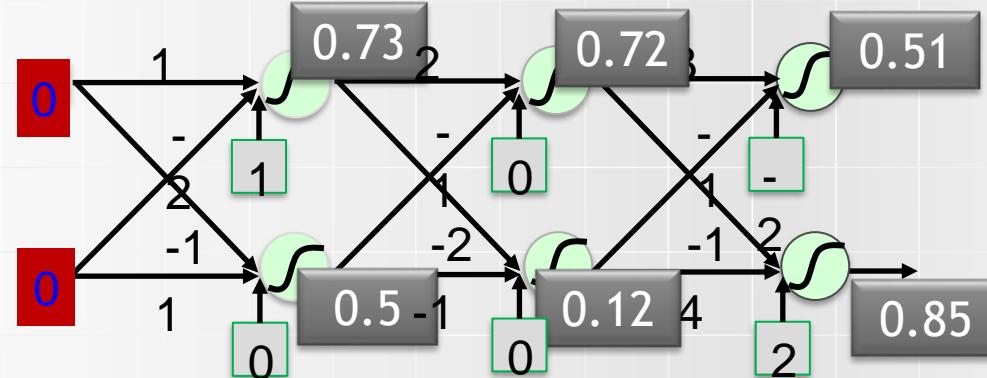
# Obliczenia w pierwszej warstwie

Przypisane wagi zostały wylosowane



Rozpoczynamy od podania wzorca [1,-1] na wejście.

# Zapis obliczeń w ujęciu macierzowym



$$f: R^2 \rightarrow R^2$$

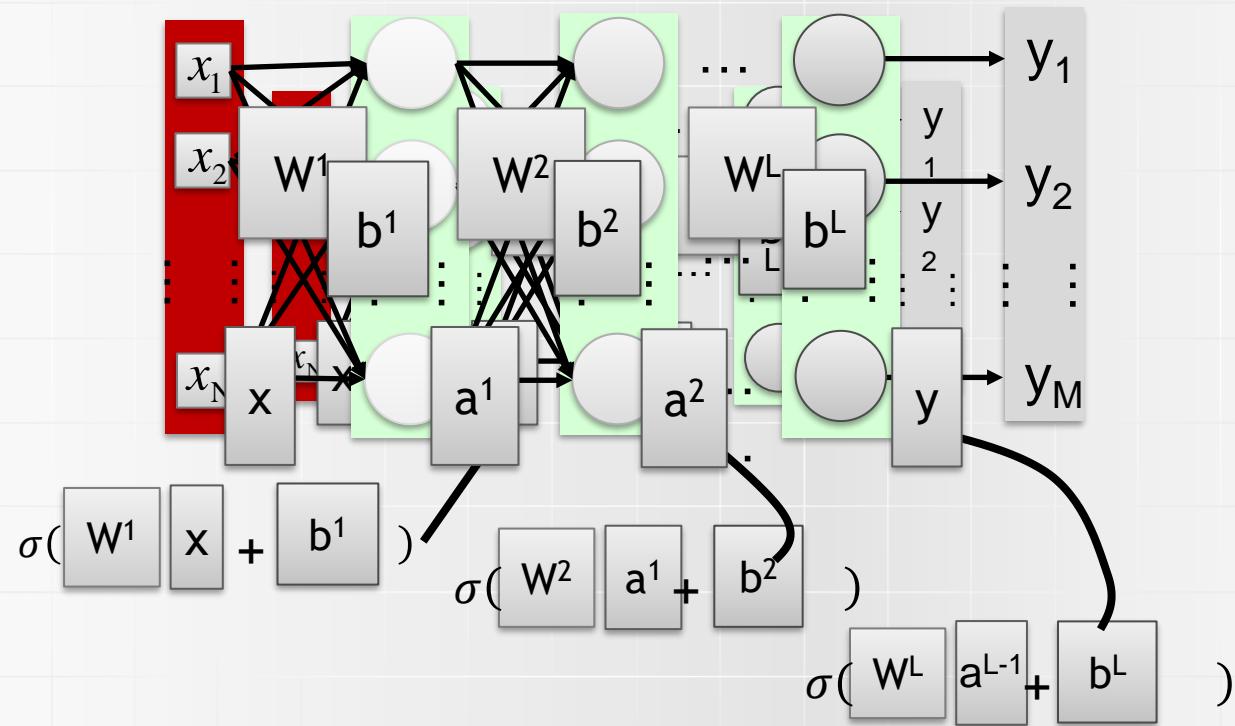
Lepiej jest reprezentować wszystkie operacje w sposób macierzowy

$$f \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix}$$

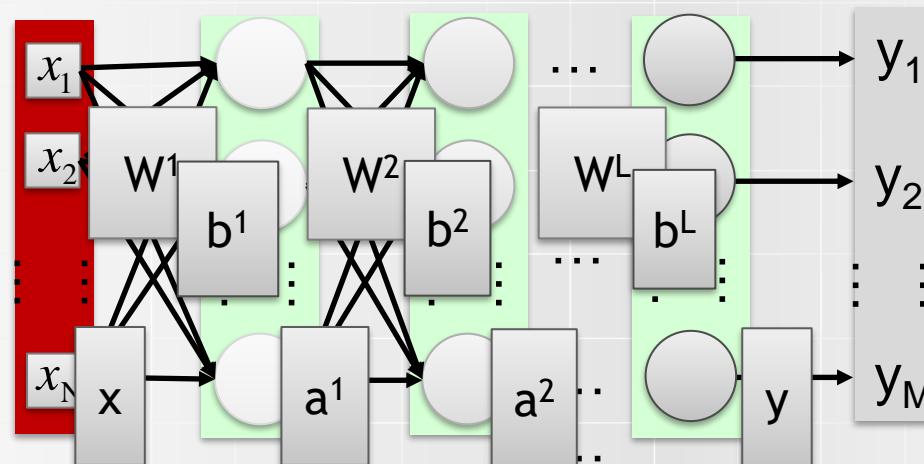
$$f \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix}$$

Różne parametry (wagi i biasy) definiują  
różne funkcje realizowane przez sieć

# Ogólnie dla całej sieci



# Funkcja realizowana przez sieć



$$y = f(x)$$

$$= \sigma(W^L \dots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \dots + b^L)$$

Użycie kart graficznych przyspiesza obliczenia



•  $\rightarrow 0$

•

•

•

•

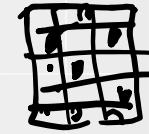
•

•

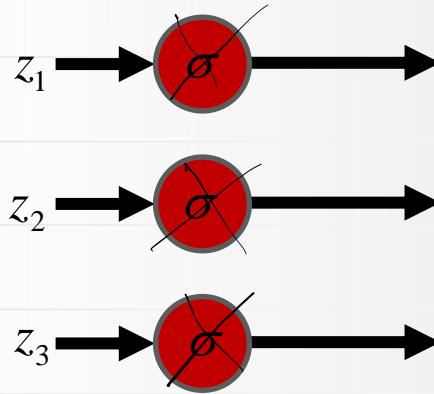
$\rightarrow 9$

6  
8

$0.78 \rightarrow$   
 $0.8 \rightarrow 8$



# Warstwa softmax



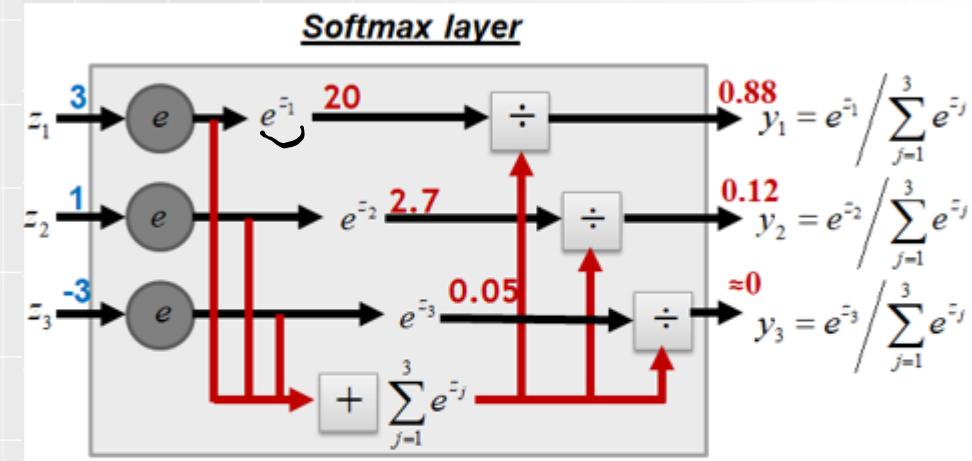
$$y_1 = \sigma(z_1)$$

$$y_2 = \sigma(z_2)$$

$$y_3 = \sigma(z_3)$$

Prawdopodobieństwo:

- $1 > y_i > 0$
- $\sum_i y_i = 1$



Wyjście może być dowolną wartością, ale mamy problem z jego interpretacją.

$$\begin{aligned} P(Y = i|x, W, b) &= \text{softmax}_i(Wx + b) \\ &= \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}} \end{aligned}$$

**Preidykcja:**

$$y_{pred} = \operatorname{argmax}_i P(Y = i|x, W, b)$$



# Backpropagation a SGD i mini batch

## SGD stochastic gradient descent

- Metoda propagacji wstecznej pozwala nam obliczyć gradienty błędów w każdej warstwie
- SGD i mini batch oraz batch training mówią o tym w jaki sposób zmieniamy parametry w sieci.
  - SGD: uaktualnianie wag (parametrów) po każdym wzorcu,
  - Minibatch: uaktualnianie wag po  $m$  wzorcach ( $m$  rozmiar minibatch),
  - Batch: uaktualnianie wag po wszystkich wzorcach uczących (po całej epoce).



# Uczenie w ujęciu macierzowym

## Oznaczenia

Przyjmiemy następujące oznaczenia:

- $\mathbf{x}$  wektor kolumna (wzorzec wejściowy),
- $\mathbf{x}^t$  wektor wiersz,
- $\mathbf{z}^l$  wektor całkowitego pobudzenia neuronów w warstwie  $l$ ,
- $\mathbf{a}^l$  wektor aktywacji neuronów w warstwie  $l$ ,
- $\mathbf{b}^l$  wektor biasów w warstwie  $l$ ,
- $\mathbf{W}^l$  macierz wag między  $l$ -tą warstwą ukrytą a  $(l-1)$ ,
- $\boldsymbol{\delta}^l$  wektor błędów w warstwie  $l$ ,
- $C$  jest funkcja kosztu
- $\odot$  iloczyn Hadamarda, czyli iloczyn odpowiadających sobie składowych wektorów.



# Algorytm w ujęciu macierzowym

Zakładamy uczenie po paczce wzorców (minibatch)

- **Wejście  $\mathbf{x}$ :** dla warstwy wejściowej ustawiamy aktywację  $\mathbf{a}^1$  równą wzorcowi wejściowemu

- **Przesłanie wejścia w przód:** For each  $l=2,3,\dots,L$  oblicz
$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l \text{ oraz } \mathbf{a}^l = f(\mathbf{z}^l)$$

- **Błąd na wyjściu:** Obliczyć wektor
$$\boldsymbol{\delta}_x^L = \frac{\partial C_x}{\partial a^L} \odot f'(\mathbf{z}_x^L)$$

- **Rzutowanie błędu wstecz:** For each  $l=L-1,L-2,\dots,2$  oblicz

$$\boldsymbol{\delta}_x^l = \left( (\mathbf{W}^{l+1})^T \boldsymbol{\delta}_x^{l+1} \right) \odot f'(\mathbf{z}_x^l)$$

- **Uaktualnianie wag** For each  $l=L,L-1,\dots,2$  uaktualnianie wag zgodnie z regułą

$$\begin{aligned}\mathbf{W}^l &= \mathbf{W}^l - \frac{\alpha}{m} \sum_{\mathbf{x}} \boldsymbol{\delta}_x^l (\mathbf{a}_x^{l-1})^T \\ \mathbf{b}^l &= \mathbf{b}^l - \frac{\alpha}{m} \sum_{\mathbf{x}} \boldsymbol{\delta}_x^l\end{aligned}$$

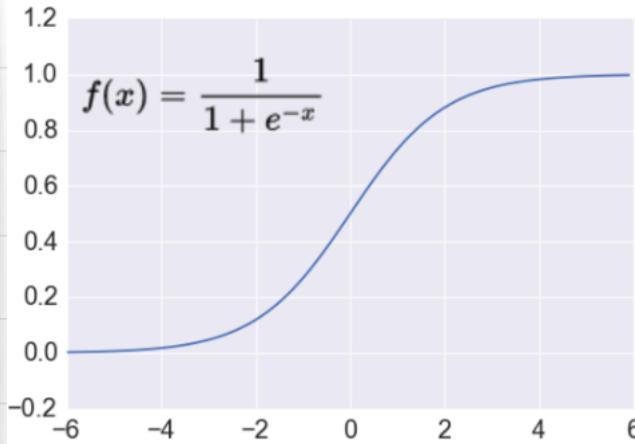


# Kilka uwag do algorytmu

- Macierzowe podejście do zapisu algorytmu pozwala na:
  - Użycie pakietów dedykowanych do przetwarzania macierzy (co znacznie przyspiesza pisanie kodu i przetwarzanie)
  - Użycie kart graficznych do przetwarzania równoległego
- Nie jest jednak najszybszym rozwiązaniem dlatego opracowano wiele innych wersji, które znajdują zastosowanie, szczególnie w odniesieniu do modeli głębokich.

# Problem zanikającego gradientu

## Funkcja aktywacji sigmoidalna



<http://adilmoujahid.com/images/activation.png>

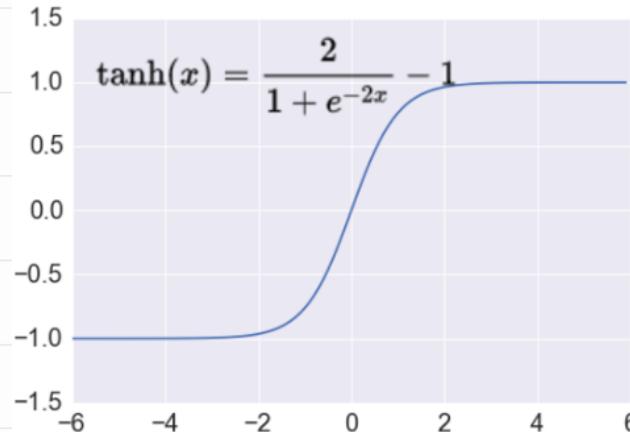
Przyjmuje wartości rzeczywiste w zakresie od 0 do 1. dla dużych ujemnych wartości dąży do 0 , dla dużych dodatnich do 1.

$$R^n \rightarrow [0,1]$$

- + Można interpretować jako zapalenie neuronu
  - 0 = nie jest zapalony
  - 1 = jest zapalony
- Neurony z sigmoidalnymi funkcjami aktywacji nasycają się i powodują, że gradienty stają się bliskie 0 a sieci przestają się uczyć.
  - Jeśli całkowite pobudzenie jest bardzo duże lub bardzo małe aktywacja jest prawie 0 lub 1 (neuron pracuje w nasyceniu)
    - (:( Gradient w tych regionach jest prawie równy 0
    - (:( Sygnał prawie nie płynie przez połączenia
    - (:( Jeśli początkowe wagi są duże, neuron będzie w nasyceniu

# Problem zanikającego gradientu

Funkcja aktywacji: tangens hiperboliczny (tanh)



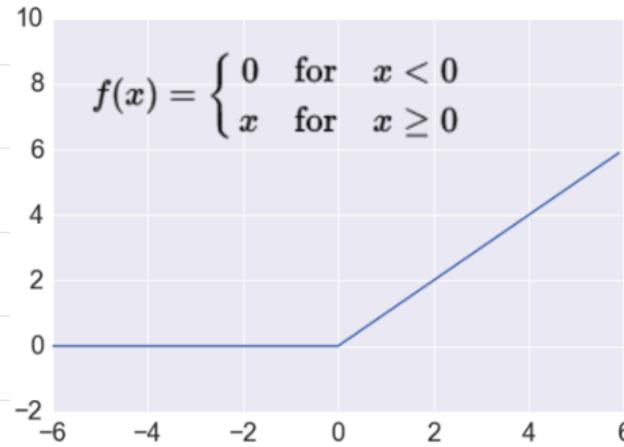
Przyjmuje wartości rzeczywiste w zakresie od -1 do 1. Ma dwie asymptoty -1 i 1.

$$R^n \rightarrow [-1,1]$$

- Podobnie jak przy funkcji sigmoidalnej, możemy pracować w zakresie nasycenia funkcji
- W odróżnieniu od f. sigmoidalnej jest funkcja jest symetryczna względem ośmiu
- Funkcje tanh możemy postrzegać jako wyskalowaną f. sigmoidalną:
- $\tanh(x) = 2\text{sigm}(2x) - 1$

# Rozwiązańe zanikającego gradientu

## Funkcja aktywacji ReLU



<http://adilmoujahid.com/images/activation.png>

Przyjmuje wartości Takes a real-valued number and thresholds it at zero

$$f(x) = \max(0, x)$$

$$\mathbb{R}^n \rightarrow \mathbb{R}_+^n$$

ReLU jest najczęściej używaną funkcją w modelach głębokich

😊 Model uczy się znacznie szybciej

- Przyspiesza zbieżność SGD
- dzięki liniowej formie

😊 Mniej złożona obliczeniowo

- W porównaniu do funkcji sigmoidalnej lub tanh
- Implementowana jako progowanie w zerze

😊 Chroni przed zanikającym gradientem

# Wybuchający gradient

## Funkcja aktywacji ReLU

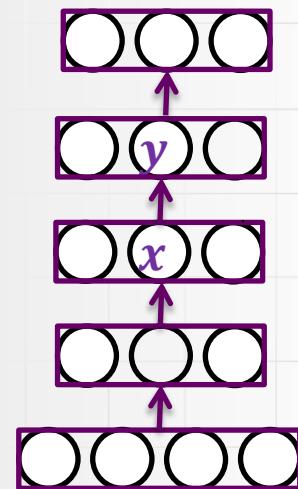
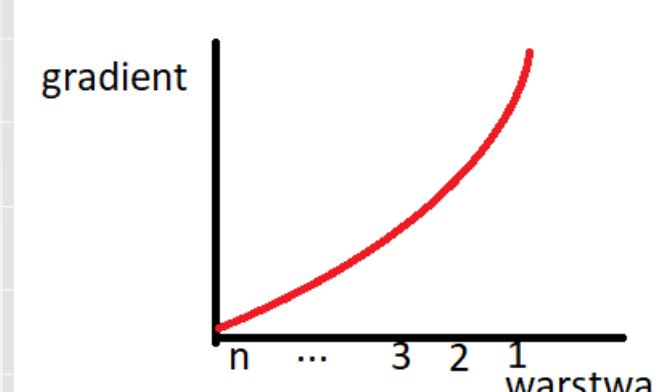
- Funkcja ReLU ,  $y = \max(0, z)$

szczególnie dla modeli głębokich, może powodować problem wybuchającego gradientu.

$$\frac{\partial y}{\partial z} = \begin{cases} 0 & z \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

$$\frac{\partial y}{\partial x} = \begin{cases} 0 & y = 0 \\ w_{yx} & \text{otherwise} \end{cases}$$

To może być problem gdy  $\left| \sum_y \frac{\partial y}{\partial x} \right| > 1$



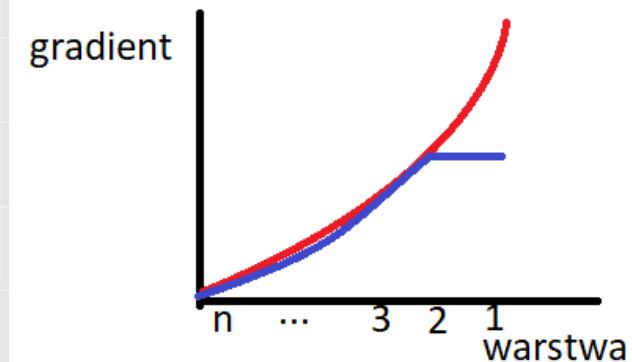
# Rozwiązańe zanikającego gradientu

- Użycie ReLU pomaga unikać zanikającego gradientu
- Obcięcie gradientu pomaga unikać eksplodującego gradientu

$$\Delta w_{xy} \sim \max\left(-\Delta_0, \min\left(\Delta_0, \frac{\partial E}{\partial w_{xy}}\right)\right)$$

- Użycie znaku gradientu pomaga unikać eksplodującego i znikającego gradientu

$$\Delta w_{xy} \sim \text{sign}\left(\frac{\partial E}{\partial w_{xy}}\right)$$





Wrocław  
University  
of Science  
and Technology

# Inne funkcje kosztu



HR EXCELLENCE IN RESEARCH



# Problem klasyfikacji

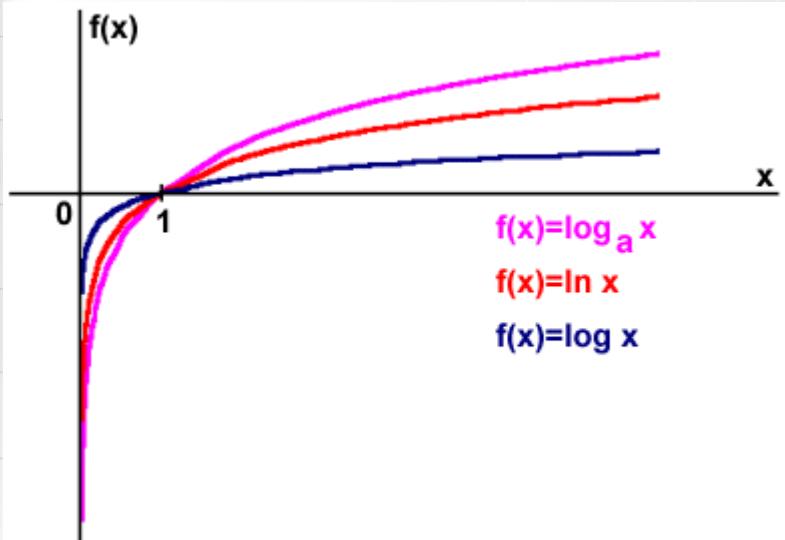
- **Klasyfikacja binarna** (mamy dwie klasy odpowiedzi kodowane jako 0 lub 1) np. XOR. Wystarczy nam jeden neuron
- **Klasyfikacja wieloklasowa** (mamy wiele klas, ale wzorzec może należeć WYŁĄCZNIE do jednej z nich). Używamy tyle neuronów wyjściowych, ile jest klas a kodowanie etykiet jest na zasadzie „1 z n”, (ang. one hot).
- **Klasyfikacja wieloetykietowa**- wzorzec może jednocześnie należeć do wielu klas, np. rozpoznajemy obrazy, które zawierają, niebo, słońce, dom, morze. Jeśli obraz przedstawia dom, drzewo i niebo, będzie jednocześnie należał do tych trzech klas. Neuronów wyjściowych mamy tyle ile rozpoznawanych klas.





- Dla funkcji kosztu wyrażonej jako błąd kwadratowy:  $E_p = 1/2 \sum_{k=0}^M (y_{pk} - \hat{y}_{pk})^2$
- Korekcja wag:  $\Delta_p w_{kj}^2(t) = \mu (y_{pk} - \hat{y}_{pk}) f_k^{2'}(z_{pk}^2) a_{pj}^1$   
zależna od pochodnej funkcji aktywacji.  
Problem z gradientem w nasyceniu krzywej.

# Logarytm



- Ta funkcja jest nieujemna  $C>0$ , ponieważ:
- Zakładamy, że wartości  $x$  są pomiędzy 0-1, dlatego każdy logarytm ma wartość ujemną, ale ponieważ mamy minus przed całym wyrażeniem, koszt jest dodatni
- Jeśli aktualne wyjście jest blisko pożądanego wyjścia dla danego wzorca wejściowego entropia skrośna jest bliska零.



## Binarna entropia krzyżowa

- Binarna entropia krzyżowa – klasyfikacja binarna

$$C = -1/P \left( \sum_{p=1}^P (y_p \ln \hat{y}_p - (1 - y_p) \ln(1 - \hat{y}_p)) \right)$$

gdzie P – liczba wszystkich wzorców,  $y_p$  to zadana odpowiedź,  $\hat{y}_p$  odpowiedź otrzymana na p-ty wzorzec.

- Obliczmy składową kosztu  $\frac{\partial C}{\partial w_{kj}^L} = \frac{\partial C}{\partial f_k^L} \frac{\partial f_k^L}{\partial z_k^L} \frac{\partial z_k^L}{\partial w_{kj}^L}$

$$\frac{y_{pk}}{\hat{y}_{pk}} - \frac{1-y_{pk}}{1-\hat{y}_{pk}} = \frac{\hat{y}_{pk}-y_{pk}}{\hat{y}_{pk}(1-\hat{y}_{pk})} \quad \frac{\partial f_k^L}{\partial z_k^L} = \hat{y}_{pk}(1-\hat{y}_{pk}) \quad a_{pj}^{L-1}$$

Zmiana wag w ostatniej warstwie nie zależy od gradientu kosztu:

$$w_{kj}^L(t+1) = w_{kj}^L(t) + \eta a_{pj}^{L-1} (y_{pk} - \hat{y}_{pk})$$

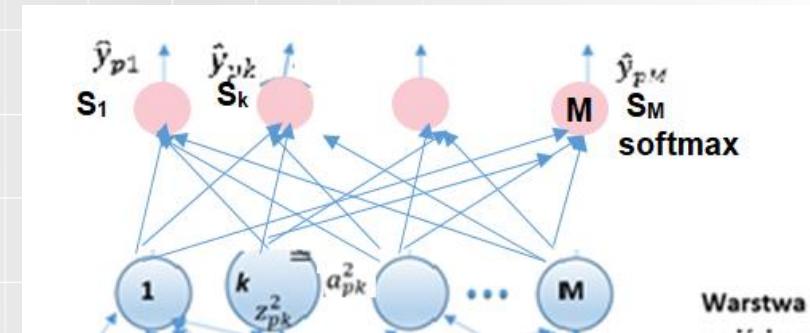


- Kategoryczna entropia krzyżowa – klasyfikacja wieloklasowa  $C = -1/P \sum_{p=1}^P (y_p \ln \hat{y}_p)$
- Oznaczenia takie jak w entropii krzyżowej binarnej
- Kodowanie wyjść 1 z n
-

# Funkcja softmax

- Funkcja softmax bierze M-wymiarowy wektor liczb rzeczywistych i produkuje inny wymiarowy wektor wartości rzeczywistych z zakresu (0, 1) który sumuje się do 1.0. Odwzorowuje on  $S(z)$ :  $R^M \rightarrow R^M$

$$S(a) : \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_M \end{bmatrix} \rightarrow \begin{bmatrix} S_1 \\ S_2 \\ \dots \\ S_M \end{bmatrix}$$



- Dzięki wprowadzeniu funkcji softmax możliwa jest probabilistyczna interpretacja wyjść:  $P(y = k|x) = \frac{e^{z_k}}{\sum_{j=1}^M e^{z_j}}$ , w dalszych wzorach proszę przyjąć, że a to jest równe z – czyli całkowite pobudzenia. Poprzednio przez a oznaczaliśmy aktywację tutaj funkcja aktywacji jest tożsamościowa.

$$S_j = \frac{e^{a_j}}{\sum_{k=1}^M e^{a_k}} \quad \forall j \in 1 \dots M$$

- Czyli:

$$S_j = P(y = j|a)$$

<https://stats.stackexchange.com/questions/265905/derivative-of-softmax-with-respect-to-weights>



# Funkcja softmax i jej pochodna

- Softmax ma M wejść i M wyjść. Dlatego licząc pochodne cząstkowe  $\frac{\partial S_i}{\partial a_j}$  otrzymamy:

$$DS = \begin{bmatrix} D_1 S_1 & \cdots & D_M S_1 \\ \vdots & \ddots & \vdots \\ D_1 S_M & \cdots & D_M S_M \end{bmatrix}$$

- Obliczmy pochodną cząstkową:

$$D_j S_i = \frac{\partial S_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^M e^{a_k}}}{\partial a_j}$$

- Mamy do czynienia z funkcją, która jest ilorazem dwóch funkcji:  $f(x) = \frac{g(x)}{h(x)}$ : jej pochodna to:  $f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{[h(x)]^2}$  gdzie:  
$$g_i = e^{a_i}$$
$$h_i = \sum_{k=1}^M e^{a_k}$$
- Pochodna  $g_i$  jest równa  $a_j$ , jeśli  $i=j$ . W każdym innym przypadku ta pochodna jest równa 0.

$$\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^M e^{a_k}}}{\partial a_j} = \frac{e^{a_i} \Sigma - e^{a_j} e^{a_i}}{\Sigma^2}$$



# Pochodna funkcji softmax

- Po przeorganizowaniu mamy:

$$\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^M e^{a_k}}}{\partial a_j} = \frac{e^{a_i} \Sigma - e^{a_j} e^{a_i}}{\Sigma^2}$$

- Czyli dla  $i=j$

$$\begin{aligned}\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^M e^{a_k}}}{\partial a_j} &= \frac{e^{a_i} \Sigma - e^{a_j} e^{a_i}}{\Sigma^2} \\ &= \frac{e^{a_i} \Sigma - e^{a_j}}{\Sigma} \\ &= S_i(1 - S_j)\end{aligned}$$

- Dla  $i \neq j$  mamy :

$$\begin{aligned}\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^M e^{a_k}}}{\partial a_j} &= \frac{0 - e^{a_j} e^{a_i}}{\Sigma^2} \\ &= -\frac{e^{a_j}}{\Sigma} \frac{e^{a_i}}{\Sigma} \\ &= -S_j S_i\end{aligned}$$

- Podsumowując:

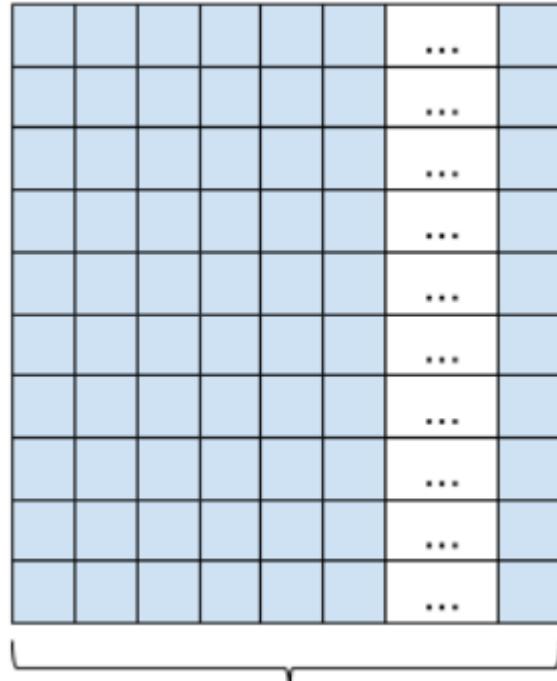
$$D_j S_i = \begin{cases} S_i(1 - S_j) & i = j \\ -S_j S_i & i \neq j \end{cases}$$

Chcemy policzyć gradient po  $W^2$ .

Prawdopodobieństwo  $P(W)$  na wyjściu to złożenie funkcji:

- $g$  mnożenia macierzy o wymiarach  $M \times L$  przez wektor  $a^1$
- softmax  $S(a)$

$$P(W) = S(g(W)) \\ = (S \circ g)(W)$$

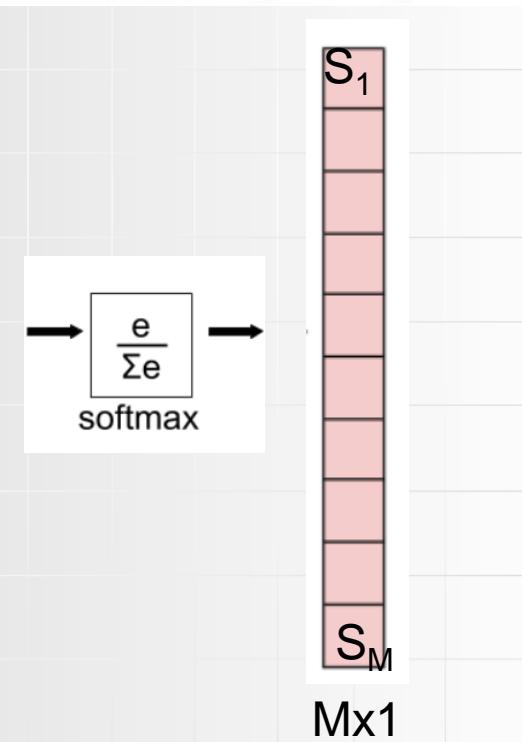
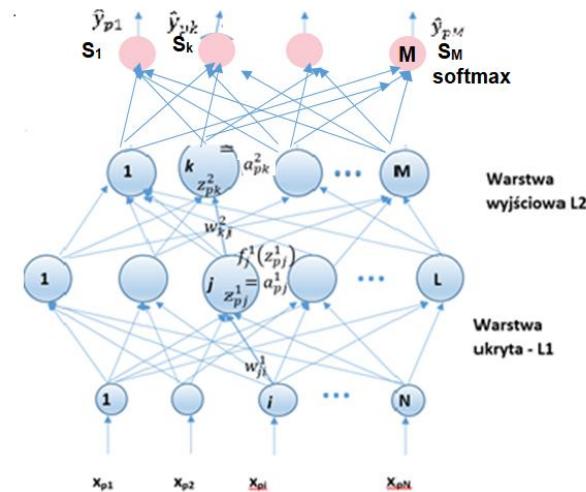


Macierz wag  $W^2$  w warstwie L2

$L \times 1$

$M \times 1$

$$a_M = z_M^2$$





- Jakobian możemy policzyć korzystając ze złożenia funkcji:

$$DP(W) = D(S \circ g)(W) = DS(g(W)) \cdot Dg(W)$$

- DS(a) policzyliśmy wcześniej musimy teraz policzyć Dg(W)
- Jakobian ma M wierszy i MxL kolumn

$$Dg = \begin{bmatrix} D_1g_1 & \cdots & D_{\text{ML}}g_1 \\ \vdots & \ddots & \vdots \\ D_Mg_1 & \cdots & D_{\text{ML}}g_M \end{bmatrix}$$

- Czym jest np.  $g_1$

$$g_1 = W_{11} \mathbf{a}_1 + W_{12} \mathbf{a}_2 + \cdots + W_{1L} \mathbf{a}_L$$

- Stąd mamy:

$$D_1g_1 = \mathbf{a}_1$$

$$D_2g_1 = \mathbf{a}_2$$

...

$$D_Lg_1 = \mathbf{a}_L$$

$$D_{L+1}g_1 = 0$$

...

$$D_{\text{ML}}g_1 = 0$$



- Otrzymujemy macierz Jakobiego

$$Dg = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_L & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_L \end{bmatrix}$$

- Jeśli będziemy się posługiwać indeksami wiersza i kolumn (i,j) możemy to zapisać jako:

$$D_{ij}g_t = \frac{\partial(W_{t1}\mathbf{a}_1 + W_{t2}\mathbf{a}_2 + \cdots + W_{tL}\mathbf{a}_L)}{\partial W_{ij}} = \begin{cases} \mathbf{a}_j & i = t \\ 0 & i \neq t \end{cases}$$

- Jeśli policzymy iloczyn skalarny i przeprowadzimy pewne uproszczenia otrzymamy:

$$\begin{aligned} D_{ij}P_t &= D_i S_t \mathbf{a}_j \\ &= S_t (\delta_{ti} - S_i) \mathbf{a}_j \end{aligned}$$



# Algorytm BP raz jeszcze dla entropii krzyżowej

Zakładamy uczenie po paczce wzorców (minibatch)

- **Wejście  $\mathbf{x}$ :** dla warstwy wejściowej ustawiamy aktywację  $\mathbf{a}^1$  równą wzorcowi wejściowemu
- **Przesłanie wejścia w przód:** For each  $l=2,3,\dots,L$  oblicz
$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l \text{ oraz } \mathbf{a}^l = f(\mathbf{z}^l)$$
- **Błąd na wyjściu:** Obliczyć wektor
$$\delta_{\mathbf{x}}^L = \frac{\partial C_{\mathbf{x}}}{\partial a^L} \odot f'(\mathbf{z}_{\mathbf{x}}^L) = (\mathbf{y}_p - \hat{\mathbf{y}}_p)$$
- **Rzutowanie błędu wstecz:** For each  $l=L-1,L-2,\dots,2$  oblicz

$$\delta_{\mathbf{x}}^l = \left( (\mathbf{W}^{l+1})^T \delta_{\mathbf{x}}^{l+1} \right) \odot f'(\mathbf{z}_{\mathbf{x}}^l)$$

- **Uaktualnianie wag** For each  $l=L,L-1,\dots,2$  uaktualnianie wag zgodnie z regułą

$$\mathbf{W}^l = \mathbf{W}^l - \frac{\alpha}{m} \sum_{\mathbf{x}} \delta_{\mathbf{x}}^l (\mathbf{a}_{\mathbf{x}}^{l-1})^T$$
$$\mathbf{b}^l = \mathbf{b}^l - \frac{\alpha}{m} \sum_{\mathbf{x}} \delta_{\mathbf{x}}^l$$



Wrocław  
University  
of Science  
and Technology

# Projektowanie aplikacji z użyciem sieci neuronowej



HR EXCELLENCE IN RESEARCH



# Problemy z użyciem sieci neuronowych

1. Proces projektowania niewystarczająco zdefiniowany w porównaniu do procesu wytwarzania oprogramowania
  - Jakie są konieczne kroki do tworzenia sieci neuronowej?
2. Jak projektować sieci neuronowe w sposób powtarzalny i przewidywalny?
3. Brak metod zapewniających jakość modeli sieci neuronowych i ich implementacji
  - Jak weryfikować implementacje?



# Kroki przy projektowaniu sieci

1. Zdefiniowanie problemu do rozwiązania ( do jakiej klasy problemów należy – klasyfikacja/regresja)
2. Zdefiniowanie cech wejściowych, podawanych na wejście sieci
  - Wybór sieci neuronowej (architektury)
  - Wymagania odnośnie funkcjonowania sieci
3. Przygotowanie wzorców
4. Implementacja sieci neuronowej
5. Uczenie sieci neuronowej
6. Informacja o jakości wyników otrzymywanych z sieci



# Faza specyfikacji problemu

- Elementy, które należy wyspecyfikować na tym etapie:
  - Wybór typu sieci neuronowej – architektury, bazując na doświadczeniu lub opublikowanych wynikach,
  - W jaki sposób zebrać i przetransformować zebrane dane, aby były użyteczne w procesie uczenia,
  - Potencjalne reprezentacje wejścia/wyjścia (sposób kodowania),
  - Metoda trenowania i testowania oraz wybór danych uczących,
  - Cele do osiągnięcia (dokładność i precyzja).



# Debugowanie sieci neuronowych

**Czasem efekt działania sieci nie jest lepszy niż losowe działanie. Co robić?**

Działania jakie możemy podjąć dotyczą:

- Wejścia
- Modelu architektury i inicjalizacji wag
- Funkcji straty
- Funkcji aktywacji
- inne



# Wejścia

- Wyucz sieć na małym zbiorze (np. 50 przykładów) wyłączając regularyzację=> sprawdzić czy f. straty szybko zbiega po kilku epokach. Jeśli nie, sieć jest zbyt głęboka
- Wyucz sieć na benchmarkowym zbiorze, żeby zobaczyć czy zbiór danych nie jest źródłem problemów
- Wycentruj średnią wzorców (wejścia powinny mieć średnią w 0).
- Sprawdź czy zbiór danych jest zrównoważony



# Model architektury i ..

- Uprość model, jeśli uważasz, że to problem architektury.
- Inicjalizacja wag - bardzo istotna (będziemy o tym mówili na następnym wykładzie).
- Martwe ReLU – jeśli wprowadzane wartości są mniejsze od 0, wówczas duża część sieci może przestać się uczyć => przyjrzyj się wagom początkowym.



# Inne wskazówki

- Zwizualizuj gradienty w każdej warstwie sieci , jeśli większość z nich jest bliska 0 => większość neuronów przestała się uczyć albo nie działa propagacja wsteczna.
- Stosunek wielkości wag do przyrostów wag. Jeśli ta wartość jest zbyt mała (mniejsza niż  $1e-4$ ) => wówczas niewiele możemy zrobić.
- Wykreśl wariancję wag w zależności od głębokości sieci. Jeśli obserwujemy że wykres jest przekrzywiony występuje problem z inicjalizacją lub normalizacją paczki wzorców (batch).



Wrocław  
University  
of Science  
and Technology



Zintegrowany  
Program Rozwoju  
Politechniki Wrocławskiej

# SIECI NEURONOWE



HR EXCELLENCE IN RESEARCH



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Politechnika Wrocławskiego

Unia Europejska  
Europejski Fundusz Społeczny





# Sieci neuronowe

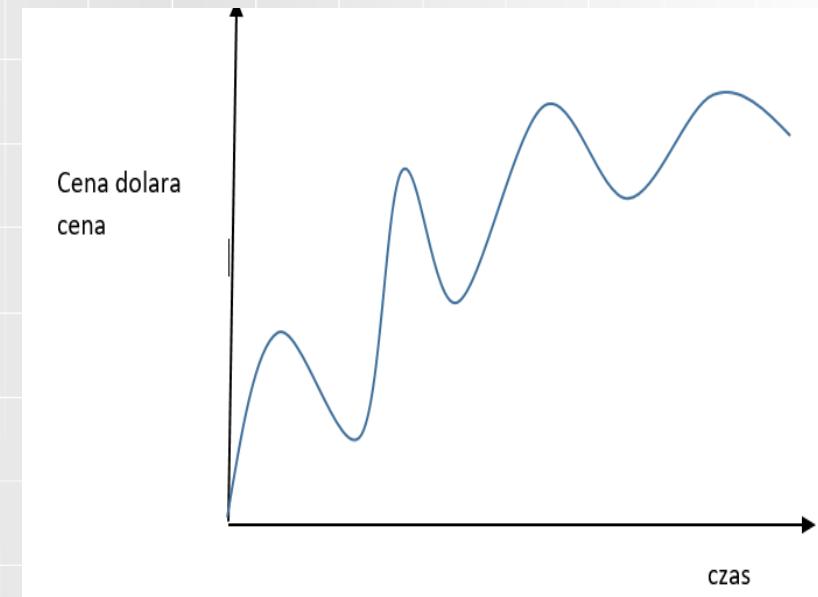
## Dobre praktyki

**URSZULA MARKOWSKA-KACZMAR**  
Katedra Inteligencji Obliczeniowej  
Wydział Informatyki i Zarządzania

# Projektowanie architektury sieci

## Warstwa wejściowa

- Liczba neuronów wejściowych jest zależna od **liczby cech i sposobu ich kodowania**
- Kodowanie dla wartości nominalnych (wyliczeniowych ) kodowanie „1 z n”, czyli musimy mieć tyle neuronów ile mamy wartości dla tej zmiennej:
  - Przykład zmiennych: kolor oczu, płeć, stan cywilny, miesiąc, dzień tygodnia itp.
- Kodowanie zmian a kodowanie wartości.
  - Jeśli występuje zmieniający się trend – lepiej uwzględniać zmiany

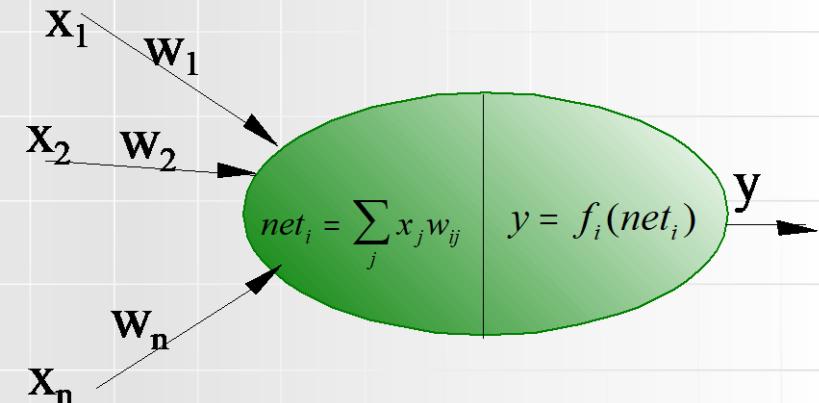


# Projektowanie architektury

## Warstwa wejściowa

- Wartości cech o wartościach rzeczywistych powinny być wyskalowane.  
Dlaczego?

- Rozważmy przykład:
  - Czy przydzielić kredyt – DECYZJA y:
  - Cechy wejściowe:
    - X1=wiek (zakres 18-80)
    - X2= zarobki (zakres 1200-40000)
    - X3 liczba osób na utrzymaniu (zakres 0-10)
  - Która cecha zdominuje działanie neuronu?





# Różne sposoby skalowania

- Najprościej:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- Normalizacja do średniej

$$x' = \frac{x - \text{mean}(x)}{\max(x) - \min(x)}$$

- Standaryzacja

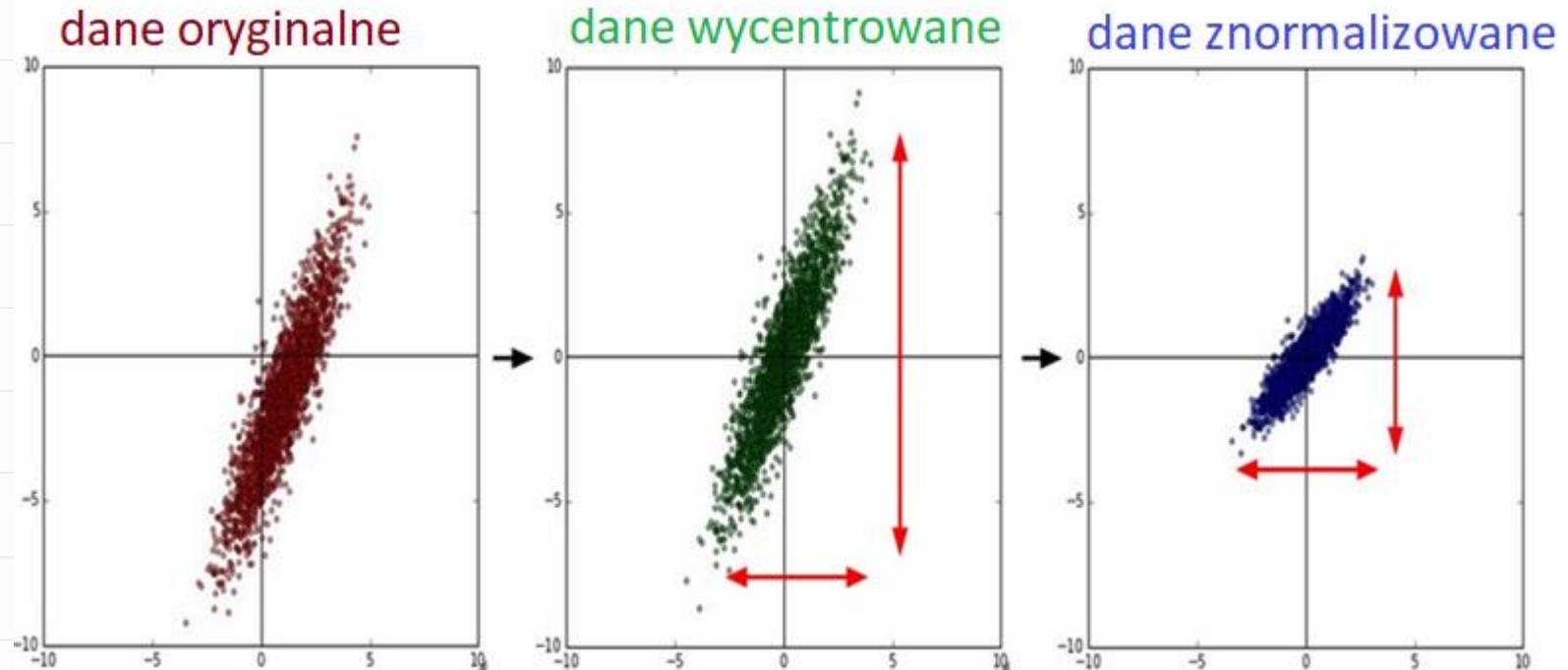
$$x' = \frac{x - \bar{x}}{\sigma}$$

- Normowanie do 1

$$x' = \frac{x}{||x||}$$

# Dane

Zanim wyskalujemy dane dobrze jest poznać ich charakterystykę



```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```



# Projektowanie architektury sieci

## Liczba neuronów wyjściowych

- Zależy od **rozwiązywanego problemu (klasyfikacja, regresja) i sposobu kodowania**
- Przykład z zadania 2 z laboratorium (rozpoznawanie cyfr). Jak możemy określić liczbę wyjść dla 10 klas?
  - binarnie
  - 1 z 10
  - 1 z 10 w skali termometrowej
  - 1 z 10, ale wyjście traktujemy jako prawdopodobieństwo (ostatnia warstwa realizuje funkcję softmax)



# Architektura sieci

Warstwa ukryta w odniesieniu do sieci płytowych

- W sieciach **płytkich**, w klasycznym podejściu liczba neuronów najczęściej jako:
- średnia arytmetyczna:

$$n_h = \frac{n_{in} + n_{out}}{2}$$

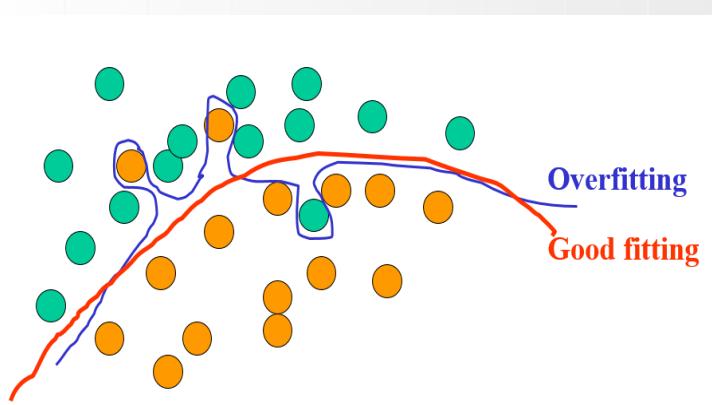
- średnia geometryczna

$$n_h = \sqrt{n_{in} n_{out}}$$

- $n_{in}$  - liczby neuronów wej.,
- $n_{out}$  - liczby neuronów wyj.

# Zdolności uogólniania a przeuczenie

## Przeuczenie (ang. overfitting)



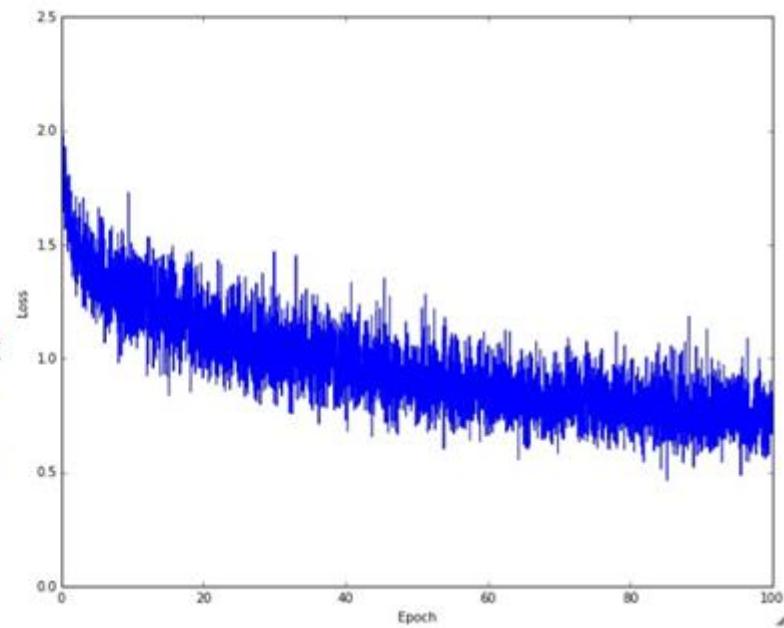
- Gdy występuje przeuczenie – sieć nie może dobrze generalizować.
- Dwie **ważne** cechy modelu:
- Zdolność do uogólniania (ang. generalization, good fitted model);
  - Przeuczenie (ang. overfitting).

- Dane trenujące zawierają informację o pewnych prawidłowościach w odwzorowaniu wejścia w wyjście.
- Dane trenujące zawierają również szum
  - Dane wyjściowe (etykiety) mogą być niewiarygodne.
  - Istnieje **błąd próbowania**. Wynika z pewnych incydentalnych regularności w przygotowanych danych.
- Kiedy już wyuczymy model, nie jest on w stanie wskazać, które regularności są prawdziwe, a które wynikają z błędu próbkowania.
  - Model dopasowuje się do obu rodzajów regularności.

Hinton, 2006

# Funkcja straty

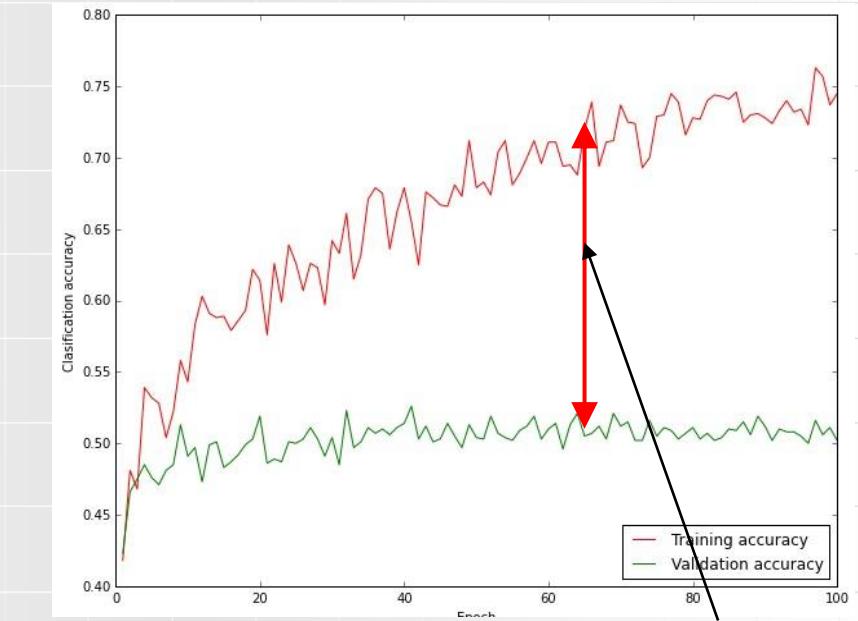
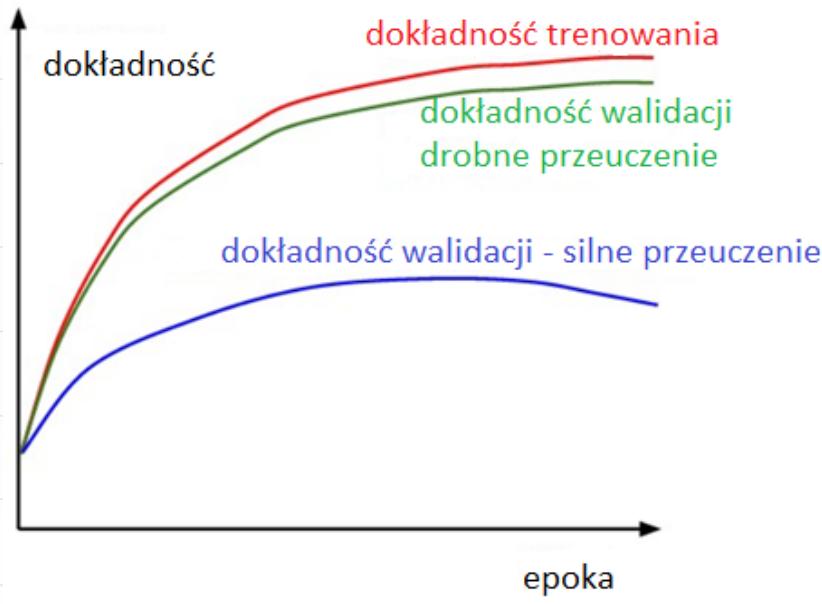
ang. Loss function



- Wpływ współczynnika uczenia na przebieg funkcji straty w kolejnych epokach

# Trenowanie i walidacja

Badamy dokładność na zbiorze testowym i walidującym



Duża różnica  
= przeuczenie

- Accuracy (dokładność)- liczba wszystkich poprawnie sklasyfikowanych wzorców d o wszystkich wzorców



Wrocław  
University  
of Science  
and Technology

# Sposoby unikania przeuczenia



HR EXCELLENCE IN RESEARCH



# Uogólnianie a struktura sieci

- Mniejsza struktura sieci => mniejszy koszt obliczeniowy w trakcie uczenia=> większa zdolność uogólniania.
- Redukcja struktury sieci (pruning)- algorytmy obcięcia wag lub redukcji neuronów.
- Metodą gradientową oblicza się wrażliwość błędu względem wagi lub neuronu.
- Wagi o najmniejszej wrażliwości są usuwane a proces uczenia jest dalej kontynuowany.



# Regularyzacja

Metoda funkcji kary

- Modyfikuje się funkcję straty ( $C$ ), aby proces uczenia eliminował najmniej potrzebne wagи.
- Regularyzacja L1 wprowadza sumę absolutnych wartości wag.

$$C = C_0 + \frac{\lambda}{P} \sum_w |w|$$

$$w(t+1) = w(t) - \eta \frac{\partial C_0}{\partial w} - \frac{\eta \lambda}{P} sign(w)$$

- Regularyzacja L2 wprowadza do  $C_0$  sumę kwadratów wag

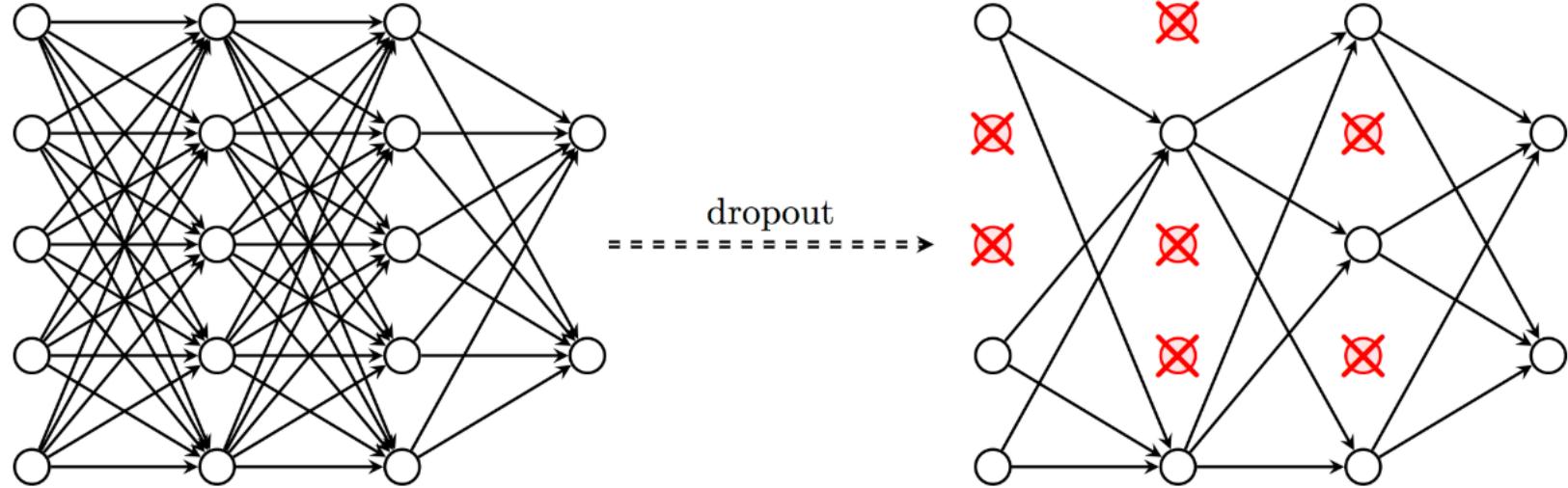
$$C = C_0 + \frac{\lambda}{2P} \sum_w w^2$$

$$w(t+1) = w(t) - \eta \frac{\partial C_0}{\partial w} - \frac{\eta \lambda}{P} w = \left(1 - \frac{\eta \lambda}{P}\right) w - \eta \frac{\partial C_0}{\partial w}$$

- **Biasy zostawiamy bez regularyzacji !!!!**

# Dropout

## Czasowe wyłączanie neuronów



- Przy podawaniu kolejnej paczki neuronów w sposób losowy z przyjętym prawdopodobieństwem  $p_d$  wyłączamy neurony (one istnieją, ale nie są brane pod uwagę przy obliczaniu przesłania sygnału i propagacji błędów dla wzorców z tej paczki).
- Może być wersja, w której połowę neuronów wyłączamy, przy każdej paczce inną część.



# Dropout

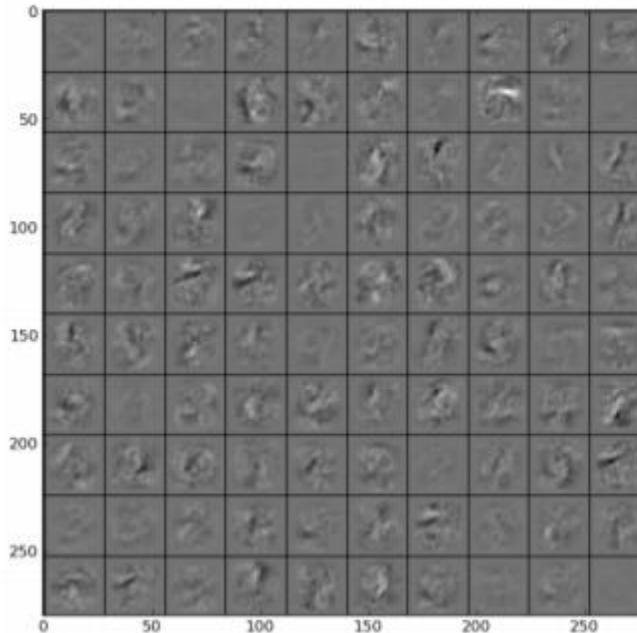
- Wejścia i wyjścia są nietknięte (w tej wersji).
- Po przesłaniu paczki wzorców modyfikujemy wagi.
- Wracamy do wszystkich neuronów i dla kolejnej paczki znowu wyłączamy losowo połowę neuronów.
- Powtarzamy to co zrobiliśmy dla pierwszej paczki. Przez powtarzanie tego procesu wagi są zmieniane przy założeniu, że połowa neuronów była nieaktywna.
- Przy odtwarzaniu, pracując z całą siecią mamy 2 x więcej neuronów. Dlatego, aby to skompensować zmniejszamy o połowę wagi na połączeniach wychodzących z warstwy ukrytej.



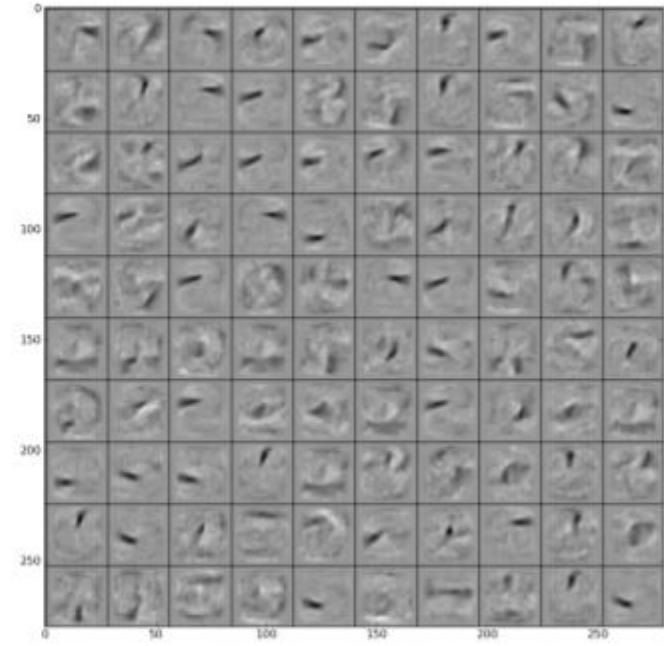
# Dropout

- Dla każdej paczki ponownie wybierana jest inna część neuronów i uczona (zmieniane są tylko wagi uczestniczące w przesłaniu aktualnej paczki wzorców)
- Po wyuczeniu sieci wracamy do sieci z pełną strukturą, ale mamy teraz 2x więcej neuronów, zakładając ze połowa neuronów była wyłączona=> zmniejszamy wszystkie wagi o połowę
- Stosuje się też **dropconnect**, czyli wyłączanie na podobnej zasadzie pewnych połączeń w sieci, ale zazwyczaj mniej skuteczne.

# Efekt dropoutu



(a)



(b)

Wizualizacja cech wyuczonych przez neurony pierwszej warstwy ukrytej używając (a) backprop (b) dropout na zbiorze MNIST.



# Dropout

Dlaczego pomaga?

- Zapobiega wspólnej adaptacji neuronów (detektorów cech w warstwie ukrytej)
- Możemy patrzeć na taką sieć jak na zespół sieci (zespół klasyfikatorów)
- Szczególnie istotny dla głębokich modeli, bo tam problem z przetrenowaniem jeszcze większy.
-



# Powiększanie zbioru uczącego

- Sztuczne powiększenie zbioru uczącego. Wykonywane są różnego rodzaju transformacje na zbiorze uczącym, np.:
  - Przesunięcia,
  - Wprowadzanie szumu,
  - Małe obroty,
  - zmiany skali,
  - Odbicia symetryczne
  - Rozmycie.

Istnieją gotowe narzędzia do tego celu

# Przykład



Przykład generowania nowych obrazów przy zastosowaniu różnych operacji na obrazie

źródło: <https://www.semanticscholar.org/paper/Data-Augmentation-for-Plant-Classification-Pawara-Okafor/a5da2c3b4174449d13dd746b7d00897c6bc1f334/figure/2>

# Wczesne zatrzymywanie uczenia

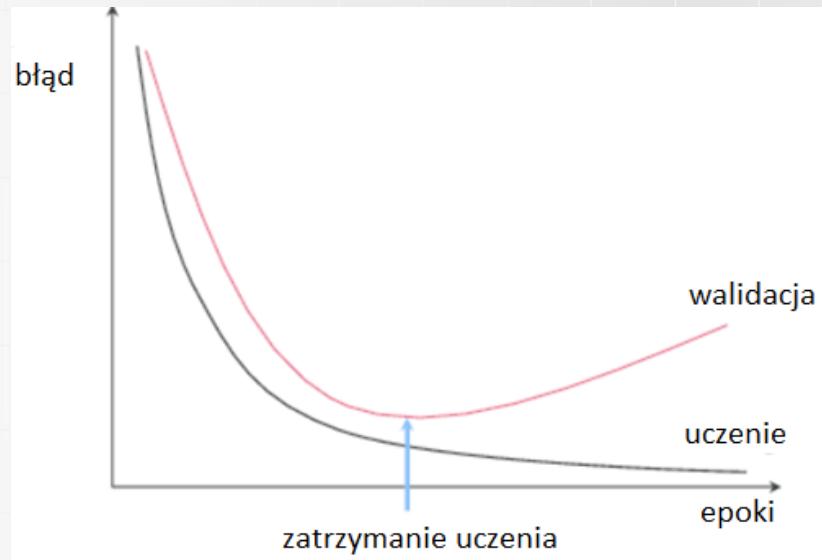
ang. early stopping



- Nie zawsze jesteśmy w stanie przy wylosowanych wagach osiągnąć zadany dopuszczalny błąd
- Jeśli błąd będzie zbyt mały sieć za bardzo dopasuje się do danych uczących i będzie źle generalizować

# Wczesne zatrzymywanie uczenia

ang. early stopping



- Dostępny zbiór danych dzielimy na trzy zbiory: trenujący, walidujący (dev – development) i testowy.
- Na zbiorze uczącym trenujemy sieć sprawdzając, co epokę na zbiorze walidującym
- Jeśli błąd na zbiorze walidującym zaczyna rosnąć o więcej niż założona dopuszczalna wartość, kończymy proces uczenia.



# Podsumowanie

Metody zapobiegające przeuczeniu

- Regularyzacja
  - Dropout
  - Wczesne zatrzymanie uczenia
  - Sztuczne powiększane zbioru uczącego
- 
- Najczęściej wszystkie metody stosowane są łącznie



Wrocław  
University  
of Science  
and Technology

# Badanie zdolności uogólniania



HR EXCELLENCE IN RESEARCH



# Metody

Cel - chcemy sieć wyuczyć cech wzorców a nie wzorców

- Sprawdzanie zdolności do uogólniania na **zbiorze testowym**
- **Walidacja krzyżowa** ( $k$ -crossvalidation)- dzielimy zbiór na  $k$  części (foldów). Uczymy na  $(k-1)$  foldach na  $k$ -tym sprawdzamy. Przeprowadzamy  $k$  procesów uczenia i testowania.
- **hold-one-out** - szczególny przypadek walidacji krzyżowej dla  $k=1$
- **Hold out** – inaczej zbiór walidujący, na którym sprawdzamy, czy sieć się nie przeuczyła
- **Bootstraping** tworzymy pewną liczbę podzbiorów danych z oryginalnego zbioru, wykorzystując losowanie ze zwracaniem (uczących i testowych)



Wrocław  
University  
of Science  
and Technology

# Miary oceny sieci neuronowej jako klasyfikatora



HR EXCELLENCE IN RESEARCH

# Macierz pomyłek (confusion matrix)

## Klasyfikacja binarna

		Actual Values				
		1	0		P aktualne	N aktualne
Predicted Values	1				P' predykowanego	True Positive TP
	0				N' predykowanego	False Negative FN

<https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62>

**True** – oznacza te wzorce, które powinny być z tej klasy

**False** – te wzorce, które wpadły do innej klasy (nie zostały właściwie rozpoznane)



# Opis zmiennych w tabeli

- True positive TP
  - Pozytywna predykcja
  - Etykieta pozytywna
- False positive FP
  - Pozytywna predykcja
  - Etykieta negatywna
- True negatives TN
  - Negatywna predykcja
  - Etykieta negatywna
- False negatives FN
  - Negatywna predykcja
  - Etykieta pozytywna

W statystyce false positive określa się jako błąd typu I “type I error” false negative jako “type II error”



# Miary oceny klasyfikatora

*Recall* inaczej *sensitivity* określa jak dobrze model unika złej klasyfikacji tzn. false negatives.

$$\text{recall} = \text{sensitivity} = \frac{TP}{TP + FN}$$

Specyficzność modelu określa jak dobrze model unika false positives (czyli wzorce z klasy negatywnej wpisuje do pozytywnej)

$$\text{specificity} = \frac{TN}{TN + FP}$$

W idealnym świecie nasz model powinien mieć 100% specificity (nikt kto nie jest chory nie będzie zaklasyfikowany jako chory i 100% *sensitivity* (wszyscy chorzy są wykryci)).



# Miary oceny klasyfikatora

## ACCURACY (dokładność)

Stopień prawidłowej klasyfikacji

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy nie jest zbyt dobrą miarą jeśli klasy nie są zbalansowane. W takiej sytuacji, jeśli model będzie klasyfikował wszystko do klasy większościowej, wtedy nasz ocena będzie wyższa mimo że model nie działa prawidłowo.

**PRECISION** (kompletność) – stopień powtarzalności wyników w tych samych warunkach. Tutaj opisuje wartość pozytywnej predykcji

$$precision \text{ (kompletność)} = \frac{TP}{TP + FP}$$

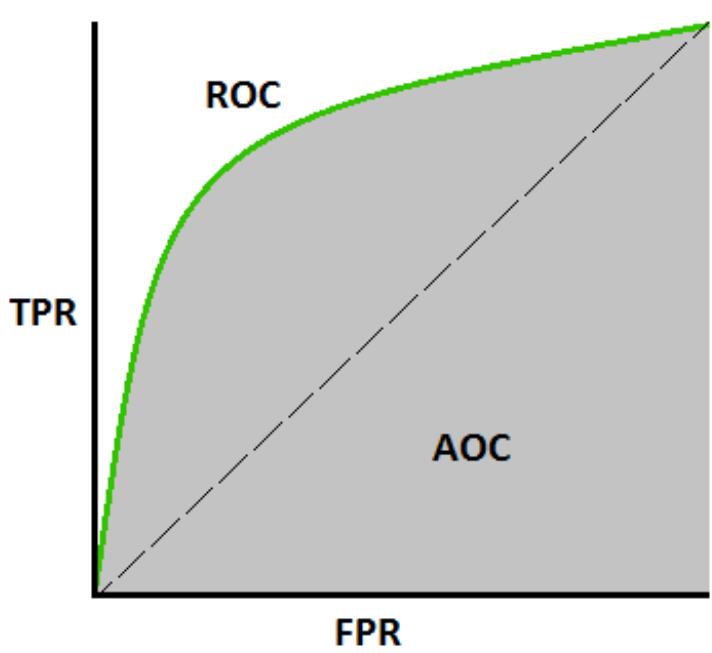
**F1 SCORE** albo **F1 MEASURE** – opisuje dokładność modelu . Jest to średnia harmoniczna precyzji i recall

$$F1 = \frac{2TP}{2TP + FP + FN}$$

F1 jest często używana do całkowitej oceny modelu

# Krzywa AUC-ROC

Mówi nam jak dobrze model rozróżnia dwie klasy



- AUC =**Area Under The Curve**
- ROC =**Receiver Operating Characteristics**
- Należy podać próg (kiedy stwierdzamy, że wzorzec należy do klasy pozytywnej)
- Im większe AUC tym lepszy klasyfikator (lepiej rozróżnia klasy). Najlepszy model ma AUC=1, słaby AUC=0 (wskażania klas są odwrotne), AUC=0,5 działa losowo.
- TPR- True Positive rate, FPR False Positive rate



# Podsumowanie

Omówiliśmy:

- Heurystyki dotyczące projektowania architektury sieci
- Metody unikania przeuczenia
- Badanie zdolności uogólniania sieci
- Miary oceny klasyfikatorów.



Wrocław  
University  
of Science  
and Technology



# SIECI NEURONOWE



HR EXCELLENCE IN RESEARCH



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Politechnika Wrocławskiego

Unia Europejska  
Europejski Fundusz Społeczny





# **Sieci neuronowe**

## **Techniki wykorzystywane w uczeniu sieci neuronowych**

**URSZULA MARKOWSKA-KACZMAR**  
Katedra Inteligencji Obliczeniowej  
Wydział Informatyki i Zarządzania



Politechnika Wrocławskiego



# Optymalizacja gradientowa

## Metoda największego spadku gradientu (Gradient descent)

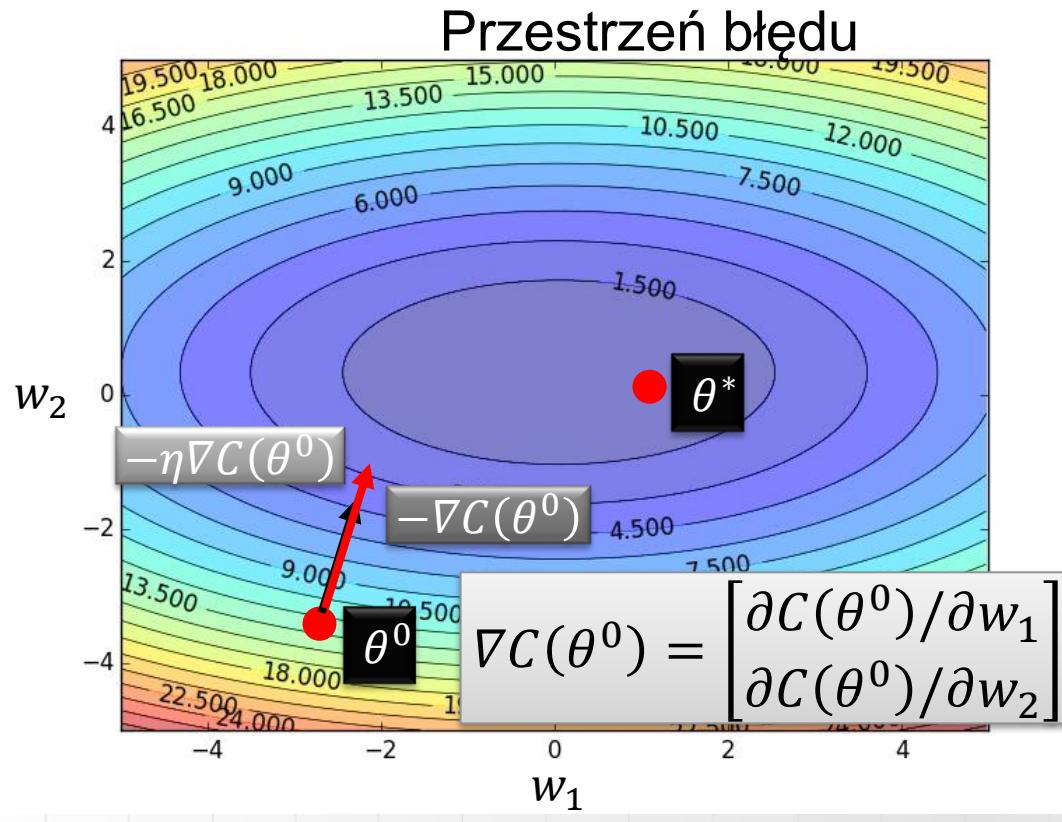
- minimalizuje funkcję kosztu (błędu)  $J(\vartheta)$  parametryzowaną przez parametry modelu  $\vartheta \in R^d$
  - służy do określania parametrów w procesie ich iteracyjnego uaktualniania, w kierunku przeciwnym do gradientu  $\nabla_{\vartheta} J(\vartheta)$ .
  - Współczynnik uczenia  $\eta$  określa rozmiar kroku używany w celu osiągnięcia minimum błędu.
- **Batch gradient descent**  
Oblicza gradient na podstawie całego zbioru uczącego  
$$\vartheta = \vartheta - \eta \cdot \nabla_{\vartheta} J(\vartheta)$$
Gwarantuje zbieżność do globalnego minimum dla funkcji wypukłych i lokalnego min dla funkcji niewypukłych
  - **Stochastic gradient descent**  
Wykonuje aktualizacje po każdym wzorcu trenującym  $x(i)$  o etykiecie  $y(i)$ :  
$$\vartheta = \vartheta - \eta \cdot \nabla_{\vartheta} J(\vartheta; x(i); y(i))$$
Duża wariancja funkcji kosztu
  - **Mini-batch gradient descent**  
Wykonuje aktualizacje po  $n$  wzorcach  
$$\vartheta = \vartheta - \eta \cdot \nabla_{\vartheta} J(\vartheta; x(i:i+n); y(i:i+n))$$
Redukuje wariancję (lepsza zbieżność)

# Gradient descent

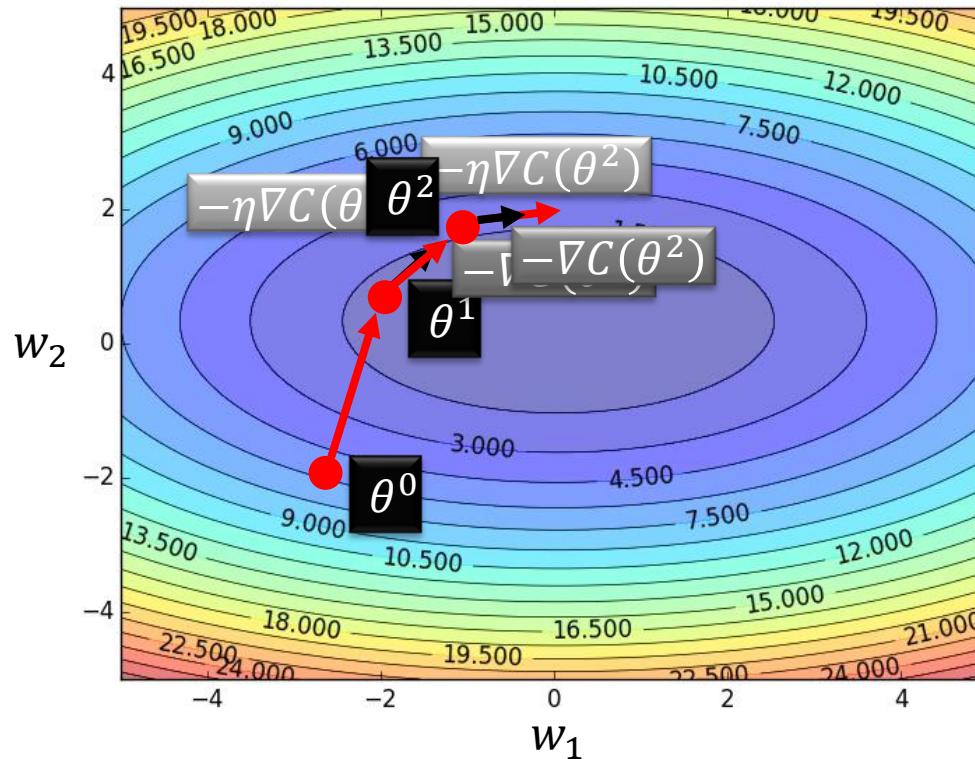
Ta część przygotowana na podstawie: <https://ruder.io/optimizing-gradient-descent/>

Kolory reprezentują wartości funkcji kosztu C

Przyjmijmy, że mamy tylko 2 wagi  $w_1$  i  $w_2$ .



# Gradient descent



Mogliśmy osiągnąć  
min.....

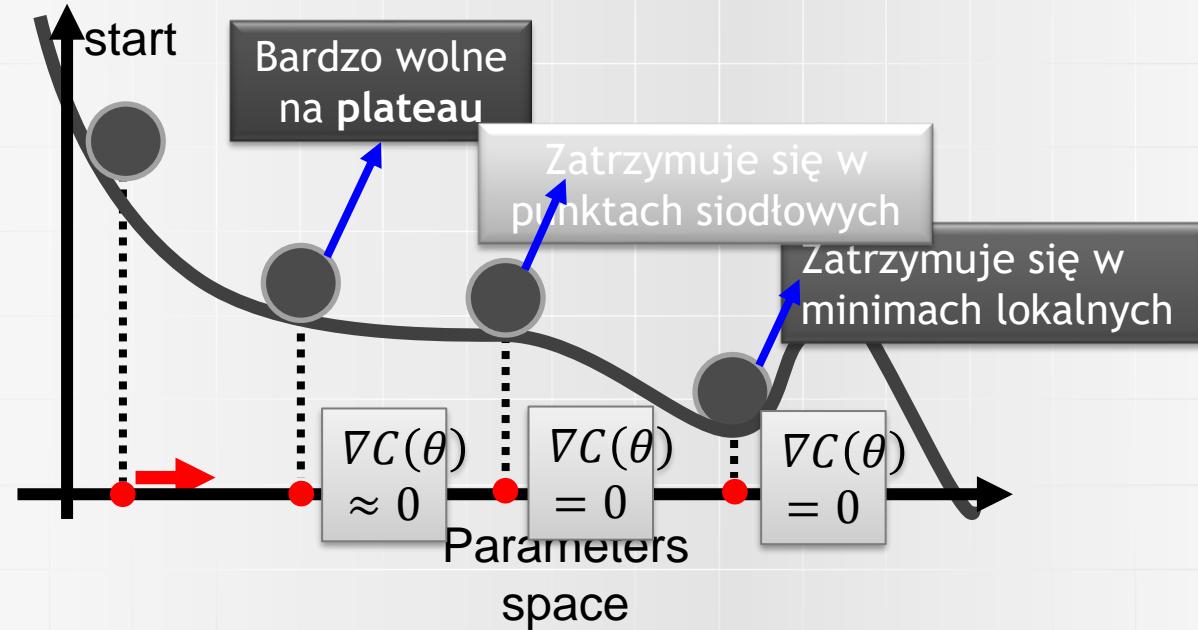


# Mini batch gradient - problemy

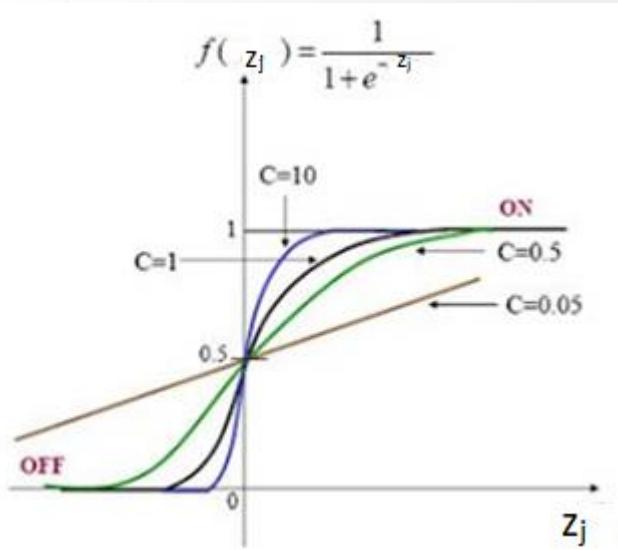
Nie gwarantuje zbieżności!!

- Wybór właściwego współczynnika uczenia jest problematyczny;
- Metody obniżania współczynnika uczenia wraz z uczeniem muszą być znane wcześniej (nie adaptują się do statystyki zbioru danych).
- Ten sam współczynnik uczenia odnosi się do wszystkich aktualnień parametrów.
- Poważnym problemem dla niewypukłych funkcji jest wpadanie w suboptimalne optima lokalne a głównie punkty siodłowe otoczone przez plateau.

# Oprócz minimów lokalnych....



# Uczenie



- Funkcja logistyczna jest bardzo płaska dla **bardzo dużych i bardzo małych wartości**
- Kiedy pobudzenie jest duże lub małe pracujemy w częściach sigmoidy, gdzie jest ona płaska, pochodna bliska 0 => (**zanikający gradient**). Mimo, że występuje błąd, sieć się nie uczy , bo pochodna jest bliska 0.
- Pamiętamy, że zdefiniowane błędy i co za tym idzie przyrost zmiany wag jest zależny od pochodnej funkcji aktywacji
$$\delta_{pk}^2 = (\hat{y}_{pk} - y_{pk}) f_k^{(2)}(z_{pk}^2)$$

$$\Delta_p w_{kj}^o(t) = \eta(y_{pk} - o_{pk}) f_k^{(o)}(net_{pk}^o) i_{pj}$$



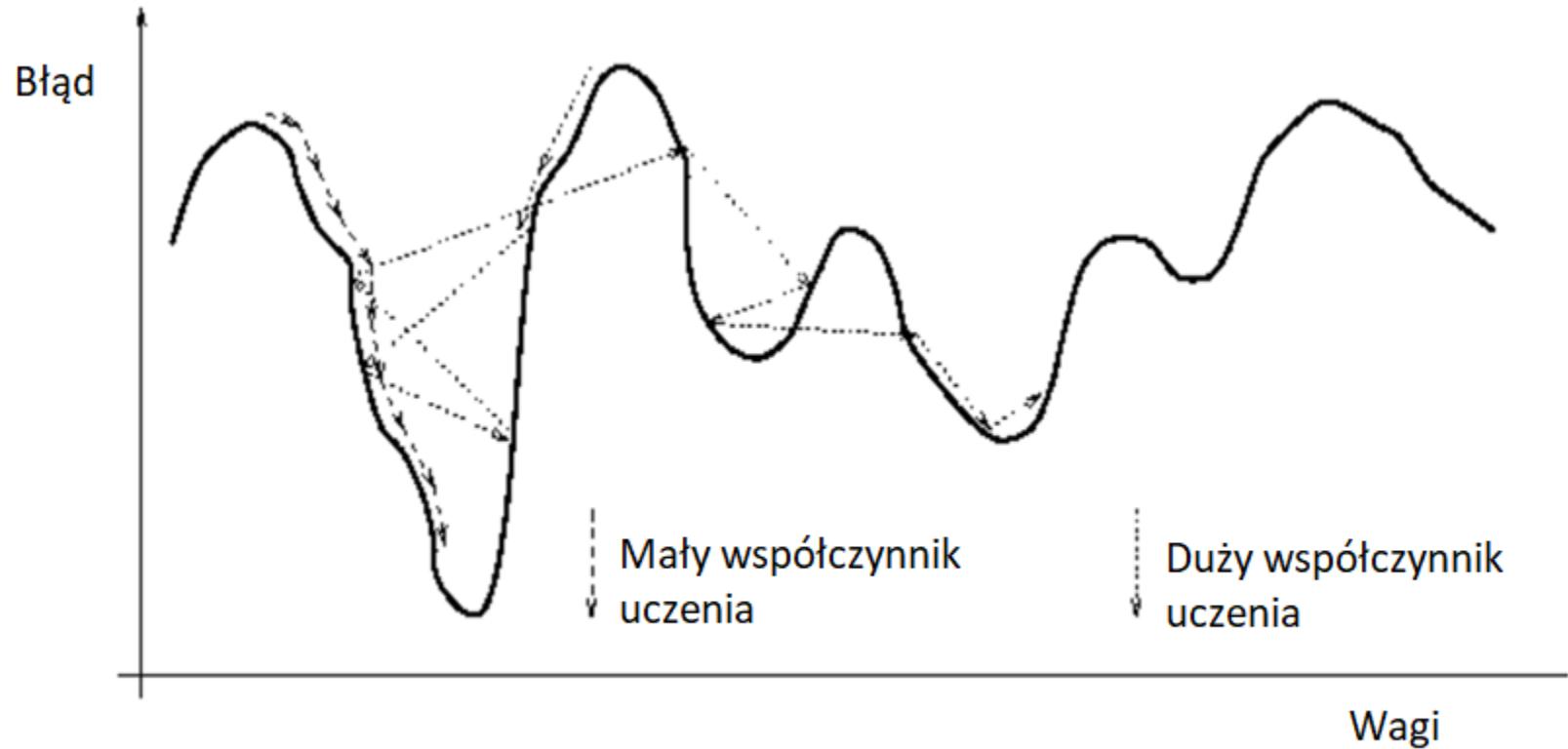
# Jak ustawić wartość współczynnika uczenia

## Wczesne podejścia

- *Stał współczynnik uczenia*
  - rzadko używany
  - Najmniej efektywny sposób. Na podstawie badań empirycznych można przyjąć, że
$$\mu \leq \min(1/n_{in}),$$
gdzie  $n_{in}$  to liczba wejść  $i$ -tego neuronu w warstwie.
- *Adaptacyjny dobór współczynnika uczenia*
  - W tej metodzie, na podstawie porównania sumarycznego błędu  $\varepsilon$  w  $i$ -tej iteracji z jej poprzednią wartością, określa się strategię zmian współczynnika uczenia.



# Adaptacyjny współczynnik uczenia



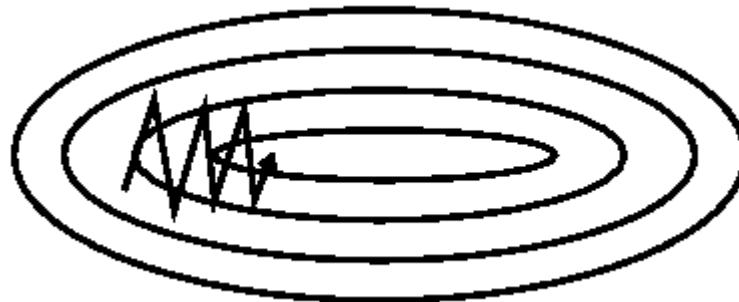


# Adaptacyjny współczynnik uczenia

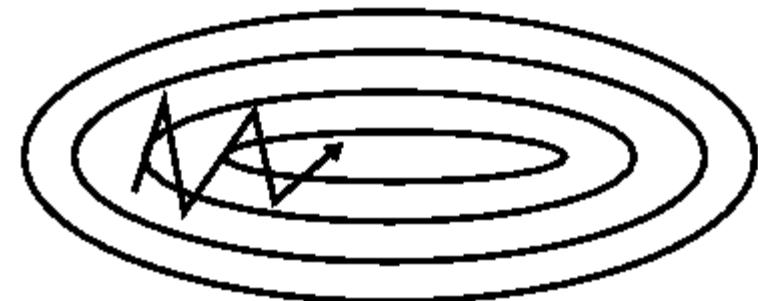
- W celu przyspieszenia uczenia, współczynnik uczenia jest zwiększany, sprawdzając czy błąd nie zacznie rosnąć. Dopuszcza się przy tym nieznaczny wzrost błędu w porównaniu z poprzednią wartością.
- Jeżeli przez  $\mu_i$  oraz  $\mu_{i-1}$  oznaczymy odpowiednio współczynnik uczenia w odpowiednich iteracjach, jeśli  $\varepsilon_i > k_w \varepsilon_{i-1}$  ( $\varepsilon_i$  to błąd w kroku i), to należy zmniejszyć wartość współczynnika uczenia  $\mu_i = \rho_d \mu_{i-1}$
- $\rho_d$  to współczynnik zmniejszania współczynnika uczenia. W przeciwnym przypadku, gdy  $\varepsilon_i \leq k_w \varepsilon_{i-1}$  przyjmuje się, że  $\mu_i = \rho_i \mu_{i-1}$ ,  
gdzie  $\rho_i$  jest współczynnikiem zwiększającym wartość współczynnika uczenia,  $\rho_i = 1.05$   $\rho_d = 0.7$ ,  $k_w = 1.04$ .
- Współczynnik uczenia określa krok uczenia i był oraz jest przedmiotem badań.

# Metody optymalizacji współczynnika uczenia

- SGD ma problem z przesuwaniem się po „wąwozach”. W takich przypadkach pojawiają się oscylacje



SGD bez momentum



SGD z momentum



# Momentum

- Używając momentum dodajemy do bieżącego wektora wag część  $\gamma$  zmian wag z kroku poprzedniego:

$$v_t = \gamma v_{t-1} + \eta \nabla_{\vartheta} J(\vartheta)$$

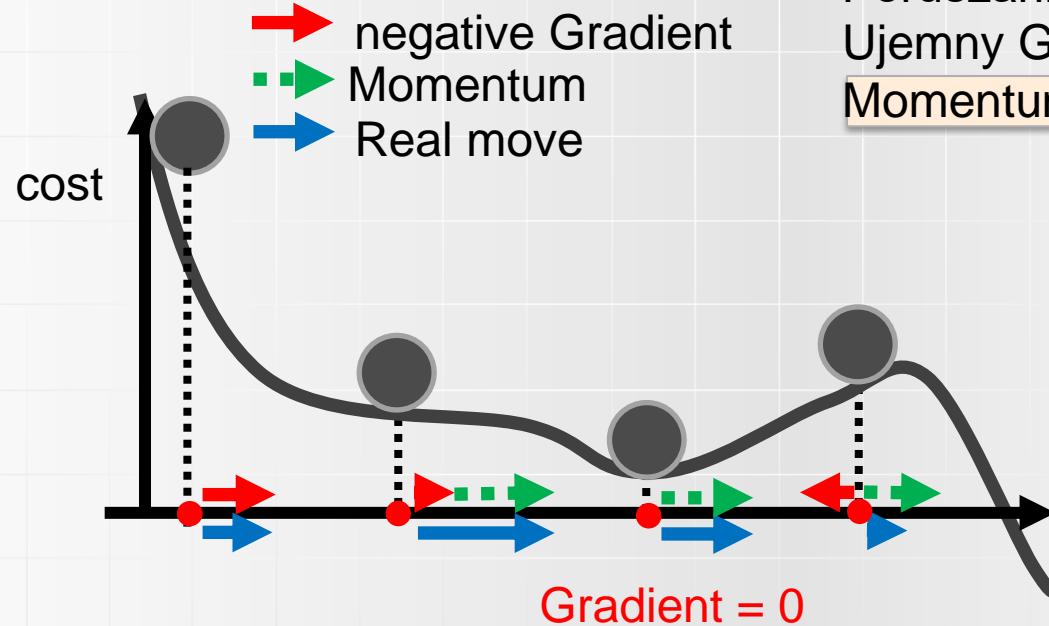
$$\vartheta = \vartheta - v_t$$

- $\gamma$  jest zazwyczaj = 0.8 - 0.9

# Momentum

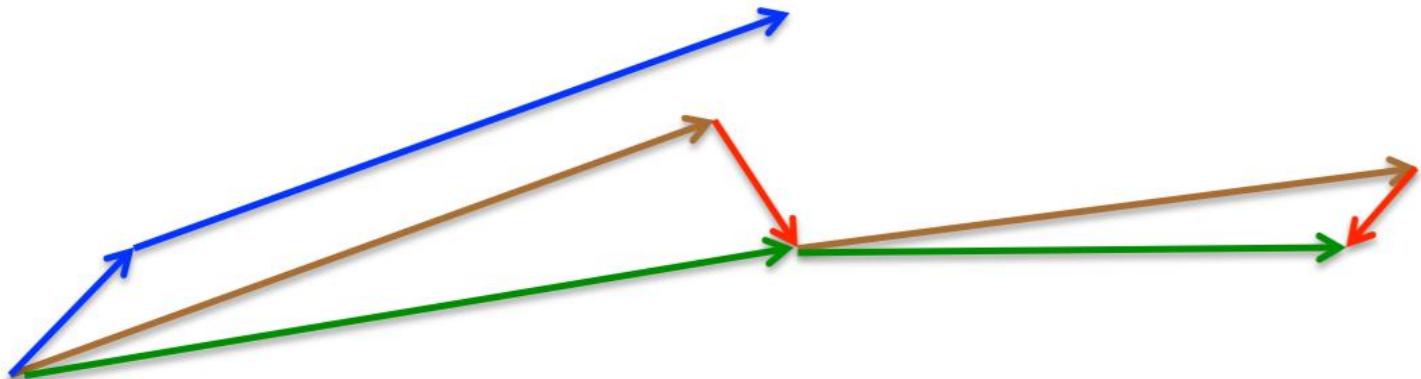
Ciągle nie mamy gwarancji, że osiągniemy globalne min, ale .....

Poruszanie się = Ujemny Gradient + Momentum



# Nesterov accelerated gradient (NAG)

- NAG najpierw skacze w kierunku poprzedniego zakumulowanego gradientu (wektor **brown**)
- Mierzy gradient w tym miejscu i wykonuje korekcję (**red** wektor)
- W wyniku otrzymujemy korekcję zgodną z NAG (**green** wektor)
- Zapobiegamy zbyt szybkiemu uaktualnianiu wag
- Wektor w kolorze **blue** odpowiada klasycznemu momentum





# Porównanie

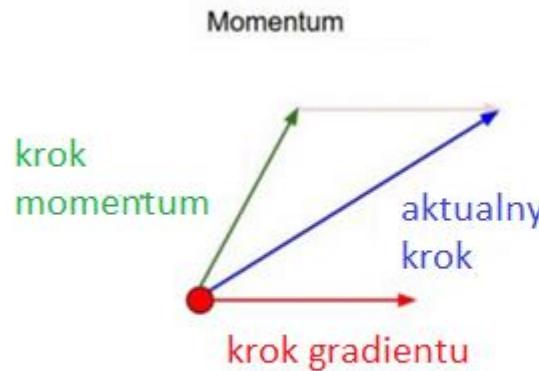
## Momentum (klasyczne)

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$
$$\theta = \theta - v_t$$



## Momentum Nesterova

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$



# Jest wiele optymalizatorów

## ADAGRAD

- Oryginalnie parametry są zmieniane następująco:

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C(\theta^{t-1})$$

- Teraz każda waga jest rozważana oddziennie

$$w^{t+1} \leftarrow w^t - \eta_w g^t$$

Współczynnik zmienia się dynamicznie w zależności od innych parametrów

$$\begin{aligned} g^t &= \frac{\partial C(\theta^t)}{\partial w} \\ \eta_w &= \frac{\eta}{\sqrt{\sum_{i=0}^t (g^i)^2}} \end{aligned}$$

constant

Suma kwadratów poprzednich gradientów



# ADAGRAD

$$\eta_w = \frac{\eta}{\sqrt{\sum_{i=0}^t (g^i)^2}}$$

$$w_1 \begin{array}{|c|}\hline g^0 \\ \hline 0.1 \\ \hline \end{array}$$

Współczynnik uczenia  
ang. learning coefficient:

$$\frac{\eta}{\sqrt{0.1^2}}$$

$$\frac{\eta}{\sqrt{0.1^2 + 0.2^2}}$$

$$w_2 \begin{array}{|c|}\hline g^0 \\ \hline 20.0 \\ \hline \end{array}$$

Współczynnik uczenia  
ang. learning coefficient:

$$\frac{\eta}{\sqrt{20^2}}$$

$$\frac{\eta}{\sqrt{20^2 + 10^2}}$$

## OBSERWACJA:

1. Współczynnik uczenia jest coraz mniejszy
2. Im mniejsza pochodna , tym większy współczynnik uczenia. Jest to prawdziwe również na odwrót.



# Adagrad

- **Korzyść:** eliminuje potrzebę ręcznego ustawiania współczynnika uczenia. Trzeba ustawić jedynie jego początkową wartość.
- **Słabość:** sumowanie gradientów w mianowniku (wszystkie wartości są dodatnie) powoduje wzrost sumy, co może powodować, że współczynnik uczenia będzie nieskończonym mały.



# Adadelta

- Redukuje Adagrad intensywne monotoniczne zmniejszanie współczynnika uczenia
- Ogranicza okno akumulowanych poprzednich gradientów do określonego rozmiaru  $w$ .
- Zamiast przetrzymywania przeszłych gradientów=> suma gradientów jest rekurencyjnie definiowana jako zmniejszająca się średnia wszystkich przeszłych gradientów (decaying average of all past squared gradients).

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma)g_t^2.$$

- W czystym SGD mamy

$$\begin{aligned}\Delta\theta_t &= -\eta \cdot g_{t,i} \\ \theta_{t+1} &= \theta_t + \Delta\theta_t\end{aligned}$$

- W Adagrad:  $\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$

- Zastępujemy  $G_t$  przez  $E[g^2]_t$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t.$$

- root mean squared (RMS)

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t.$$



# Adadelta

- Jednak w tym przypadku nie zgadzają się hipotetyczne jednostki
- Definiujemy inną eksponencjalnie malejącą średnią (parametrów podniesionych do kwadratu):  
$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1 - \gamma) \Delta\theta_t^2.$$
- 

- RMS:  $RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}.$

- Ponieważ ta wartość jest nieznana, aproksymujemy ją za pomocą

$$RMS[\Delta\theta]_{t-1}$$

- Podstawiając zamiast  $\eta$   $RMS[\Delta\theta]_{t-1}$  otrzymujemy regułę uaktualniania wag:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

- **Nawet nie potrzebujemy współczynnika uczenia**



# RMSprop

- Nieopublikowany pomysł Hintona
- RMSprop i Adadelta pojawiły się w tym samym czasie z potrzeby rozwiązania problemów związanych z radykalnym zmniejszaniem współczynnika uczenia.
- RMSprop jest identyczny z początkowym opisem Adadelta:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

- RMSprop też dzieli współczynnik uczenia przez zmniejszającą się średnią kwadratów gradientów
- Hinton sugerował, żeby  $\gamma$  było równe 0.9, a domyślna wartość dla współczynnika uczenia  $\eta = 0.001$



# Adaptive Moment Estimation (Adam)

- Oprócz pamiętania eksponencjalnie zmniejszającej się średniej poprzednich kwadratów gradientów  $v_t$  jak w Adadelta albo RMSprop, Adam utrzymuje też eksponencjalnie zmniejszającą się średnią poprzednich gradientów  $m_t$ , podobnych do momentum.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

- $m_t$  i  $v_t$  estymują pierwszy moment (średnią) i drugi moment (wariancję) gradientów – stąd nazwa metody
- Przy zainicjowaniu  $m_t$   $v_t$  jako wektory = 0, Autorzy zaobserwowali, że są one obciążone w kierunku 0, dlatego wprowadzono korekty:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- Zmiana wag:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

- Sugestia autorów:  $\beta_1=0.9$ ,  $\beta_2=0.999$ ,  $\epsilon=10^{-8}$



# ADAM - algorytm

**Require:**  $\alpha$ : Stepsize

**Require:**  $\beta_1, \beta_2 \in [0, 1]$ : Exponential decay rates for the moment estimates

**Require:**  $f(\theta)$ : Stochastic objective function with parameters  $\theta$

**Require:**  $\theta_0$ : Initial parameter vector

$m_0 \leftarrow 0$  (Initialize 1<sup>st</sup> moment vector)

$v_0 \leftarrow 0$  (Initialize 2<sup>nd</sup> moment vector)

$t \leftarrow 0$  (Initialize timestep)

**while**  $\theta_t$  not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)

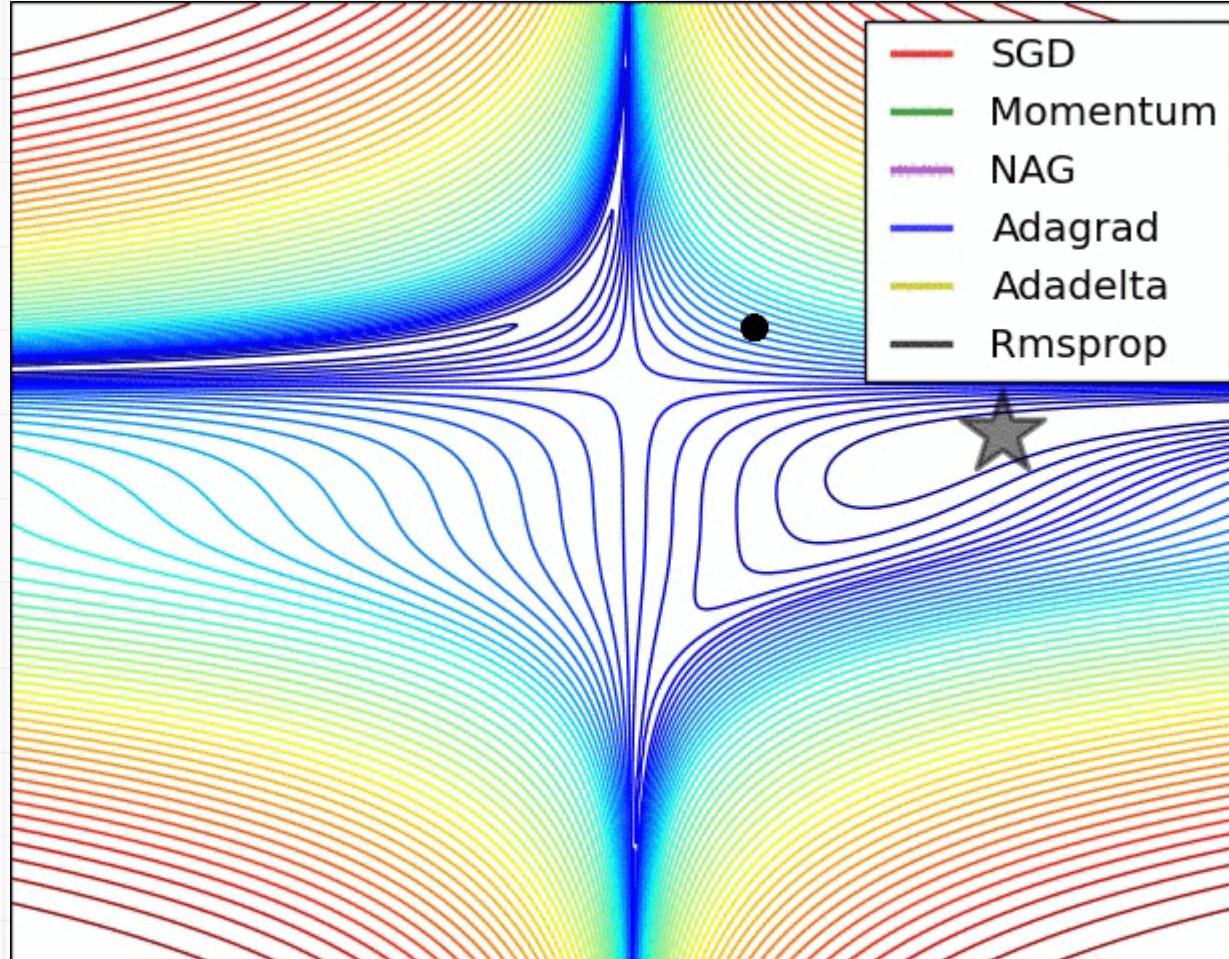
$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)

**end while**

**return**  $\theta_t$  (Resulting parameters)

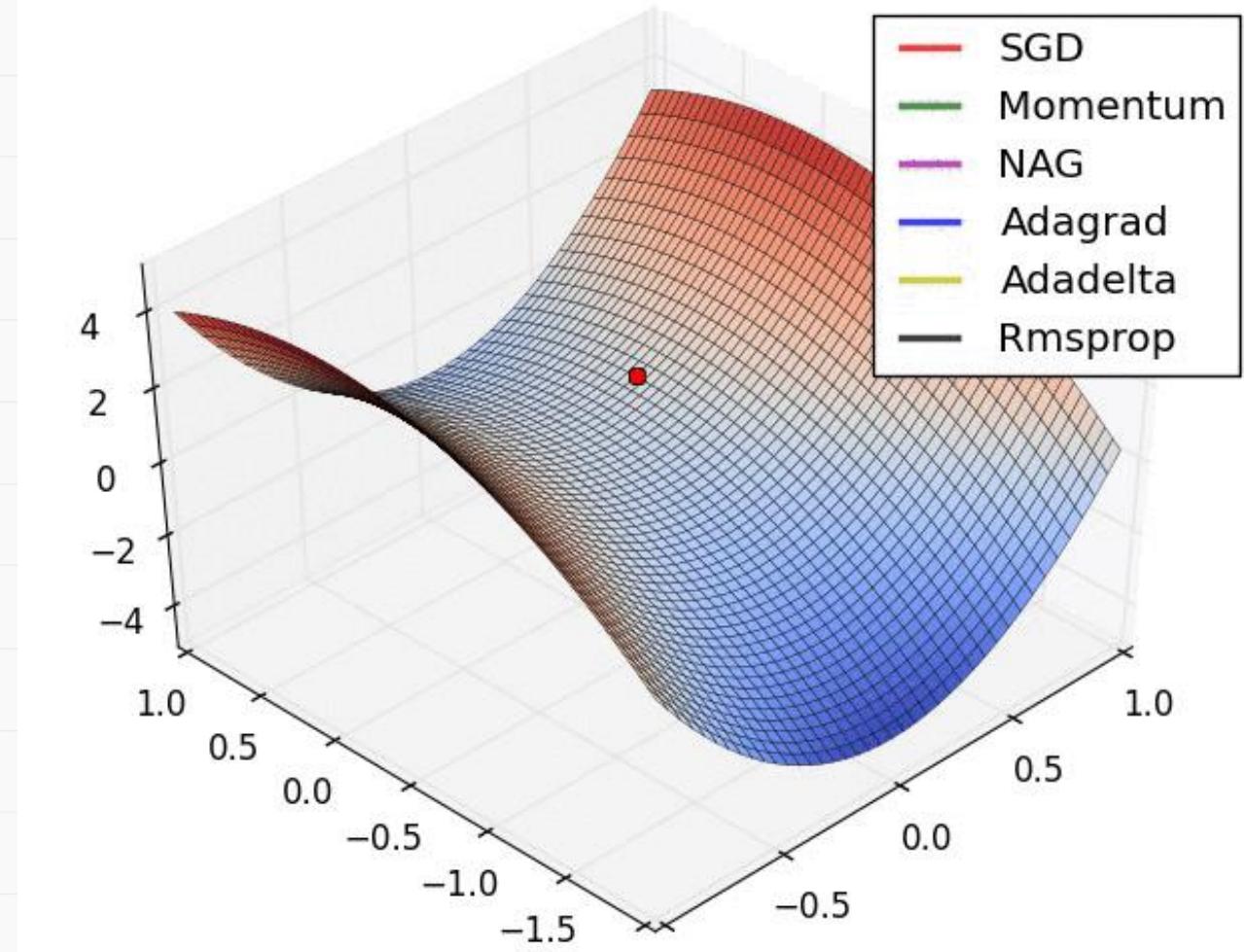
# Wizualizacja



• źródło: [Alec Radford](#)



# Optymalizacja SGD dla punktach siodłowych



źródło: [Alec Radford](#)



# Metody inicjalizacji wag

Pamiętajmy o zanikającym lub  
eksplodującym gradiencie





# Charakterystyka problemu

- Wagi inicjalizowane są losowo.
- Metoda uczenia zapewnia (przy odpowiedniej liczbie powtórzeń) zbieżność do najbliższego minimum funkcji straty.
- Ten rodzaj inicjalizacji jest podatny na pojawianie się zanikającego lub wybuchającego gradientu.
- Metoda powinna być uzależniona od zastosowanej funkcji aktywacji.
- UWAGA: inicjalizacja wag jednakową wartością np. =0 prowadzi do jednakowego wpływu każdego neuronu na funkcję kosztu => jednakowe gradienty. Każdy neuron uczy się tej samej cechy. **NALEŻY UNIKAĆ TAKIEJ SYMETRII**



# Sieci płytkie

- Stosowane podejście do inicjalizacji wag:

1. Randomizacja w zakresie  $[-\frac{a}{\sqrt{n_{in}}}, \frac{a}{\sqrt{n_{in}}}]$

- $n_{in}$  liczba wejść do danego neuronu
- $a$  jest tak dobrane, żeby wariancja wag odpowiadała punktowi maksymalnej krzywizny funkcji aktywacji (dla standardowej sigmoidy 2.38)

2. Randomizacja wag w zakresie  $[-\frac{2}{\sqrt{n_{in}}}, \frac{2}{\sqrt{n_{in}}}]$

3. Randomizacja wag w zakresie  $[-\sqrt[n_{in}]{N_h}, \sqrt[n_{in}]{N_h}]$

$N_h$  - liczba neuronów w warstwie ukrytej

$N_{in}$  - jak poprzednio

Randomizacja wag w warstwie wyjściowej – losowo w zakresie {-0.5; 0.5}



# W sieciach głębokich

- **Dla ReLU w sieciach głębokich :**
- Wygeneruj wagi z rozkładu normalnego ze średnią w 0 i odchyleniu standardowym równym =1
- Pomnóż wylosowane wagi przez wartość  $[-\sqrt{2} \frac{1}{\sqrt{n_{in}}}, \sqrt{2} \frac{1}{\sqrt{n_{in}}}]$
- gdzie  $n_{in}$  jest liczbą wejść do danego neuronu

Xavier Glorot and Yoshua Bengio: [Understanding the difficulty of training deep feedforward neural networks](#),



# W sieciach głębokich

- **Jeśli Tanh jest funkcją aktywacji :**
- Wygeneruj wagi z rozkładu Gaussa dla rozkładu ze średnią w 0 i odchyleniem standardowym = 1
- Wszystkie próbki przemnóż przez pierwiastek kwadratowy z  $(1/n_{in})$ ; gdzie  $n_{in}$  jest liczbą neuronów wejściowych dla danego neuronu.

Xavier Glorot and Yoshua Bengio: *Understanding the difficulty of training deep feedforward neural networks,*



# Inicjalizacja Xaviera

- Wszystkie wagi w warstwie  $l$  są wybierane losowo z rozkładu normalnego ze średnią (mean)  $\mu=0$  i wariancją  $\sigma^2=1/n^{l-1}$  gdzie  $n_{l-1}$  jest liczbą neuronów w warstwie  $l-1$ . Biasy inicjowane są jako 0

$$W^{[l]} \sim \mathcal{N}(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}})$$
$$b^{[l]} = 0$$

- Wariancja może też zamiast  $\mathcal{N}(0, \frac{1}{n^{l-1}})$  być określana jako średnia harmoniczna  $\mathcal{N}(0, \frac{2}{n^{[l-1]} + n^{[l]}})$  z  $\frac{1}{n^{[l-1]}}$
- Inicjalizacja Xaviera ustawia wagi w warstwie z rozkładu normalnego ograniczonego przez

$$\pm \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}$$

- gdzie  $n_i$  jest liczbą wchodzących połączeń (“fan-in,” ) do danej warstwy  $n_{i+1}$  jest liczbą wychodzących połączeń z tej warstwy ( “fan-out.” )



# Iinicjalizacja He

- He trenując 30-warstwową sieć konwolucyjną z ReLU, używając iinicjalizacji Xaviera zauważył, że sieć przestała się uczyć.
- **Zaproponowali następującą procedurę**
  - Utwórz tensor o wymiarach odpowiednich dla macierzy wag i ustaw losowo wartości wag zgodnie z rozkładem normalnym.
  - Każdą wygenerowaną losowo liczbę przemnóż przez  $\sqrt{2}/\sqrt{n}$  gdzie  $n$  jest liczbą wchodzących połączeń do danej warstwy z poprzedniej warstwy (“fan-in”).
  - Biasy zainicjuj jako 0.



# Podsumowanie

- Istotne elementy wpływające na szybkość i efektywność modelu sieci neuronowej to:
  - Optymalizacja współczynnika uczenia
  - Zastosowanie momentum
  - Odpowiednia inicjalizacja wag.



Wrocław  
University  
of Science  
and Technology



# SIECI NEURONOWE



HR EXCELLENCE IN RESEARCH



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Politechnika Wrocławskiego

Unia Europejska  
Europejski Fundusz Społeczny





# Sieci neuronowe

## Przykłady zastosowań pływkich sieci neuronowych

**URSZULA MARKOWSKA-KACZMAR**  
Katedra Inteligencji Obliczeniowej  
Wydział Informatyki i Zarządzania



Politechnika Wrocławskiego

# Wczesne podejście

Ręcznie określane cechy, mały rozmiar problemu



Politechnika Wrocławskiego



# Przykład





# Przykład - rozpoznawanie warzy

- 20 osób
- 32 obrazy na osobę
- Zmienne to cechy wyciągane z obrazów: wyraz twarzy, kierunek patrzenia, okulary słoneczne, tło, ubiór pozycja twarzy na obrazie
- 624 obrazy w skali szarości, rozdzielcość 120x128, intensywność piksela 0 do 255
- Można określić wiele celów rozpoznawania: identyczność, kierunek patrzenia, płeć, czy są okulary.. itd.



# Rozpoznawanie twarzy

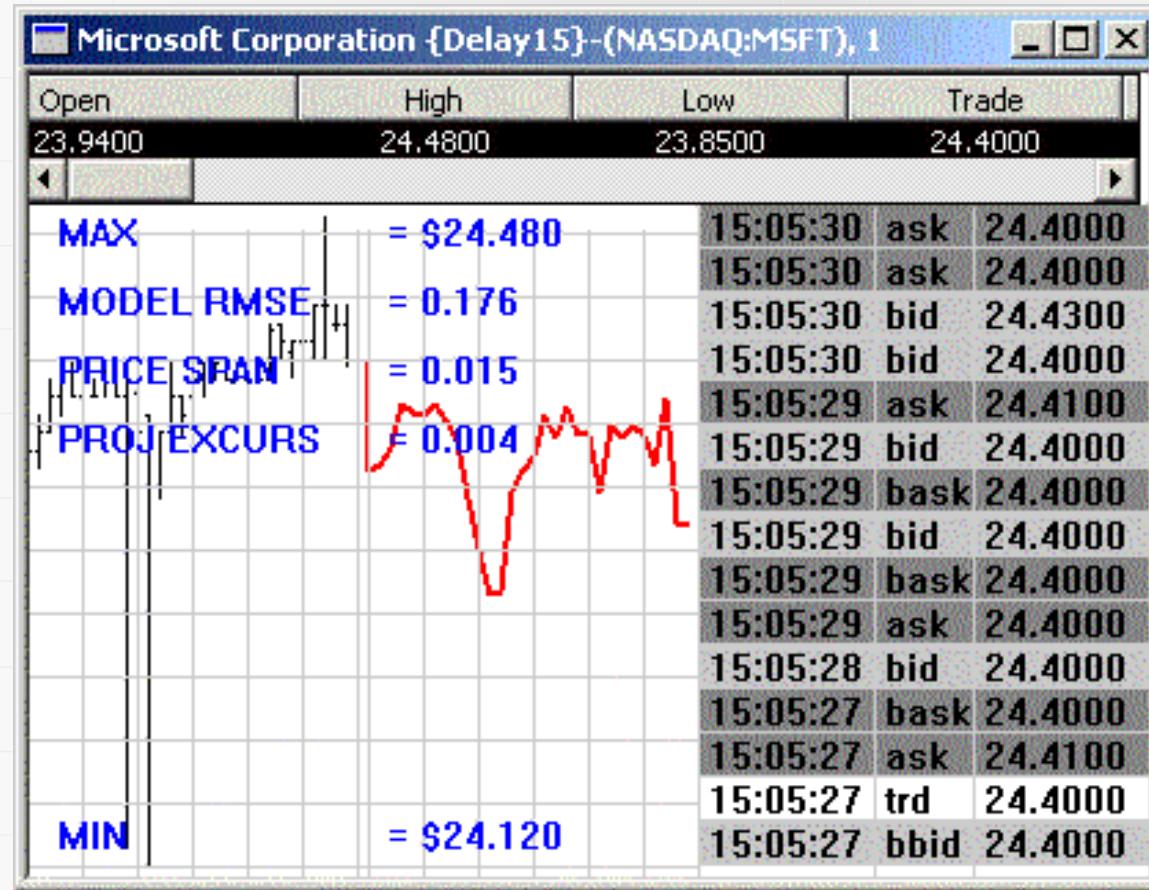


Typowe obrazy podawane jako wejścia

90% rozpoznawanie ustawienia twarzy i  
osoby 1-of-20



# Predykcja

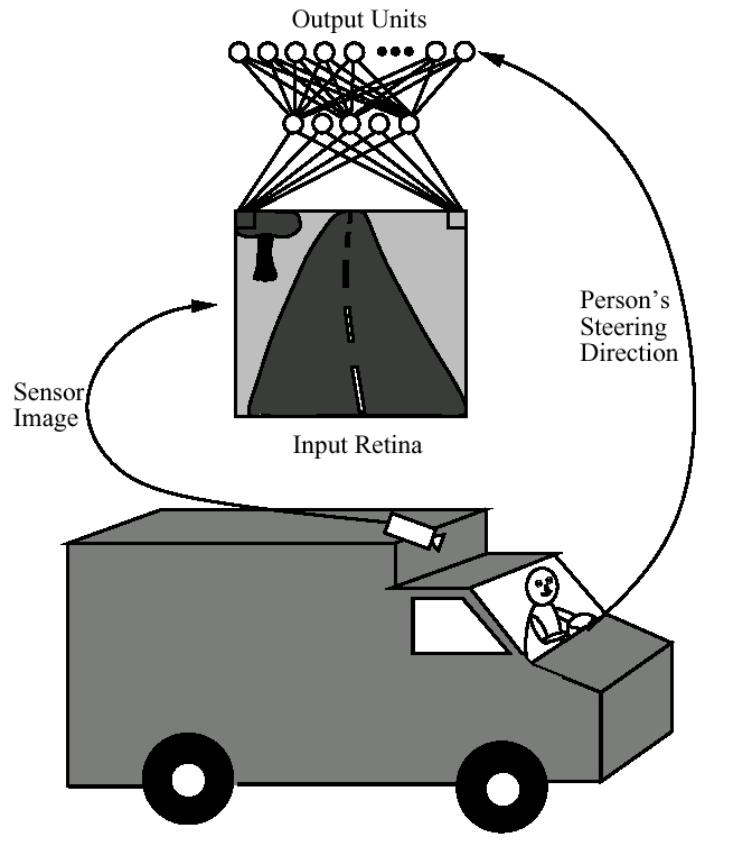


Predykcja cen produktów, pogody, sprzedaży



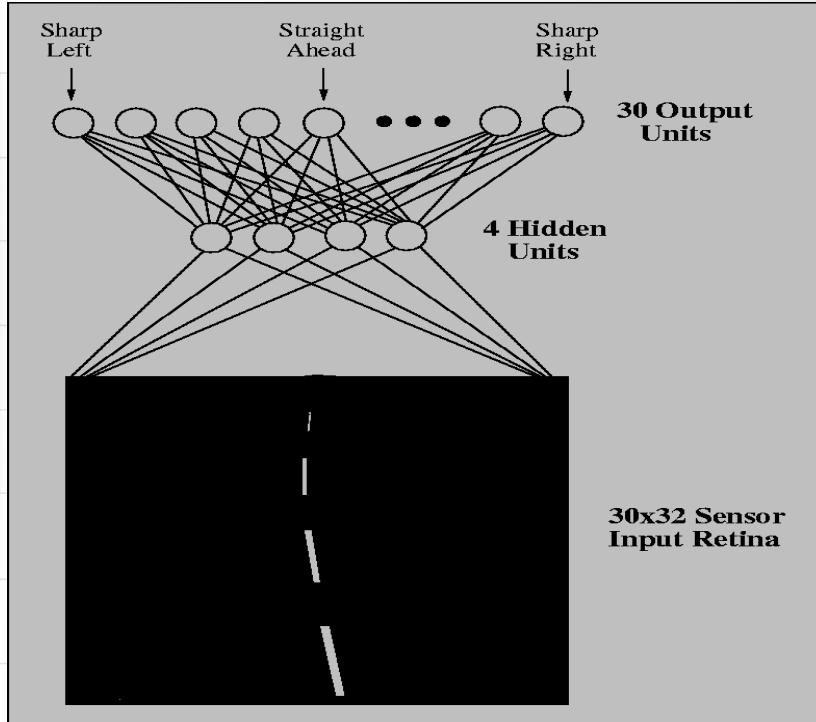
Wrocław  
University  
of Science  
and Technology

# ALVINN kieruje autem 70 mil/h na autostradzie





# ALVINN



- Sieć neuronowa jako sterownik jazdy samochodu
- Wejście składa się z obrazu 30x32 piksele z wideo kamery zainstalowanej na samochodzie oraz obraz 8x32 z odległościomierza też w skali szarości

\*Początek: praca doktorska DA Pomerleau **1990** rok



# Samochody są coraz lepiej wyposażone - DARPA rajd 2005

- **Grand Challenge 2005** - 175-mil przez pustynię Mojave Desert.
- **Stanford Racing Team** wygrał nagrodę 2 milion dolarów – czas: 6 godz., 53 min
- 5 załóg ukończyło rajd na pustyni (4 z czasem mniejszym niż 10 godz.)
- W listopadzie 2007 odbył się trzeci wyścig, którego trasa przebiegała przez ćwiczebne sztuczne tereny miejskie w Kalifornii.



# DARPA Grand Challenge 2005, cd.





# Selfdriving Cars- Sebastian Thrun



obciete.wmv



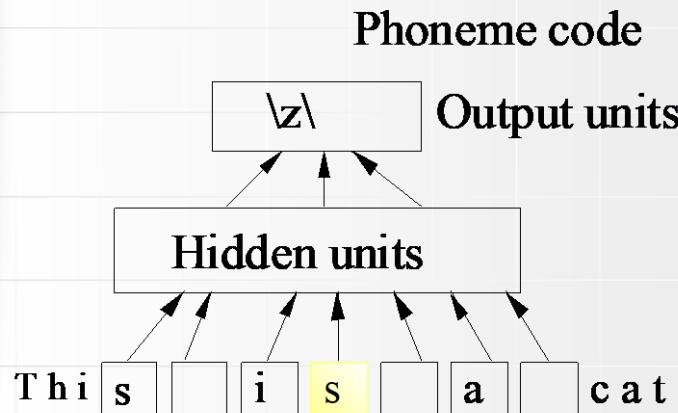
# Pierwszy komeryjny projekt - NetTalk

- NETtalk wytrenowana do wymawiania tekstu w języku angielskim
- Wejście składa się z 7 kolejnych znaków z napisanego tekstu. Są one pobierane z przesuwającego się okna.
- Pożądane wyjście jest kodem fonemu, który jest podawany do syntezatora dźwięku, który produkuje wymowę litery znajdującej się w centrum wejściowego okna.
- Architektura sieci ma 7x29 wejść, bo kodujemy 7 siedem znaków z 29 liter, 80 neuronów ukrytych i 26 wyjść kodujących fonemy.

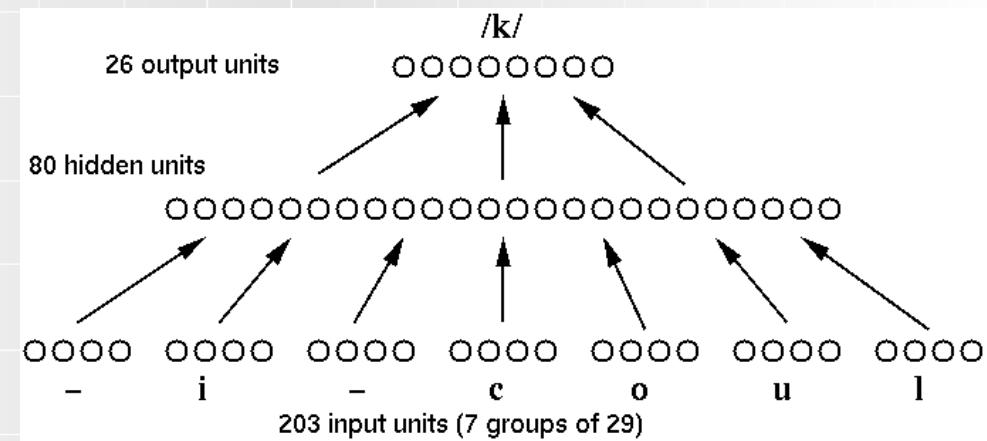


# NetTalk – idea

Konwersja tekstu w języku angielskim na mowę – trudny problem,  
różna wymowa w zależności od kontekstu



Ogólna idea przetwarzania





# NetTalk

<https://www.youtube.com/watch?v=gakJlr3GecE>

- Wybrano losowo 16,000 słów z 20,000 słów ze słownika
- Sieć dobrze generalizowała wymowę, słów, których nie była uczona
- Sieć odpowiada prawidłowo na 90% z 4000 pozostałych słów w słowniku
- Uczy się regularności
- Działanie na nowych słowach było porównywalne do systemu bazującego na dużej liczbie ręcznie określonych reguł.
- **Pierwsze zastosowanie komercyjne !!!**

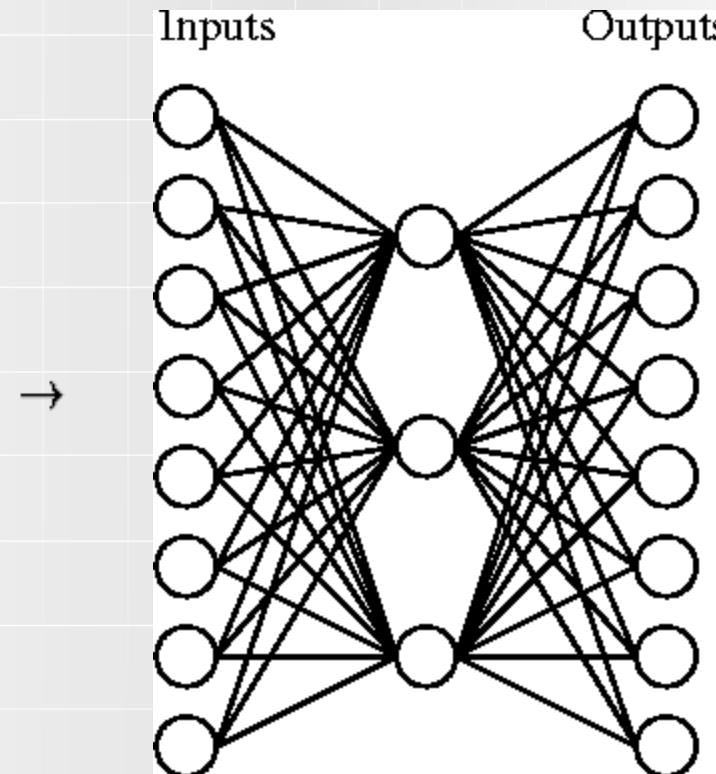
# Uczenie się ukrytej reprezentacji – autokoder (ang.encoder)

Pierwotnie, jak tutaj, do kompresji danych

A target function:

Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

Can this be learned??



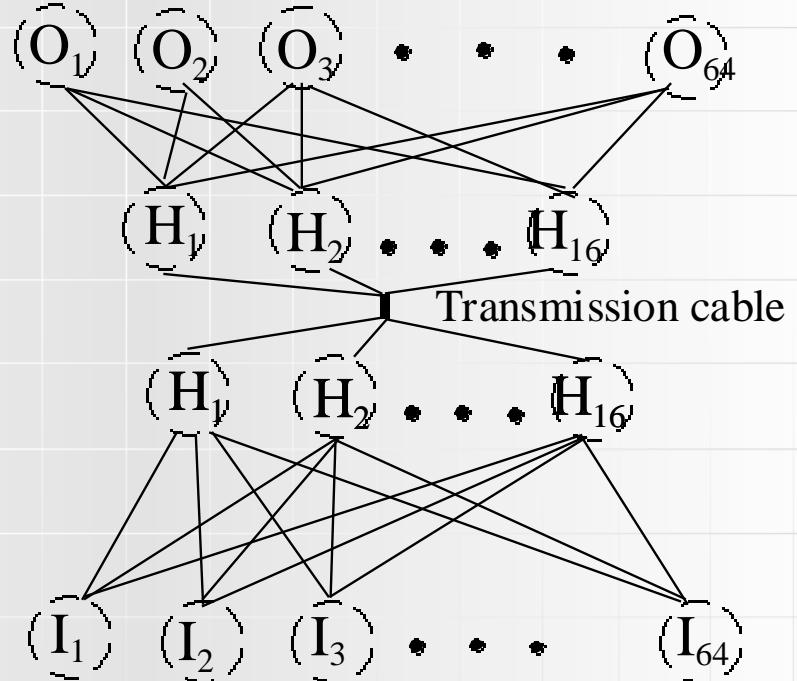
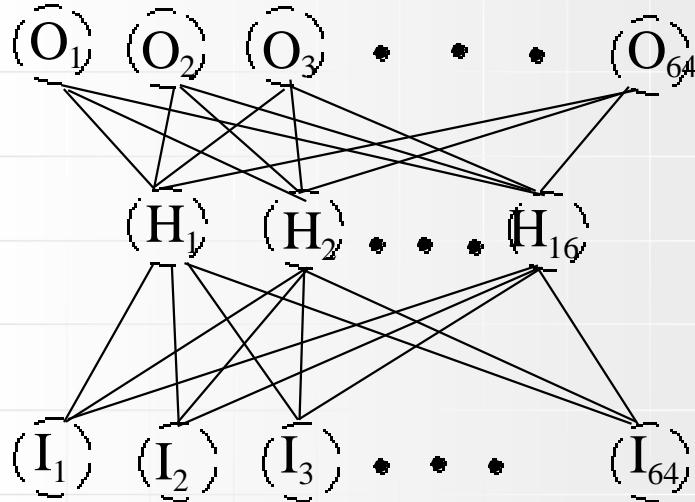


# Autokoder

- Problem zakodowania wejścia polega na znalezieniu zbioru wzorców aktywacji w neuronach ukrytych.
- W tym wydaniu liczba neuronów powinna być mała, aby uzyskać efektywny kod.
- Problem autoasocjacji.
- Idea autokodera intensywnie wykorzystywana w głębokich sieciach neuronowych, ale z większą liczbą neuronów w warstwie ukrytej.

# Autokoder – kompresja danych

Kompresja 4:1



# Współczesne zastosowanie pływtkich sieci

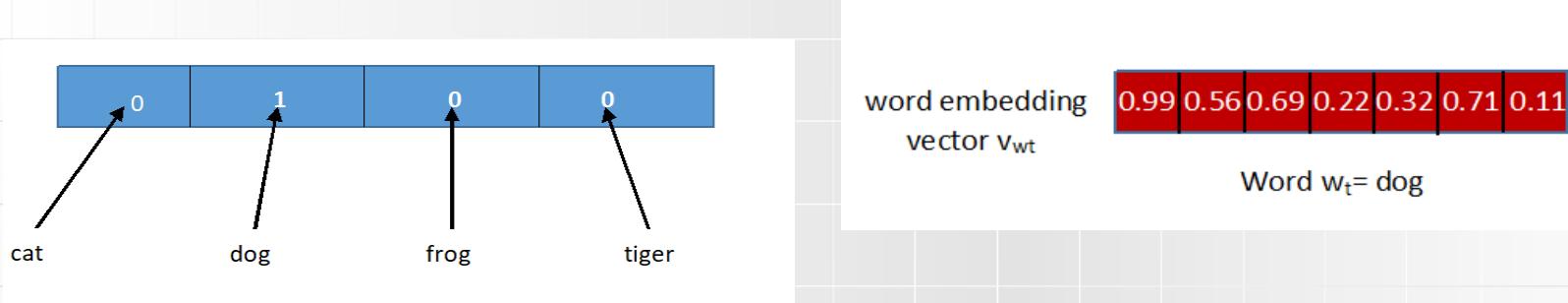
Word embeddings



Politechnika Wrocławska



# Word2vec



Tak było.

Pomyślmy iluwymiarowy musiałby być wektor dla słownika składającego się z 10000 słów

Tak jest teraz:

Wymiar wektora dla słowa jest zadany i równy liczbie neuronów warstwie ukrytej.

Wartości składowych są liczbami rzeczywistymi

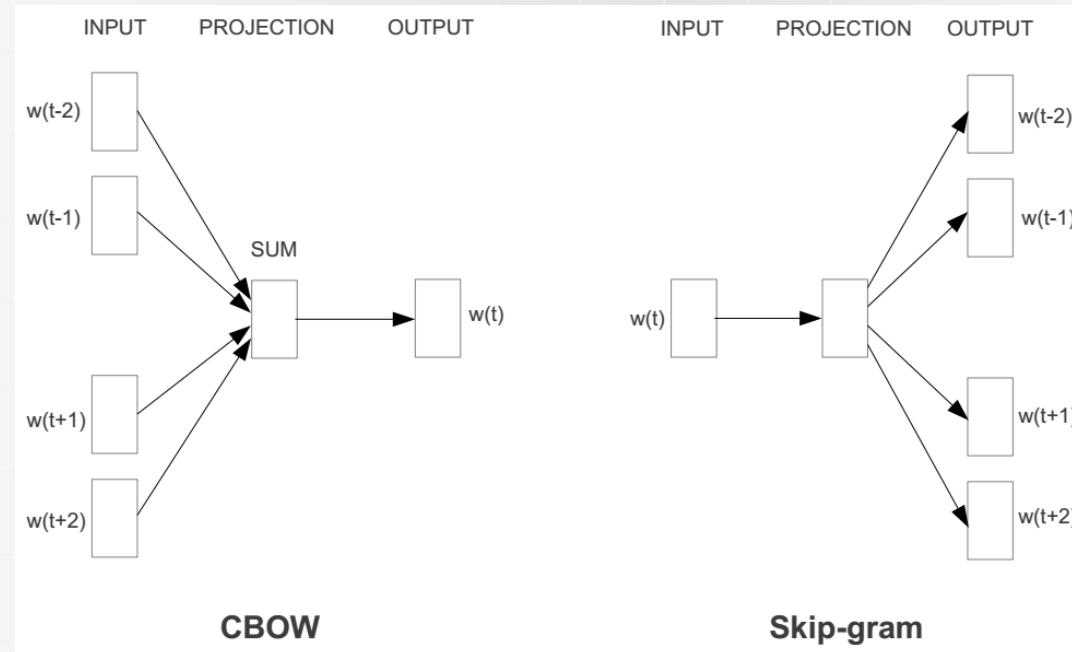


# Word2vec reprezentuje znaczenie słowa

- Reprezentuje każde słowo za pomocą wektora o niskiej wymiarowości
- Podobieństwo słów = podobieństwo wektorów
- Kluczowy pomysł: przewidywać otaczające słowa dla każdego słowa
- Szybsze i łatwo możemy dodać nowe zdanie/dokumenty lub słowo do słownika.

# Reprezentacja znaczenia słów – word2vec

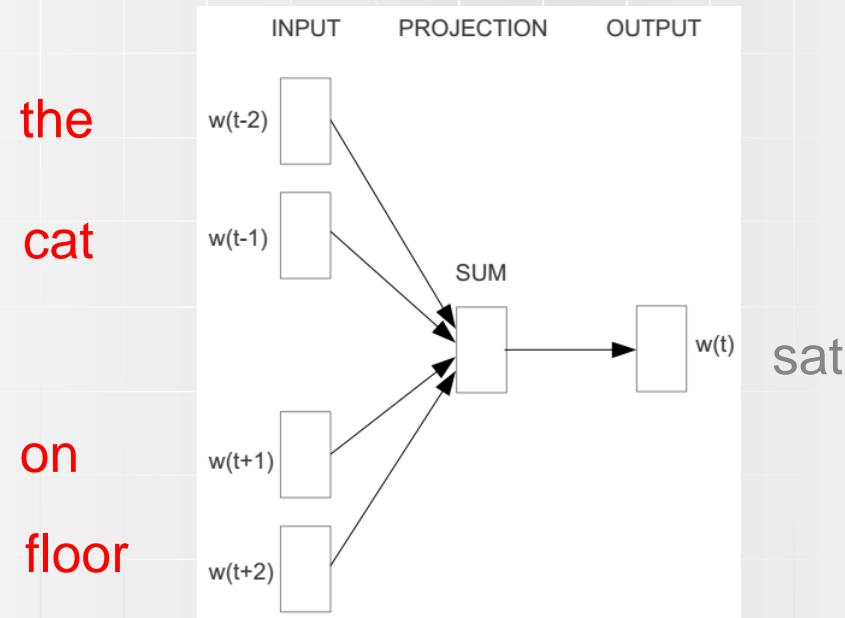
Dwa modele



- Obejmuje dwa modele:
  - Continuous Bag of Words (CBOW): używa worka słów, żeby przewidzieć środkowe słowo.
  - Skip-gram (SG): używa słowa, żeby przewidzieć otaczające dane słowo, słowa znajdujące się w oknie.

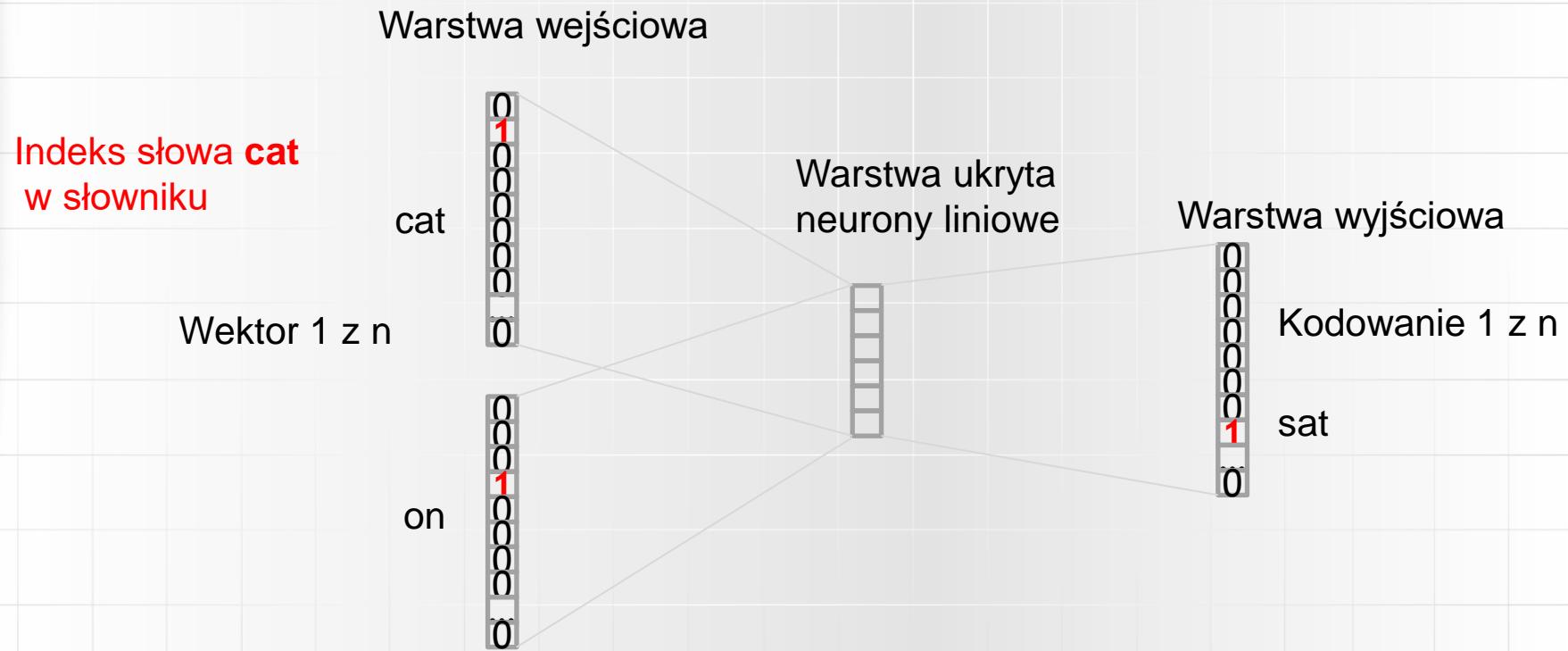
# Word2vec – Continuous Bag of Words (CBOW)

- Przykład: The cat sat on floor
  - Rozmiar okna = 2



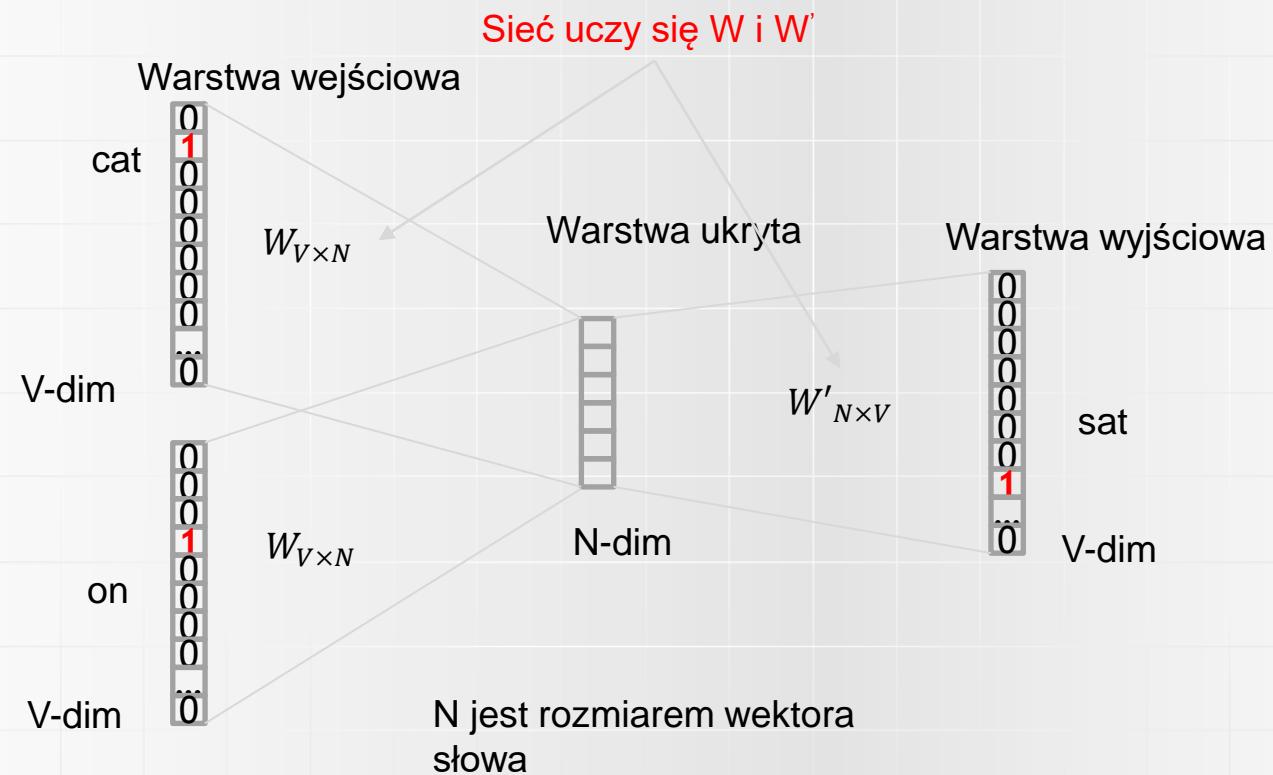


# CBOW

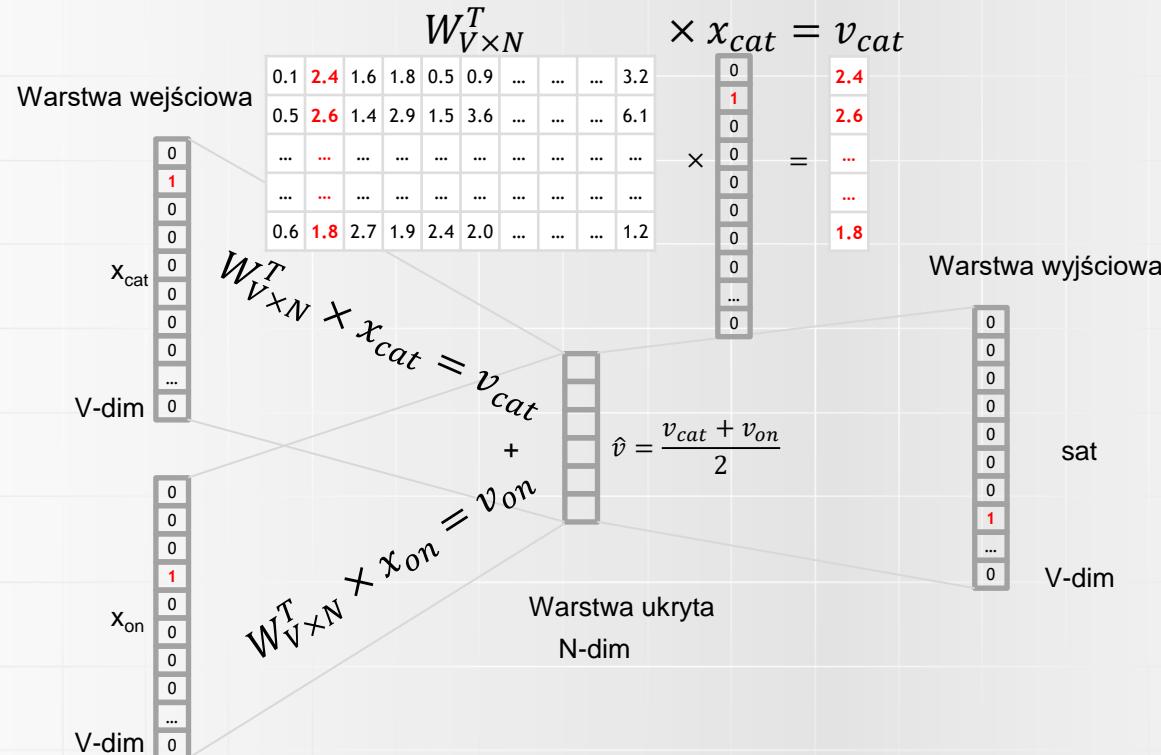




# CBOW - uczenie sieci



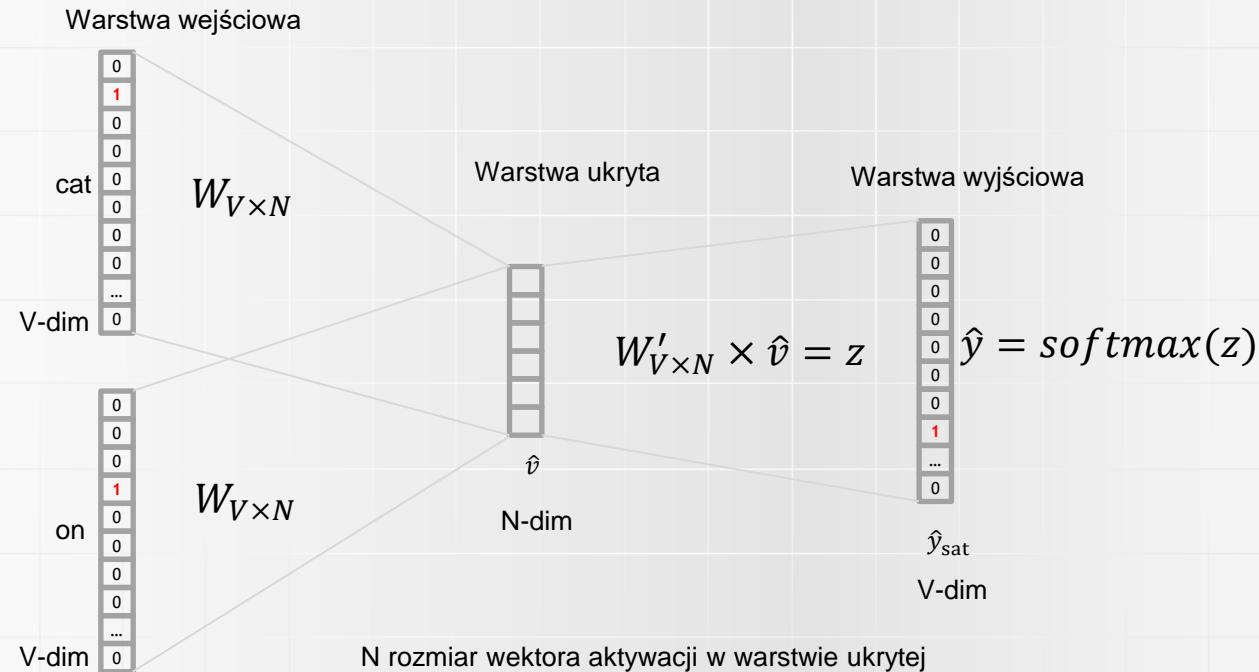
# CBOW - działanie



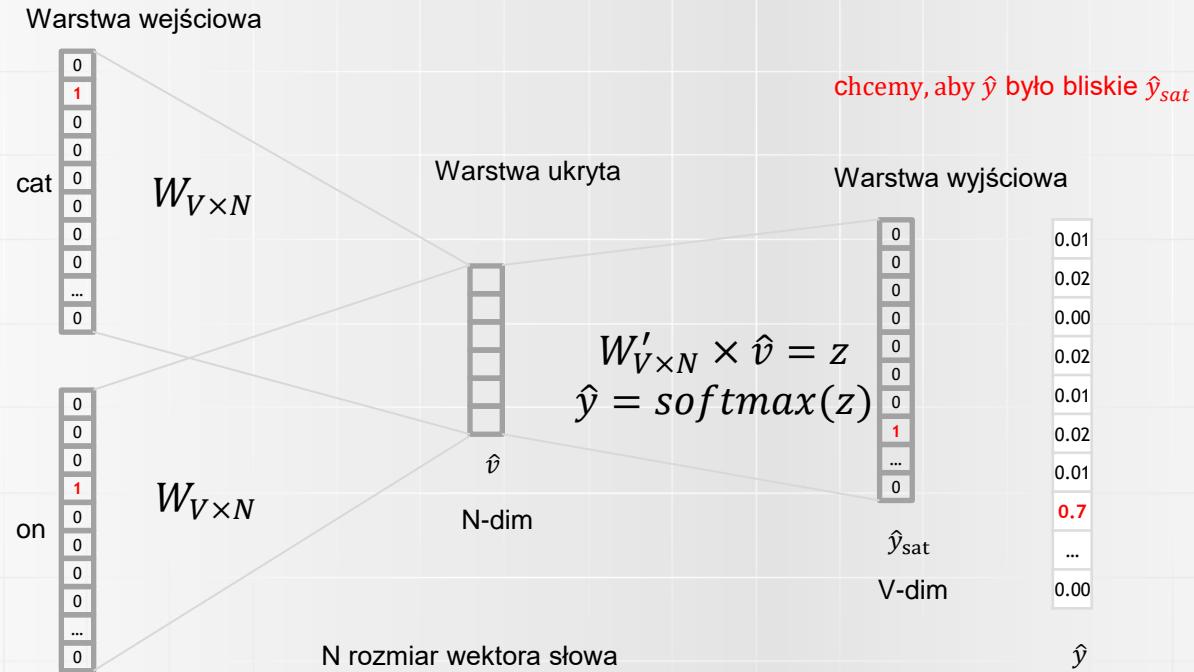
W macierzy  $W^T$ :

- wiersze odpowiadają indeksom słów w słowniku,
- Kolumny - neuronom w warstwie ukrytej

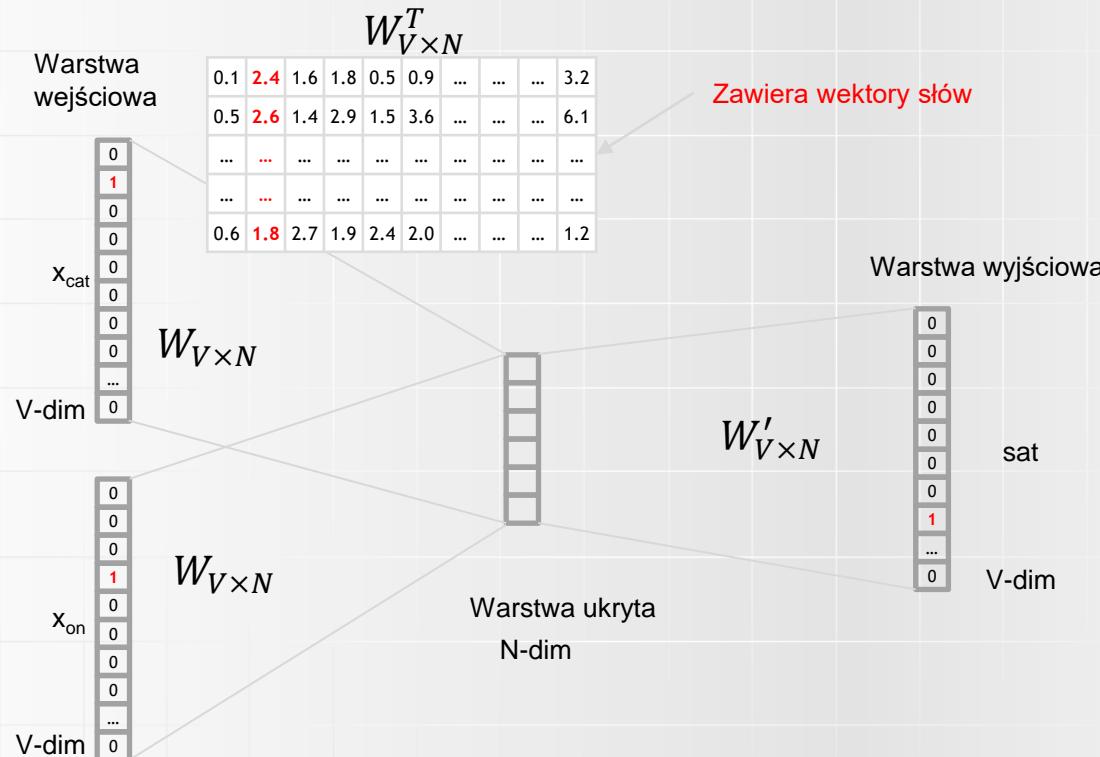
# CBOW – określanie wyjścia



# CBOW - działanie



# Co jest reprezentacją słowa (embedding)



Jako wektory słów możemy rozważyć wiersze  
 $W$  or  $W'$ , albo nawet wziąć średnią.

# Interesujące wyniki- analogie słów

## Word Analogies

Test for linear relationships, examined by Mikolov et al. (2014)

$$a:b :: c:?$$



$$d = \arg \max_x \frac{(w_b - w_a + w_c)^T w_x}{\|w_b - w_a + w_c\|}$$

man:woman :: king:?

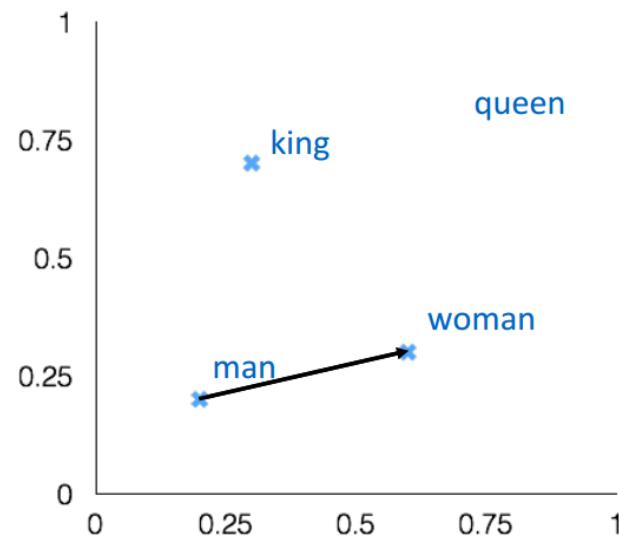
+ king [ 0.30 0.70 ]

- man [ 0.20 0.20 ]

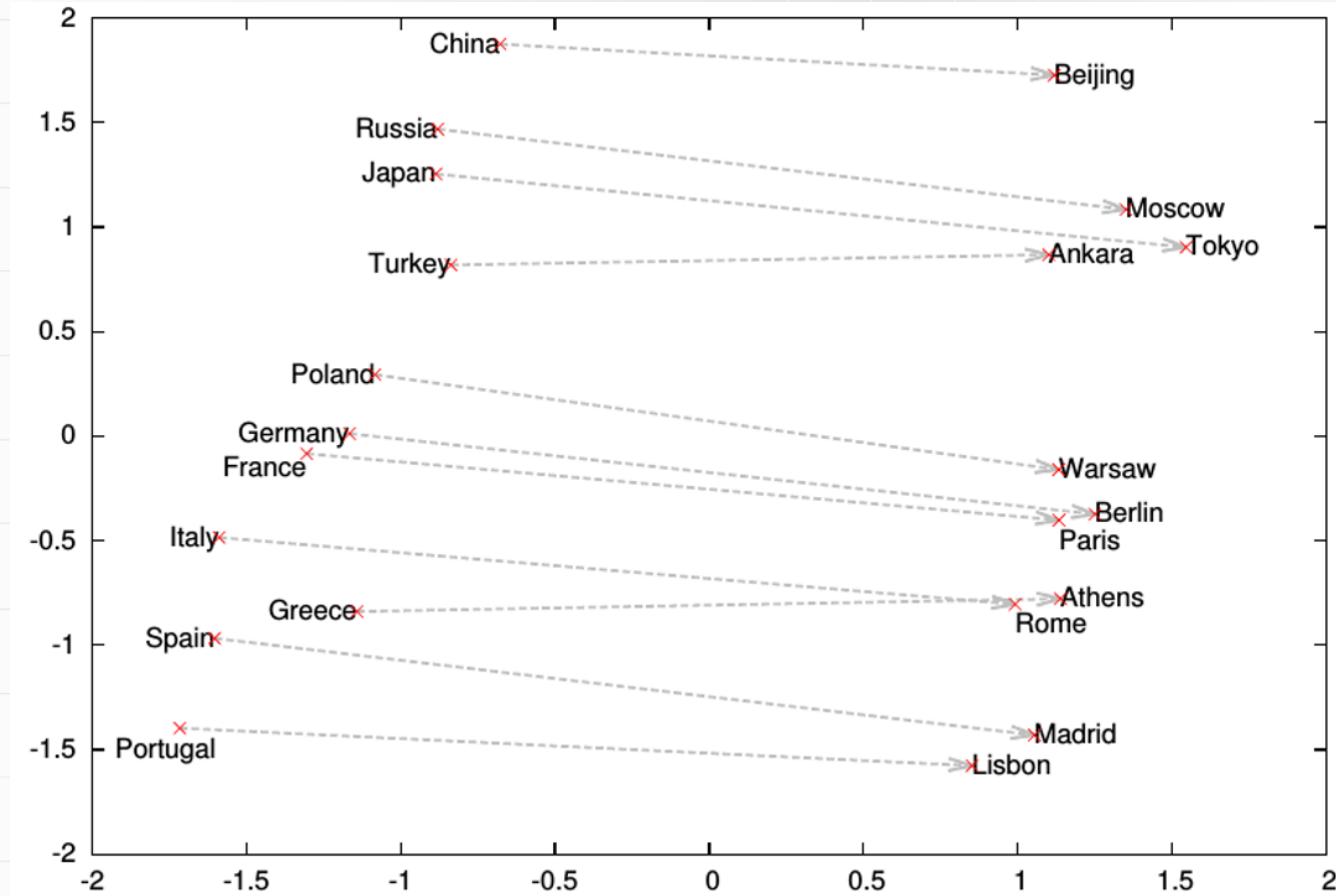
+ woman [ 0.60 0.30 ]

---

queen [ 0.70 0.80 ]



# Analogie słów

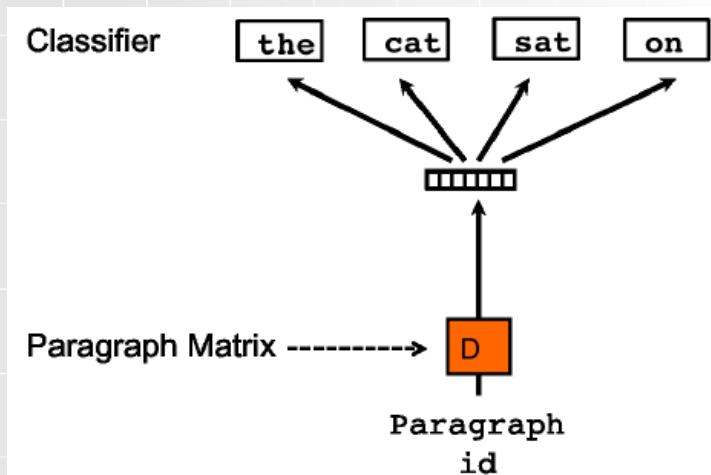
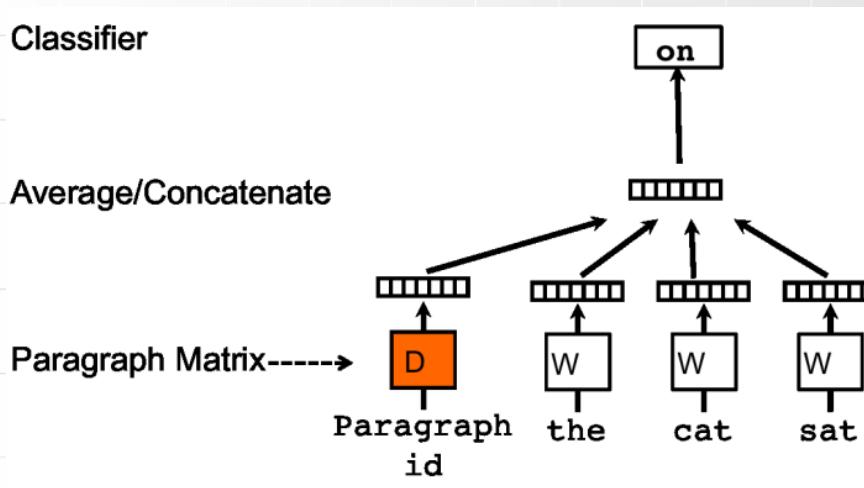


# Reprezentacja znaczenia zdania/tekstu

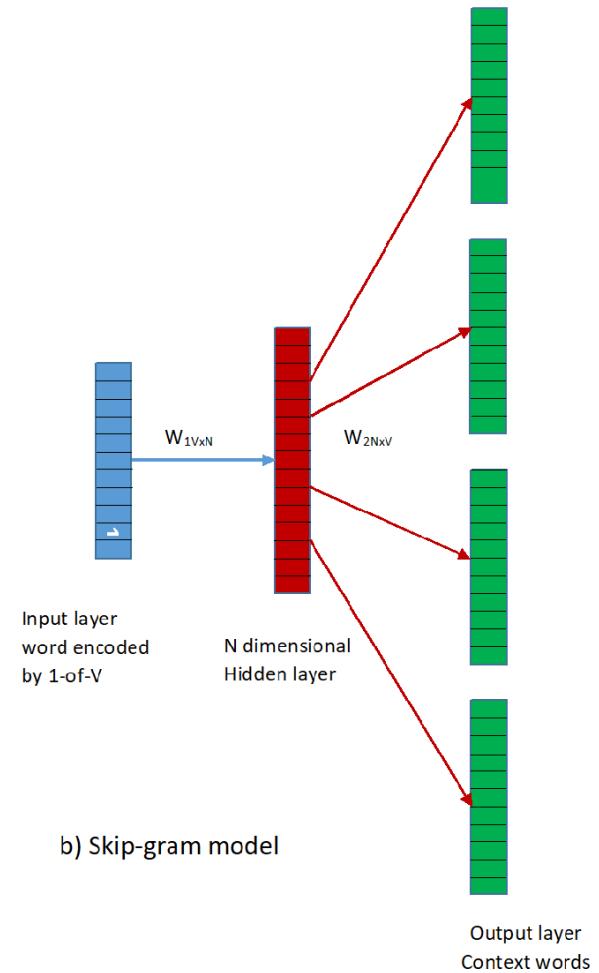
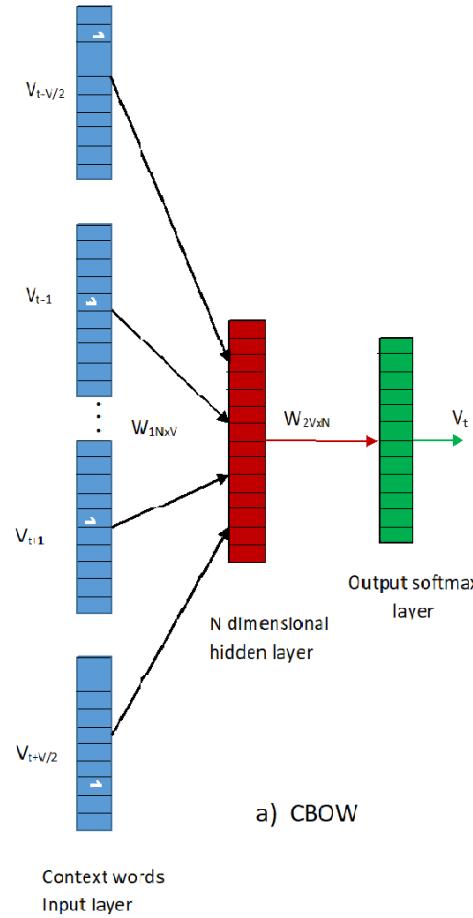
Wektor reprezentacji paragrafu tekstu(2014, Quoc Le, Mikolov)

Rozszerzenie word2vec do poziomu tekstu

Też istnieją dwa modele: dodajemy wektor paragrafu jako wejście



# CBOW a SKIP-GRAM





# Model Skip-gram

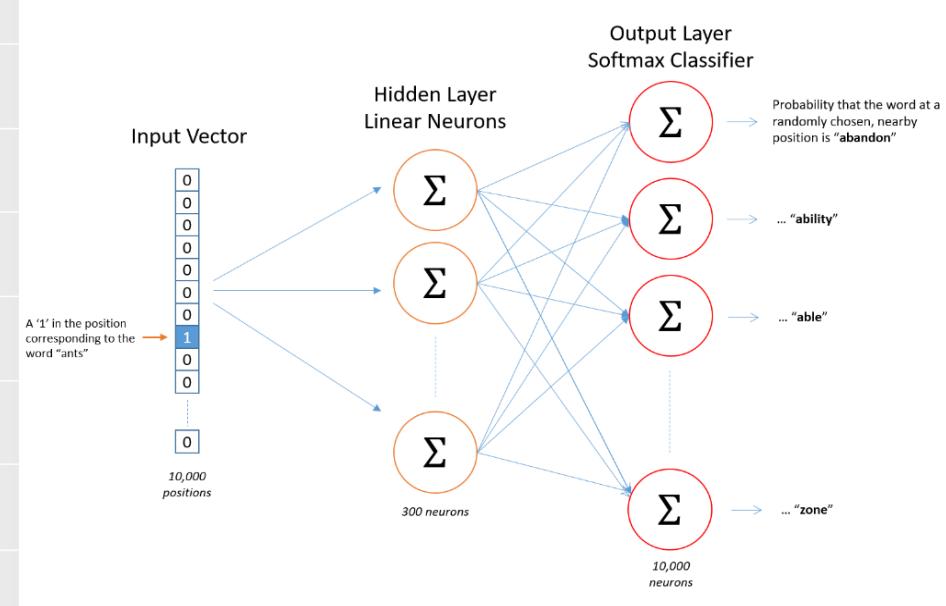
- Wyjściowe prawdopodobieństwa będą nam mówić jak bardzo prawdopodobne jest znalezienie każdego słowa ze słownika w pobliżu słowa wejściowego.
- Zakładamy rozmiar okna ze słowami, np. 10, które stanowią kontekst i sieć przewiduje prawdopodobieństwo każdego słowa ze słownika dla tego kontekstu
- W naszym przykładzie okno=2
- Tworzymy zbiór dla zdania: **The quick brown fox jumps over the lazy dog**

Source Text	Training Samples
The <b>quick</b> brown fox jumps over the lazy dog. ➔	(the, quick) (the, brown)
The <b>quick</b> brown <b>fox</b> jumps over the lazy dog. ➔	(quick, the) (quick, brown) (quick, fox)
The <b>quick</b> brown <b>fox</b> jumps <b>over</b> the lazy dog. ➔	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The <b>quick</b> brown <b>fox</b> jumps <b>over</b> the lazy dog. ➔	(fox, quick) (fox, brown) (fox, jumps) (fox, over)



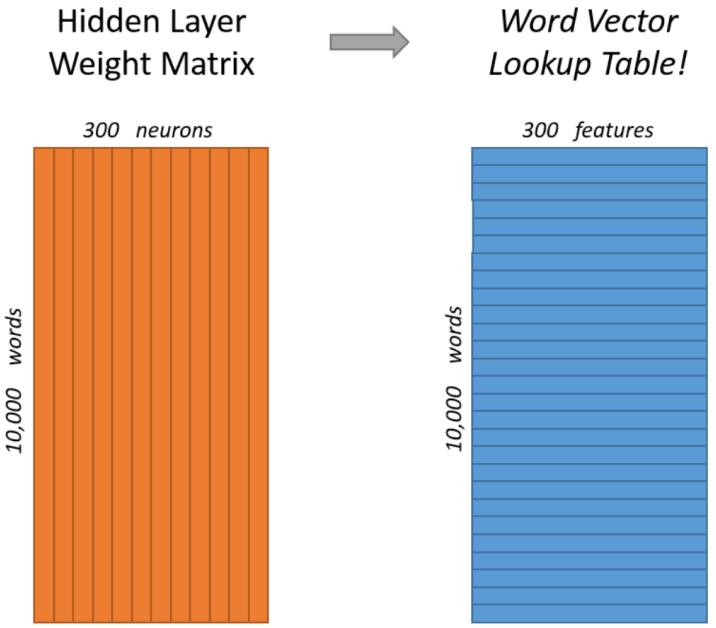
# Architektura sieci

- Funkcja aktywacji w warstwie ukrytej jest liniowa, neurony w warstwie wyjściowej używają softmax.
- Trenowanie odbywa się na parach wektorów,
  - wejściem jest wektor 1 z n reprezentujący słowo wejściowe
  - wyjściem jest słowo z pary także kodowane jako 1 z n.
- Po wyuczeniu, po podaniu wejściowego słowa sieć, na wyjściu określa rozkład prawdopodobieństwa słów ze słownika jako kontekst dla słowa wejściowego.





# Działanie sieci na przykładzie



$$\begin{bmatrix} 0 & 0 & 0 & \textcolor{green}{1} & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \textcolor{green}{10} & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

# Przykłady zachowanych relacji słów

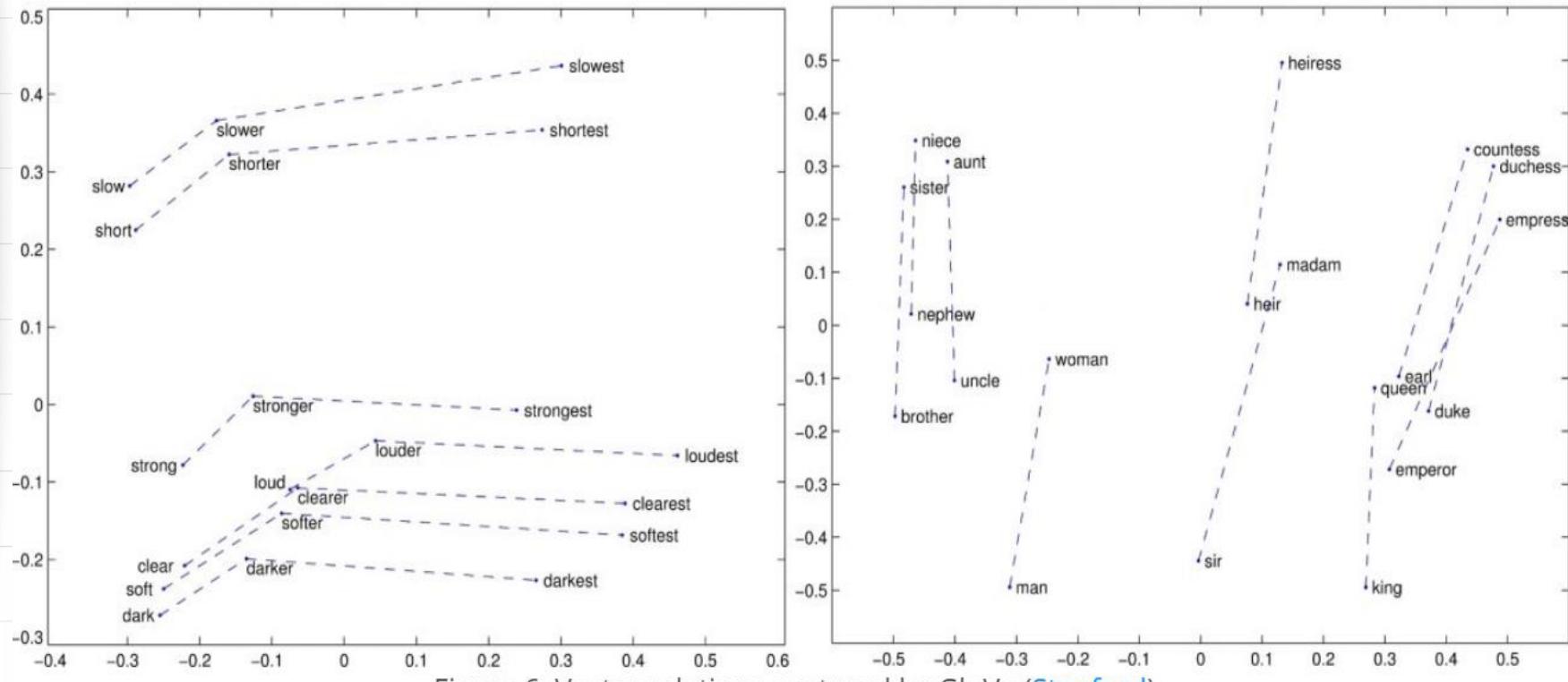
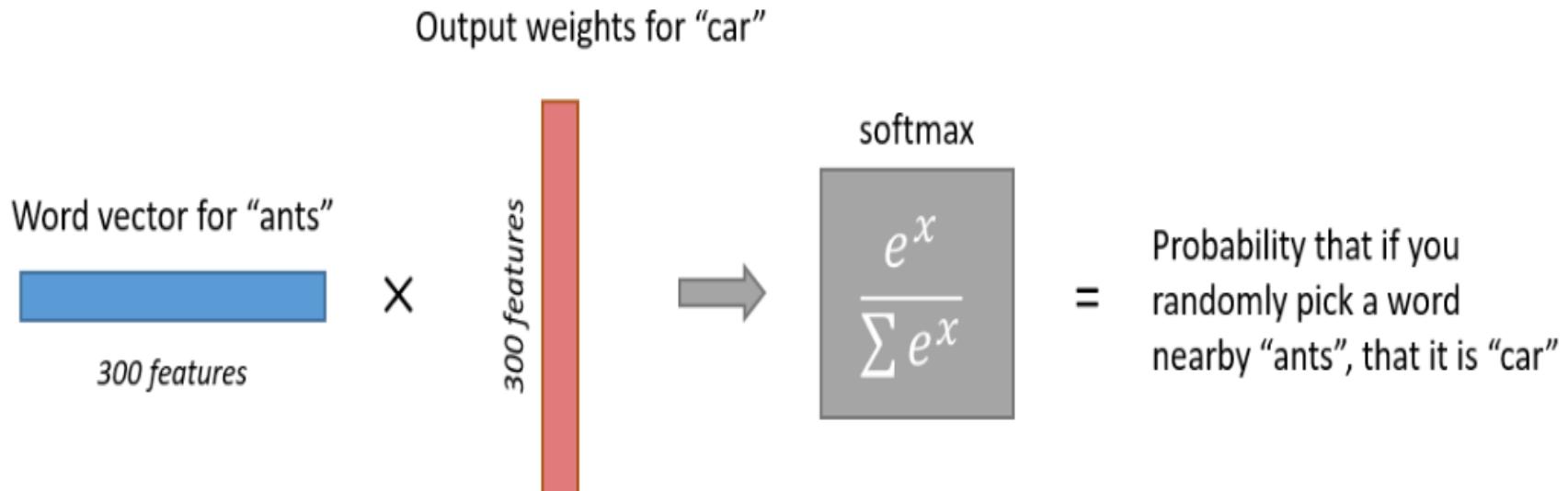


Figure 6: Vector relations captured by GloVe ([Stanford](#))



# Działanie warstwy wyjściowej





# Problemy

- Jeśli dwa różne słowa mają bardzo podobny kontekst (czyli słowa jakie się pojawiają wokół nich), wówczas model powinien produkować bardzo podobne wyniki dla tych słów.
- Jeśli dwa słowa mają bardzo podobny kontekst, wówczas sieć powinna się uczyć dla nich podobnych wektorów słów, np. gazeta i czasopismo będą miały podobne konteksty.
- Taka sieć dla 300 neuronów ukrytych i 10000 słów ma 3 miliony wag.
- **Wady:**
  - Obliczanie gradientu jest bardzo wolne.
  - Potrzebujemy olbrzymiej liczby danych trenujących



# Rozwiązańe

- Traktowanie wspólnych **par słów lub fraz** jako pojedynczych słów w ich modelu.
- **Próbkowanie częstych** słów, aby zmniejszyć liczbę wzorców trenujących.
- Modyfikacja celu optymalizacji za pomocą techniki nazywanej ***Negative Sampling***, która powoduje uaktualniane niewielkiej liczby wag.
- Próbkowanie częstych słów i zastosowanie ***Negative sampling***, poprawia jakość otrzymywanych wektorów słów



# Próbkowanie

- Każde przejście w trakcie uczenia poszukuje kombinacji 2 słów, ale możemy je wykonać wiele razy i otrzymywać dłuższe frazy. Na przykład pierwsze przejście będzie używało New York a potem możemy wybrać New York City jako kombinację New York i City.
- Narzędzie **oblicza ile razy każda kombinacja dwóch słów** pojawia się w tekście użytym do trenowania a następnie ta wykorzystywana jest do określenia jaką kombinację słów użyć we frazie. Zapobiega to także tworzeniu fraz z takich słów jak „and the” i „this is”.
- Dwa główne problemy ze słowem **the** to:
  - niewiele nam ono mówi o znaczeniu słowa fox
  - Mamy zbyt dużo par z *the*
- **Rozwiążanie:**
  - *Próbkowanie* - Dla każdego słowa w tekście jest szansa, że możemy je z tekstu usunąć. Przy oknie 10 mamy 10 razy mniej wzorców trenujących.



- Word2vec oblicza prawdopodobieństwo, z którym dane słowo jest zatrzymywane w słowniku.
- Niech  $w_i$  jest słowem,  $z(w_i)$  opisuje jaką część wszystkich słów w korpusie stanowi dane słowo. Na przykład słowo “peanut” pojawiło się 1,000 razy w milionie słów, wówczas  $z('peanut') = 1E-6$ .
- Istnieje także parametr, nazwany *sample*, który określa jak wiele razy wykonujemy próbkowanie (subsampling). Domyślna wartość to 0.001. Im mniejsza wartość ‘sample’, tym mniej jest prawdopodobne utrzymywanie średnich słów.
- $P(w_i)$  to prawdopodobieństwo utrzymania słowa w zbiorze uczącym:

$$P(w_i) = \left( \sqrt{\frac{z(w_i)}{0.001}} + 1 \right) \cdot \frac{0.001}{z(w_i)}$$



# Negatywne próbkowanie (ang.negative sampling)

Trenowanie sieci oznacza dopasowywanie wag na podstawie wzorców trenujących rozmiar naszego słownika powoduje, że liczba wag jest ogromna (biliony !!).

- **Negative sampling** powoduje, że będziemy modyfikować **niewielki procent** neuronów w porównaniu do wszystkich.
- Jak to działa?
- Przypomnijmy, że kiedy uczymy sieć na parach słów(fox, quick) właściwa odpowiedź sieci jest kodowana jako 1 z n, czyli wyjście odpowiadające z quick ma wyjście 1 a pozostałe neurony (tysiące neuronów) powinny mieć wyjście zero.
- Z **negatywnym próbkowaniem losowo wybieramy małą liczbę negatywnych** słów np. 5 i uaktualniamy wagi dla nich (czyli **negatywne słowa to takie, dla których sieć powinna odpowiedzieć 0**).
- Uaktualniamy wówczas wagi dla pozytywnego słowa (takiego które znajduje się w kontekście – w naszym przypadku słowa quick) i dla małego zbioru słów negatywnych. Zazwyczaj wybiera się ok. 15-20 słów negatywnych dla małych zbiorów danych.
- Przypomnijmy, że warstwa wyjściowa modelu miała  $300 \times 10000$  wag.
- Uaktualniając wagi dla pozytywnego przykładu (quick) plus wagi dla 5 innych słów, gdzie wyjście powinno być ustawione na 0 mamy tylko do uaktualnienia 1600 wag, czyli 0.06 wszystkich wag w warstwie wyjściowej.



# Podsumowanie

- Word2vec jest powszechnie stosowaną techniką w przetwarzaniu języka naturalnego (MLP)
- Tworzy reprezentację wektorową (w postaci wektora liczb rzeczywistych ) ang. **word embedding**
- Istnieją gotowe implementacje metody.



# Źródła do NLP

<http://web.stanford.edu/class/cs224n/>

“word2vec Parameter Learning Explained”, Xin Rong

<https://ronxin.github.io/wevi/>

Word2Vec Tutorial - The Skip-Gram Model

<http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>



Wrocław  
University  
of Science  
and Technology



Zintegrowany  
Program Rozwoju  
Politechniki Wrocławskiej

# SIECI NEURONOWE



HR EXCELLENCE IN RESEARCH



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Politechnika Wrocławskiego

Unia Europejska  
Europejski Fundusz Społeczny





# Sieci neuronowe

## Sieci konwolucyjne

**URSZULA MARKOWSKA-KACZMAR**

Katedra Inteligencji Obliczeniowej

Wydział Informatyki i Zarządzania



# Chcemy rozpoznawać obrazy

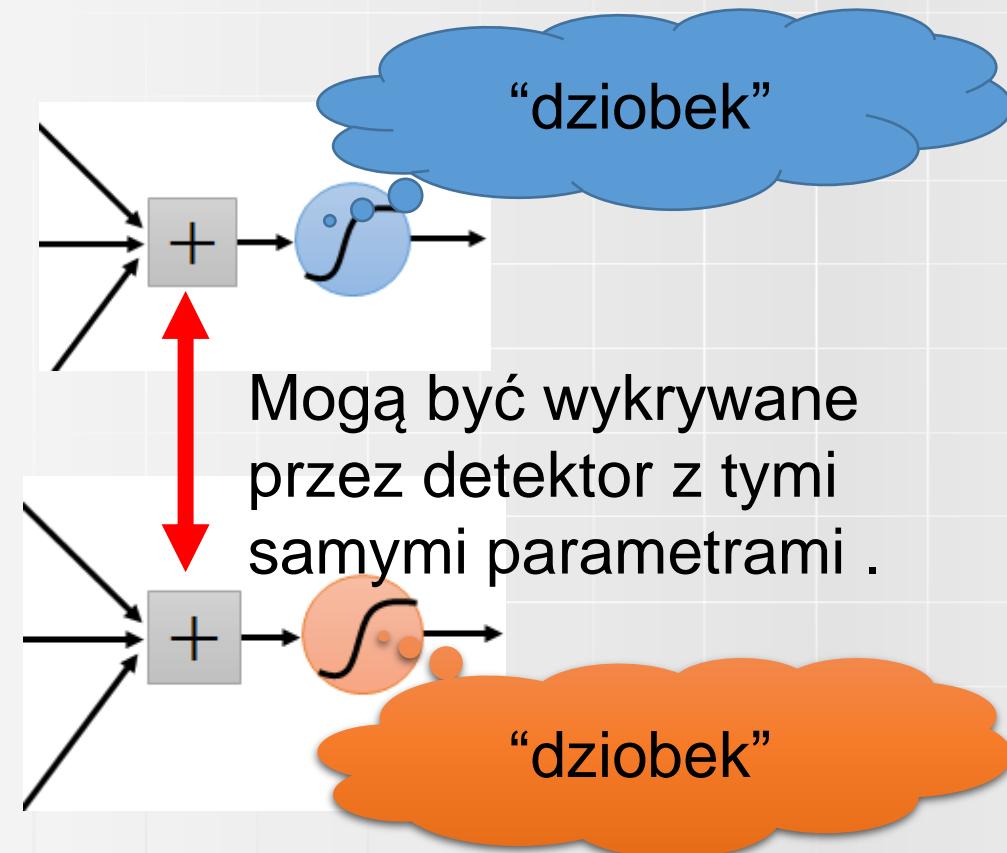
Małe obszary z niewielką liczbą parametrów mogą być pożyteczne



- Taki detektor może być istotny w przypadku rozpoznawania ptaków

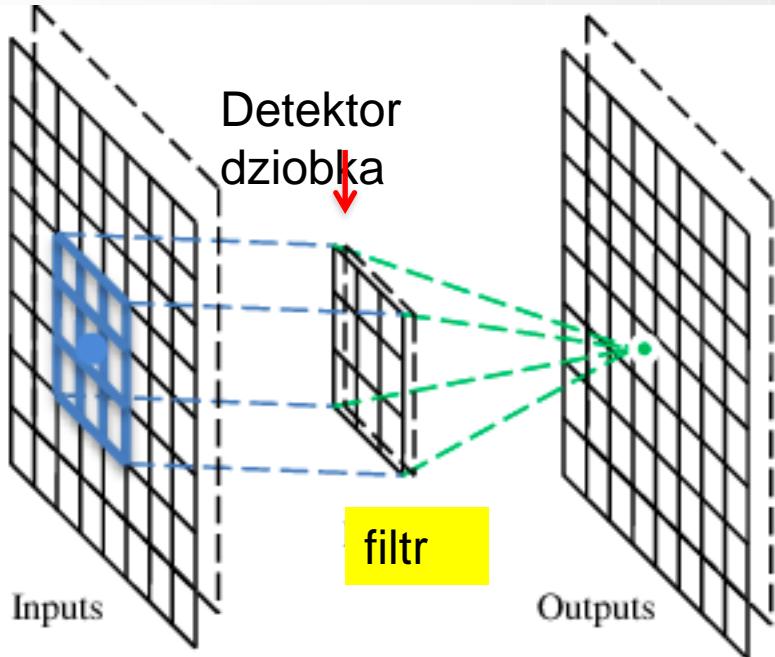


Ta sama cecha może się pojawiać w różnym miejscu na obrazie





# Sieci konwolucyjne CNN



- Składają się z warstw konwolucyjnych
- (i innych warstw)
- Każda warstwa konwolucyjna ma pewną liczbę filtrów (detektorów)



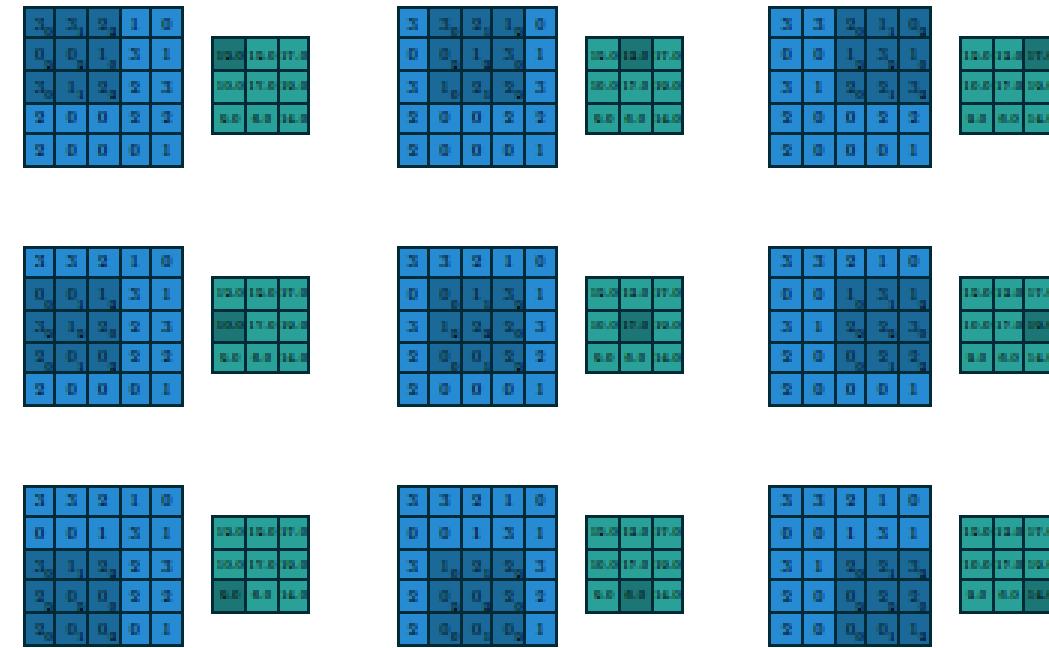
# Podstawowa operacja – konwolucja

## Inaczej operacja splotu

- Wprowadza się pojęcia **filtra**, inaczej zwanego jądrem (tutaj: o wymiarach 3x3 piksele)

0	1	2
2	3	0
0	1	2

- Jądro przechodzi po obrazie (bierzemy pod uwagę jaskrawość pikseli) obliczając wartość wyjściową używając w tym celu operacji konwolucji
- Krok** o który przesuwa się filtr nazywa się w j. ang. stride – hiperparametr
- Jeśli filtr nie mieści się na obrazie całkowitą liczbę razy, dodajemy **ramkę** (ang. padding) samych 0 – liczba dodanych wierszy i kolumn hiperparametr

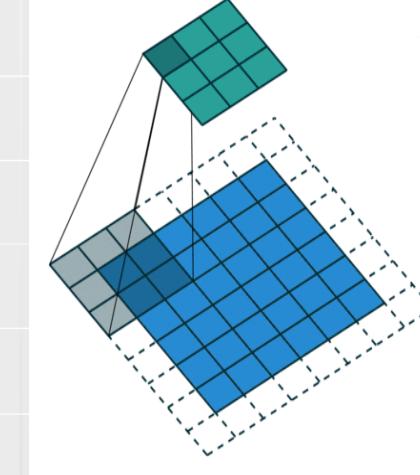
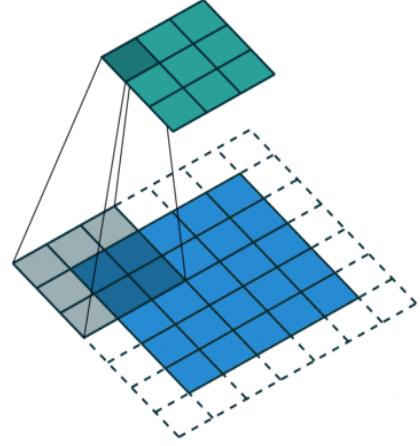
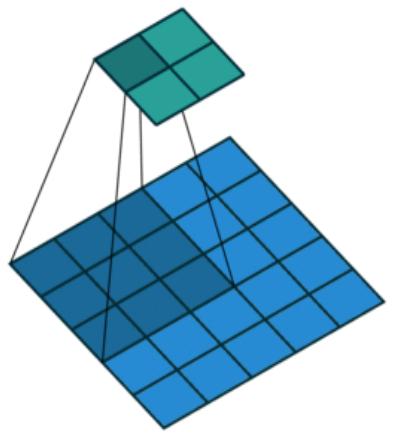
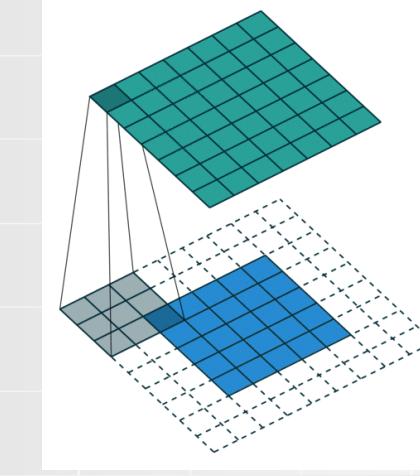
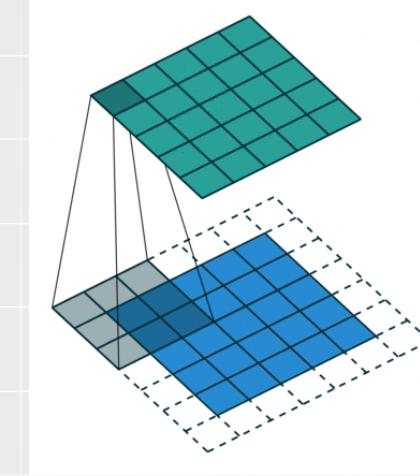
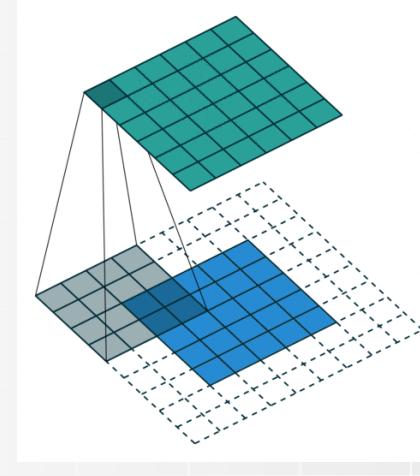
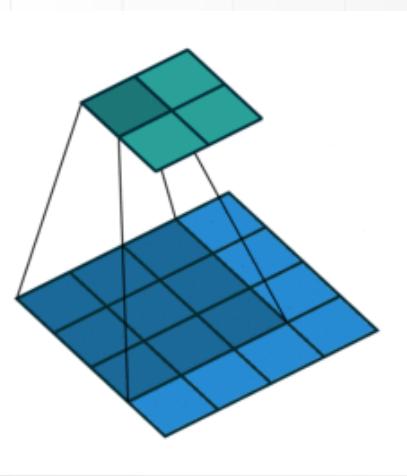


Dyskretna operacja konwolucji



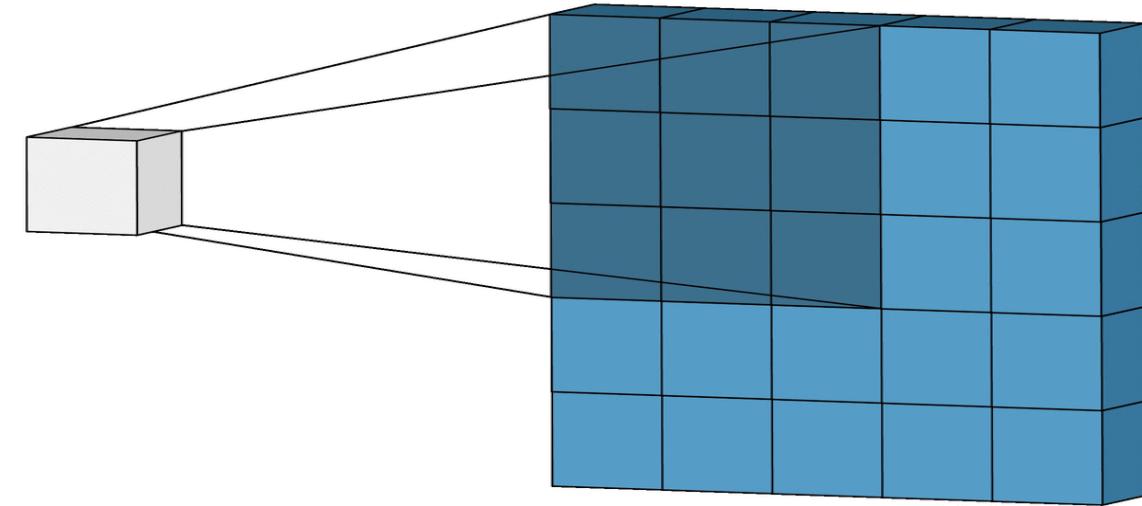
# Operacja konwolucji przy różnym kroku i ramce

[https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)





# Konwolucja 2D intuicyjnie



0	1	2
2	2	0
0	1	2

Kernel =  
Filter in 2D

3 <sub>0</sub>	3 <sub>1</sub>	2 <sub>2</sub>	1	0
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2	0	0	2	2
2	0	0	0	1

Image

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

output



# Konwolucja stary pomysł

## Wykrywanie krawędzi

Wejście



-1	-1	-1
-1	8	-1
-1	-1	-1

Jądro= filtr  
dla obrazów w szarości.  
Wagi w filtrze z góry  
zadane.

wyjście



- jądro (ang. kernel) = mała macierz wag
- Takie jądro przesuwa się po obrazie
  - Mnożenie odpowiednich składowych i
  - Sumowanie daje wyjście dla pojedynczego piksela.



# Konwolucja formalnie

$I_{11}$	$I_{12}$	$I_{13}$	$I_{14}$	$I_{15}$	$I_{16}$	$I_{17}$	$I_{18}$	$I_{19}$
$I_{21}$	$I_{22}$	$I_{23}$	$I_{24}$	$I_{25}$	$I_{26}$	$I_{27}$	$I_{28}$	$I_{29}$
$I_{31}$	$I_{32}$	$I_{33}$	$I_{34}$	$I_{35}$	$I_{36}$	$I_{37}$	$I_{38}$	$I_{39}$
$I_{41}$	$I_{42}$	$I_{43}$	$I_{44}$	$I_{45}$	$I_{46}$	$I_{47}$	$I_{48}$	$I_{49}$
$I_{51}$	$I_{52}$	$I_{53}$	$I_{54}$	$I_{55}$	$I_{56}$	$I_{57}$	$I_{58}$	$I_{59}$
$I_{61}$	$I_{62}$	$I_{63}$	$I_{64}$	$I_{65}$	$I_{66}$	$I_{67}$	$I_{68}$	$I_{69}$

$K_{11}$	$K_{12}$	$K_{13}$
$K_{21}$	$K_{22}$	$K_{23}$

- Operacja konwolucji przesuwa jądro K przez obraz I, rozpoczynając od lewego górnego wierzchołka i przesuwając dalej dopóki jądro całkowicie mieści się w obrazie
- Każda pozycja w jądrze odpowiada jednemu pikselowi. Wartość wyjściowa dla piksela  $I_{57}$  jest obliczana następująco:

$$O_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} + I_{69}K_{23}$$

- Przyjmijmy, że obraz ma  $M$  wierszy i  $N$  kolumn, a jądro ma  $m$  wierszy i  $n$  kolumn, wówczas wyjście po przetworzeniu obrazu ma rozmiar  $(M - m + 1)$  wierszy i  $(N - n + 1)$  Kolumn.
- Matematycznie możemy opisać operację konwolucji jako:

$$O(i, j) = \sum_{k=1}^m \sum_{l=1}^n I(i+k-1, j+l-1)K(k, l)$$

- Gdzie  $i$  zmienia się od 1 do  $(M - m + 1)$  a  $j$  od 1 do  $(N - n + 1)$ .



# Korelacja wzajemna a konwolucja

- Mając dany obraz  $I$  oraz filtr  $K$  o wymiarach  $k_1 \times k_2$  operacja korelacji wzajemnej opisana jest wzorem:

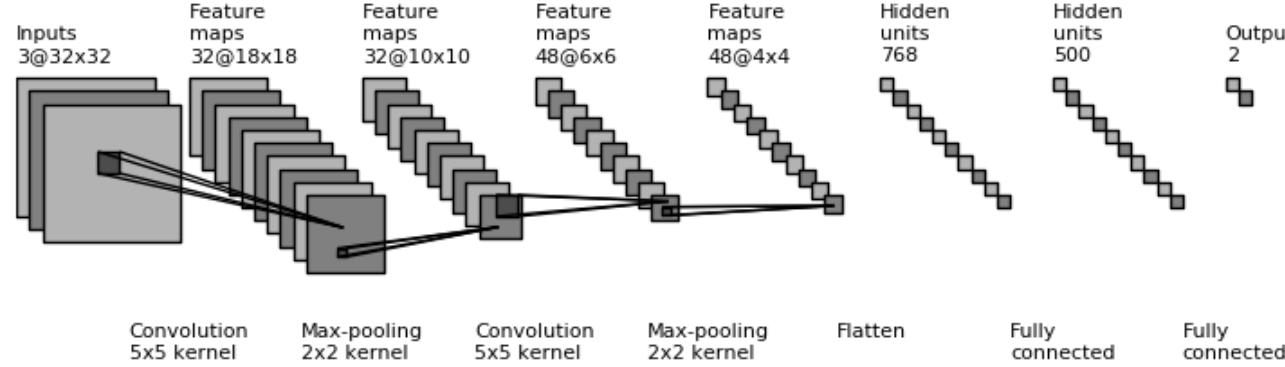
$$(I \otimes K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n)K(m, n)$$

- Mając dany obraz  $I$  oraz filtr  $K$  o wymiarach  $k_1 \times k_2$  operacja konwolucji opisana jest wzorem:

$$\begin{aligned}(I \star K)_{ij} &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i-m, j-n)K(m, n) \\ &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n)K(-m, -n)\end{aligned}$$

- Łatwo zauważyć, że konwolucja jest tym samym co korelacja wzajemna z obróconym o 180 stopni filtrem

# Sieć konwolucyjna

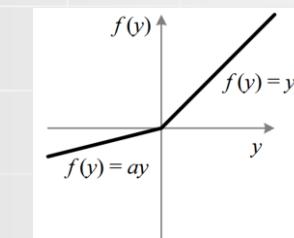
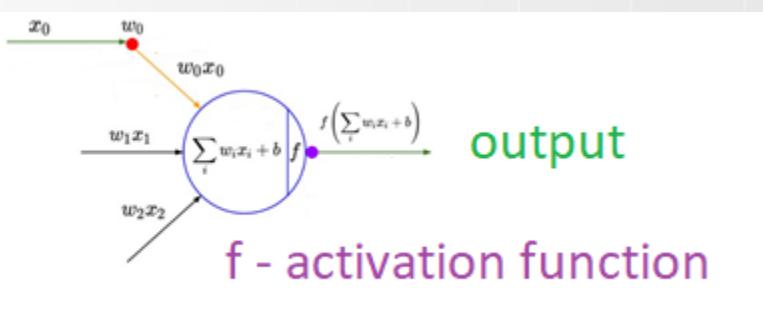
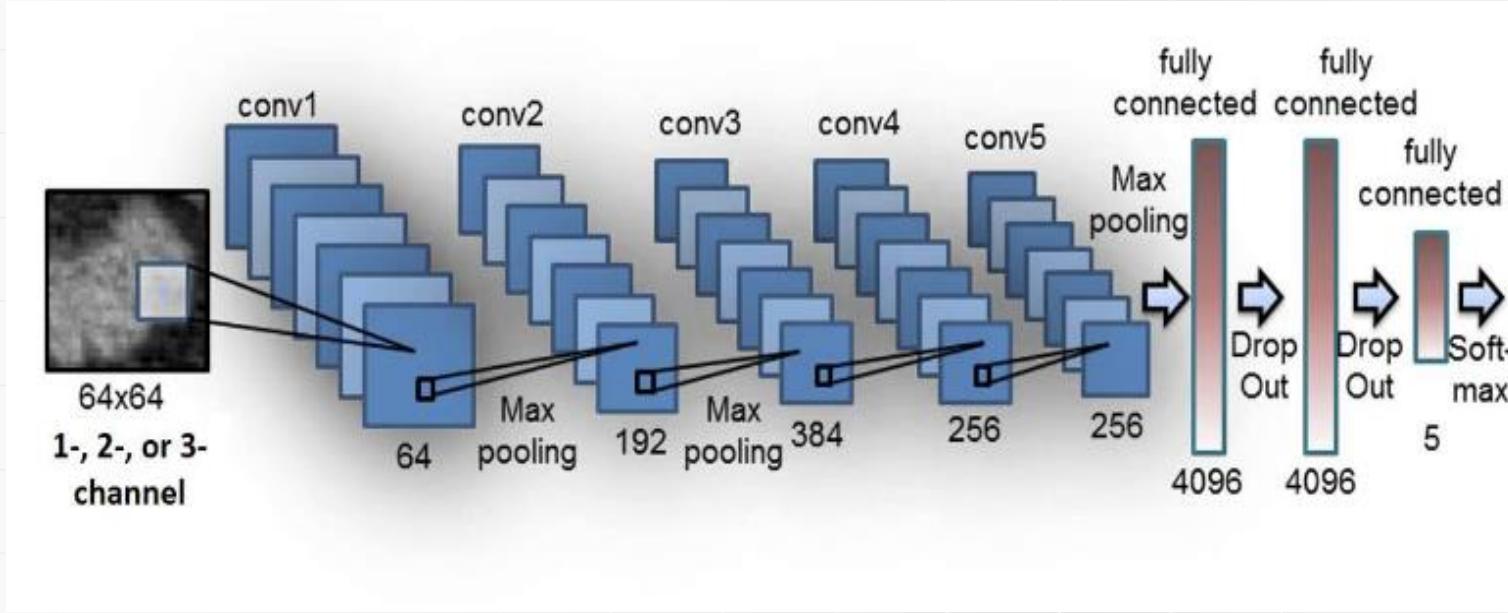


- Podstawowe elementy:
  - Warstwy konwolucyjne (mapy cech)
  - Warstwy wykonujące sampling (max pooling lub average pooling)
  - Warstwy w pełni połączone



# CNN

## Struktura sieci konwolucyjnej



Leaky ReLu jako funkcja aktywacji

Neurony w warstwie konwolucyjnej mają klasyczną budowę



# Jak określić rozmiar map cech

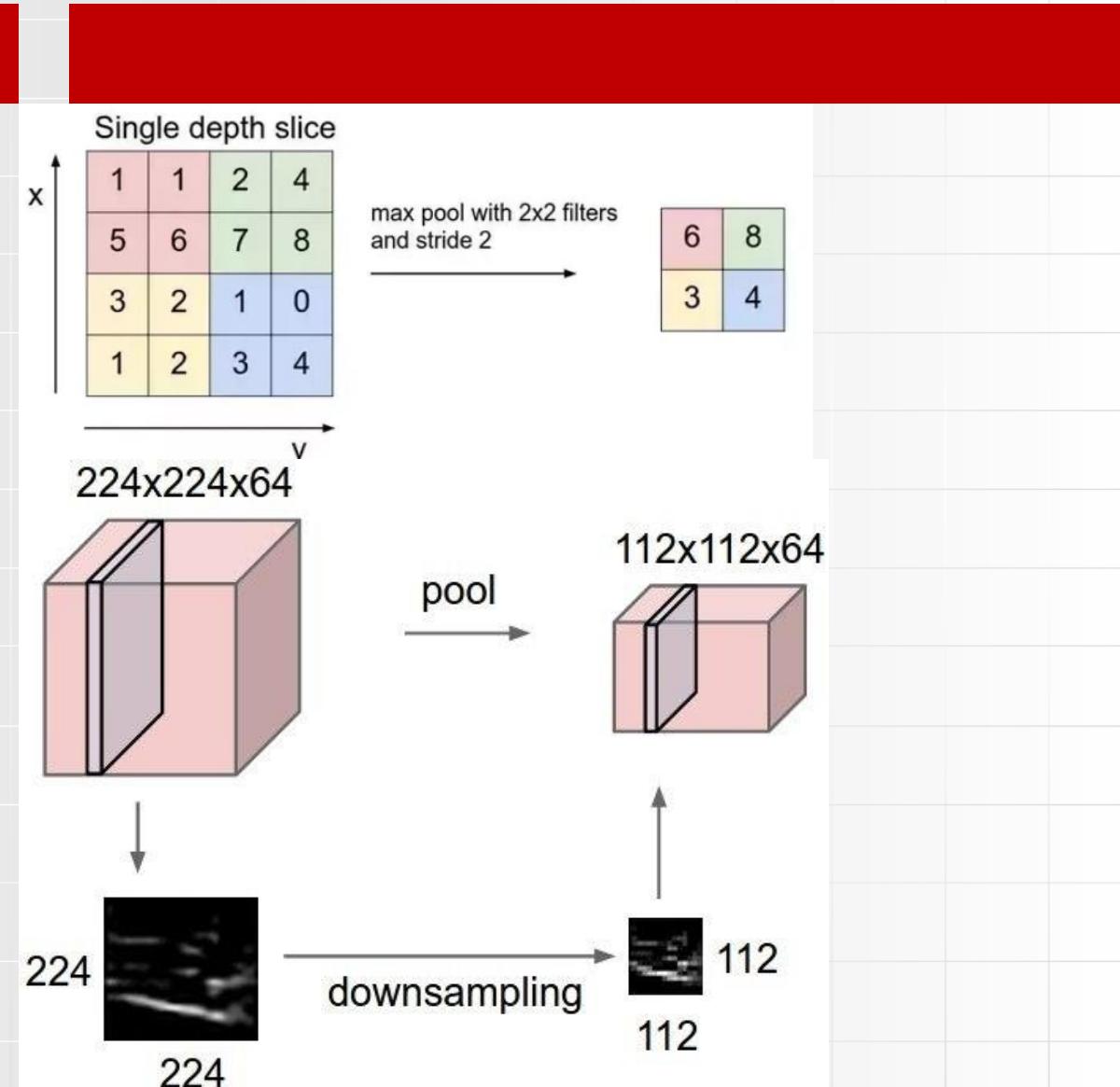
- Możemy obliczyć rozmiar wyjścia (mapy cech) w zależności od wejścia W, filtra F, kroku S i ramki P. Jest on dany wzorem:
  - $(W-F+2P)/S+1$
- Dla przykładu, jeśli  $W=10$ ,  $P=0$ ,  $F=3$ , nie jest możliwe użycie kroku  $S=2$ , ponieważ  $(W-F+2P)/S+1=(10-3+0)/2+1=4.5$
- Filtr musi się mieścić całkowitą liczbę razy



# Pooling (subsampling)

Max pooling – rozwiązuje następujące problemy

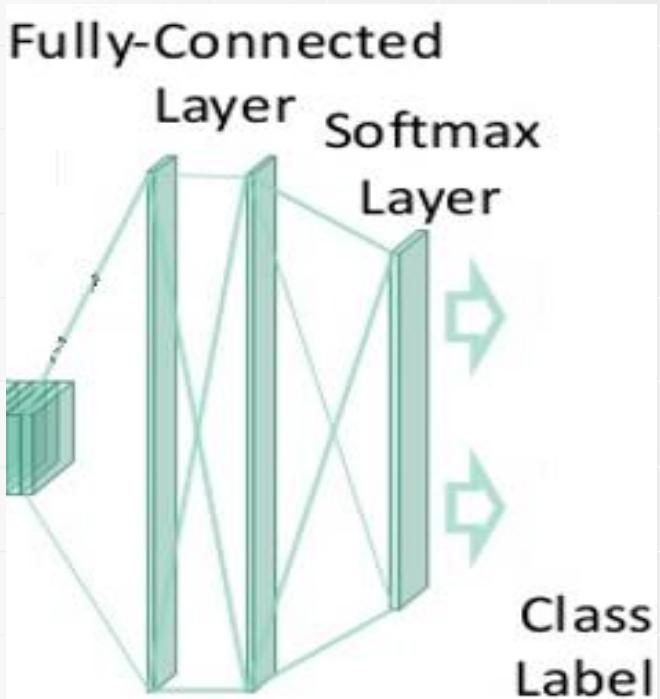
- Redukuje liczbę parametrów modelu – określa się to czasem subsampling.
- Uogólnia wyniki z filtrów konwolucyjnych – przyczynia się (w pewnym stopniu) do uniezależnienia od skali i przesunięcia.
- UWAGA: wszystkie połączenia w warstwie pooling są = 1 i nie podlegają uczeniu.



Credit: <http://cs231n.github.io/convolutional-networks/>



# Warstwy w pełni połączone



- Ostatnia warstwa to softmax.
- Modeluje rozkład prawdopodobieństwa klas.
- Czasem stosowany jest SVM w ostatniej warstwie.
- Każde połączenie ma przypisaną wagę, poszukiwaną w trakcie uczenia
- Często używa się dropout.



# Jak możemy wyuczyć filtry?

## Używając BP

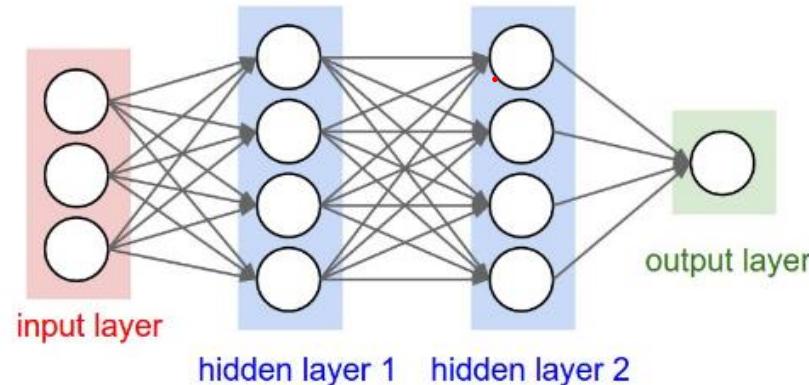
- SGD – gradient obliczany na mini batchach – paczkach obrazów.
- Wagi filtrów są współdzielone przez cały kanał (mapę cech).

## Parametry & hiperparametry

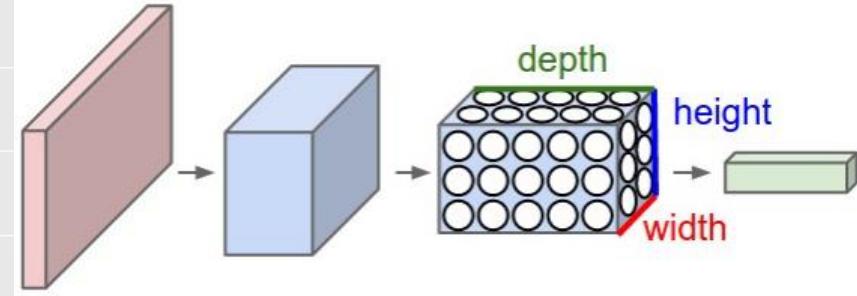
- Parametry = wagi i biasy znajdowane w procesie uczenia
- Hiperparametry:
  - Rozmiar filtra, np: 3x3, 5x5, 11x11)
  - Krok (stride)
  - Ramka (Padding)
  - Liczba warstw w sieci
  - Liczba filtrów
  - Współczynnik uczenia

# MLP a sieć konwolucyjna

## MLP



## Convolution



Credit: <http://cs231n.github.io/convolutional-networks/>

Przyjmijmy, że mamy obraz  $4 \times 4$  i dokonujemy przekształcenia na siatkę  $2 \times 2$ .

**Dla MLP**, przekształcamy wejście  $4 \times 4$  na wektor o 16 współrzędnych, Macierz wag **W** w warstwie ma 64 parametry:

$$\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} & w_{1,6} & w_{1,7} & w_{1,8} & w_{1,9} & w_{1,10} & w_{1,11} & w_{1,12} & w_{1,13} & w_{1,14} & w_{1,15} & w_{1,16} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} & w_{2,6} & w_{2,7} & w_{2,8} & w_{2,9} & w_{2,10} & w_{2,11} & w_{2,12} & w_{2,13} & w_{2,14} & w_{2,15} & w_{2,16} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} & w_{3,6} & w_{3,7} & w_{3,8} & w_{3,9} & w_{3,10} & w_{3,11} & w_{3,12} & w_{3,13} & w_{3,14} & w_{3,15} & w_{3,16} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} & w_{4,6} & w_{4,7} & w_{4,8} & w_{4,9} & w_{4,10} & w_{4,11} & w_{4,12} & w_{4,13} & w_{4,14} & w_{4,15} & w_{4,16} \end{bmatrix}$$

V. Dumoulin, F. Visin: A guide to convolution arithmetic for deep learning,  
<https://arxiv.org/pdf/1603.07285.pdf>

**CNN** z jądrem **K** o rozmiarze 3 na wejściu  $4 \times 4$  daje wyjście  $2 \times 2$  odpowiednia macierz transformacji wygląda następująco:

$$\begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix}$$

Mamy tylko **9** parameterów, dzięki **lokalnym** połączeniom



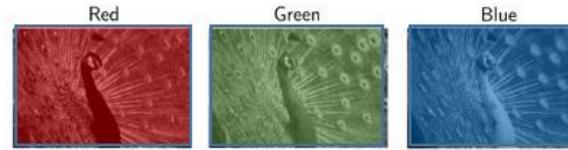
# Różne funkcje aktywacji

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) \underset{x=0}{\text{is undefined}} \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$



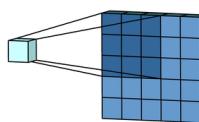
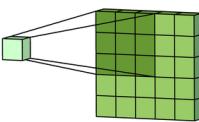
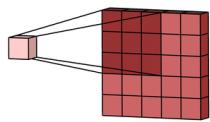
# Wiele kanałów

## Obraz kolorowy – 3 kanały RGB

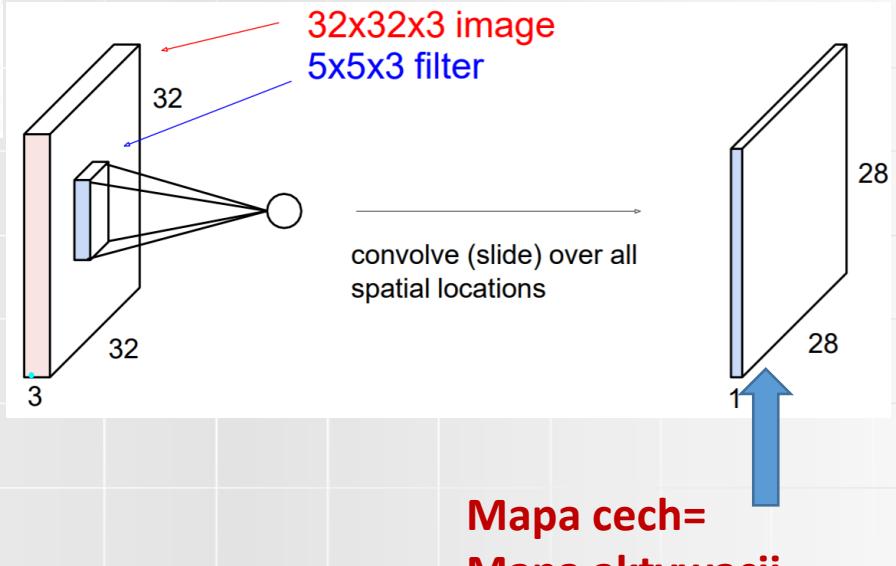


(Credit: [Andre Mouton](#))

Trzy kanały dla każdego komponentu RGB



Dla każdego kanału jedno jądro channel,



Jedno **jądro (filtr)** dla każdego pojedynczego kanału  
**kanału, (mapy aktywacji)**  
Każde jądro jest **unikalne**.

Filtr jako całość jest jeden dla całego kanału wyjściowego.



# Definiowanie warstwy konwolucyjnej

Przykład inicjalizacji warstwy konwolucyjnej w Pytorch

```
15. # initialize first set of CONV => RELU => POOL layers
16. self.conv1 = Conv2d(in_channels=numChannels, out_channels=20,
17.     kernel_size=(5, 5))
18. self.relu1 = ReLU()
19. self.maxpool1 = MaxPool2d(kernel_size=(2, 2), stride=(2, 2))
20.
```

```
21. # initialize second set of CONV => RELU => POOL layers
22. self.conv2 = Conv2d(in_channels=20, out_channels=50,
23.     kernel_size=(5, 5))
24. self.relu2 = ReLU()
25. self.maxpool2 = MaxPool2d(kernel_size=(2, 2), stride=(2, 2))
26.
```

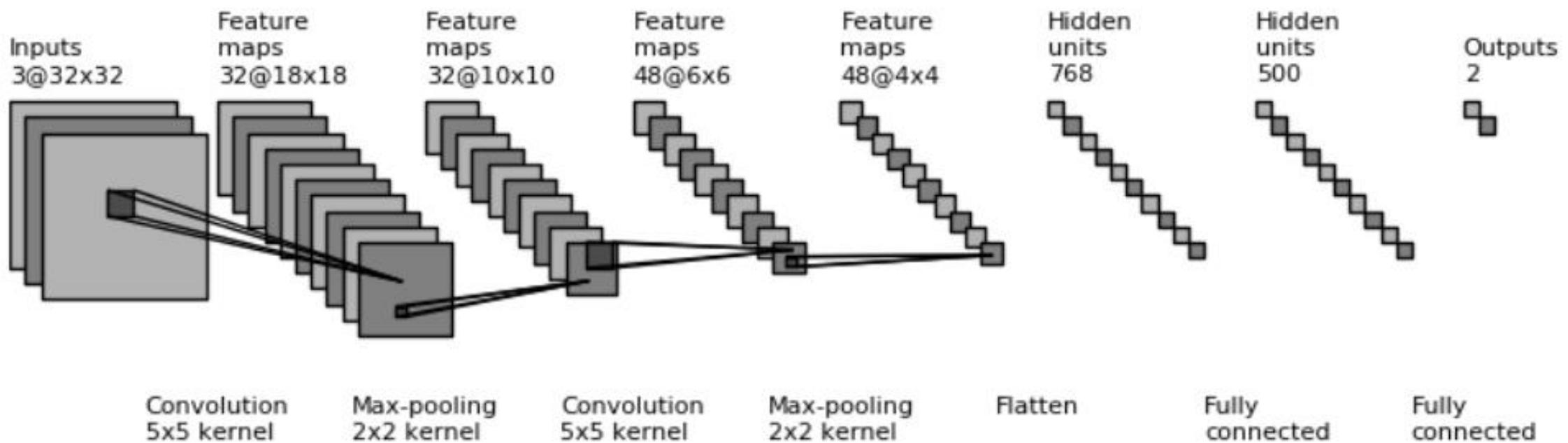
- Pierwsza warstwa konwolucyjna uczy się 20 filterów, każdy ma rozmiar  $5 \times 5$ . Stosuje sięReLU a następnie warstwę  $2 \times 2$  z krokiem  $2 \times 2$ , aby zmniejszyć rozmiar mapy cech.
- Przeanalizujmy drugą warstwę konwolucyjną

# **Uczenie warstw konwolucyjnych**

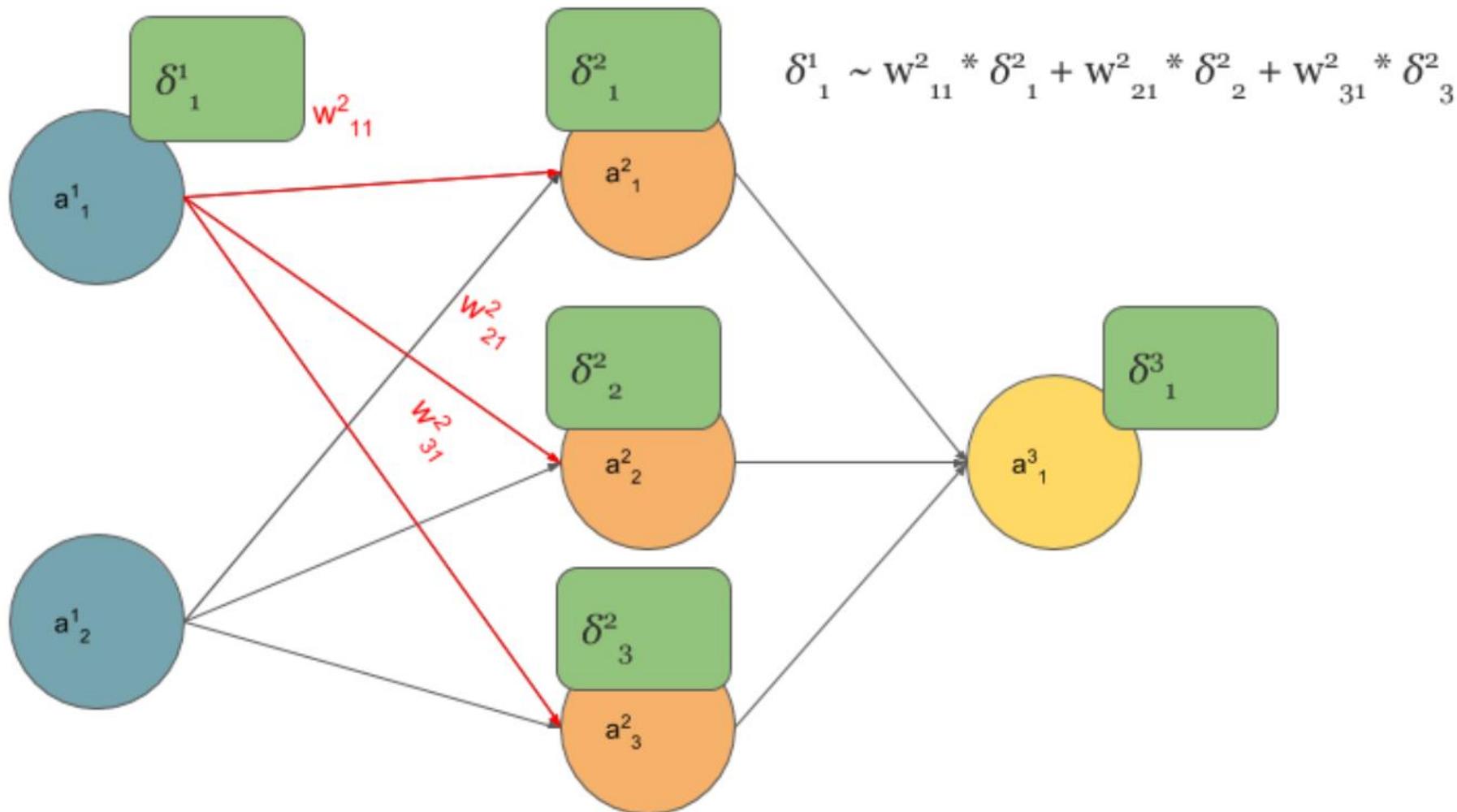


Politechnika Wrocławská

# Sieć konwolucyjna

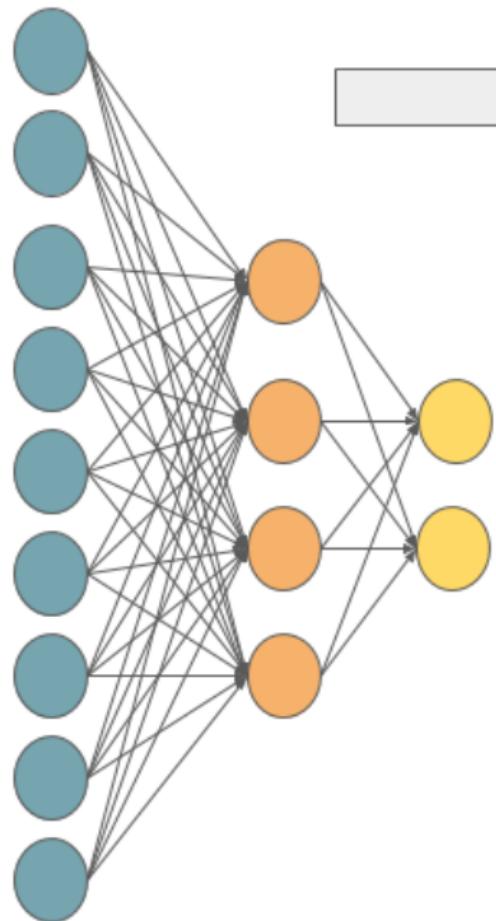


# Jak obliczamy błędy w MLP?

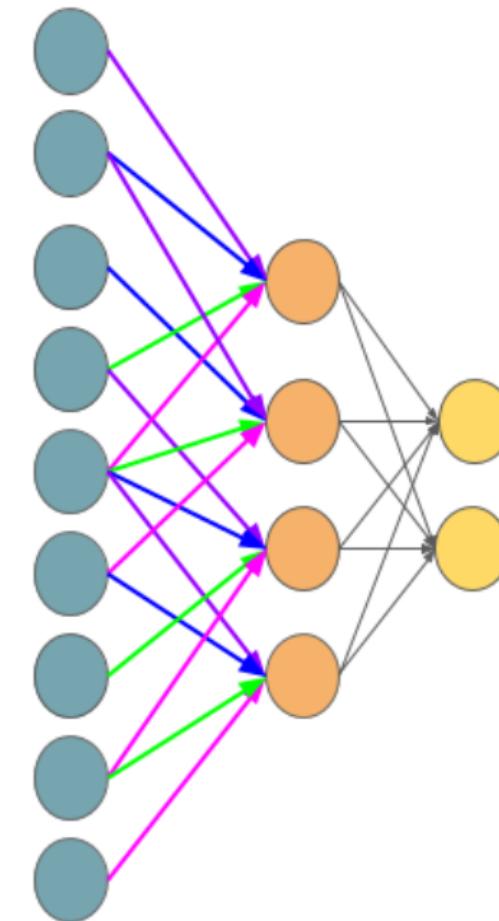
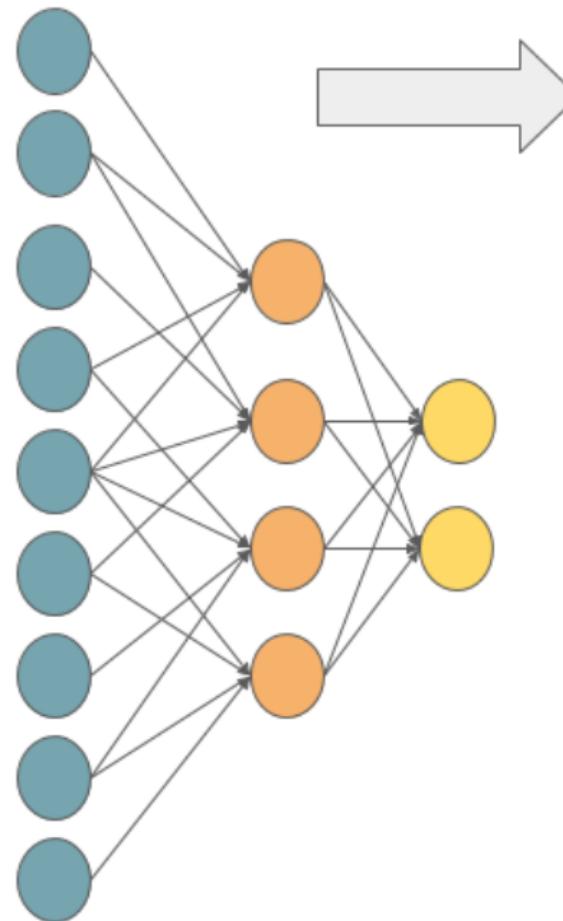


# Sieć CNN też jest jednokierunkowa

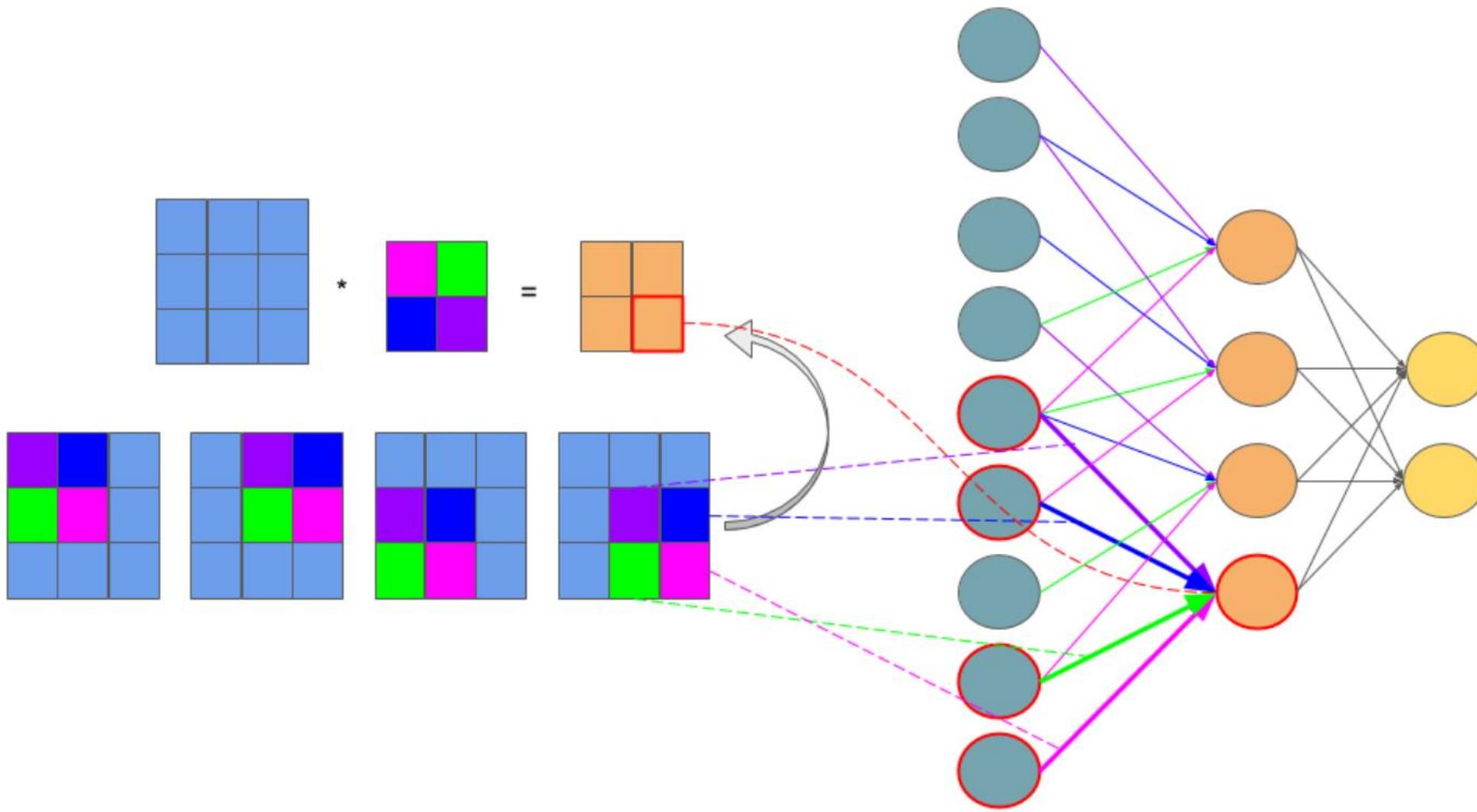
connections cutting



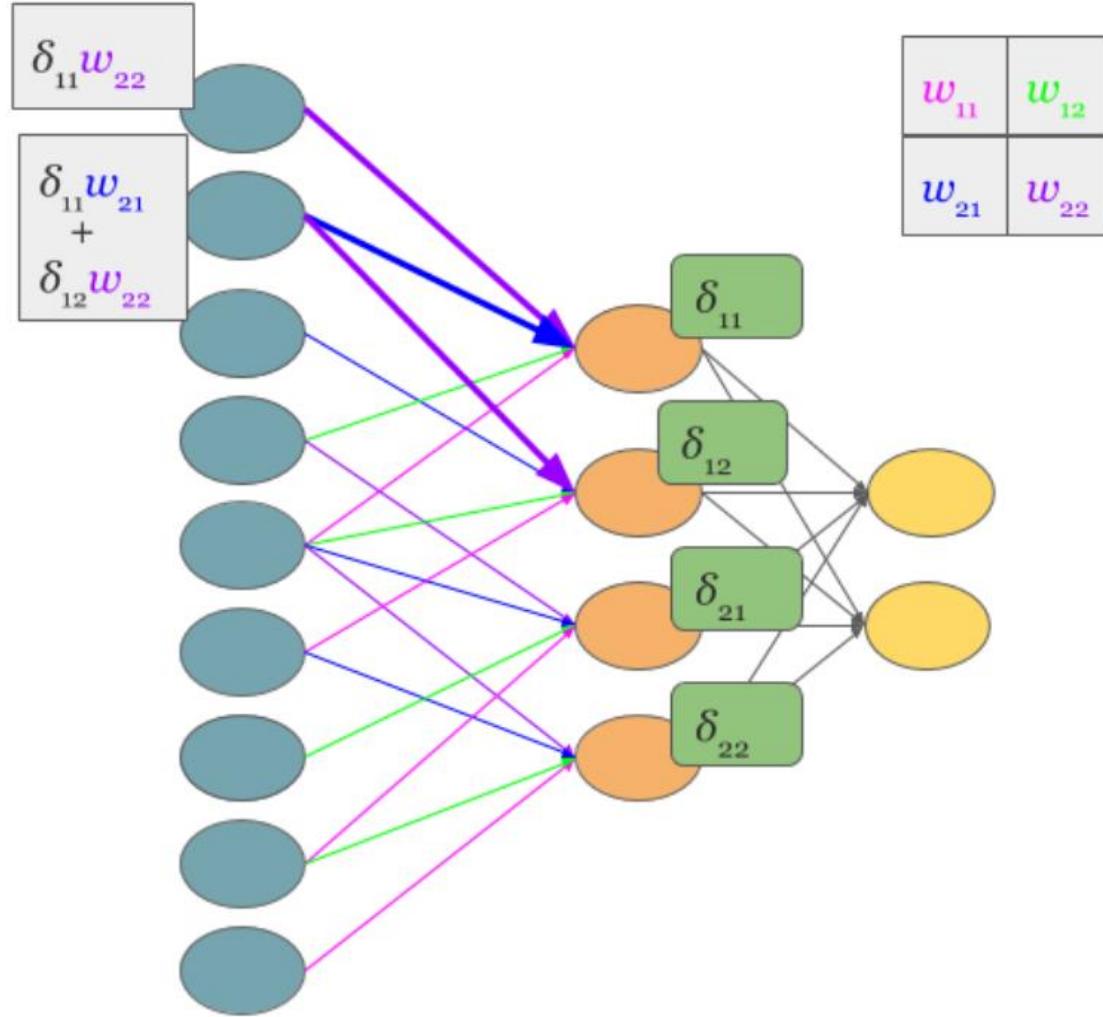
weights sharing



# Sieć CNN z jedną warstwą konwolucyjną



# Obliczanie błędów dla CNN



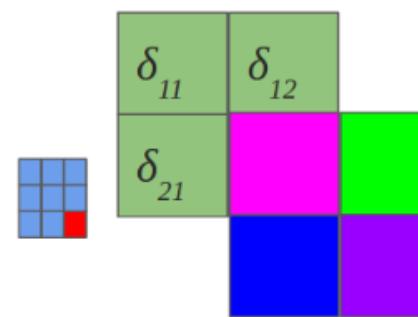
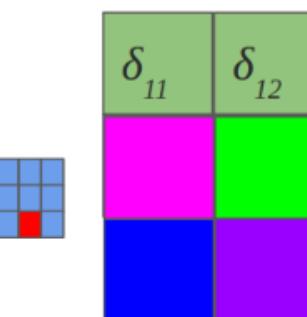
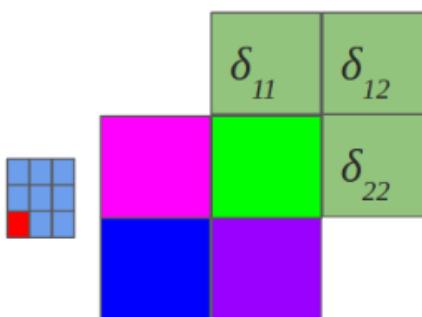
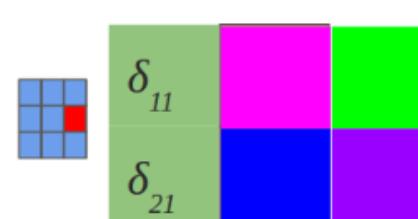
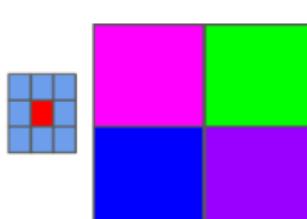
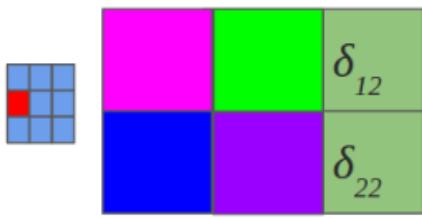
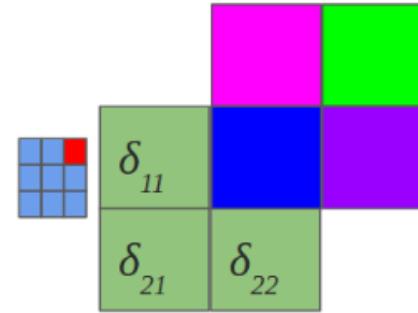
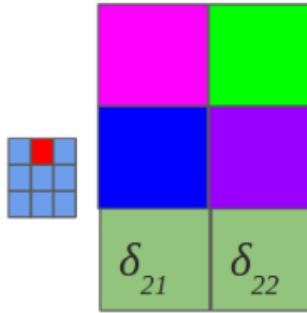
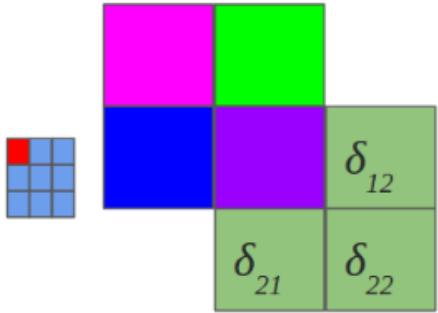
$$\begin{array}{c}
 \text{rot\_180}(w) * \text{grads from orange layer} = \\
 \begin{array}{c}
 \begin{array}{|c|c|} \hline w_{22} & w_{21} \\ \hline w_{12} & w_{11} \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|} \hline \delta_{11} & \delta_{12} \\ \hline \delta_{21} & \delta_{22} \\ \hline \end{array} = \\
 \begin{array}{|c|c|} \hline \delta_{11} w_{22} & \delta_{11} w_{21} + \delta_{12} w_{22} & \delta_{12} w_{21} \\ \hline \delta_{11} w_{12} + \delta_{12} w_{12} + \delta_{21} w_{21} + \delta_{22} w_{22} & \delta_{12} w_{11} + \delta_{21} w_{12} & \delta_{12} w_{11} + \delta_{22} w_{21} \\ \hline \delta_{21} w_{12} & \delta_{21} w_{11} + \delta_{22} w_{12} & \delta_{22} w_{11} \\ \hline \end{array}
 \end{array}
 \end{array}$$

Legend:

- $w_{11}, w_{12}, w_{21}, w_{22}$  (top row)
- $w_{12}, w_{11}$  (middle row)
- $\delta_{11}, \delta_{12}$  (top row)
- $\delta_{21}, \delta_{22}$  (middle row)

Diagram showing the rotation of the weight matrix  $w$  by 180 degrees and its multiplication with the gradients from the orange layer. The result is a 3x3 grid of terms representing the product of specific weight elements and their corresponding gradients.

# To też jest konwolucja



# Jak zmieniamy wagi w MLP

- Błąd dla j-tego neuronu

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

- Gdzie  $z_j^l$  jest całkowitym pobudzeniem

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

- Natomiast  $a_j^l$  jest wyjściem po przetworzeniu przez funkcję aktywacji  $\sigma$

$$a_j^l = \sigma(z_j^l)$$

# Przechodzimy od MLP do CNN

- Mnożenie macierzy zastępujemy konwolucją - \* (neuron ma pozycję  $(x,y)$  na mapie cech w  $(l+1)$ -ej warstwie (jest to faza przesłania w przód- forward)

$$z_{x,y}^{l+1} = w^{l+1} * \sigma(z_{x,y}^l) + b_{x,y}^{l+1} = \sum_a \sum_h w_{a,h}^{l+1} \sigma(z_{x-a,y-h}^l) + b_{x,y}^{l+1}$$

- $\sigma$  jest funkcją aktywacji (zazwyczaj ReLU)
- Najpierw obliczymy błąd:

$$\delta_{x,y}^l = \frac{\partial C}{\partial z_{x,y}^l} = \sum_{x'} \sum_{y'} \frac{\partial C}{\partial z_{x',y'}^{l+1}} \frac{\partial z_{x',y'}^{l+1}}{\partial z_{x,y}^l}$$

# Cd liczenia błędu

- I znowu reguła łańcuchowa

$$\frac{\partial C}{\partial z_{x,y}^l} = \sum_{x'} \sum_{y'} \frac{\partial C}{\partial z_{x',y'}^{l+1}} \frac{\partial z_{x',y'}^{l+1}}{\partial z_{x,y}^l} = \sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} \frac{\partial (\sum_a \sum_b w_{a,b}^{l+1} \sigma(z_{x'-a,y'-b}^l) + b_{x',y'}^{l+1})}{\partial z_{x,y}^l}$$

- Pierwszy komponent jest błędem w warstwie wyższej
- W drugim - wszystkie składniki są równe 0, oprócz tego o indeksach:  
 $x = x' - a$   
 $y = y' - b$
- Stąd otrzymujemy:

$$\sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} \frac{\partial (\sum_a \sum_b w_{a,b}^{l+1} \sigma(z_{x'-a,y'-b}^l) + b_{x',y'}^{l+1})}{\partial z_{x,y}^l} = \sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{a,b}^{l+1} \sigma'(z_{x,y}^l)$$

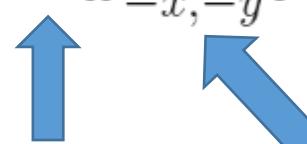
# Dalej liczymy błąd

- Jeśli  $x = x' - a$  oraz  $y = y' - b$  to  $a = x' - x$  oraz  $b = y' - y$
- Możemy przepisać równanie do postaci:

$$\sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{a,b}^{l+1} \sigma'(z_{x,y}^l) = \sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{x'-x,y'-y}^{l+1} \sigma'(z_{x,y}^l)$$

- A to możemy przedstawić jako:

$$\sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{x'-x,y'-y}^{l+1} \sigma'(z_{x,y}^l) = \delta^{l+1} * w_{-x,-y}^{l+1} \sigma'(z_{x,y}^l)$$

  
konwolucja  ROT180( $w_{x,y}^{l+1}$ )

# Zmiana wag

- Pochodna cząstkowa:

$$\frac{\partial C}{\partial w_{a,b}^l} = \sum_x \sum_y \frac{\partial C}{\partial z_{x,y}^l} \frac{\partial z_{x,y}^l}{\partial w_{a,b}^l} = \sum_x \sum_y \delta_{x,y}^l \frac{\partial (\sum_{a'} \sum_{b'} w_{a',b'}^l \sigma(z_{x-a',y-b'}^l) + b_{x,y}^l)}{\partial w_{a,b}^l} =$$
$$\sum_x \sum_y \delta_{x,y}^l \sigma(z_{x-a,y-b}^{l-1}) = \delta_{a,b}^l * \sigma(z_{-a,-b}^{l-1}) = \delta_{a,b}^l * \sigma(ROT180(z_{a,b}^{l-1}))$$

# Algorytm BP dla CNN

- Zastosuj wzorzec wejściowy  $x$  do warstwy wejściowej
- Dla każdej warstwy  $l = 2, 3, \dots, L$  oblicz  $z_{x,y}^l = w^l * \sigma(z_{x,y}^{l-1}) + b_{x,y}^l$  oraz  $a_{x,y}^l = \sigma(z_{x,y}^l)$
- Oblicz błąd w warstwie wyjściowej  $\delta^L = \nabla_a C \odot \sigma'(z^L)$
- Rzutuj błąd wstecz dla warstw  $l=L-1, L-2, \dots, 2$  obliczając:

$$\delta_{x,y}^l = \delta^{l+1} * ROT180(w_{x,y}^{l+1})\sigma'(z_{x,y}^l)$$

- Przyrost wagi  $\Delta w_{a,b}^l(t)$  jest proporcjonalny do składowej gradientu
- $\frac{\partial C}{\partial w_{a,b}^l} = \delta_{a,b}^l * \sigma'(ROT180(z_{a,b}^{l-1}))$

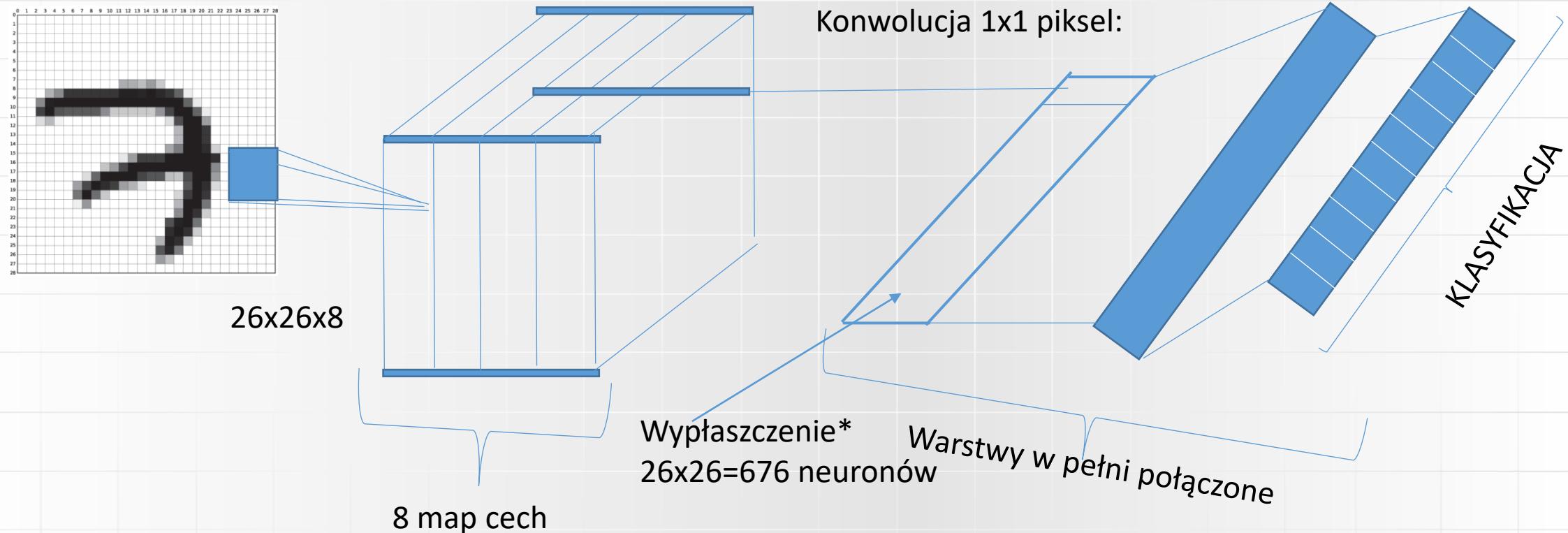
# Warstwa poolingu

- Może być realizowana jako:
  - operacja uśredniania lub
  - wybierania max
- Służy do redukcji rozmiaru mapy (i liczby parametrów). NxN blok zastępowany jest pojedynczą wartością.
- Uczenie w tej warstwie nie zachodzi (każde połączenie ma wagę stałą równą 1)
- Dla operacji max w trakcie Backpropagation obliczany jest błąd uzyskiwany przez zwycięską jednostkę (pozostałe błędy są równe 0, dlatego trzeba pamiętać indeks zwycięskiej jednostki).
- Dla operacji uśredniania błąd jest przypisywany do całego bloku i mnożony przez  $1/(NxN)$



# Płytki sieć konwolucyjna

Poznanie operacji konwolucji – implementacja płytkej sieci z warstwą konwolucyjną w numpy



\*UWAGA W prostszej wersji wypłaszczenie możemy zrealizować bez konwolucji 1x1.

Wtedy pierwsza płaska warstwa będzie miała  $26 \times 26 \times 8 = 676 \times 8$  neuronów



# Podsumowanie

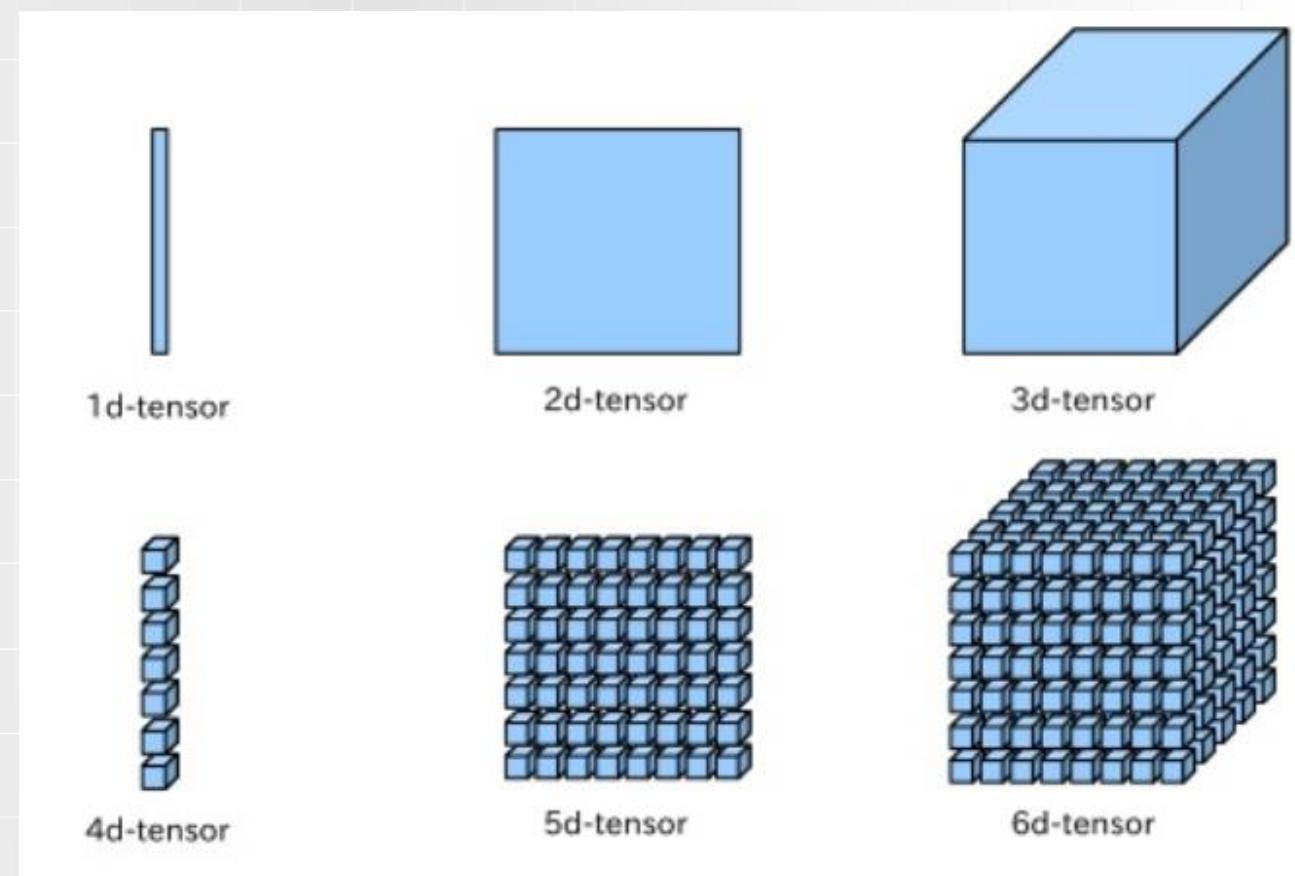
- Sieci konwolucyjne są przede wszystkim wykorzystywane do przetwarzania obrazów, ale też do klasyfikacji tekstów, rozpoznawania mowy.
- Wszystkie zawierają warstwy konwolucyjne, różnią się wielkością filtrów, liczbą warstw po których stosowany jest pooling, normalizacją wag w sieci
- Sieci z połączeniami rezydualnymi pozwalają na budowanie sieci (nawet kilkaset warstw)



# Tensor

## Ilustracja pojęcia

W pewnym uproszczeniu, **tensor** jest uogólnieniem macierzy. We wszystkich platformach do DL operacje wykonuje się na tensorach, np. mapy cech są tensorem 3d.





Wrocław  
University  
of Science  
and Technology



# SIECI NEURONOWE



HR EXCELLENCE IN RESEARCH



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Politechnika Wrocławskiego

Unia Europejska  
Europejski Fundusz Społeczny





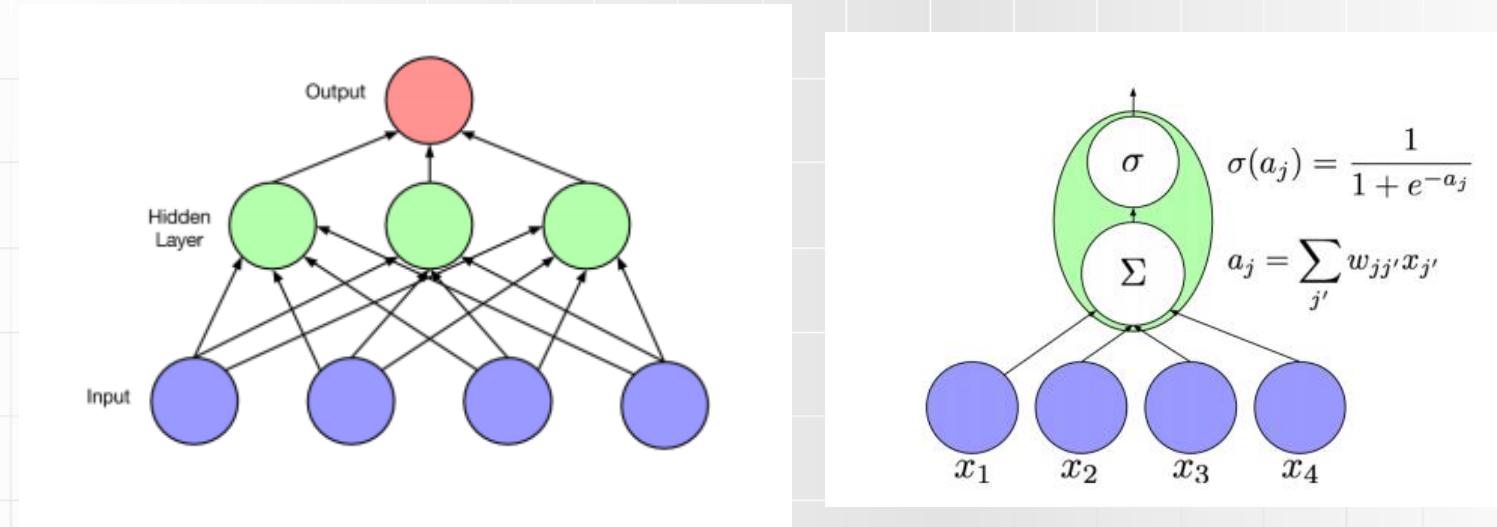
# Sieci neuronowe

## Wstęp

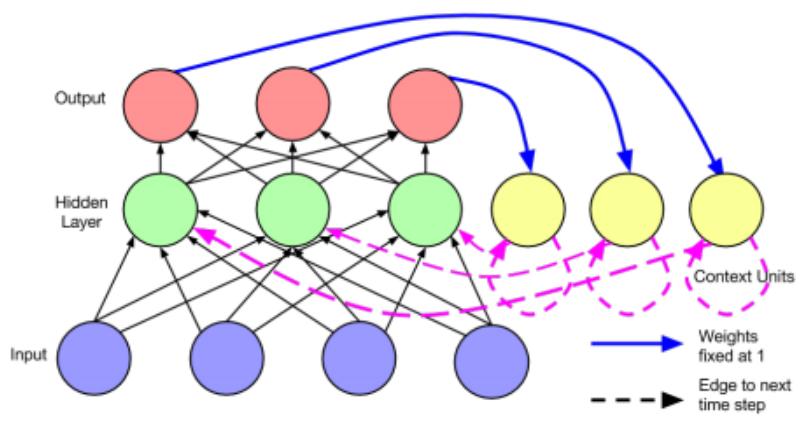
**URSZULA MARKOWSKA-KACZMAR**  
Katedra Inteligencji Obliczeniowej  
Wydział Informatyki i Zarządzania

# Dlaczego potrzebujemy sieci rekurencyjnych

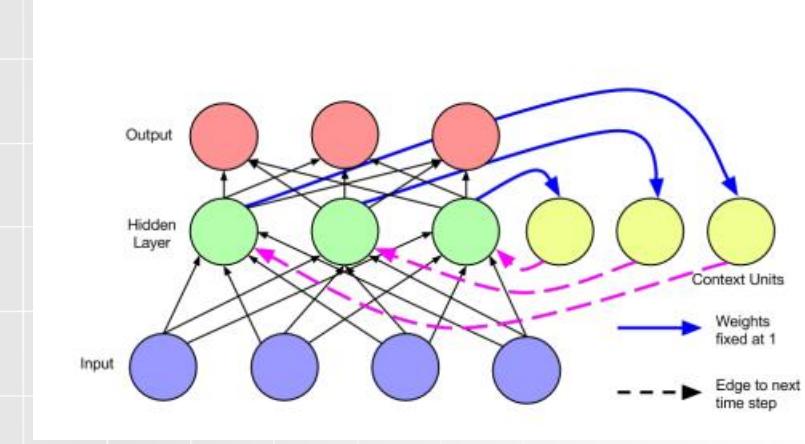
- Sieci typu feedforward są dobrą techniką do klasyfikacji.
- Jak możemy je zastosować do predykcji serii czasowych?
- Jak zamodelować relacje czasowe we wzorcach (mowa, rozpoznawanie i rozumienie wideo)?



# Przykłady klasycznych sieci neuronowych



Sieć Jordana



Sieć Elmana

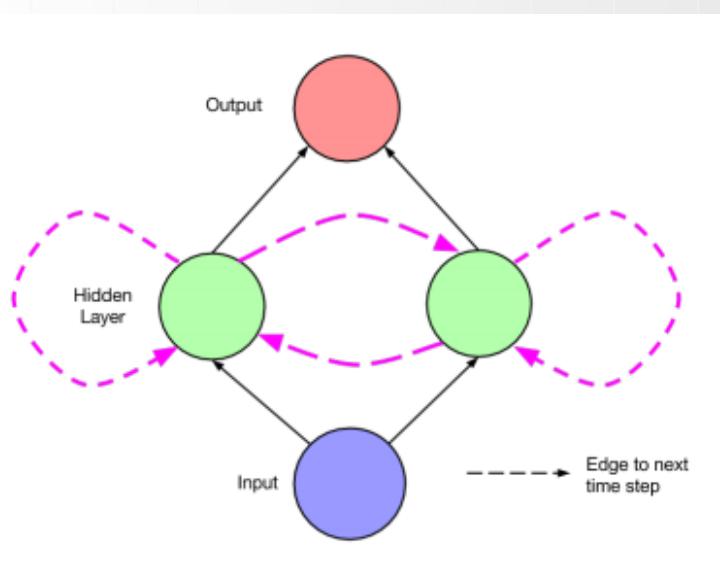
Sieci mają tę samą architekturę jak sieci jednokierunkowe  
Sposób zmian połączeń:

Cykle pomiędzy neuronami,

Połączenia na siebie,

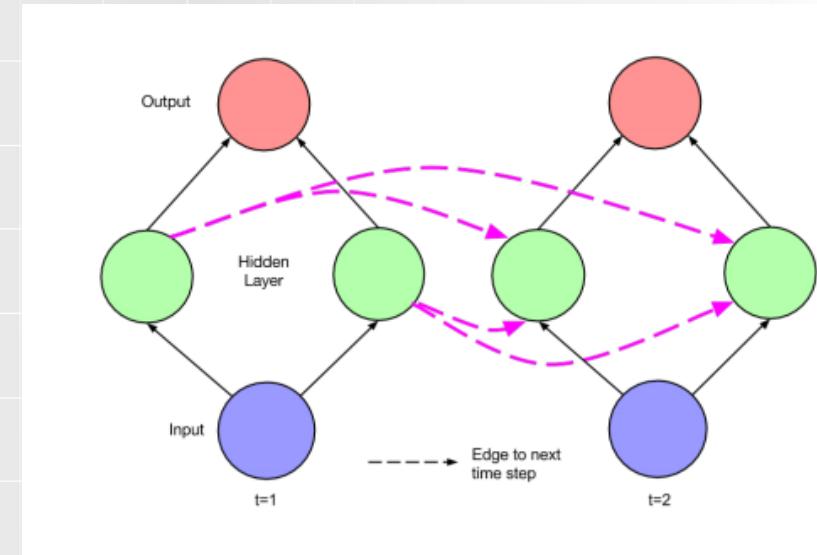
Czasem pojawiają się połączenia między wyjściem a wejściem.

# Klasyczne RNN dla sekwencji czasowych



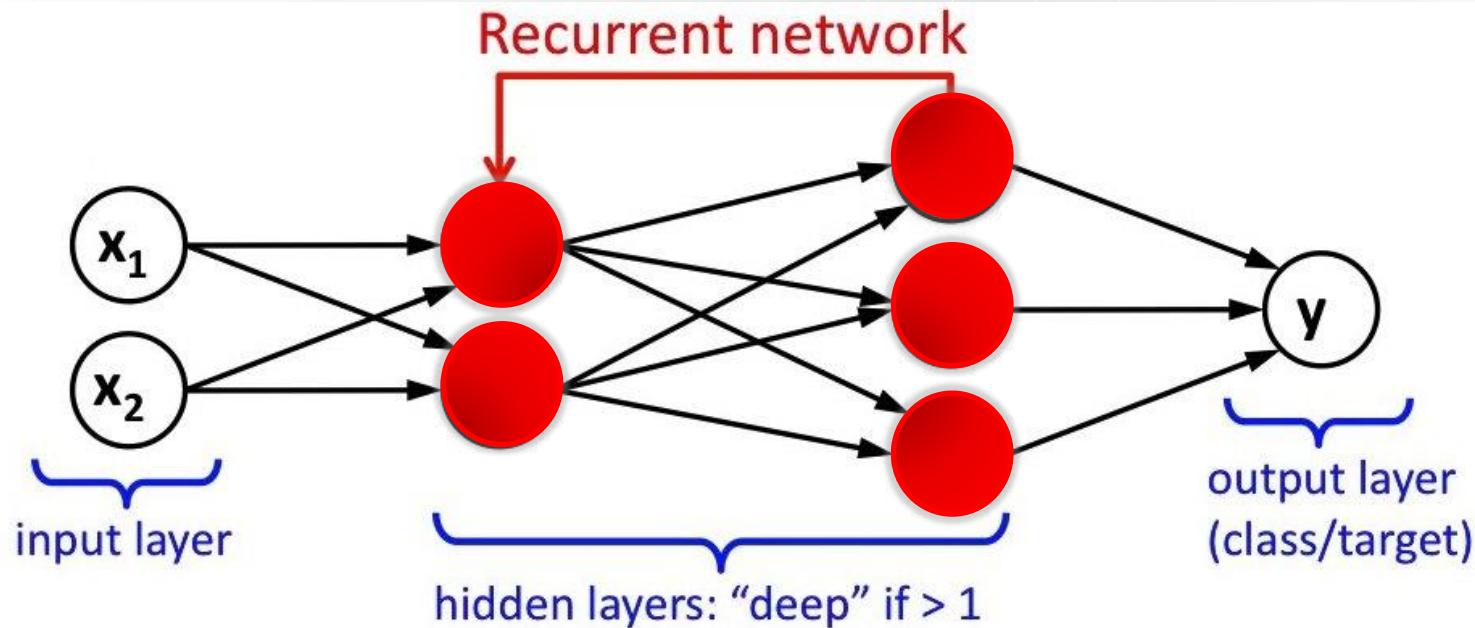
$$\mathbf{h}^{(t)} = \sigma(W^{\text{hx}} \mathbf{x}^{(t)} + W^{\text{hh}} \mathbf{h}^{(t-1)} + \mathbf{b}_h)$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(W^{\text{yh}} \mathbf{h}^{(t)} + \mathbf{b}_y).$$



Sieć rozwinięta w relacji do czasu

# Rekurencyjna sieć neuronowa(RNN)

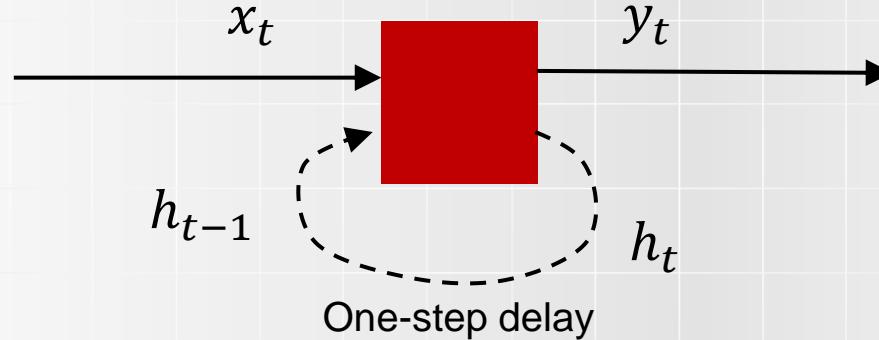


Source of image: [https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/recurrent\\_neural\\_networks.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/recurrent_neural_networks.html)



# Sieci rekurencyjne

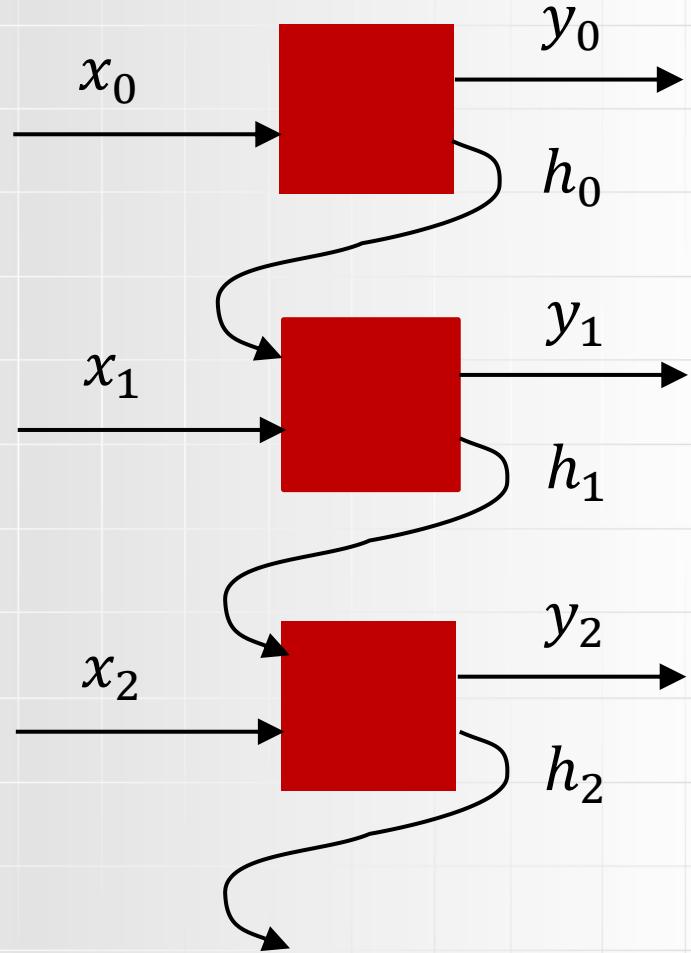
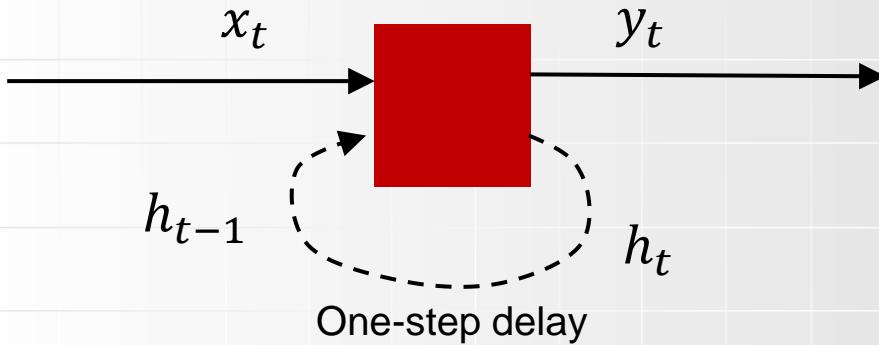
Zależą od czasu i mają cykle.



Przetwarzają wejściową sekwencję  $x_1, \dots, x_n$   
I produkują wyjściową sekwencję  $y_1, \dots, y_m$ .

Source: CS294-129:

# Rozwijjanie w czasie



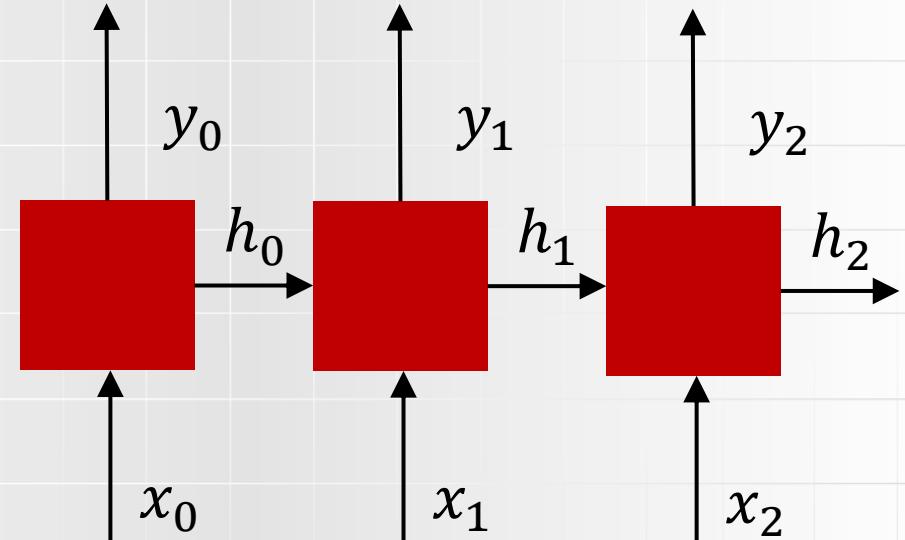
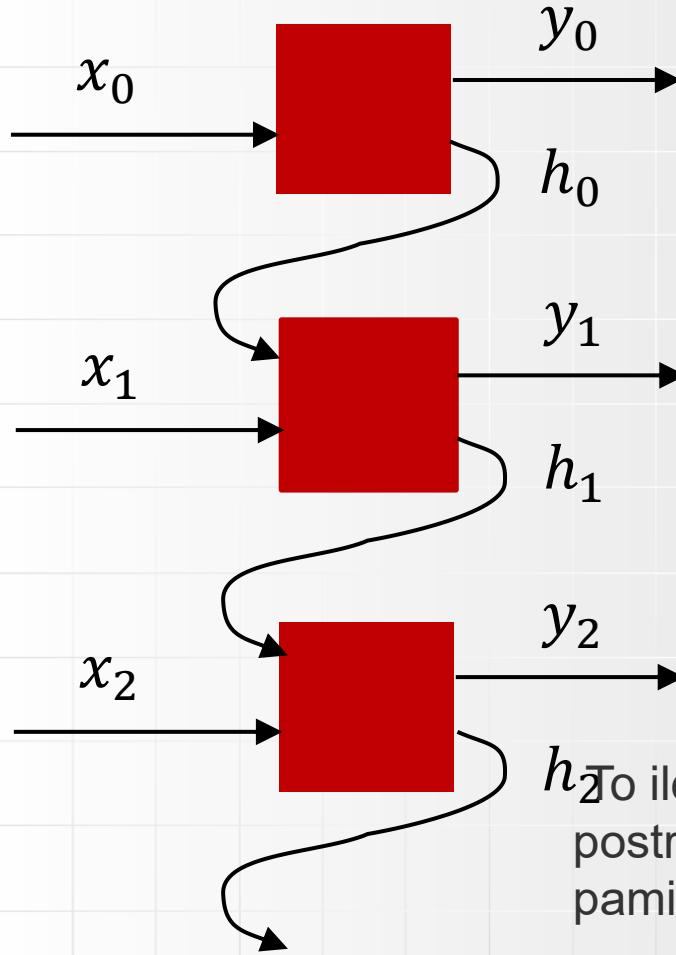
Wyniki rozwijania sieci w prosty  
acykliczny graf (DAG).

Daje możliwość zastosowania  
Backpropagation.

Od czego zależy rozmiar  
rozwijania sieci?

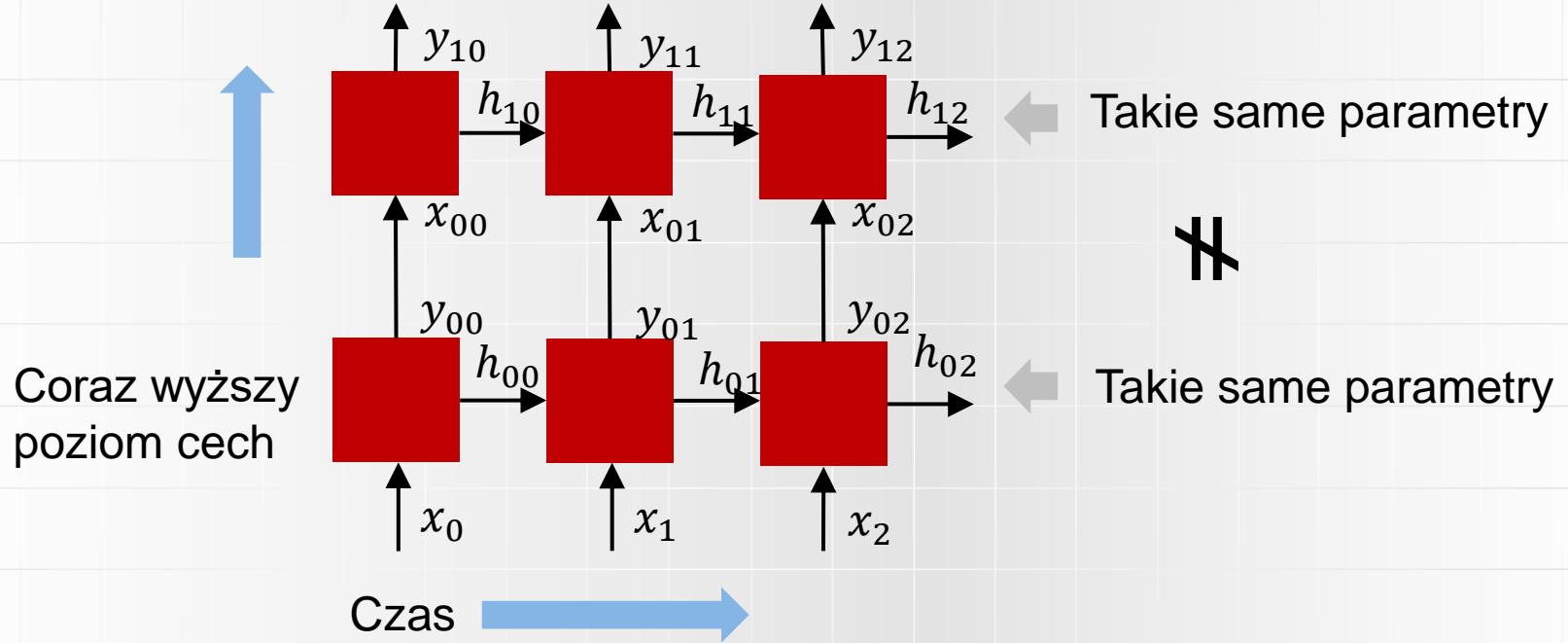
# Rozwijanie RNN

Zwykle są prezentowane jako:



h<sub>2</sub> To ile razy rozwijamy sieć (unroll , unfold)?  
postrzegamy to z punktu widzenia historii jaką sieć ma pamiętać. W ten sposób mamy sieć jednokierunkową.

# Struktura sieci



Tworzy się warstwy w pionie (deep RNN)

Deep zazwyczaj odnosi się do czasu.

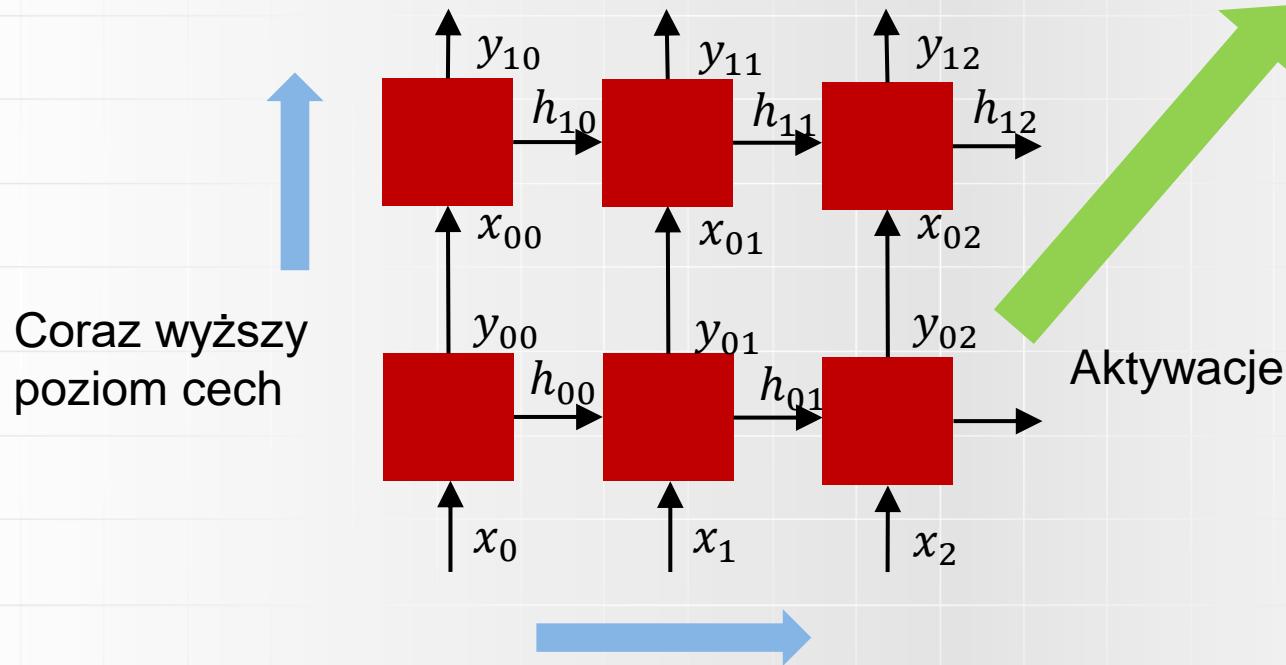


# BackPropagation Through Time -

## BPTT

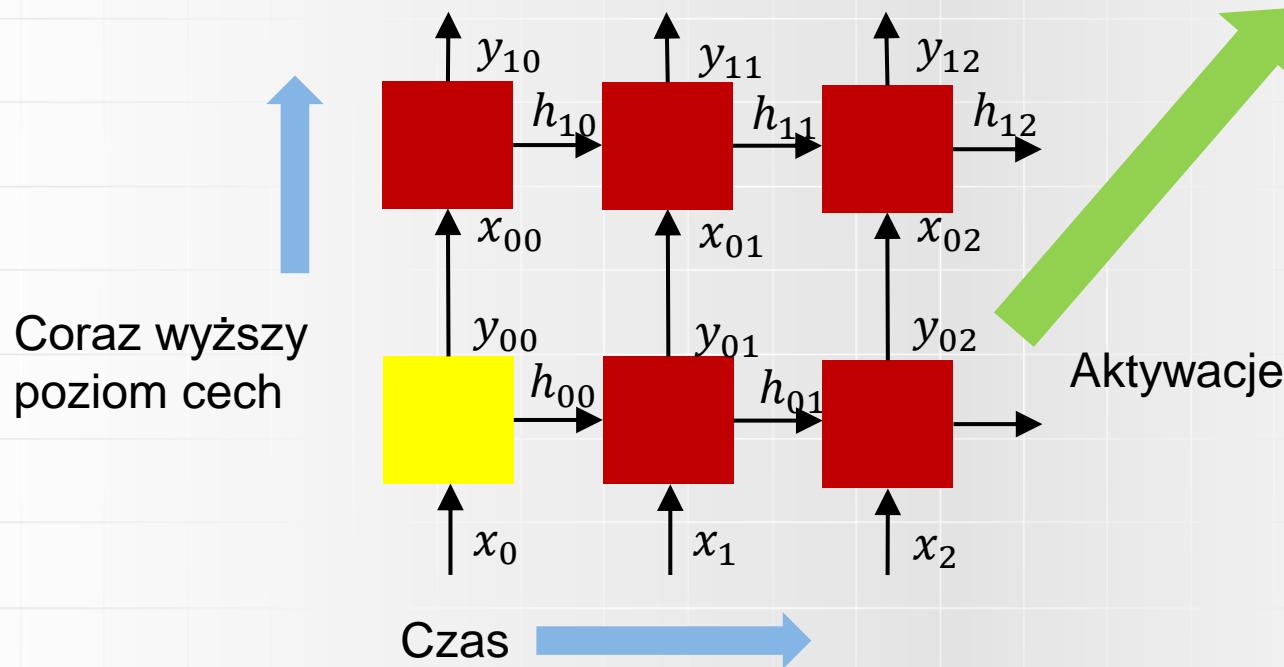
- Unfolding (rozwijanie) sieci w czasie prowadzi do głębokich sieci RNN.
- Wejścia, wyjścia i neurony ukryte są replikowane w każdym kroku czasowym.
- Jednostki ukryte są połączone w następujący sposób: kiedy oryginalna RNN ma połączenia zwrotne z wagą  $w$  od neuronu  $i$  do neuronu  $j$ , wówczas w sieci feedforward istnieje połączenie z wagą  $w$  pomiędzy neuronami  $i$  w warstwie  $t_k$  i neuronem  $j$  w warstwie  $t_{k+1}$  (współdzielenie wag).
- podsumowując: transformujemy sieć i stosujemy BP.

# Struktura sieci

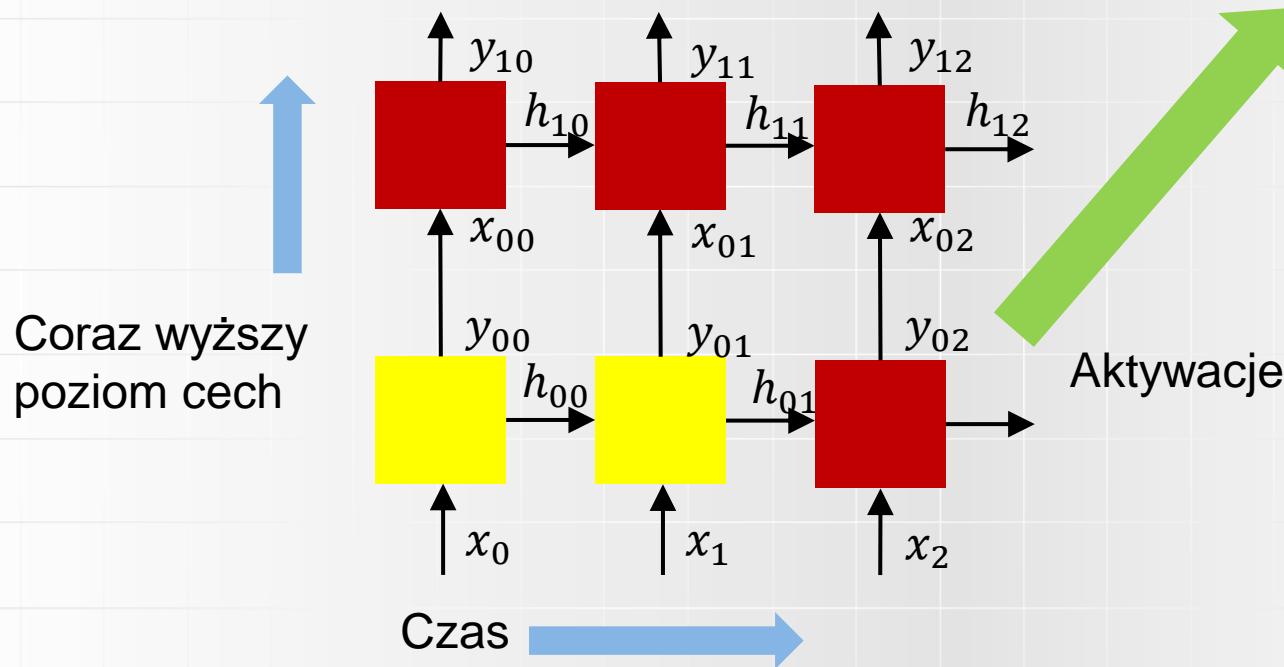


Jak możemy trenować taką sieć?

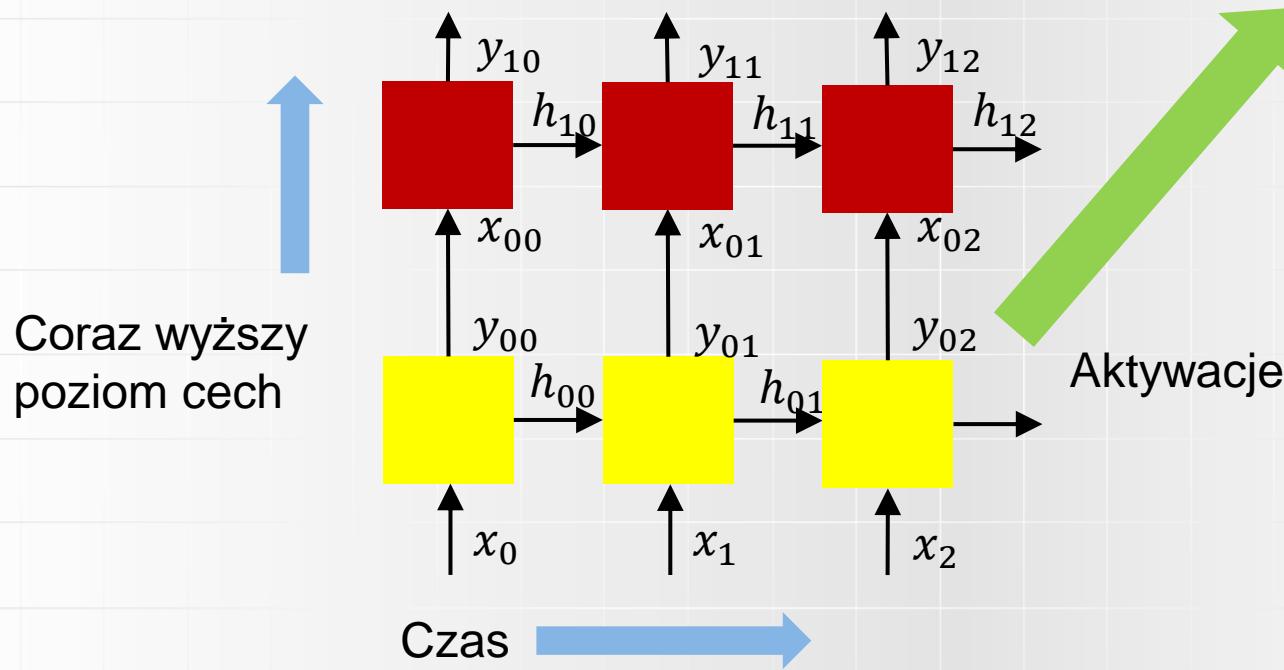
# Faza przesłania w przód



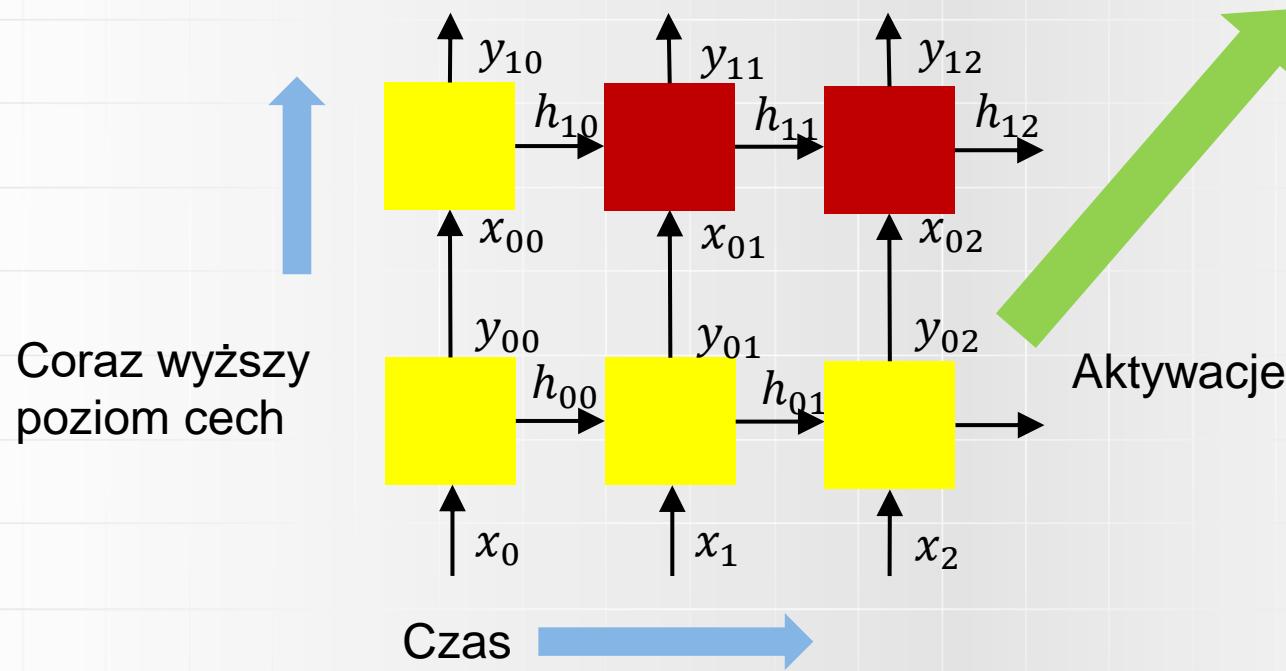
# Faza przesłania w przód



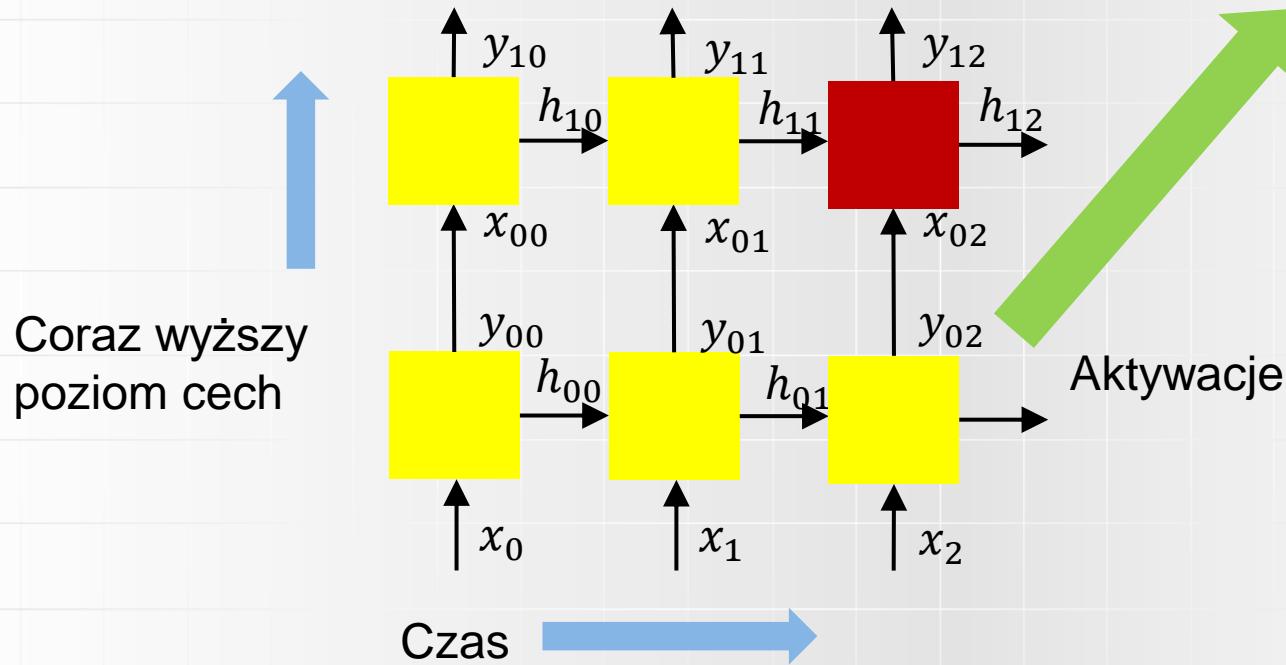
# Faza przesłania w przód



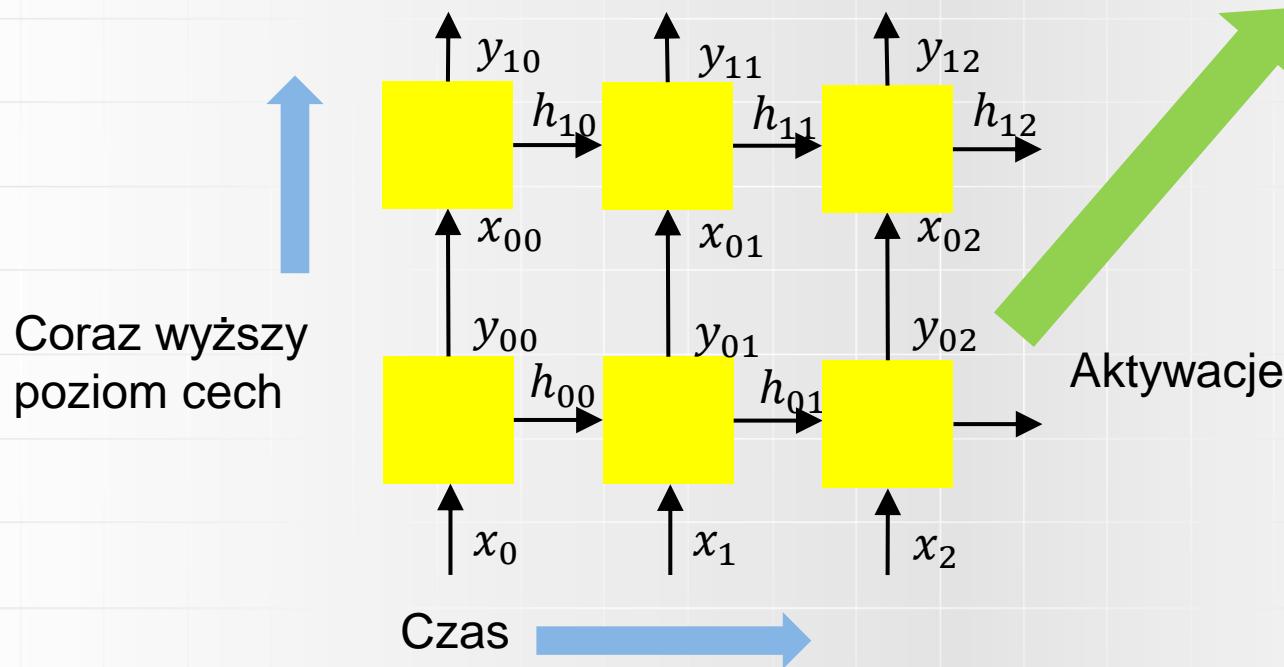
# Faza przesłania w przód



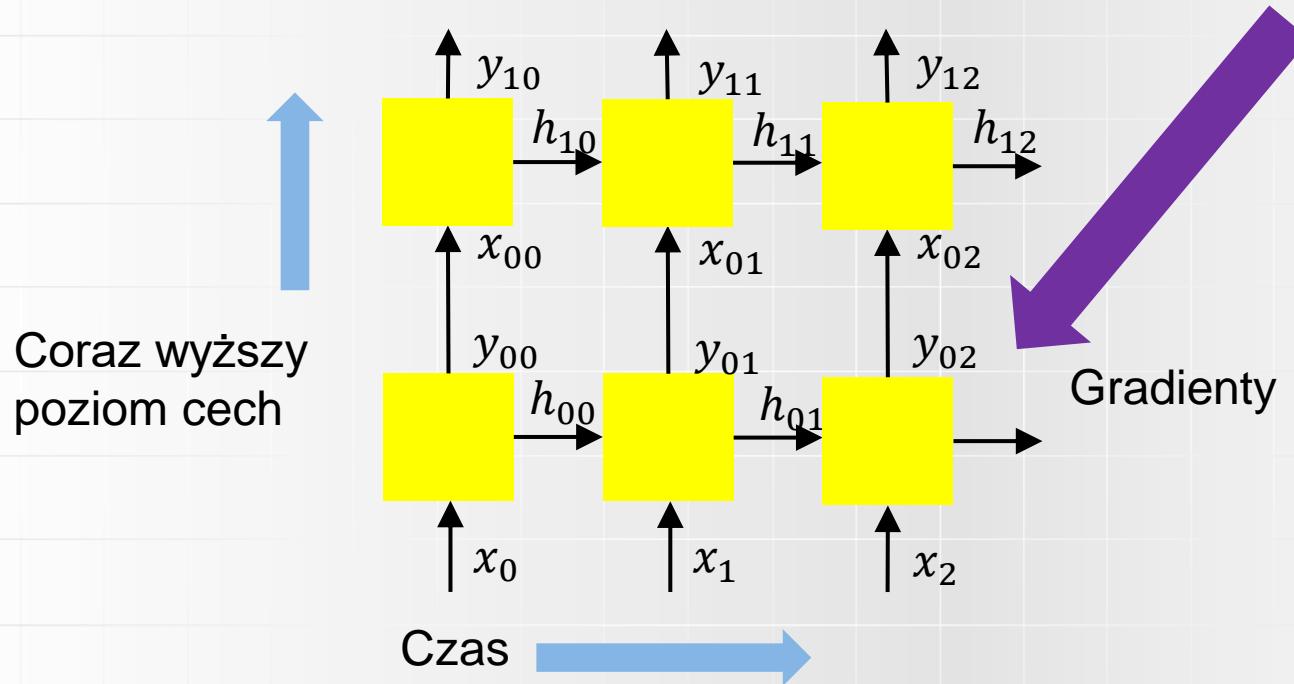
# Faza przesłania w przód



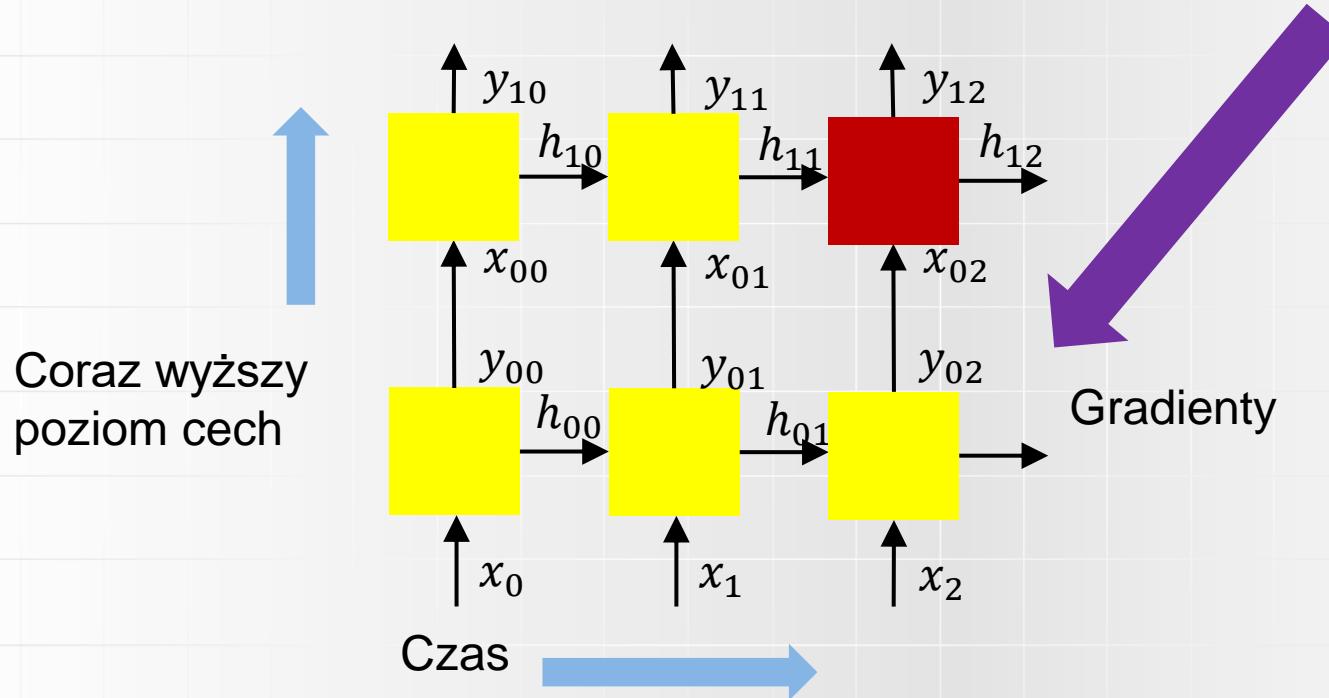
# Faza przesłania w przód



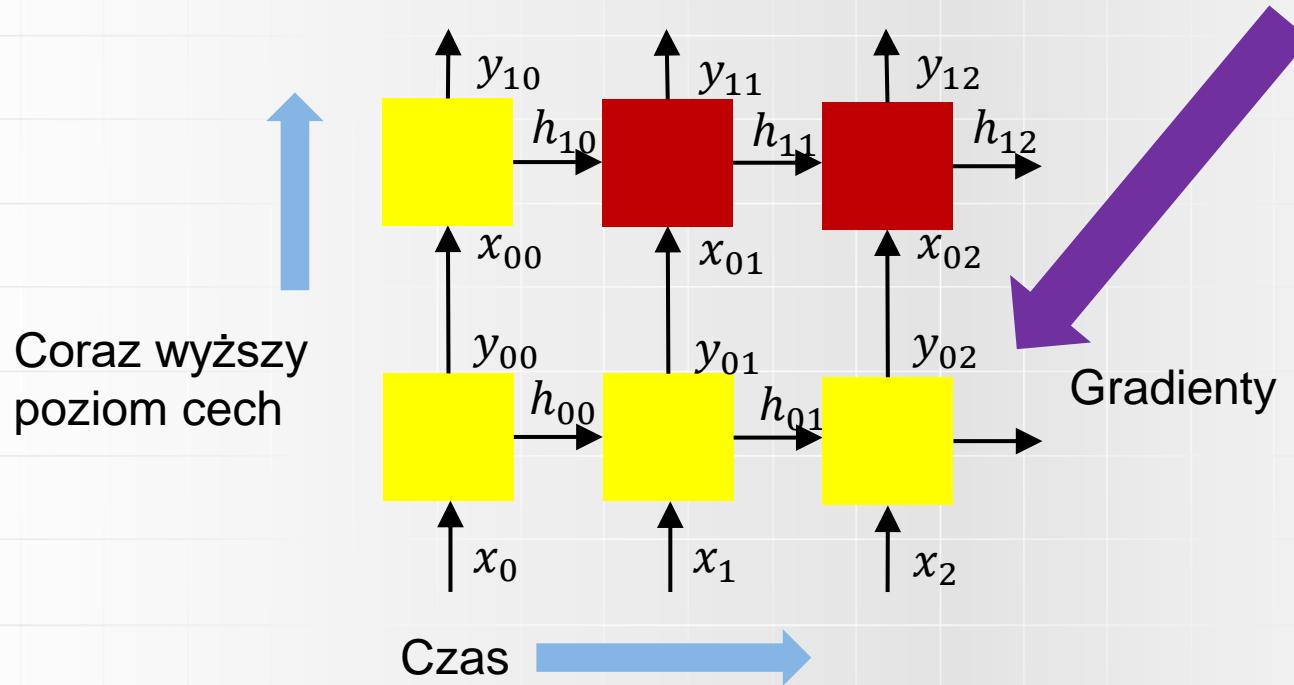
# Faza rzutowania gradientów wstecz



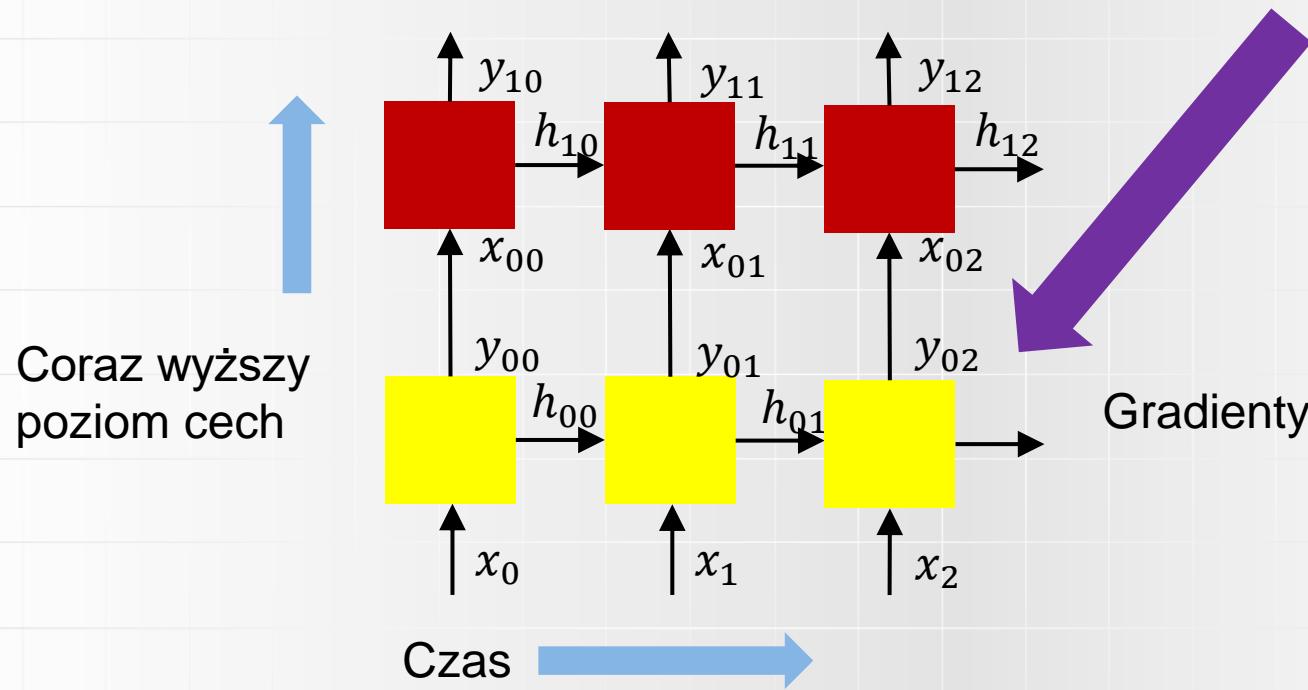
# Faza rzutowania gradientów wstecz



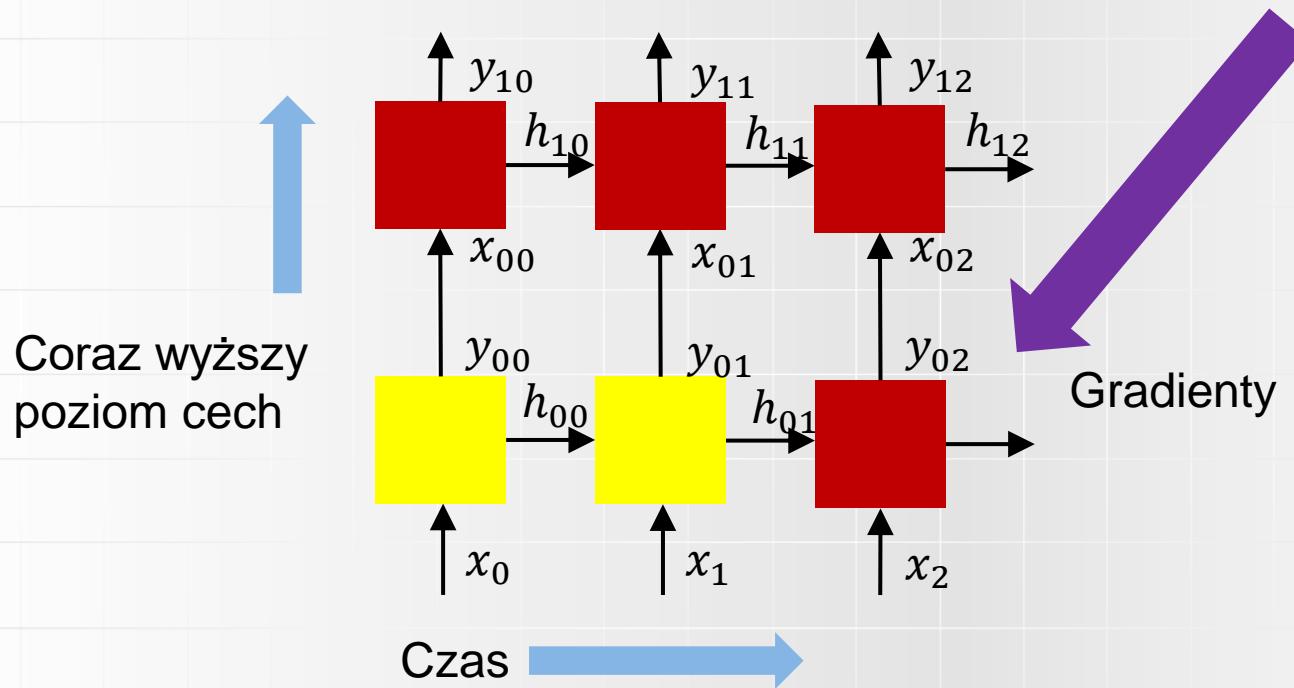
# Faza rzutowania gradientów wstecz



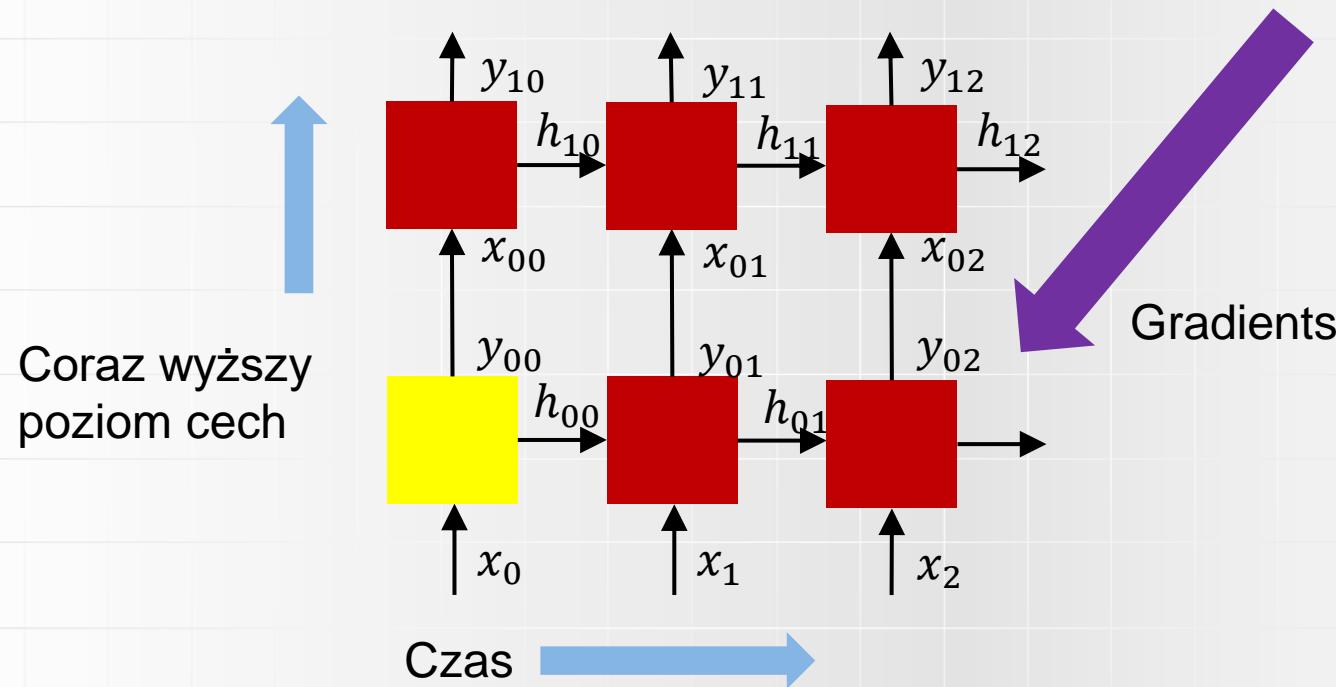
# Faza rzutowania gradientów wstecz



# Faza rzutowania gradientów wstecz

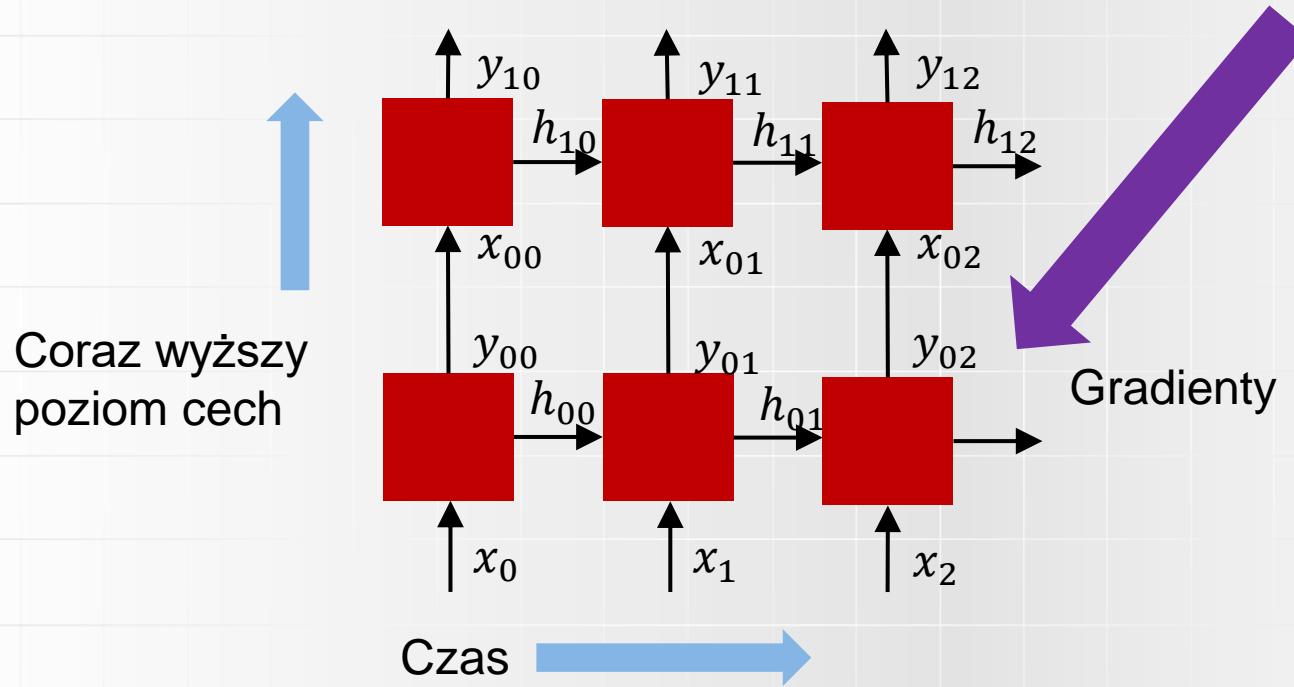


# Backward phase in BP



Source: CS294-129:

# Struktura sieci



Jeden cykl uczenia sieci BP.

Source: CS294-129:



# Uczenie – uwagi

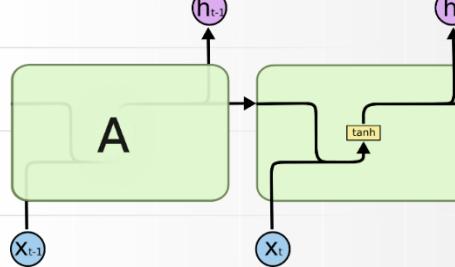
- Po każdej paczce wzorców (batchu), wagi są modyfikowane przy użyciu gradientów.
- Po takiej transformacji mamy zbiór połączeń odpowiadających danemu połączeniu z oryginalnej sieci.
- Gradient błędu na każdym połączeniu w rozwinięciu będzie inny. Dlatego sumuje się lub uśrednia gradient na odpowiadających sobie połączeniach i wszystkie one są modyfikowane o tę samą wartość.



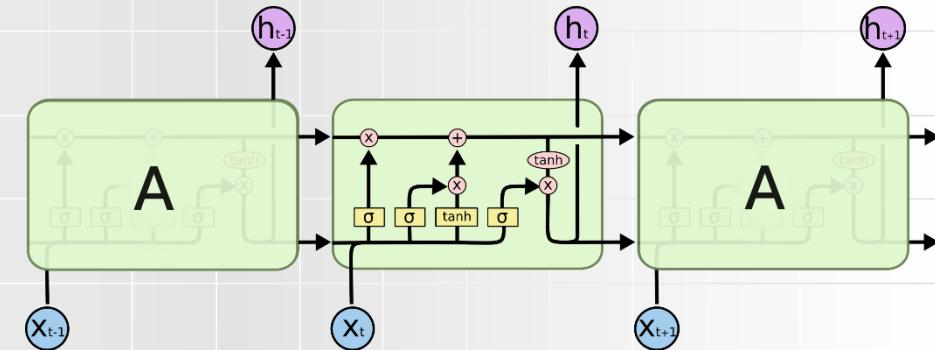
# Sieci rekurencyjne podsumowanie

- Pozwalają elastycznie zaprojektować architekturę
- Klasyczne RNN są proste, ale nieefektywne
- Często pojawia się eksplozja gradientu albo zanikający gradient
- Eksplodujący gradient jest łatwiejszy do rozwiązania (obcięcie wag - ang. truncated activation function)
- Zanikający gradient rozwiązywany przez nowe architektury: **LSTM** or **GRU**
- Posiadają wewnętrzne bramki, które zapobiegają znikowi gradientu.

# LSTM (Long Short Term Memory) a RNN

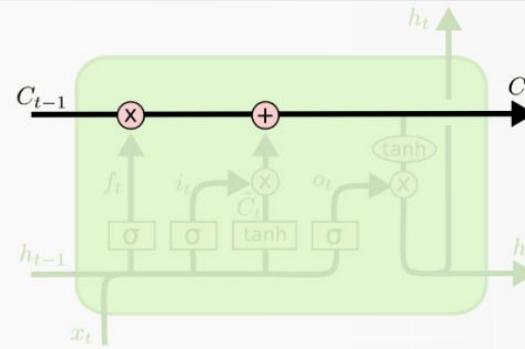


Klasyczne RNN

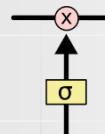


RNN z siecią LSTM

# LSTM - idea

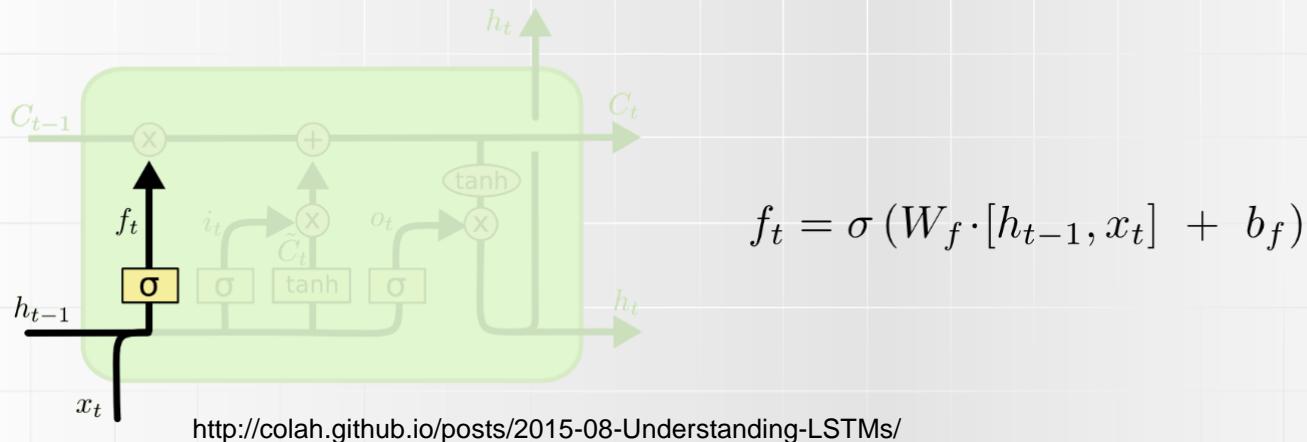


- Czarna pozioma linia odpowiada linii czasu.
- Stan komórki może być zmieniony przez aktywność bramki.
- Każda bramka jest implementowana jako warstwa neuronów sigmoidalnych która wykonuje iloczyn skalarny na dwóch wektorach i przekształca wynik przez funkcję sigmoidalną.
- Sieć ma 3 różne rodzaje bramek.



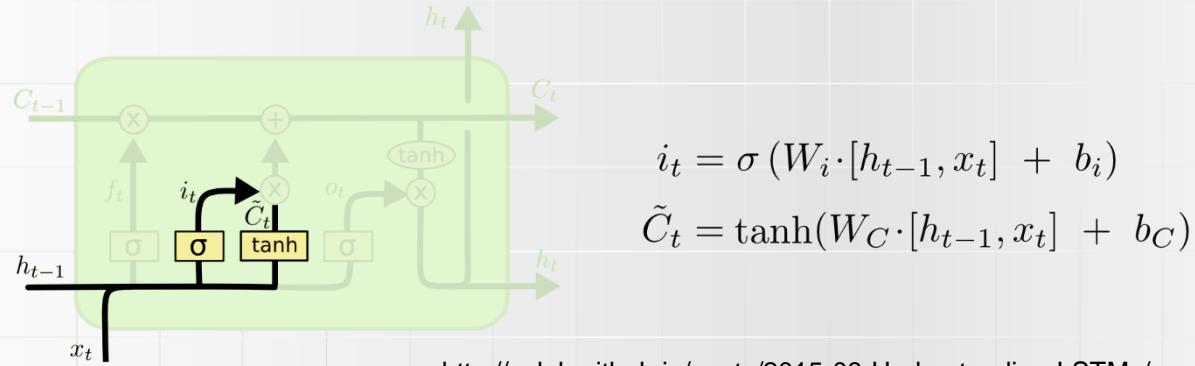
# LSTM – krok po kroku

- **Forget gate. Bramka zapominania.**
- Bierze  $h_{t-1}$  i produkuje wektor wartości z zakresu  $(0,1)$ , który mówi jak bardzo stan będzie zmieniony.



# LSTM- krok po kroku

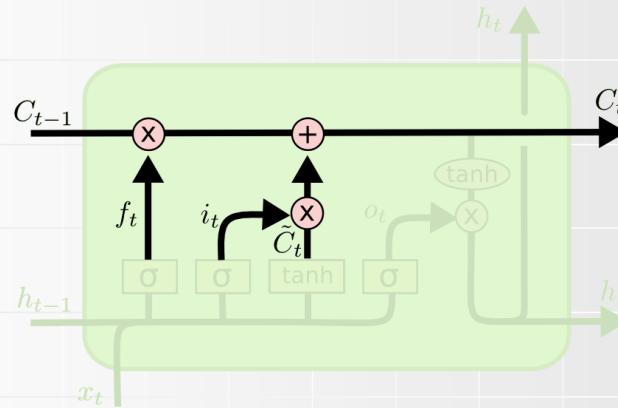
- **Write (input) gate.** Decyduje która wielkość będzie uaktualniana. Jest implementowana jako funkcja sigmoidalna.
- Warstwa *tahn* tworzy nowy wektor  $\tilde{C}_t$ . Jest to kandydat, że stan aktualny zmieni się o taką wartość.



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# LSTM – krok po kroku

- Uaktualnienie stanu komórki z  $C_{t-1}$  na  $C_t$



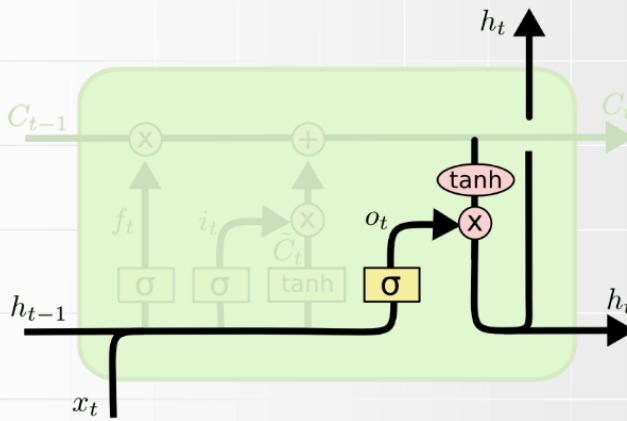
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

$f_t$  wyjście z *forget* – ile pozostaje z poprzedniego stanu  $i_t$  jeśli będzie modyfikowany przez kandydata  $\tilde{C}_t$  na nowy wektor stanu.

# LSTM – krok po kroku

- **Output gate** – obliczenie wyjścia z komórki



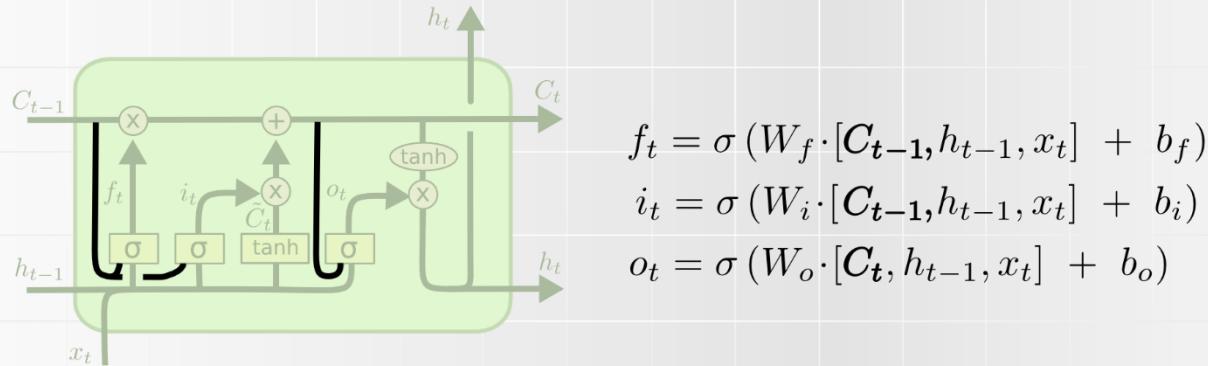
$$o_t = \sigma (W_o [ h_{t-1}, x_t ] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Warianty LSTM (1)

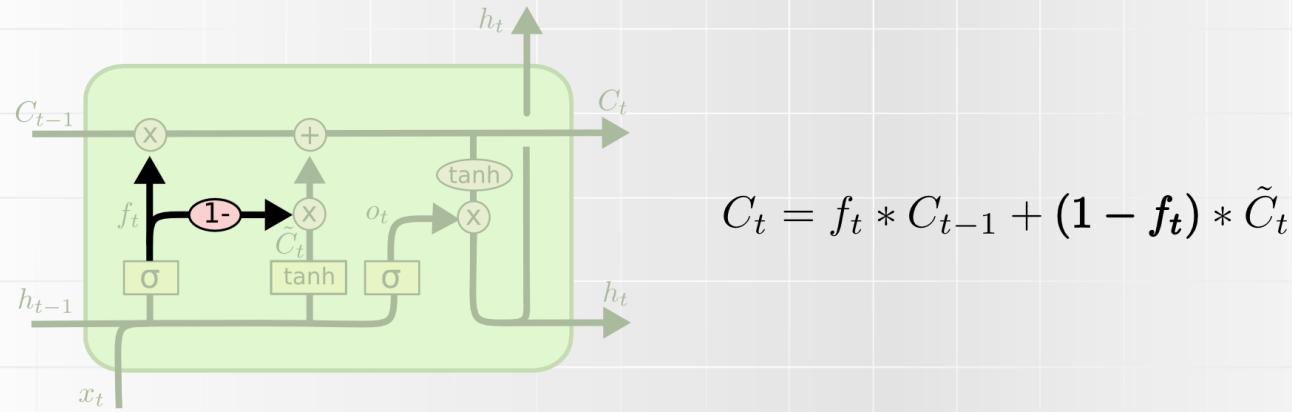
- Gers & Schmidhuber (2000), dodali “peephole connections.” Te bramki zależne są od stanu komórki.



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Warianty LSTM (2)

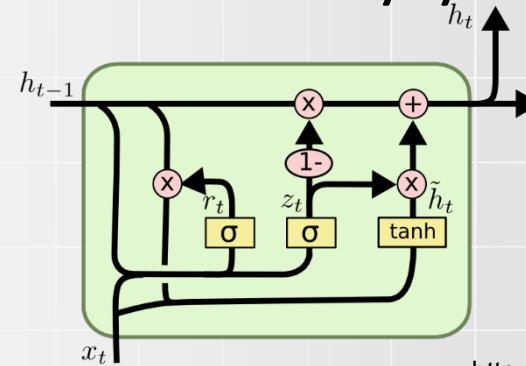
- Połączenie bramek „input” i „forget”



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Warianty LSTM (3)

- **Gated Recurrent Unit (GRU) Cho, et al. (2014)** - łączy bramki **forget** i **input** w jedną bramkę **update**. Łączy stan komórki ze stanem ukrytym czyniąc model prostszym.



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

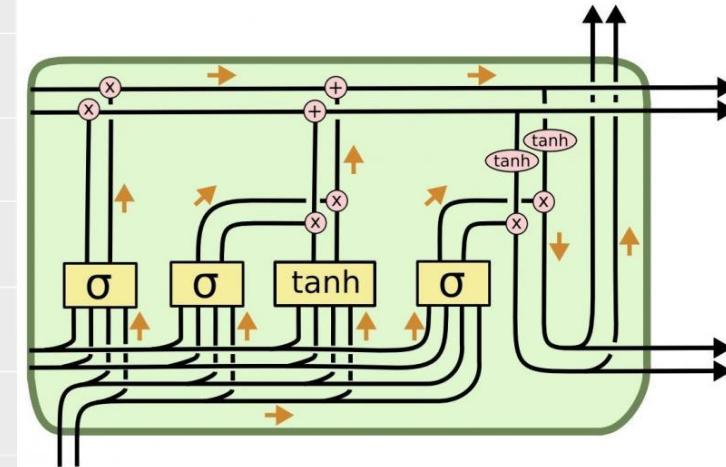
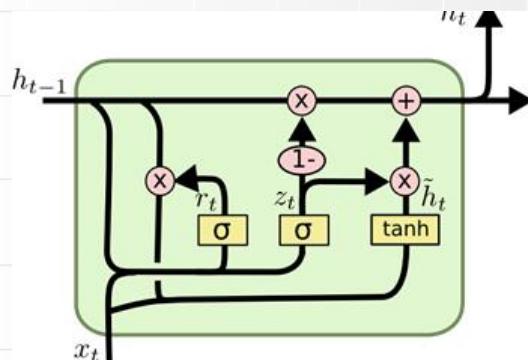
## LSTM vs GRU

### Główne różnice:

- 3 bramki vs 2 bramki w GRU (update i reset)
- Nie ma wyjściowej bramki, nie ma drugiej nieliniowości.
- Nie mają pamięci  $C_t$ , redukuje się ją do stanu ukrytego w GRU.
- GRU ma mniej parametrów, może się łatwiej wyuczać i lepiej generalizować.

# LSTM – obliczenia na wektorach

- Ważne – wszystkie grube linie na diagramie z lewej strony oznaczają przesłanie informacji w postaci wektorowej tak jak z prawej strony.



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>



# Klasyczna RNN a LSTM

RNN:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$h \in \mathbb{R}^n \quad W^l \quad [n \times 2n]$$

---

LSTM:

$$W^l \quad [4n \times 2n]$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

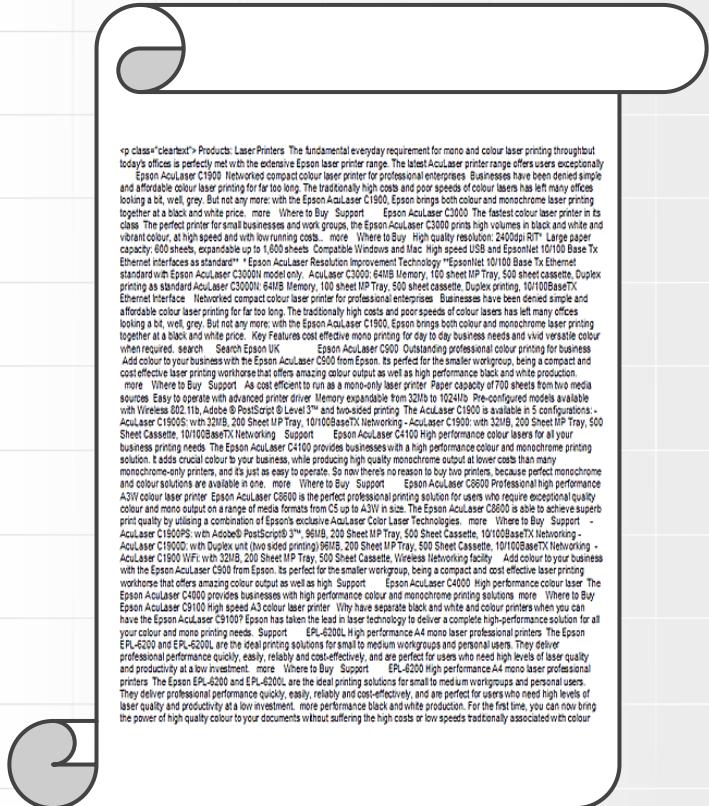


# Możliwe zastosowania

- Generowanie tekstu
- Translacja maszynowa (angielski --> francuski)
- Transkrypcja mowy na tekst
- Przewidywanie rynku
- Opis scen w wideo (w połączeniu z CNN)
- Sterowanie kołami w pojazdach (w połączeniu z CNN)



# Text generation



<p class="list-item-l1"> Product: Laser Printer. The Aculaser is every day required for mono and colour laser printing throughout today's office. It is perfect with the extensive range of paper sizes. The basic Aculaser is very cost effective and uses exceptionally fast print speeds. The Aculaser C1500 Networked compact colour laser printer for professional enterprises. Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit well, grey. But not any more with the Epson Aculaser C1500. Epson brings both colour and monochrome laser printing together at a black and white price. more... Where to Buy, Support... Epson Aculaser C3000 The fastest colour laser printer in its class. The perfect printer for small businesses and work groups, the Epson Aculaser C3000 prints high volumes in black and white and vibrant colour, at high speed and with low running costs... more... Where to Buy, High quality resolution: 2400dpi RTT. Large paper capacity: 600 sheets, expandable up to 1,800 sheets. Compatible Windows and Mac. High speed USB and EpsonNet 10/100 Base Tx Ethernet. Add colour to your workspace with the Epson Aculaser C3000N. Outstanding professional colour printing for business standard with Epson Aculaser C3000N model only. Aculaser C3000N: 64MB Memory, 100 sheet MFP Tray, 500 sheet cassette, Duplex printing as standard.Aculaser C3000N: 64MB Memory, 100 sheet MFP Tray, 500 sheet cassette, Duplex printing, 10/100BaseTX Ethernet Interface. Networked compact colour laser printer for professional enterprises. Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit well, grey. But not any more with the Epson Aculaser C1500. Epson brings both colour and monochrome laser printing together at a black and white price. Key Features cost effective mono printing for day to day business needs and vivid versatile colour when required. search... Search Epson UK... Epson Aculaser C900 Outstanding professional colour printing for business. Add colour to your workspace with the Epson Aculaser C900. Outstanding professional colour printing for business. Compact and cost effective colour printing workspace that offers amazing colour output as well as high performance black and white production. more... Where to Buy, Support... As cost efficient to run as a mono-only laser printer. Paper capacity of 770 sheets from two media sources. Easy to operate with advanced printer driver. Memory expandable from 32MB to 1024MB. Pre-configured models available with Wireless 802.11b, Adobe® PostScript® 3™ and two-sided printing. The Aculaser C1900 is available in 5 configurations - Aculaser C1900S with 52MB, 200 Sheet MFP Tray, 10/100BaseTX Networking - Aculaser C1900 with 32MB, 200 Sheet MFP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - Aculaser C4100 High performance colour lasers for all your business needs. The Epson Aculaser C4100 provides businesses with a high performance colour and monochrome printing solution. A wide colour palette to your business which includes the latest in colour printing technology. Outstanding monochrome-only printers, and as you can see, it's just as easy to operate. So no more reason to buy two printers, because perfect monochrome and colour solutions are in one, and more... Where to Buy, Support... Epson Aculaser C9600 Professional high performance A3W colour laser printer. Epson Aculaser C9600 is the perfect professional printing solution for users who require exceptional quality colour and mono output on a range of media formats from C5 up to A3W in size. The Epson Aculaser C9600 is able to achieve superb print quality by utilising a combination of Epson's exclusive Aculaser Color Laser Technologies. more... Where to Buy, Support... Aculaser C1900P with Asperio PostScript 3™, 36MB, 200 Sheet MFP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - Aculaser C1900P with Duplex unit, 200 Sheet MFP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - Aculaser C4100P with Duplex unit, 200 Sheet MFP Tray, 500 Sheet Cassette, 10/100BaseTX Networking. Outstanding professional business with the Epson Aculaser C900 from Epson. It's perfect for the smaller workgroups, being a compact and cost effective laser printing workspace that offers amazing colour output as well as high. Support... Epson Aculaser C4000 High performance colour laser. The Epson Aculaser C9100 high speed A3 colour laser printer. Why have separate black and white and colour printers when you can have the Epson Aculaser C9100? Epson has taken the lead in laser technology to deliver a complete high-performance solution for all your colour and mono printing needs. Support... EPL-6200L High performance A4 mono laser professional printer. The Epson EPL-6200 and EPL-6200L are designed specifically for small to medium workgroups and personal users. They deliver professional performance, ease of use, reliability and cost-effectiveness, and are perfect for users who need high levels of quality and productivity at a low investment. more... Where to Buy, Support... EPL-6200 High performance A4 mono laser professional printers. The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more performance black and white production. For the first time, you can now bring the power of high quality colour to your documents without suffering the high costs or low speeds traditionally associated with colour.

## Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds  
Admit impediments. Love is not love  
Which alters when it alteration finds,  
Or bends with the remover to remove:  
O no! it is an ever-fixed mark  
That looks on tempests and is never shaken;  
It is the star to every wandering bark,  
Whose worth's unknown, although his height be taken.  
Love's not Time's fool, though rosy lips and cheeks  
Within his bending sickle's compass come:  
Love alters not with his brief hours and weeks,  
But bears it out even to the edge of doom.  
If this be error and upon me proved,  
I never writ, nor no man ever loved.

- Dane trenujące: wszystkie dzieła Shakespeare złączone w jeden plik (4.4MB).
- 3-warstwowa RNN (LSTM) z 512 ukrytymi węzłami w każdej warstwie.



# Wyniki poprawiane w trakcie trenowania

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e plia tkldrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng



"Tmont thithey" fomesscerliund  
Keushey. Thom here  
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwyl on aseterlome  
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."



Aftair fall unsuch that the hall for Prince Velzonski's that me of  
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort  
how, and Gogition is so overelical and ofter.



"Why do what that day," replied Natasha, and wishing to himself the fact the  
princess, Princess Mary was easier, fed in had oftened him.  
Pierre aking his soul came to the packs and drove up his father-in-law women.



# Przykład końcowego wyniku

## VIOLA:

Why, Salisbury must find his flesh and thought  
That which I am not aps, not a man and in fire,  
To show the reining of the raven and the wars  
To grace my hand reproach within, and not a fair are hand,  
That Caesar and my goodly father's world;  
When I was heaven of presence and our fleets,  
We spare with hours, but cut thy council I am great,  
Murdered and by thy master's ready there  
My power to give thee but so much as hell:  
Some service in the noble bondman here,  
Would show him to her wine.

## KING LEAR:

O, if you were a feeble sight, the courtesy of your law,  
Your sight and several breath, will wear the gods  
With his heads, and my hands are wonder'd at the deeds,  
So drop upon your lordship's head, and your opinion  
Shall be against your honour.



# Geometria algebraiczna (latex)

## The Stacks Project

[home](#) [about](#) [tags explained](#) [tag lookup](#) [browse](#) [search](#) [bibliography](#) [recent comments](#) [blog](#) [add slogans](#)

### Browse chapters

Part	Chapter	online	TeX source	view pdf
Preliminaries	1. Introduction	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	2. Conventions	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	3. Set Theory	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	4. Categories	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	5. Topology	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	6. Sheaves on Spaces	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	7. Sites and Sheaves	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	8. Stacks	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	9. Fields	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>
	10. Commutative Algebra	<a href="#">online</a>	<a href="#">tex</a>	<a href="#">pdf</a>

### Parts

- [Preliminaries](#)
- [Schemes](#)
- [Topics in Scheme Theory](#)
- [Algebraic Spaces](#)
- [Topics in Geometry](#)
- [Deformation Theory](#)
- [Algebraic Stacks](#)
- [Miscellany](#)

### Statistics

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

- Plik ze źródłami w Latex (a 16MB file) służył do trenowania wielowarstwowego LSTM.



# Zadziwiające wyniki

For  $\bigoplus_{n=1,\dots,m}$  where  $\mathcal{L}_{m,\bullet} = 0$ , hence we can find a closed subset  $\mathcal{H}$  in  $\mathcal{H}$  and any sets  $\mathcal{F}$  on  $X$ ,  $U$  is a closed immersion of  $S$ , then  $U \rightarrow T$  is a separated algebraic space.

*Proof.* Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by  $\coprod Z \times_U U \rightarrow V$ . Consider the maps  $M$  along the set of points  $\text{Sch}_{fppf}$  and  $U \rightarrow U$  is the fibre category of  $S$  in  $U$  in Section, ?? and the fact that any  $U$  affine, see Morphisms, Lemma ???. Hence we obtain a scheme  $S$  and any open subset  $W \subset U$  in  $\text{Sh}(G)$  such that  $\text{Spec}(R') \rightarrow S$  is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that  $f_i$  is of finite presentation over  $S$ . We claim that  $\mathcal{O}_{X,x}$  is a scheme where  $x, x', s'' \in S'$  such that  $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}'_{X',x'}$  is separated. By Algebra, Lemma ?? we can define a map of complexes  $\text{GL}_{S'}(x'/S'')$  and we win.  $\square$

To prove study we see that  $\mathcal{F}|_U$  is a covering of  $\mathcal{X}'$ , and  $\mathcal{T}_i$  is an object of  $\mathcal{F}_{X/S}$  for  $i > 0$  and  $\mathcal{F}_p$  exists and let  $\mathcal{F}_i$  be a presheaf of  $\mathcal{O}_X$ -modules on  $\mathcal{C}$  as a  $\mathcal{F}$ -module. In particular  $\mathcal{F} = U/\mathcal{F}$  we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\text{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)_{fppf}^{\text{opp}}, (\text{Sch}/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longrightarrow (U, \text{Spec}(A))$$

is an open subset of  $X$ . Thus  $U$  is affine. This is a continuous map of  $X$  is the inverse, the groupoid scheme  $S$ .

*Proof.* See discussion of sheaves of sets.  $\square$

The result for prove any open covering follows from the less of Example ???. It may replace  $S$  by  $X_{\text{spaces},\text{étale}}$  which gives an open subspace of  $X$  and  $T$  equal to  $S_{\text{Zar}}$ , see Descent, Lemma ???. Namely, by Lemma ?? we see that  $R$  is geometrically regular over  $S$ .

**Lemma 0.1.** Assume (3) and (3) by the construction in the description.

Suppose  $X = \lim |X|$  (by the formal open covering  $X$  and a single map  $\underline{\text{Proj}}_X(\mathcal{A}) = \text{Spec}(B)$  over  $U$  compatible with the complex

$$\text{Set}(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that  $\mathcal{Q} \rightarrow \mathcal{C}_{Z/X}$  is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If  $T$  is surjective we may assume that  $T$  is connected with residue fields of  $S$ . Moreover there exists a closed subspace  $Z \subset X$  of  $X$  where  $U$  in  $X'$  is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1)  $f$  is locally of finite type. Since  $S = \text{Spec}(R)$  and  $Y = \text{Spec}(R)$ .

*Proof.* This is form all sheaves of sheaves on  $X$ . But given a scheme  $U$  and a surjective étale morphism  $U \rightarrow X$ . Let  $U \cap U = \coprod_{i=1,\dots,n} U_i$  be the scheme  $X$  over  $S$  at the schemes  $X_i \rightarrow X$  and  $U = \lim_i X_i$ .  $\square$

The following lemma surjective restrocomposes of this implies that  $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\dots,0}$ .

**Lemma 0.2.** Let  $X$  be a locally Noetherian scheme over  $S$ ,  $E = \mathcal{F}_{X/S}$ . Set  $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$ . Since  $\mathcal{I}^n \subset \mathcal{I}^n$  are nonzero over  $i_0 \leq p$  is a subset of  $\mathcal{J}_{n,0} \circ \overline{A}_2$  works.

**Lemma 0.3.** In Situation ???. Hence we may assume  $q' = 0$ .

*Proof.* We will use the property we see that  $p$  is the next functor (??). On the other hand, by Lemma ?? we see that

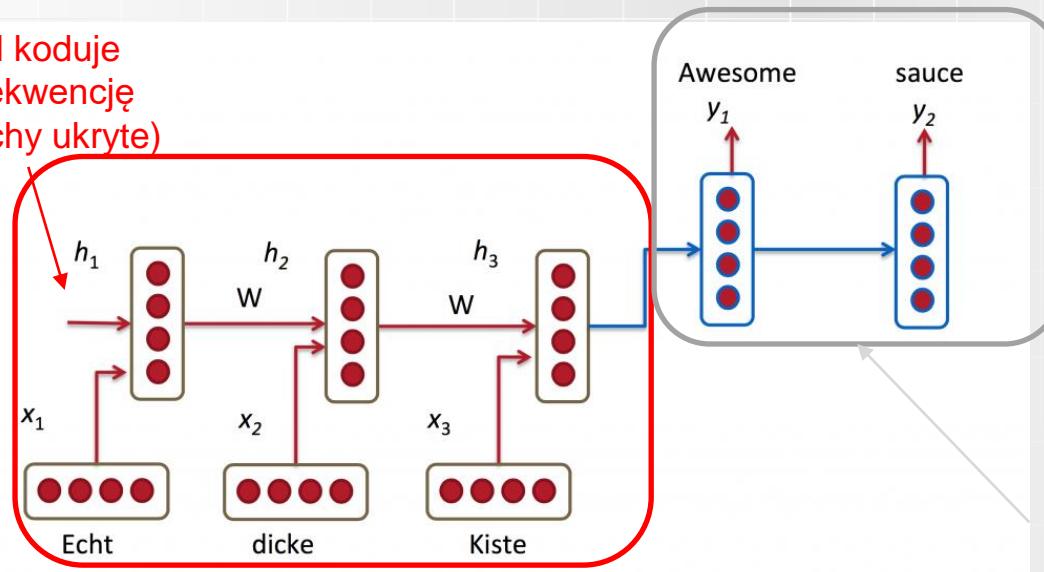
$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where  $K$  is an  $F$ -algebra where  $\delta_{n+1}$  is a scheme over  $S$ .  $\square$

# Maszynowa translacja

- Zdanie w języku źródłowym jest wprowadzane na wejście kodera (encodera).
- Koder znajduje swoją ukrytą reprezentację, która jest dostarczana do dekodera..

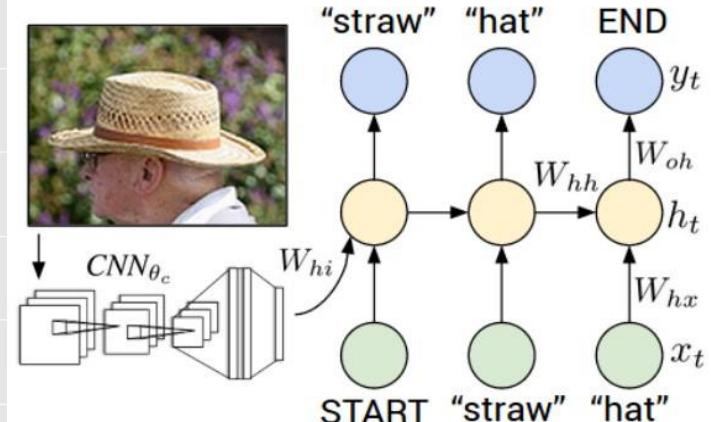
**Koder:** RNN koduje wejściową sekwencję (znajduje cechy ukryte)



**Dekoder:** RNN na podstawie reprezentacji ukrytej zdania wejściowego produkuje na wyjściu zdanie w języku docelowym.

Architektura **Encoder-decoder** do translacji maszynowej

# CNN i RNN – podpisywanie obrazów



- Artykuły z archive:
  - Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
  - Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
  - Show and Tell: A Neural Image Caption Generator, Vinyals et al.
  - Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
  - Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick



# Podpisywanie obrazów

Obraz  
testowy





# Podpisywanie obrazów



Obraz  
testowy



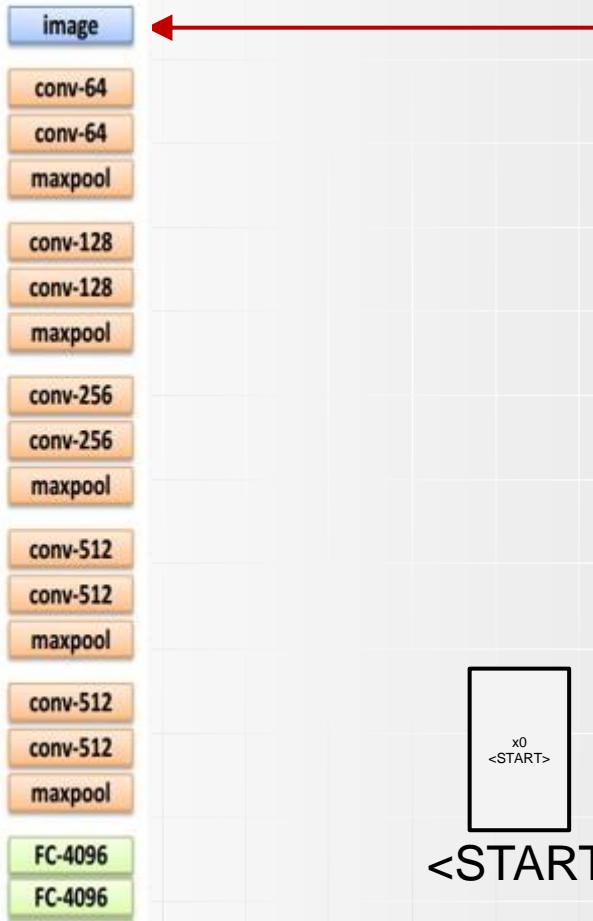
# Podpisywanie obrazów



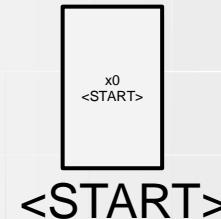
Obraz  
testowy



# Podpisywanie obrazów



Obraz  
testowy



# Podpisywanie obrazów

image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

V



Obraz  
testowy

y0

h0

x0  
<START>

$W_{ih}$

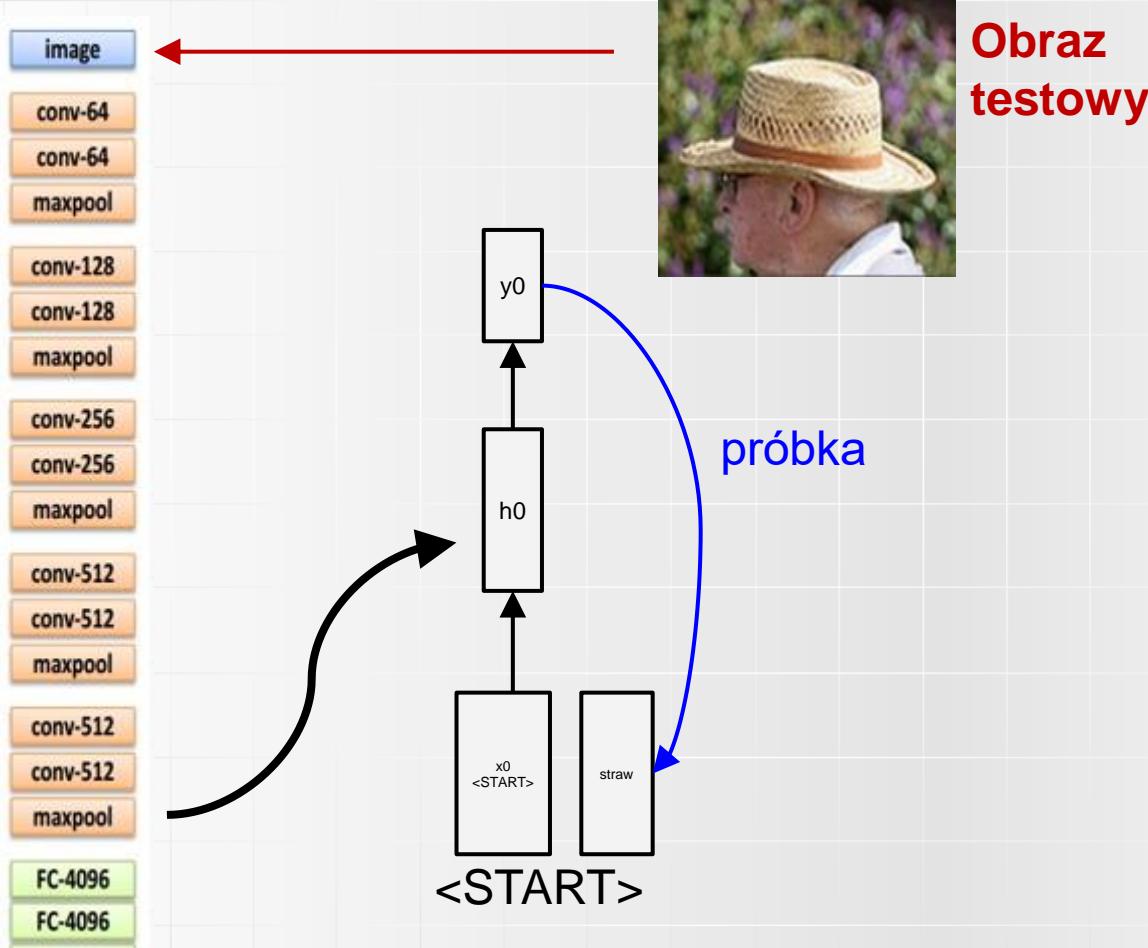
wcześniej:

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

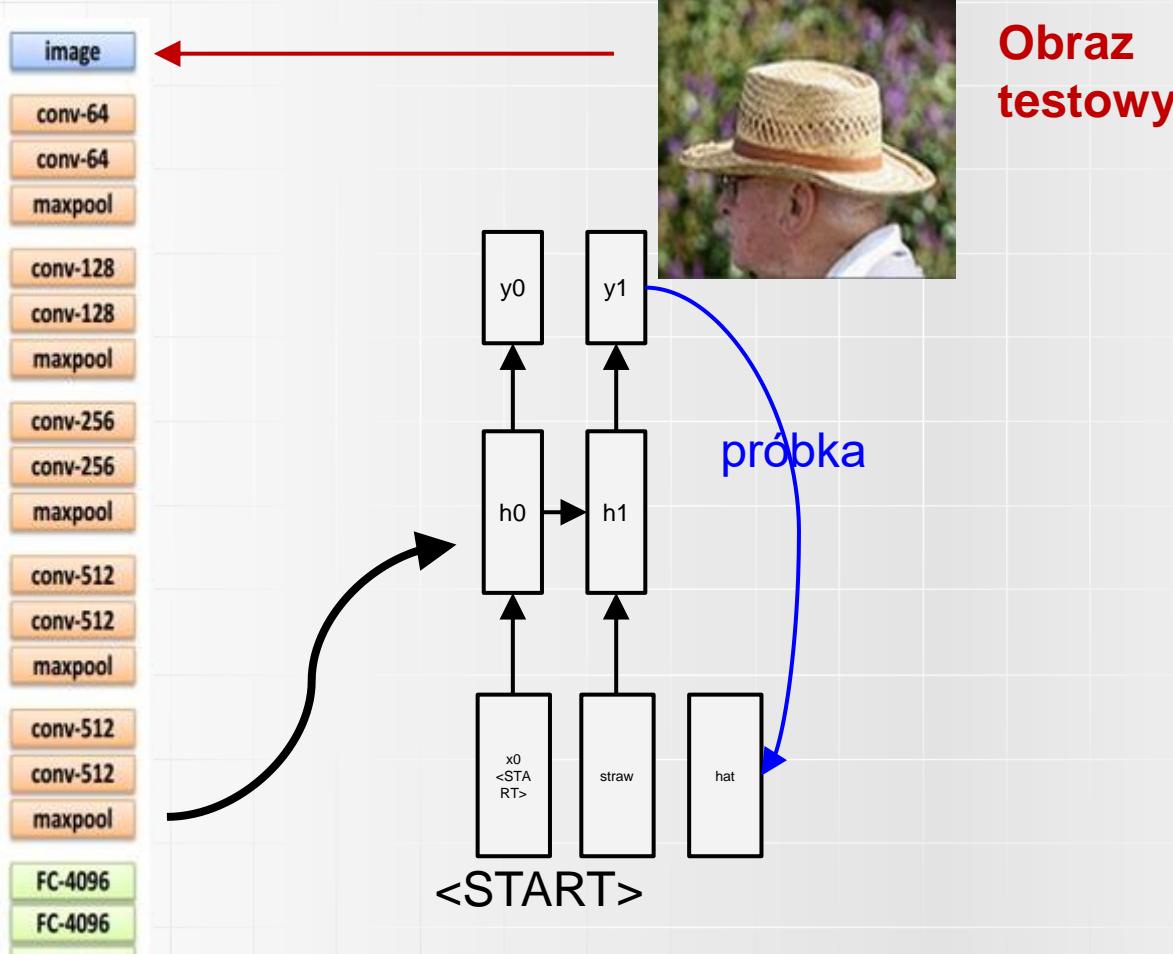
teraz:

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

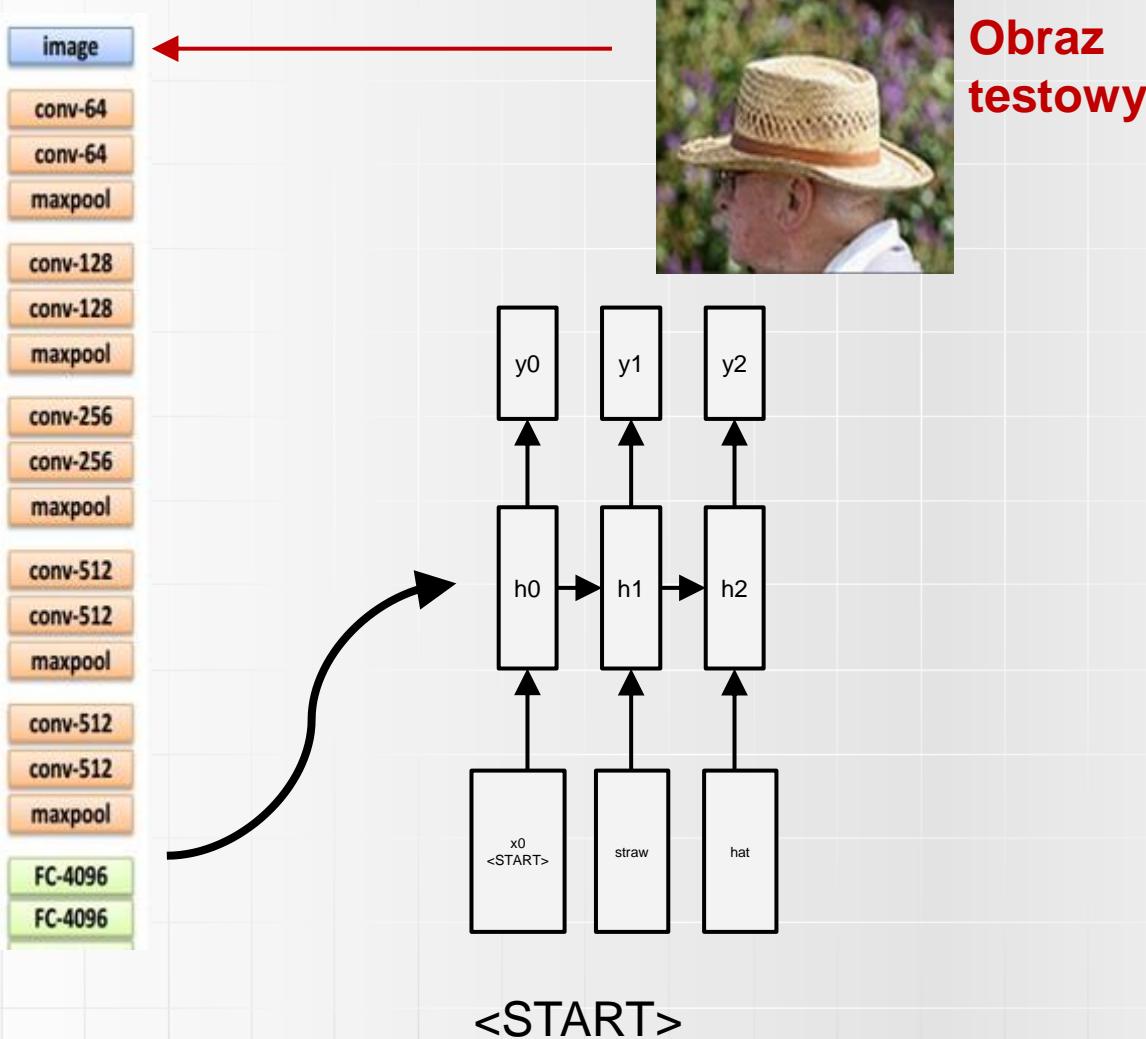
# Podpisywanie obrazów



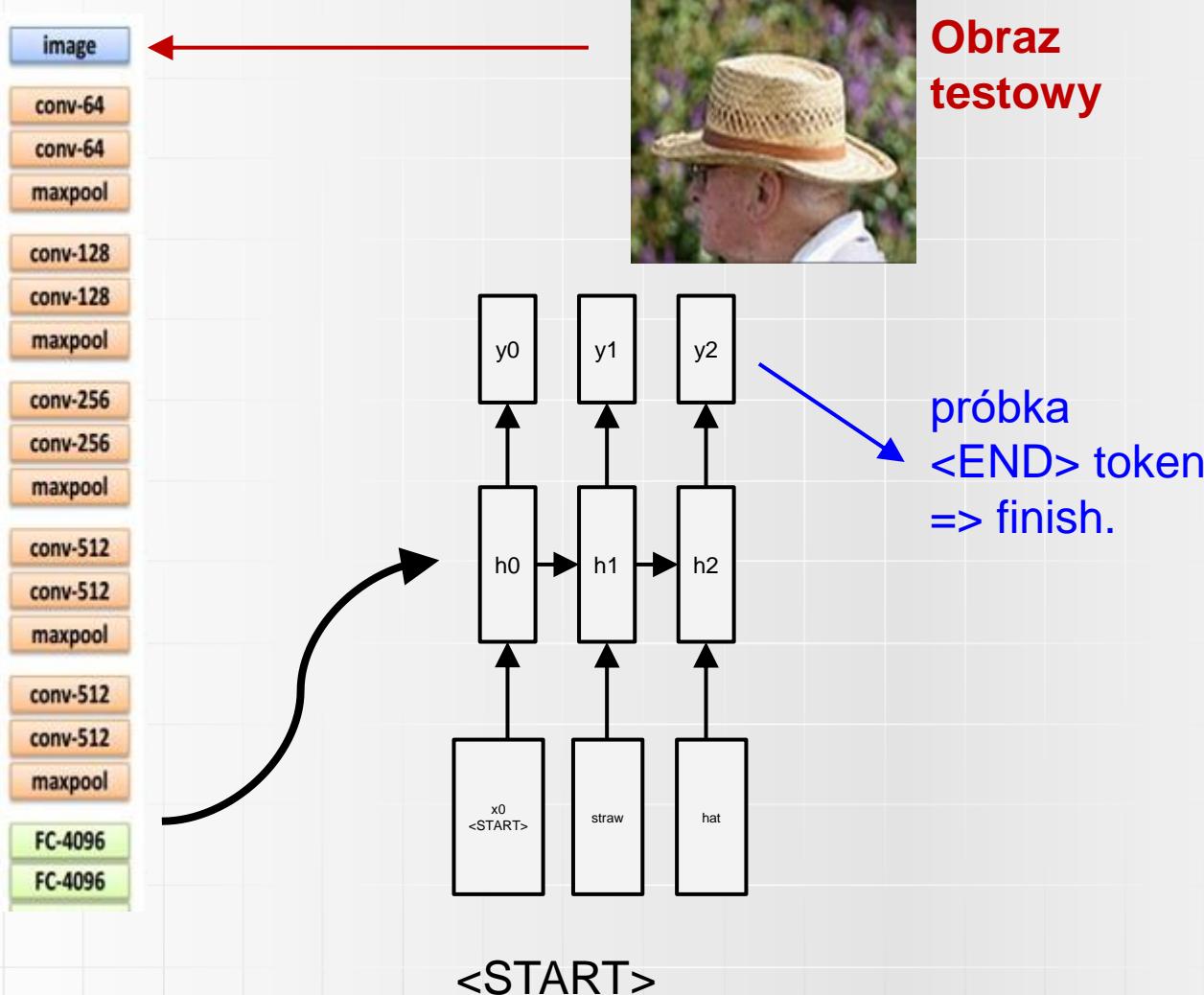
# Podpisywanie obrazów



# Podpisywanie obrazów



# Podpisywanie obrazów





# Podsumowanie

- RNN umożliwiają wprowadzenie relacji czasowych pomiędzy wzorcami.
- Lepsze rezultaty dla LSTM lub GRU, ponieważ rozwiązują problem zanikającego gradientu.
- Są uczone metodą Backpropagation Through Time (BPTT)
- Do podpisywania obrazów LSTM jest wykorzystywane łącznie z CNN.

model do podpisywania obrazów Microsoft's CaptionBot.  
demo: <https://www.captionbot.ai/>