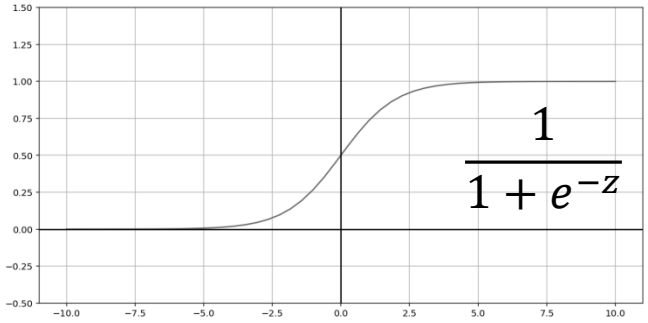
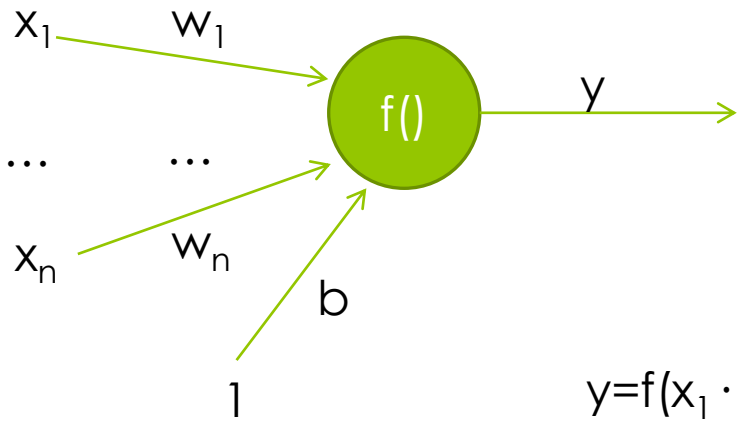
Sztuczne sieci neuronowe 2

dr inż. Piotr Szczuko

Perceptron



- Pojedynczy neuron



$y=f(x_1 \cdot w_1+\ldots+x_n \cdot w_n+b)$
$=f(b + \Sigma x_i w_i)$

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

$\mathbf{w}=[w_1 \ldots w_n, b]^T$

„*bias*" Wartość progowa (zwykle ujemna, ustala kiedy argument funkcji jest większy od zera)

Funkcje aktywacji

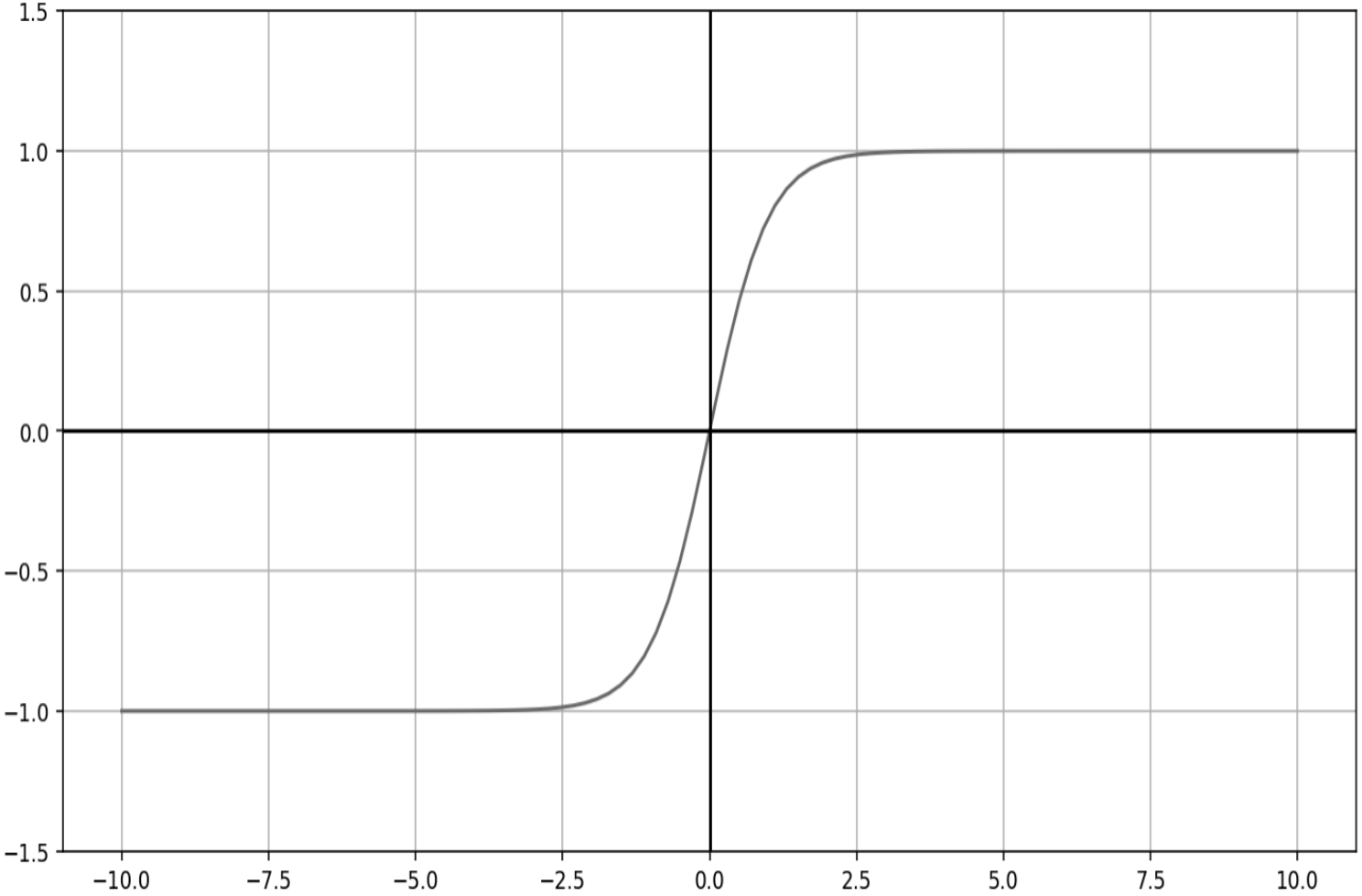
Hyperbolic Tangent Function

$$tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

$$tanh(0) = 0$$
$$tanh(\infty) = 1$$
$$tanh(-\infty) = -1$$

Zastosowania:

Różniczkowalna aproksymacja funkcji sign
Włącznik/Wyłącznik
Detekcja
Progowanie (ograniczenie wartości)


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

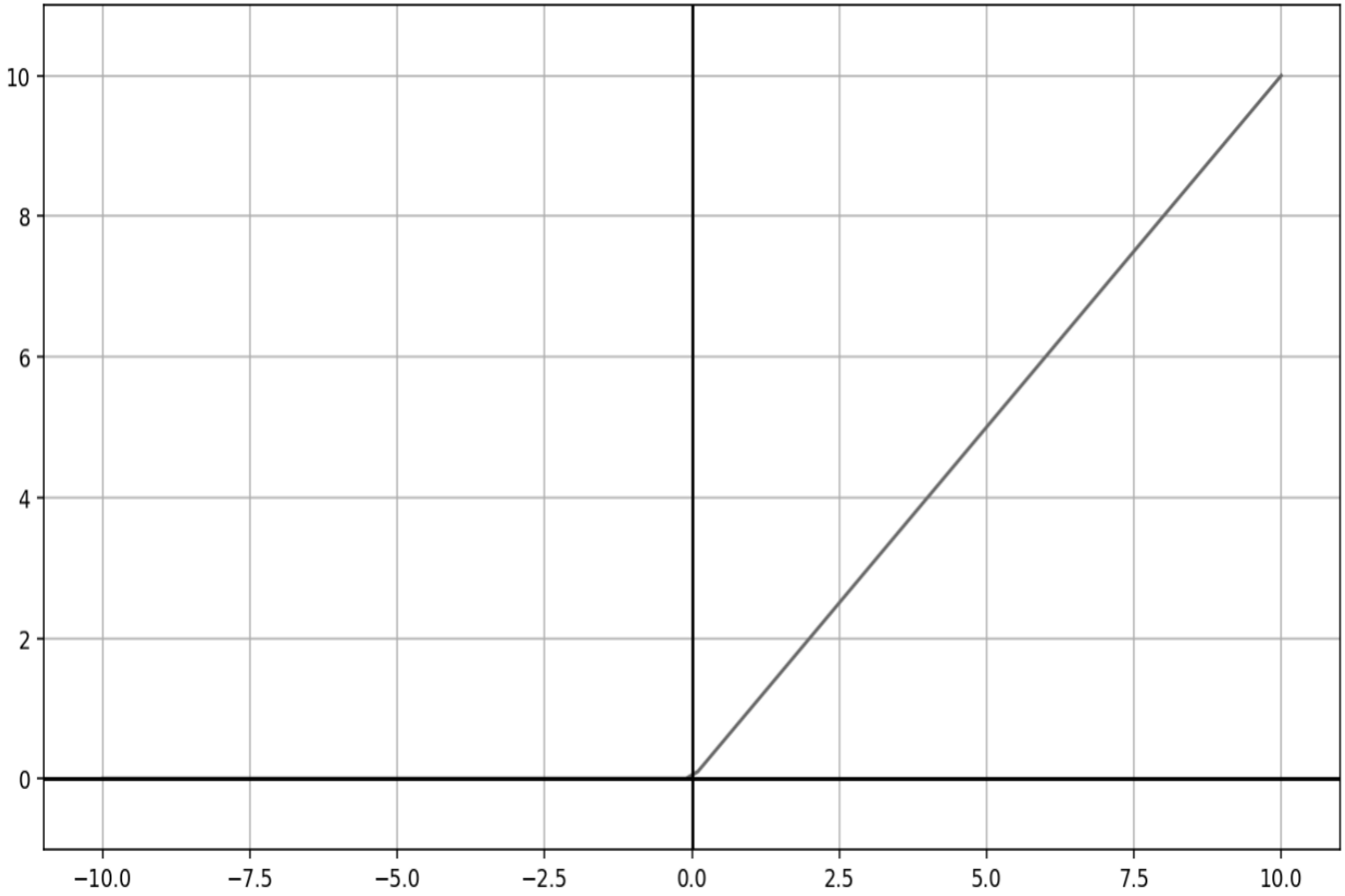
Rectified Linear Unit (ReLU)

$$ReLU(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$$
$$= \max(0, z)$$

$$ReLU(0) = 0$$
$$ReLU(z) = z$$
$$ReLU(-z) = 0$$

Zastosowania:

Zachowanie wartości bez zmian
Przetwarzanie wyników sumowania i wartości bezwzględnych
Dla średnich ważonych
Dla problemów liniowych


10
8
6
4
2
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

“Leaky” Rectified Linear Unit (ReLU)

$$LReLU(z) = \begin{cases} \alpha z, & z < 0 \\ z, & z \geq 0 \end{cases}$$
$$= \max(\alpha z, z) \qquad \text{for } (\alpha < 1)$$

$$LReLU(0) = 0$$
$$LReLU(z) = z$$
$$LReLU(-z) = -\alpha z$$

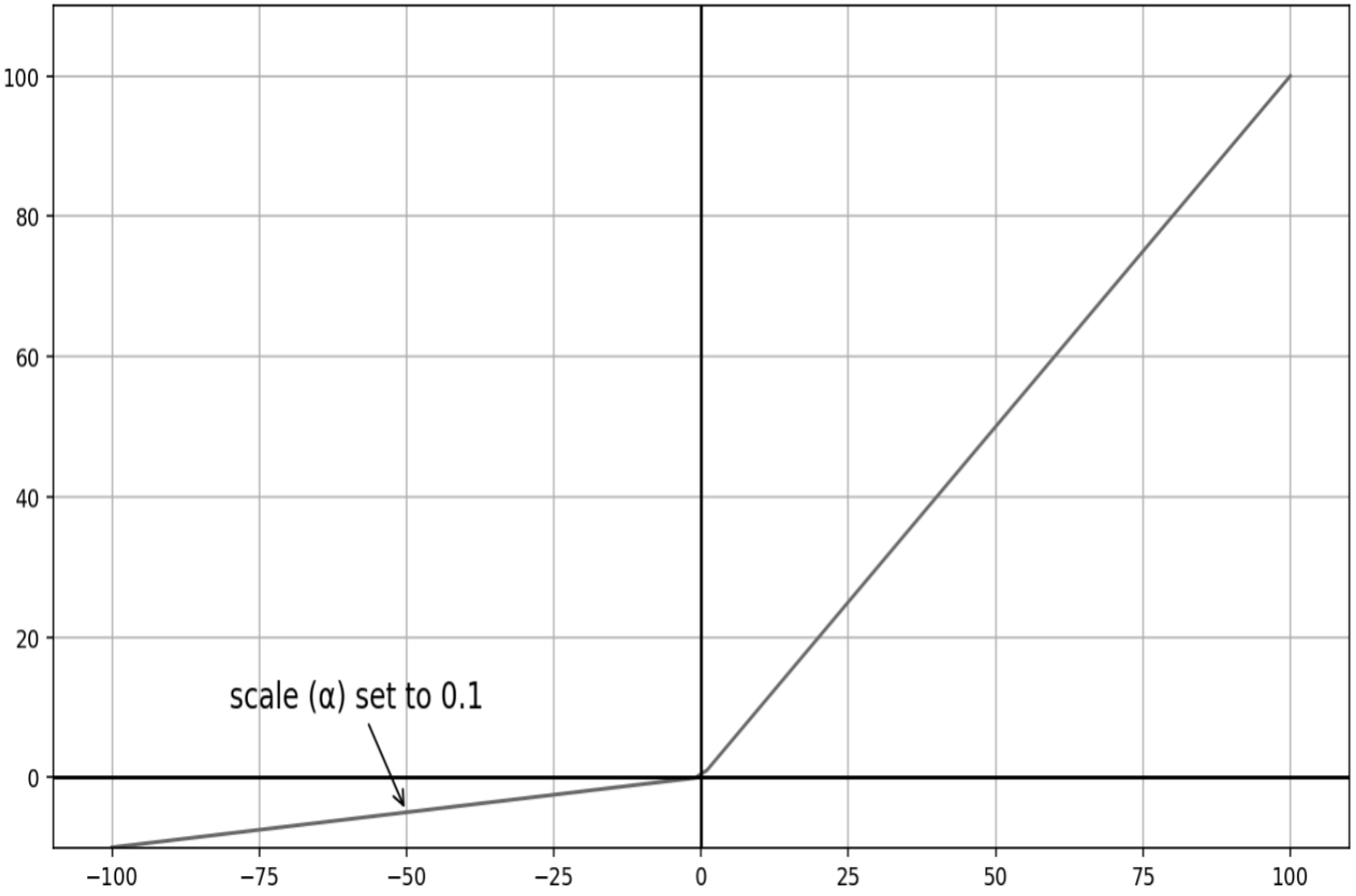
Zastosowania:

Analogiczne jak ReLU

+ niezerowa pochodna dla wartości ujemnych

Istotne ujemnych wartości „bias” w argumencie f.aktywacji:

$$y = f(x_1 \cdot w_1 + \ldots + x_n \cdot w_n + b)$$
$$= f(b + \Sigma x_i w_i)$$


100
80
60
40
20
0
scale (α) set to 0.1
−100
−75
−50
−25
0
25
50
75
100

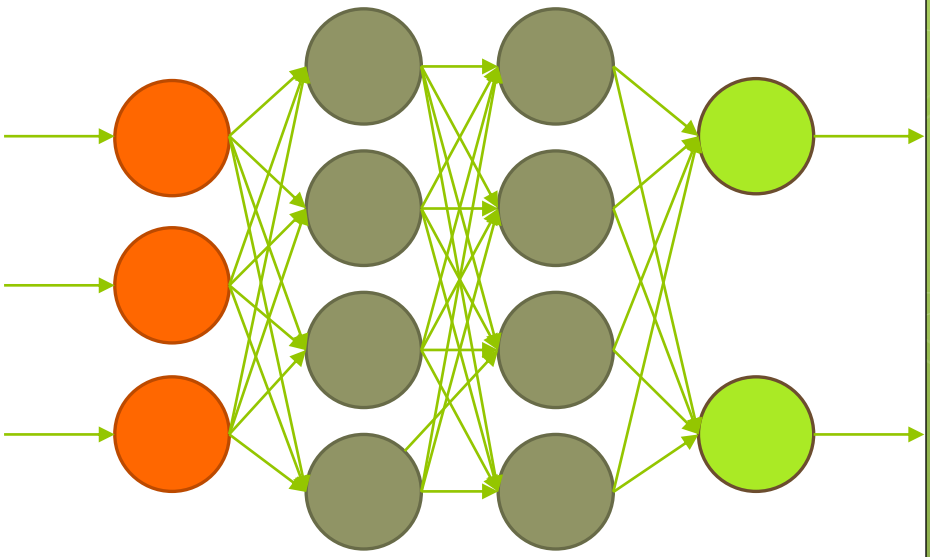
Jeden neuron – sieć neuronów?

- **Jeden neuron** – liniowa granica między klasami
- **Warstwy neuronów** – wyliczanie cech/atrybutów (*feature*) przydatnych dla kolejnych neuronów

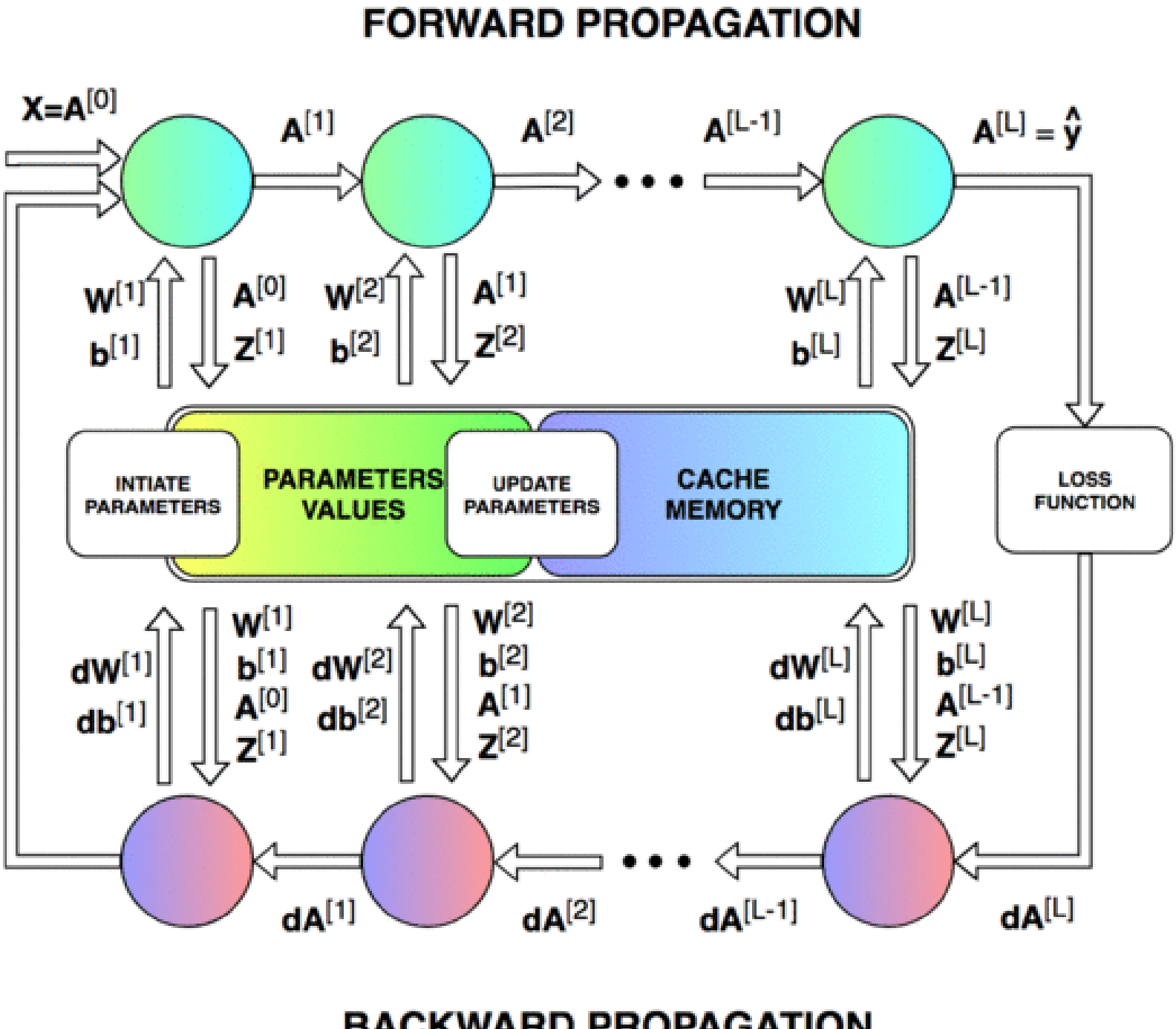
Warstwy:
- Wejściowa
- N ukrytych
- Wyjściowa

Macierze wag np. 4x2

Funkcje aktywacji każdego neuronu

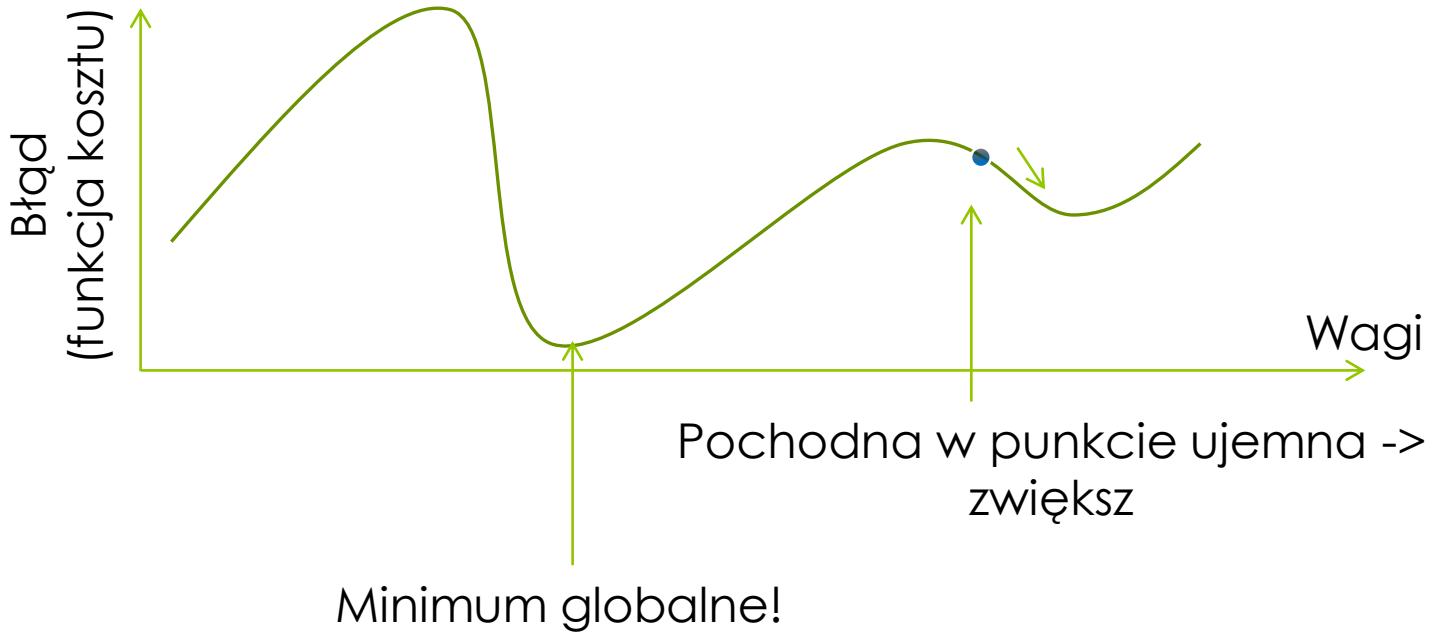


Trening sieci - Propagacja wsteczna

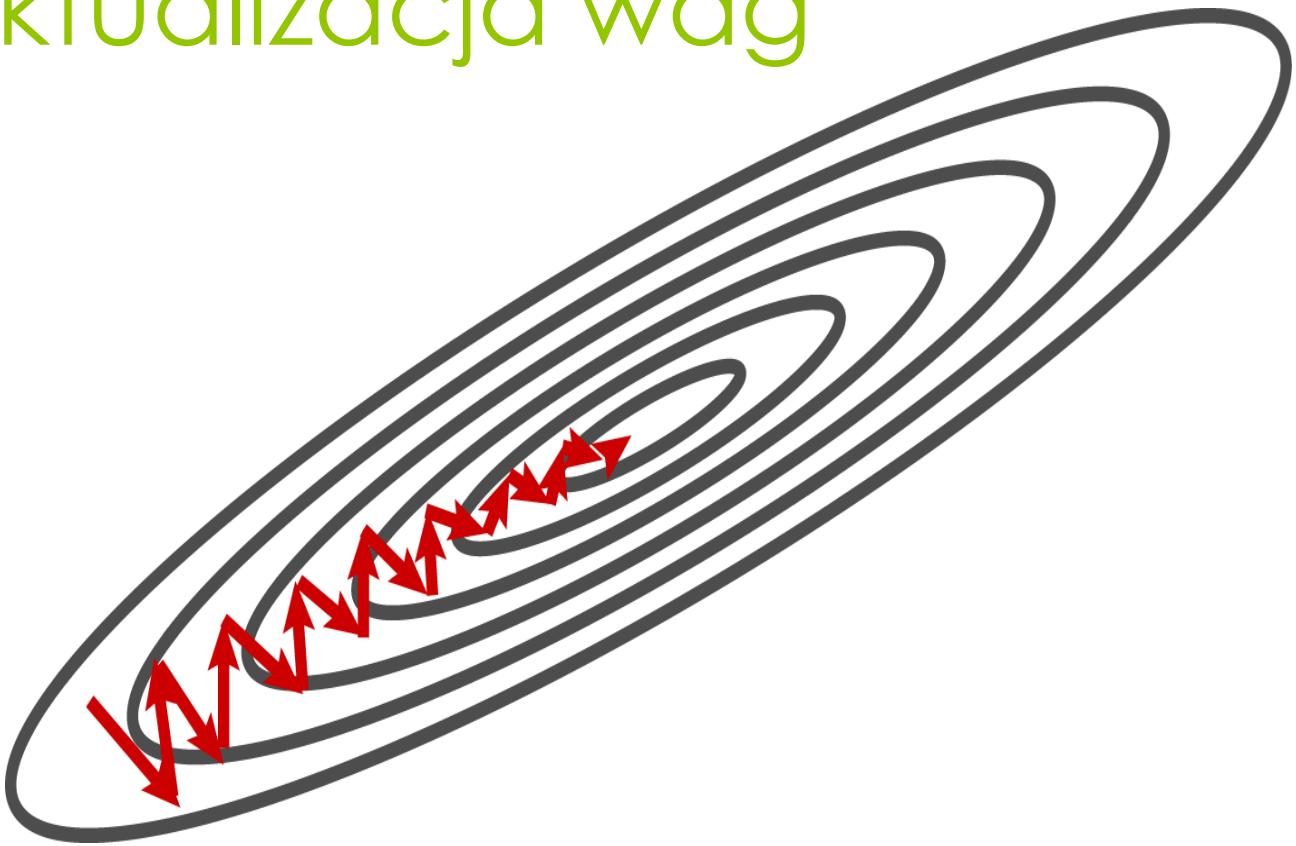

FORWARD PROPAGATION
X=A[0]
A[1]
A[2]
A[L-1]
A[L] = ŷ
W[1]
b[1]
A[0]
Z[1]
W[2]
b[2]
A[1]
Z[2]
W[L]
b[L]
A[L-1]
Z[L]
INTIATE PARAMETERS
PARAMETERS VALUES
UPDATE PARAMETERS
CACHE MEMORY
LOSS FUNCTION
dW[1]
db[1]
W[1]
b[1]
A[0]
Z[1]
dW[2]
db[2]
W[2]
b[2]
A[1]
Z[2]
dW[L]
db[L]
W[L]
b[L]
A[L-1]
Z[L]
dA[1]
dA[2]
dA[L-1]
dA[L]
BACKWARD PROPAGATION

Metody korekty wag

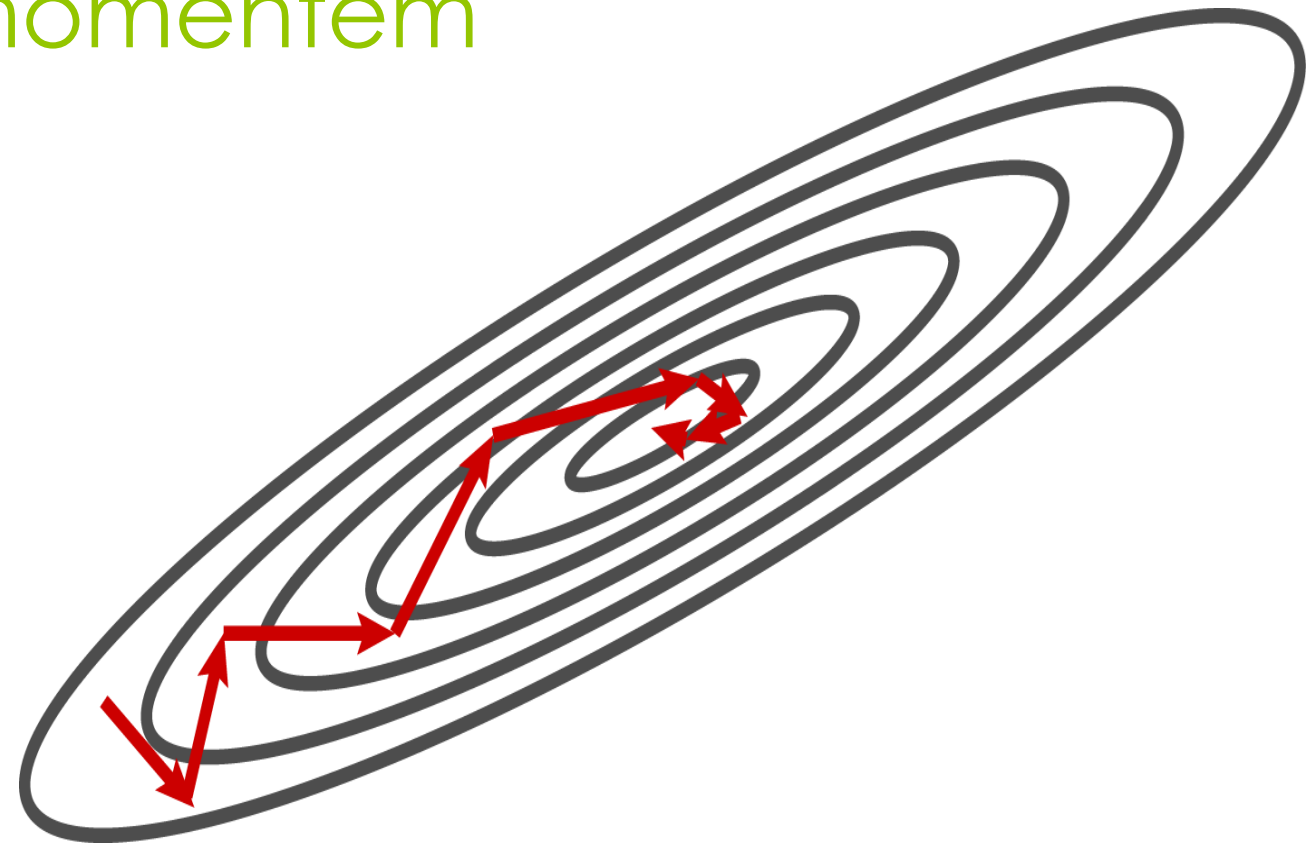
- Iteracyjne poprawki w stronę malejącego błędu



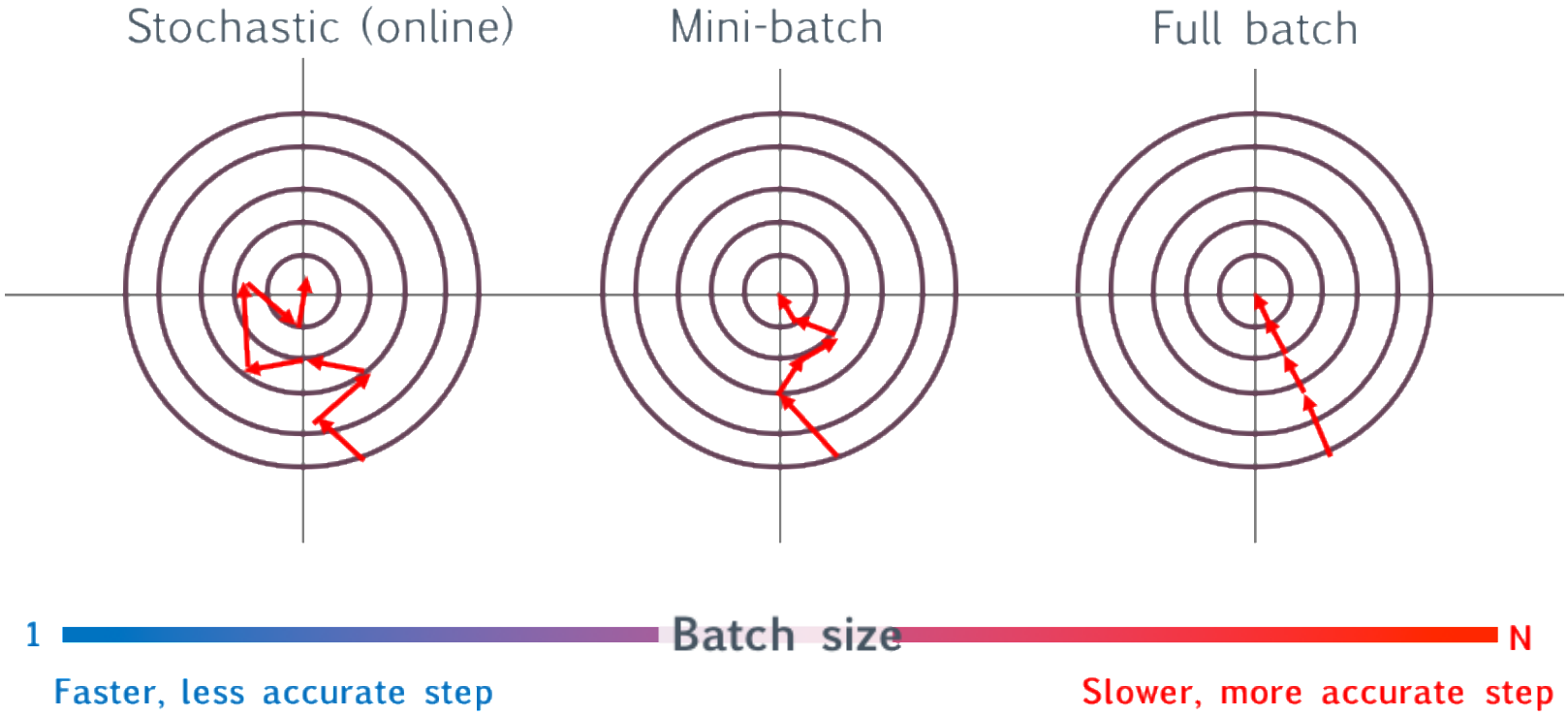
Aktualizacja wag



Aktualizacja wag z momentem

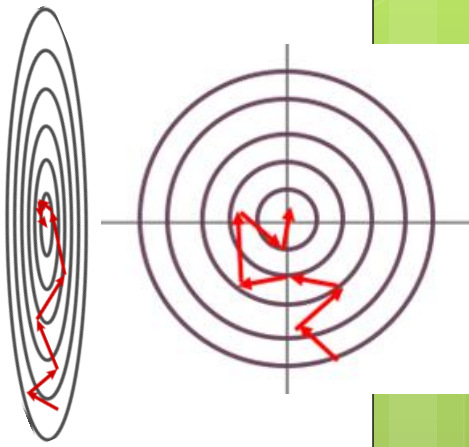


Aktualizacja wag - podsumowanie



- Źródło: INTEL AI Academy

Normalizacja danych wejściowych



- Przedział [0, 1]

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

- Przedział [-1, 1]

$$x_i = 2\left(\frac{x_i - \bar{x}}{x_{max} - x_{min}}\right) - 1$$

- Standaryzacja do rozkładu normalnego

$$x_i = \frac{x_i - \bar{x}}{\sigma}; \qquad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Metody optymalizacji momentu

- http://ruder.io/optimizing-gradient-descent/

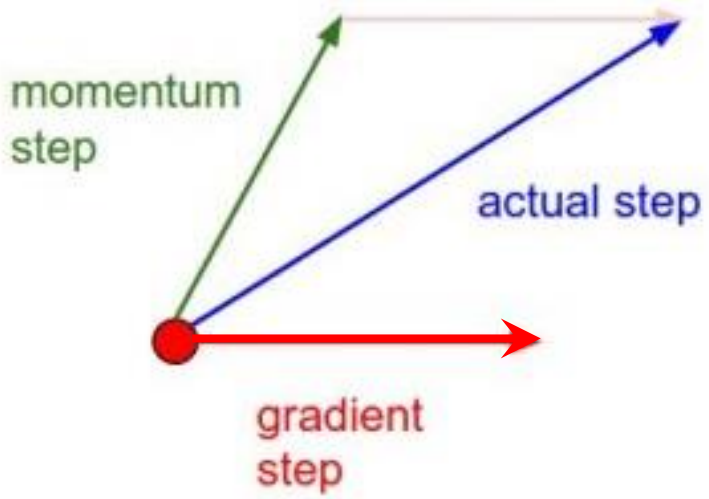
Moment Nesterova (Accelerated Nesterov Gradient)

- Przeszukiwanie wprzód, tj. gradient wyznaczany w punkcie „przyszłym”
- Poprawka kieruje wagi w kierunku tego spadku błędu

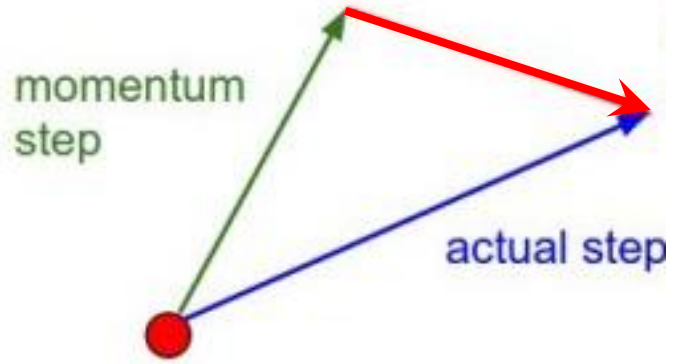
$$v_t = \eta \cdot v_{t-1} - \alpha \cdot \nabla(J - \eta \cdot v_{t-1})$$
$$\mathrm{W} := W - v_t$$

Moment Nesterova (Accelerated Nesterov Gradient)

Klasyczne podejście:



Nesterov: liczenie gradientu w przyszłym punkcie



Wektor momentu

Gradient

Gradient Nesterova

Wynikowa korekta

AdaGrad
(metoda adaptacyjna)

- Skalowanie poprawki dla **każdej wagi osobno**
- Zmniejszanie poprawki dla wag **często** aktualizowanych:
  - Obliczanie sumy G wszystkich dotychczasowych poprawek dla danej wagi
  - Skalowanie nowych poprawek przez G

$$W := W - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \nabla J$$

G_t to suma gradientów do chwili t
ϵ - Unikamy dzielenia przez 0

- Duża waga dla poprawek **występujących rzadko**, tj. gdy w dotychczasowych krokach czynnik G_t był mały, a w aktualnym gradient jest niezerowy
- …ale suma stale rośnie, zmniejszając (w granicy) tempo nauki do zera

Adadelta

- Wariant AdaGrad
- Zamiast sumy wszystkich poprzednich poprawek, uwzględnia „wiek” gradientów:
  - **wygaszanie starszych poprawek** bardziej niż nowych poprawek
- Aktualizacja wag przebiega głównie na podstawie najnowszych poprawek
- Szybkość nauki nie zanika

Adam - Adaptive Moment Estimation

- Użycie jednocześnie momentu pierwszego i drugiego rzędu i wygaszanie obu w czasie.

$$W := W - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \odot \hat{m}_t$$

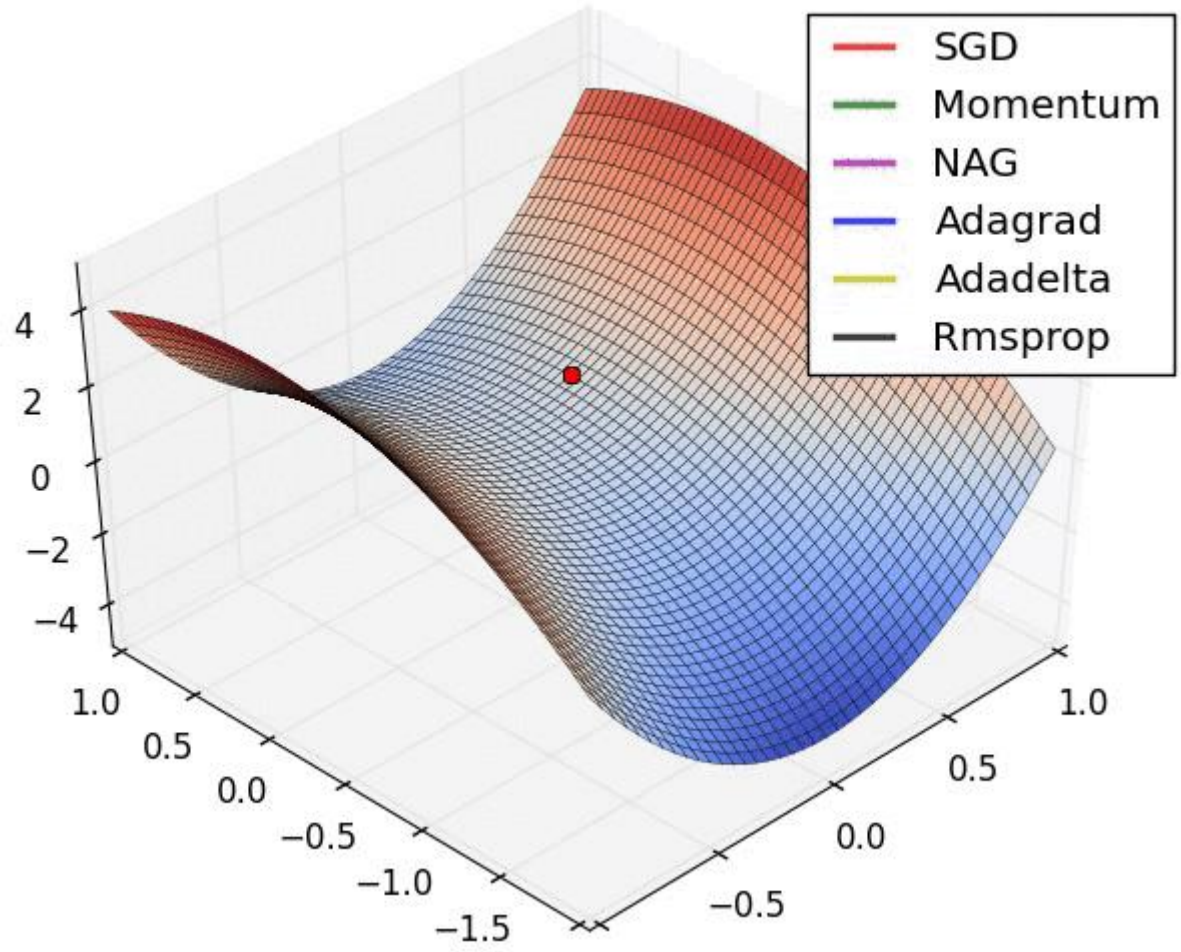
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

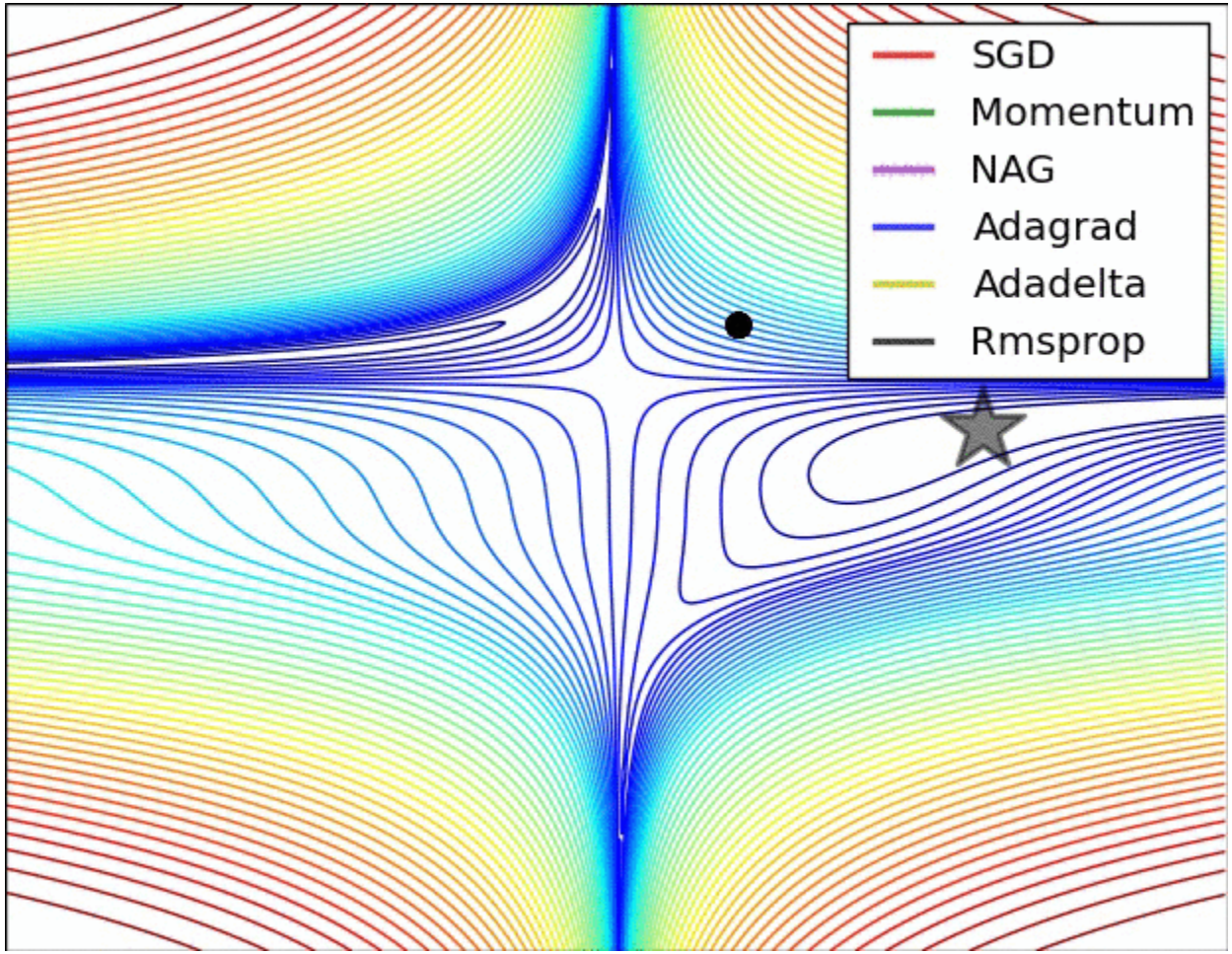
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla J$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla J^2$$

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


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

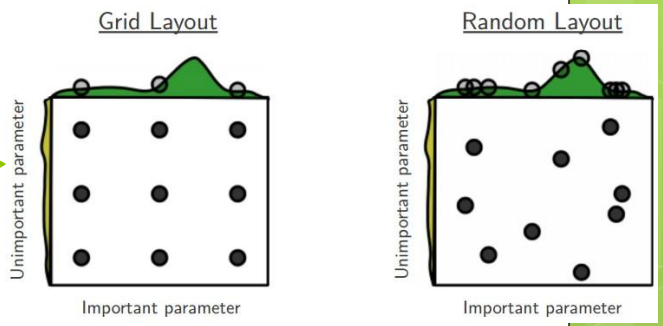

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

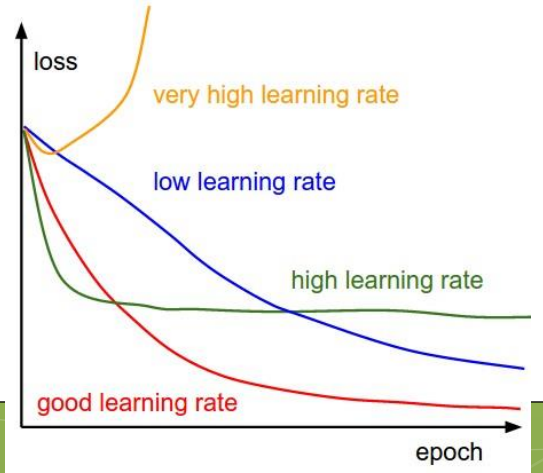
Zanikający gradient

- Problem dla wielu warstw z nasycającymi się f.aktywacji
- Wraz z propagacją wstecz:
  - Gradient mnożony przez pochodną bliską zeru (nasycenie), wyliczana poprawka jest coraz mniejsza (*vanishing gradient*)
- Dlatego w głębokich sieciach, stosuje się nienasycające się funkcje, np. ReLU (y=x, dla x>0)

Strojenie hiperparametrów

- Losowo w danym przedziale, wykładniczo, np. $10^{\text{random}(-6, 1)}$
  - Przyrost o 0.3 dla 0.001 to ogromna zmiana,
  - Przyrost o 0.3 dla 10 to nieistotna zmiana.
- Poszukiwanie losowych kombinacji
- Uwaga na skrajne wartości
  - Może najlepszy wynik dla 10^{-6} oznacza, że dla jeszcze mniejszych będzie lepszy?
- Sukcesywnie zawężać zakres, trenując w rosnącej liczbie kroków (epok)





Klasyfikacja – kodowanie typu „One-hot”

- Warstwa wyjściowa posiada n neuronów (n=liczba_klas)
- Każda klasa to „1” na odpowiednim wyjściu, pozostałe zera

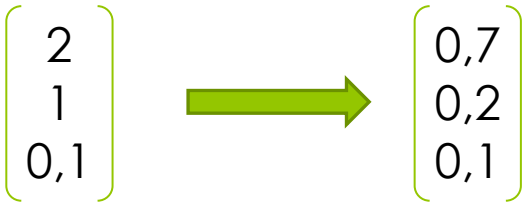
$$\begin{bmatrix}1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\end{bmatrix}$$

Rower Osobowy Ciężarowy

Neurony wyjściowe

- Funkcja aktywacji – softmax:
  - Wartości z zakresu <0,1>, sumujące się do 1

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$



„Logits”

Prawdopodo-
bieństwa

Regularyzacja

- Unikanie dużych wartości pojedynczych wag, gdyż zwykle oznacza to:
  - Niewłaściwe „faworyzowanie" wybranych wejść dużymi wagami
  - Niezdolność do generalizacji – radzenia sobie z nowymi danymi, wcześniej nie reprezentowanymi
  - …czyli ryzyko przetrenowania
- Błąd (funkcja kosztu) uwzględnia wartości wag (ocenia model)

$$J = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda\sum_{j=1}^{m}W_i^2$$

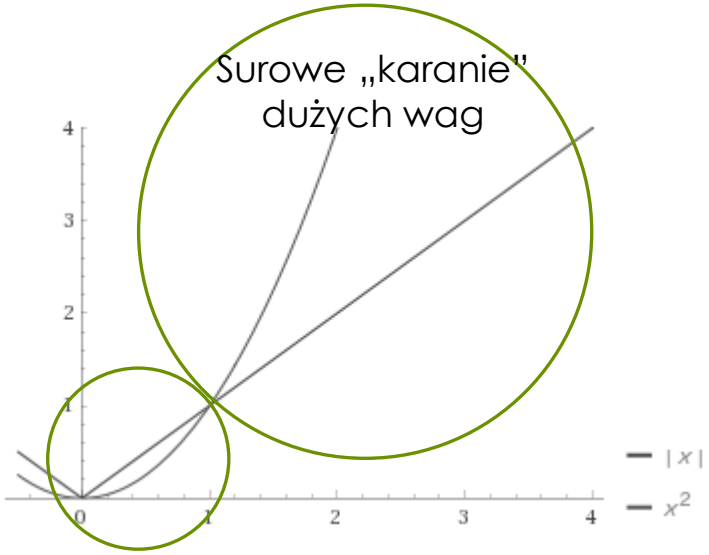
$$J = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda\sum_{j=1}^{m}|W_i\ |$$

Błąd nauki

„Kara" za duże wartości wag, czyli za niewłaściwy model

Warianty regularyzacji

- L2 (odległość) miara Euklidesowa
- L1 (suma) – miara Manhattan
- (x to przykładowa waga)



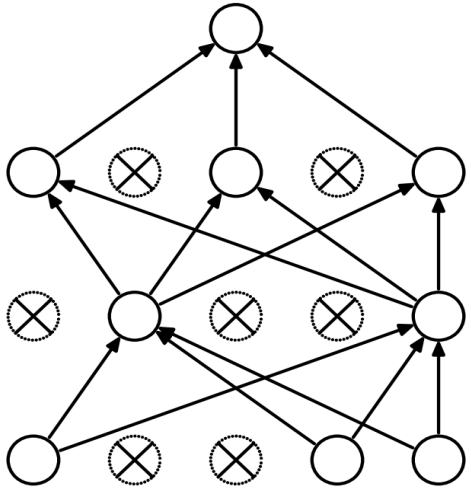
- https://playground.tensorflow.org
- regularyzacja

„Dropout” - koncepcja

- **Pojedynczy neuron nie powinien mieć przeważającego wpływu na decyzję**
- Strategia „dropout” wymusza udział wszystkich neuronów w decyzji

- Jest to dodatkowa metoda regularyzacji sieci

„Dropout” - koncepcja

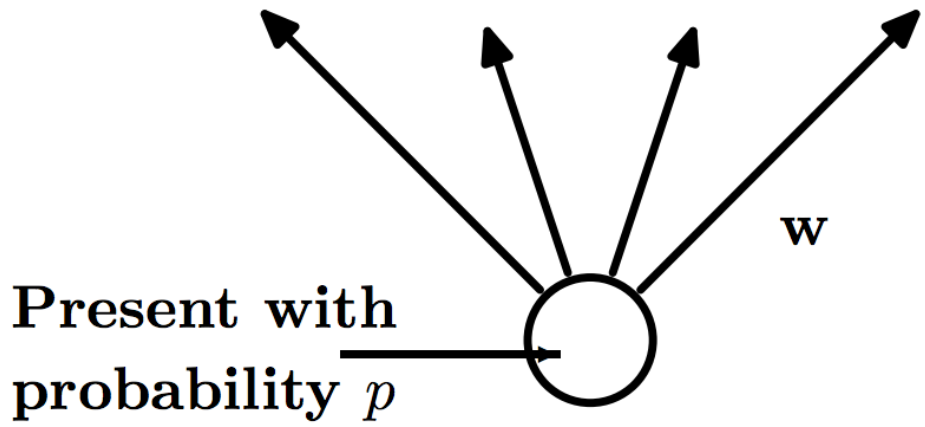


- Na etapie treningu **wyłącza się losowo neurony** (wyjścia ustawia na zero) na czas jednego cyklu nauki; sterowane prawdopodobieństwem *p*
- Neurony wyłączone nie są aktualizowane
- Trenuje się pozostałe neurony
- Ponawia się proces losowego włączenia/wyłączenia i kontynuuje trening

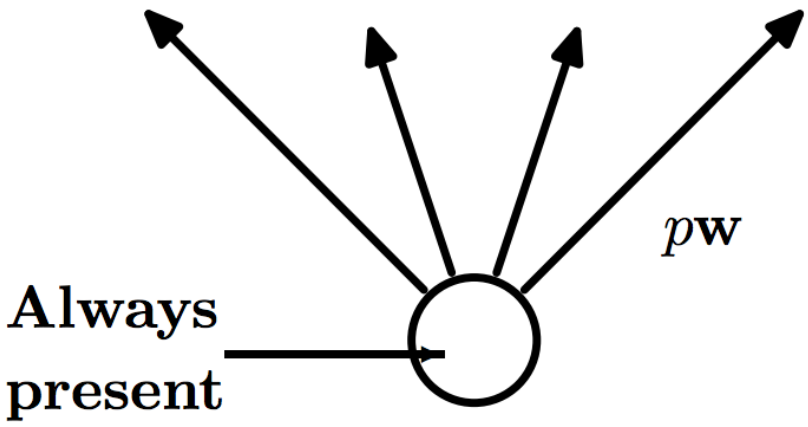
- Na etapie testu decyzja podejmowana jest łącznie przez wszystkie neurony
- To technika “*Model ensamble*”, zespołu modeli

Dropout – podejmowanie decyzji

- Jeśli neuron był włączany z prawdopodobieństwem p, to w trakcie wnioskowania jego wagi wyjściowe są skalowane: $\mathbf{w'}=p\mathbf{w}$



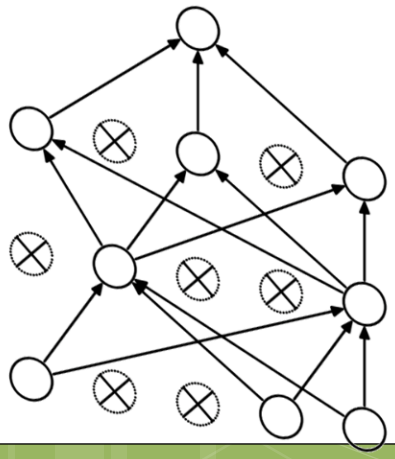
(a) At training time



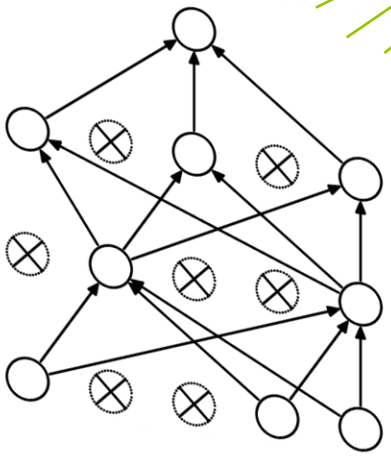
(b) At test time

Dropout – zespół modeli

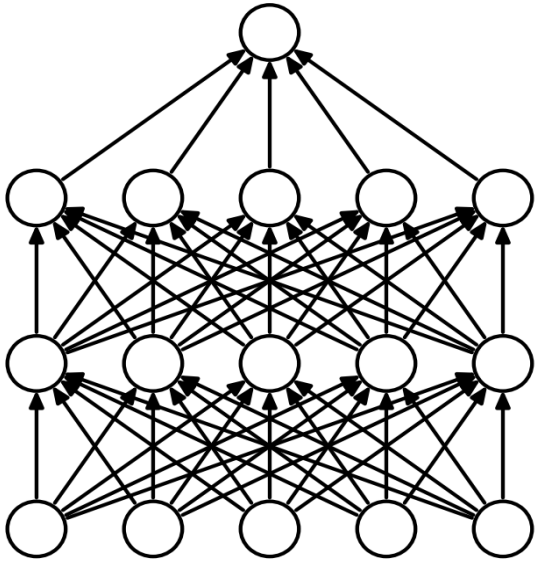
Trenować różne modele osobno:



…



Uśrednić modele i decyzje



Głębokie sieci neuronowe

Głębokie (splotowe) sieci neuronowe

- Sieci o wielu warstwach
- **Dodatkowe typy neuronów**
  - Splotowe
  - „Kumulujące”, ang. *pooling*

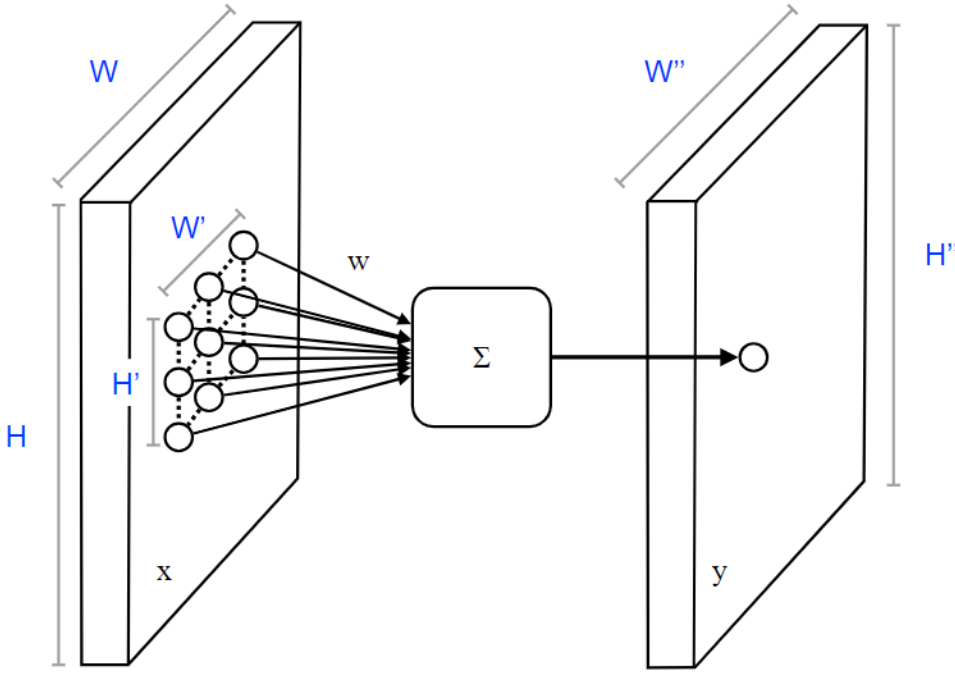
- Cel: stopniowa (z warstwy na warstwę) **redukcja reprezentacji** próbki wejściowej do zestawu **cech** opisujących
- Powiązanie cech (a nie wartości sygnału/pikseli) z decyzją

Splot 2D

- Typowo: filtracja **obrazu**, tj. jednej/trzech macierzy 2D: wys.×szer.×3 (1 lub 3 kolory składowe, tylko skala szarości lub RGB)
- Wariant: filtracja **spektrogramu**, traktowanego jako obraz 2D: wys.×szer., czyli częstotliwość×czas

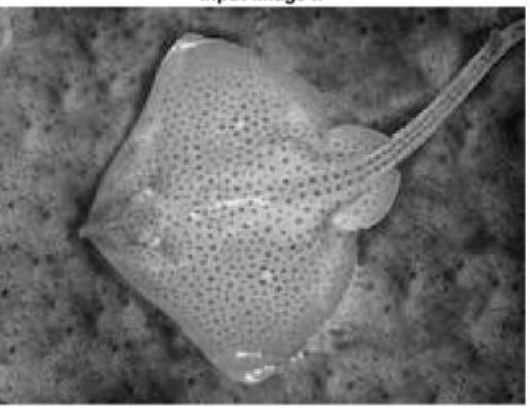
- Użycie kilku filtrów (zestawów wag), na wszystkich wycinkach macierzy, przesuwane z krokiem 1×1

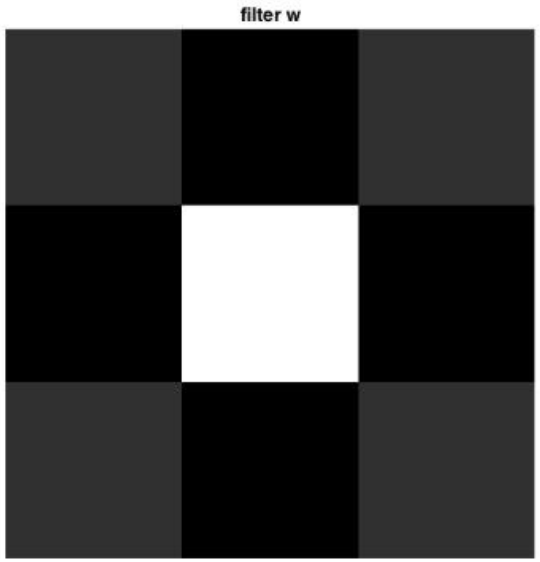
- „wycinek" = „pole recepcyjne" (analogia do komórek horyzontalnych siatkówki oka)

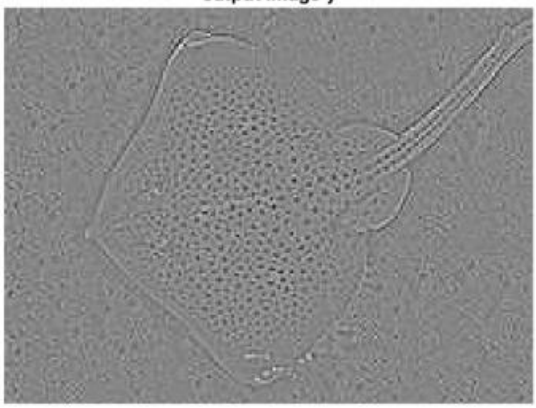


Przykład splotu

- Detektor krawędzi:







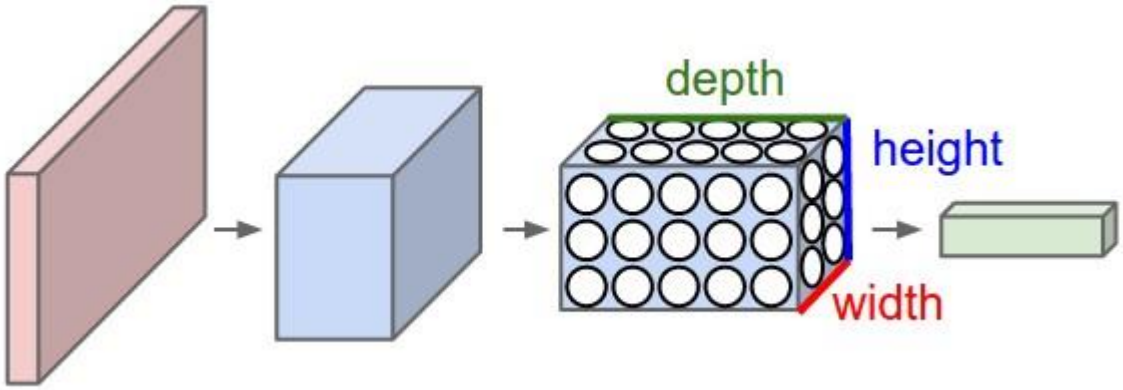
$$\mathbf{w} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$y_{ij} = \sum_{uv} w_{uv}\, x_{i+u,\, j+v}$$

Zastosowania splotu 2D

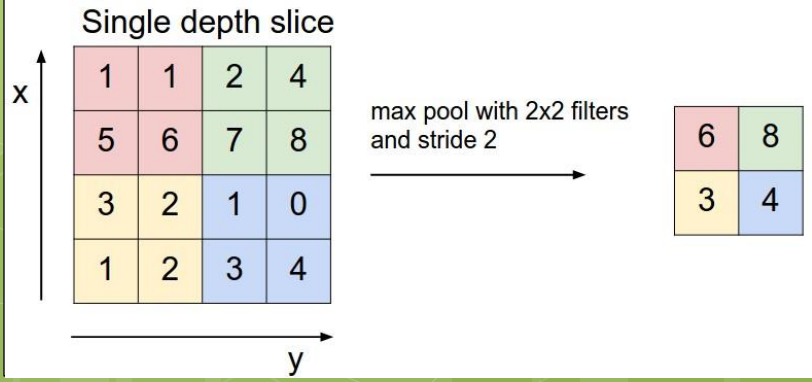
- **Wykrywanie krawędzi**, liczenie pochodnej
- **Wykrywanie punktów,** kierunków, faktur, częstotliwości
- **Usuwanie szumu**, tj. wyliczanie tła, czyli lokalnej tendencji
- Przykładowe filtry (ang. *kernels*):

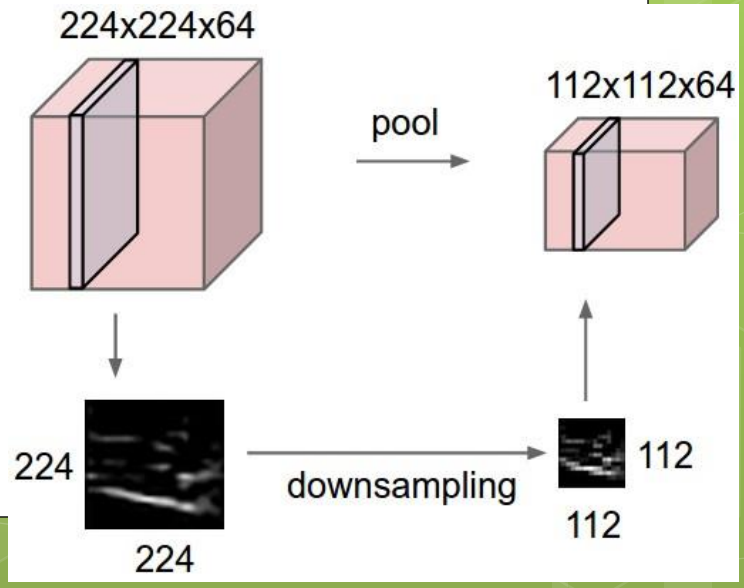
Sieć głęboka - wiele warstw, wiele filtrów



Kumulowanie „Pooling”

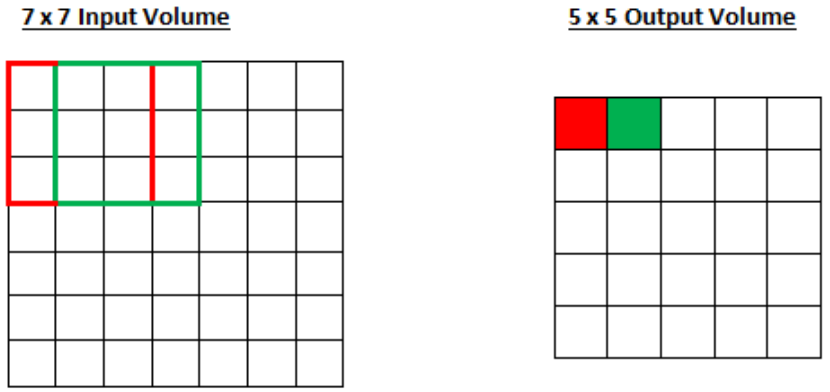
- Pozycja obiektu na obrazie z dokładnością co do piksela nie jest istotna! Ważne są lokalne tendencje (max, średnia, mediana)
- Przesuwane okno z zadanym krokiem (ang. *stride*)
- Zmniejszenie rozdzielczości (podpróbkowanie)



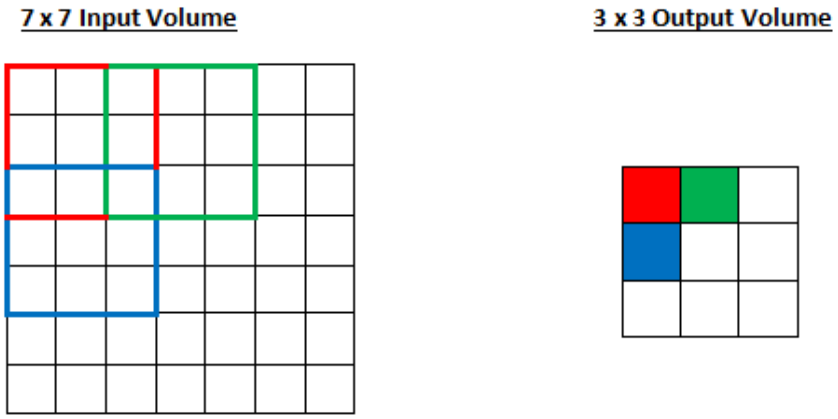


Kumulowanie „Pooling”

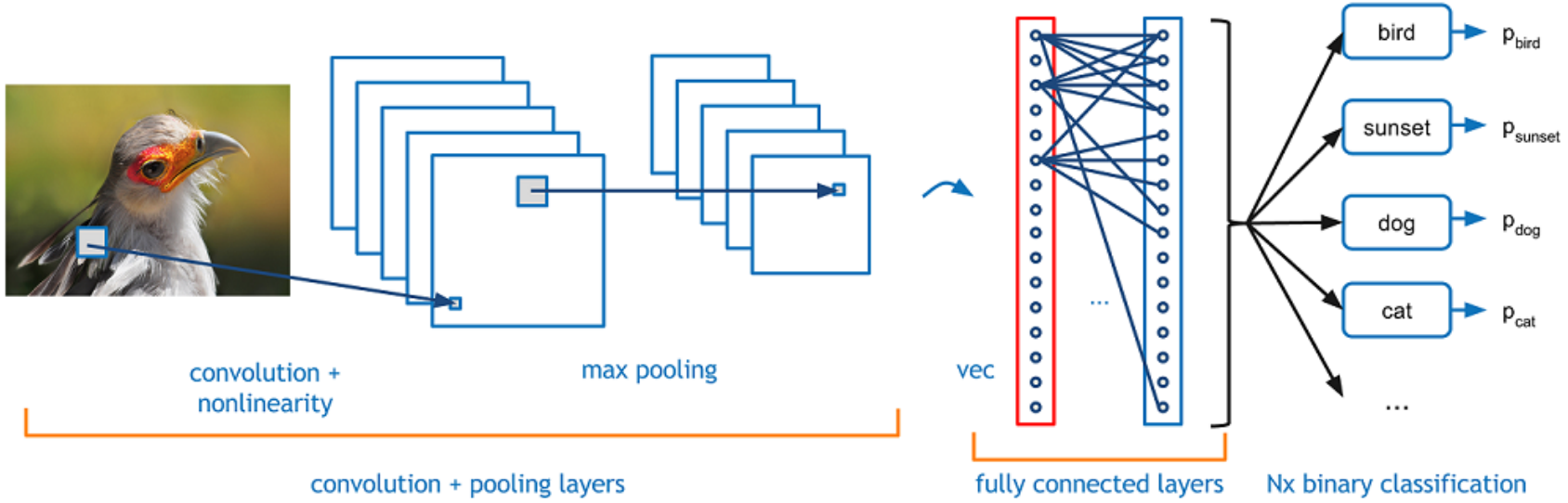
- Dobór rozmiaru zakładki
- Rozmiar 3x3, stride 1:

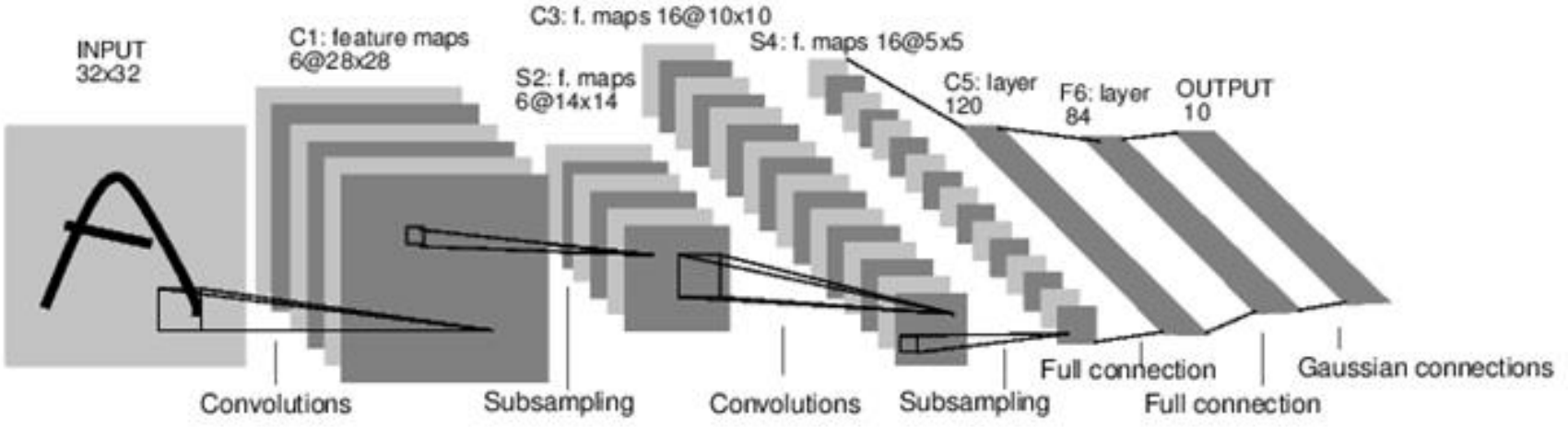


- Rozmiar 3x3, stride 2:



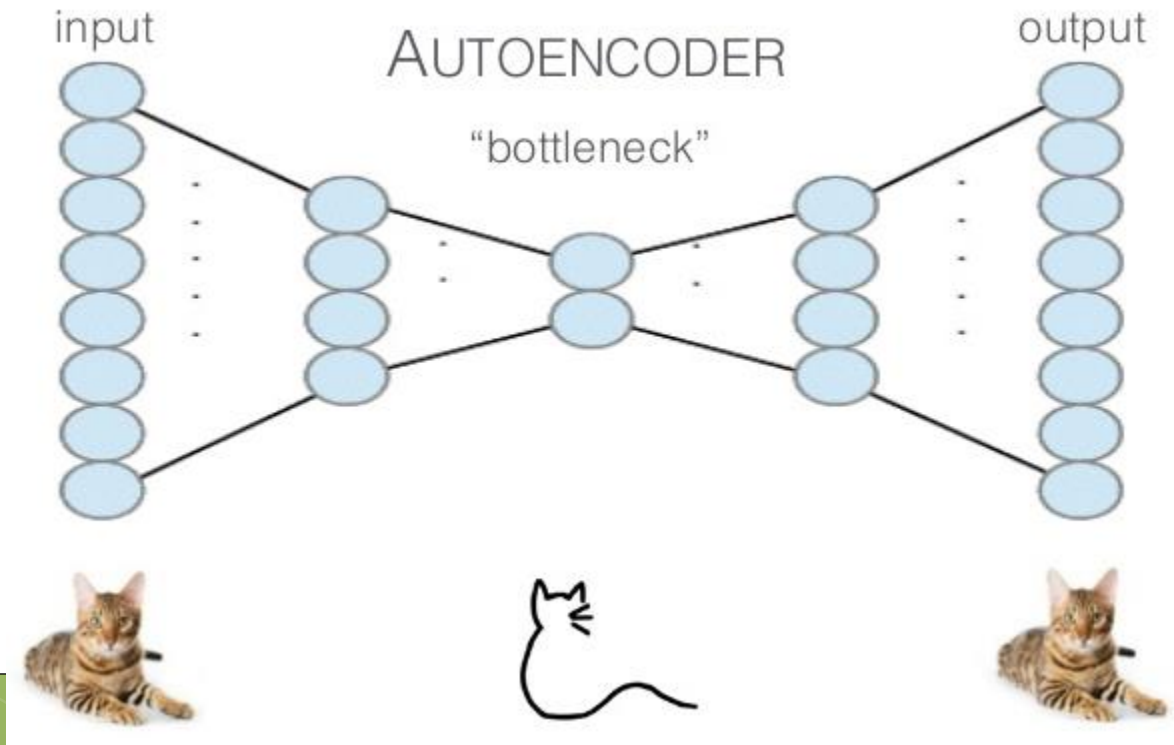
Przykłady





A Full Convolutional Neural Network (LeNet)

„Representation learning” i kompresja danych siecią typu „Autoencoder”



Transfer learning

- Bazowanie na **gotowej sieci** wytrenowanej wcześniej na milionach próbek, w celu uzyskania **nowej sieci** dla innego zagadnienia o **małej liczbie próbek**

http://cs231n.github.io/transfer-learning/

Transfer learning

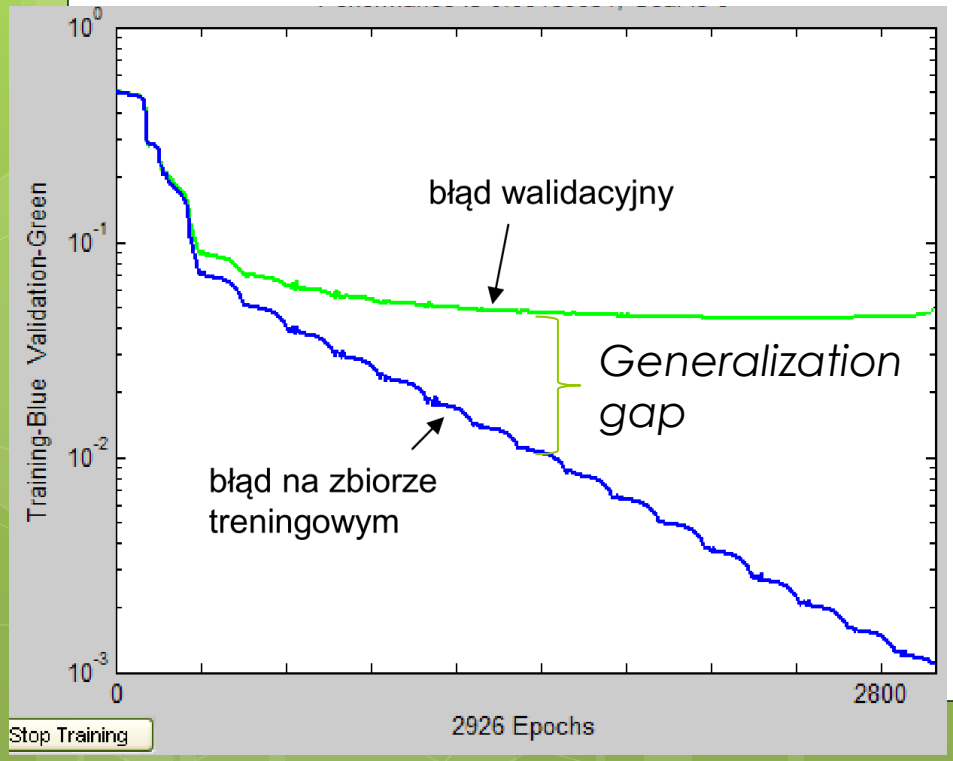
- Wytrenowana sieć stosowana jako:
  - **Ekstraktor przydanych cech**
    - wymaga dotrenowania głównie ostatniej warstwy, wiążącej cechy z decyzjami
    - Regulacja prędkości nauki:
      - Wysoka prędkość dla ostatniej warstwy
      - Niska prędkość dla całej sieci
    - stosowany dla małego zbioru danych, podobnych do oryginalnego

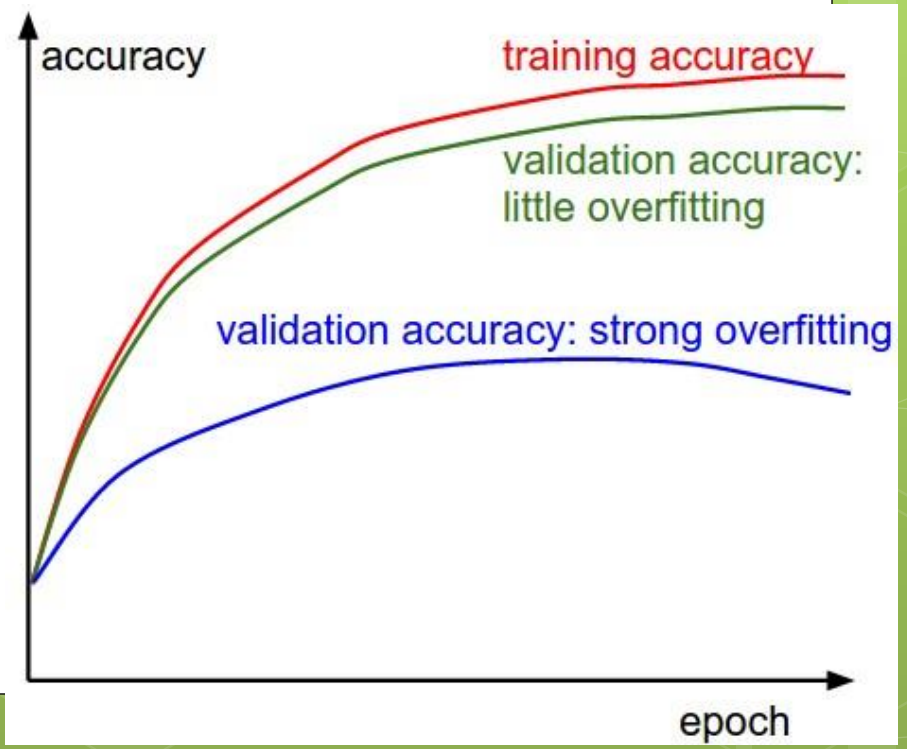
Transfer learning

- Wytrenowana sieć stosowana jako:
  - **Punkt startu do dalszego treningu**
    - założenie, iż najprzydatniejsze cechy z warstw pierwszych pozostaną (nieznacznie zmienione), a cechy z warstw głębszych dostosują się do nowego zagadnienia
    - prędkość nauki dla wszystkich warstw taka sama
    - dla dużego zbioru, o innej naturze niż oryginalny

Wczesne zatrzymanie treningu

- Obserwacja błędu walidacyjnego
- Jeśli przestaje maleć, zakończyć trening





Przetrenowanie

- Sieć można „przeuczyć”, gdy algorytm powtarza się w zbyt wielu krokach
- Sieć przetrenowana generuje bardzo trafne wyniki dla danych treningowych, ale nie działa prawidłowo dla danych testowych
- Metoda przeciwdziałania – sprawdzanie działania sieci w każdym kroku dla danych walidacyjnych

Dziękuję za uwagę