Podstawy Sztucznej Inteligencji (PSI)

Lab 7 - omówienie

Pytania? Wątpliwości? Uwagi?

Częste kwestie... brak!

Wstęp do sieci neuronowych

Idea

- początkowo inspirowane neuronami w mózgu
- później - czysta matematyka
- zdecydowana większość ma **budowę warstwową:**
  - **neurony (neurons)** układane w **warstwy (layers)**
  - neurony w warstwie nie są ze sobą połączone, tylko warstwy po kolei
  - **wyjście** poprzedniej warstwy to **wejście** następnej
- na wyjściu całej sieci dostajemy np. wyniki dla poszczególnych klas

Rodzaje sieci neuronowych

Rodzaj sieci	Numer laboratorium	Dziedzina zastosowań
MultiLayer Perceptron (MLP)	3	Dane tabelaryczne
Convolutional Neural Network (CNN)	4	Obrazy, audio, klasyfikacja szeregów czasowych
Recurrent Neural Network (RNN)	-	Szeregi czasowe, dane sekwencyjne
Attentional / transformers	5	Przetwarzanie języka naturalnego (NLP) oraz ostatnio wszystko inne
Graph Neural Networks (GNN)	6 (zad. dodatkowe)	Grafy, dane przestrzenne, ostatnio też obrazy, szeregi czasowe

Sieci typu MultiLayer Perceptron (MLP)

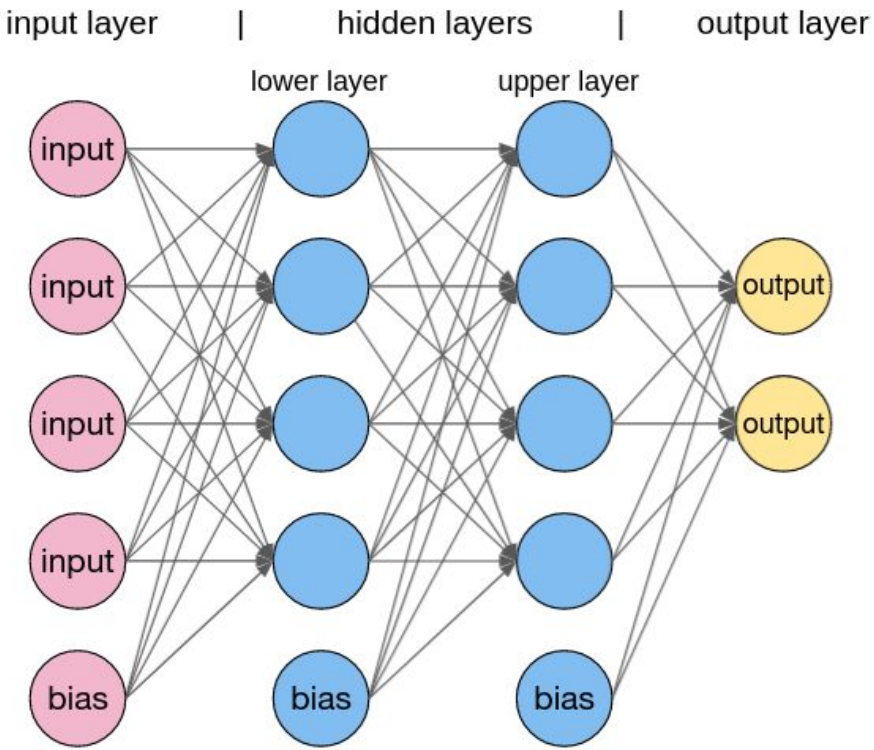
MultiLayer Perceptron (MLP)

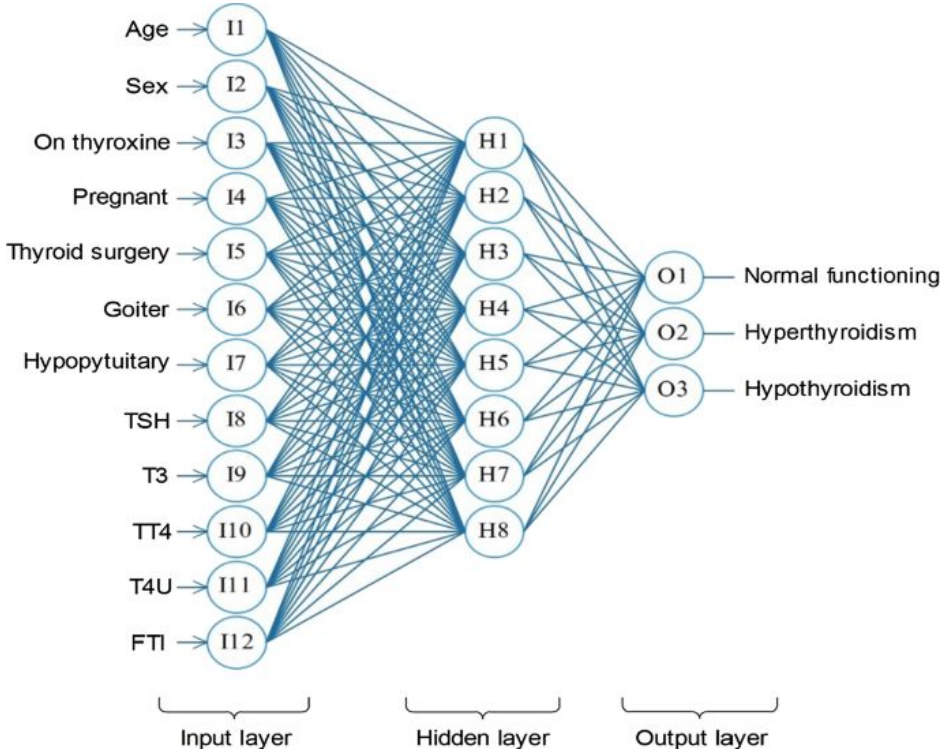
- najstarszy i podstawowy rodzaj sieci
- regresja lub klasyfikacja na danych tabelarycznych
- niezbędny element budowy innych sieci, typowo na końcu, jako tzw. **głowa (head)**
- wejście: wektor
- wyjście: skalar (regresja) lub wektor **logitów (logits)**, długość = liczba klas

Budowa MLP

- wejście sieci: wektor
- składa się z **warstw w pełni połączonych (fully / densely connected)**
- wyjście poprzedniej warstwy to wejście następnej
- każdy neuron z poprzedniej warstwy łączy się z każdym z następnej (stąd nazwa)
- wyjście sieci:
  - skalar (regresja)
  - wektor **logitów (logits)**, długość = liczba klas

Budowa MLP


input layer | hidden layers | output layer
lower layer
upper layer
input
input
input
input
bias
bias
bias
output
output


Age
Sex
On thyroxine
Pregnant
Thyroid surgery
Goiter
Hypopytuitary
TSH
T3
TT4
T4U
FTI
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
I12
H1
H2
H3
H4
H5
H6
H7
H8
O1
O2
O3
Normal functioning
Hyperthyroidism
Hypothyroidism
Input layer
Hidden layer
Output layer

Warstwa w pełni połączona

- zbiór neuronów, każdy to **wektor parametrów**, o zadanej **wymiarowości**
- cała warstwa to **macierz wag W**
- neuron realizuje **funkcję**, której wejściem jest macierz X: $f(X) = \sigma(XW^T)$
- σ to **funkcja aktywacji (activation function)**, która wyznacza, jak bardzo wejście **aktywuje** dany neuron
- dodaje się też bias, jak w regresji liniowej - kolumna samych 1 w X
- wyjście nazywa się często **ukrytą reprezentacją (hidden representation)**

Sieć MLP

- mamy warstwy po kolei, przekazują sobie wejście - to **złożenie funkcji:**

$$H_1 = f_1(X) = \sigma(XW_1^T)$$
$$H_2 = f_2(H_1) = \sigma(H_1W_2^T)$$
$$\dots$$
$$H_n = f_{n-1}(H_{n-1}) = \sigma(H_{n-1}W_n^T)$$

- funkcja σ musi być **nieliniowa (non-linear)**, bo złożenie funkcji liniowych byłoby liniowe
- warstwy w środku nazywa się **warstwami ukrytymi (hidden layers)**

Wyjście sieci

Regresja:

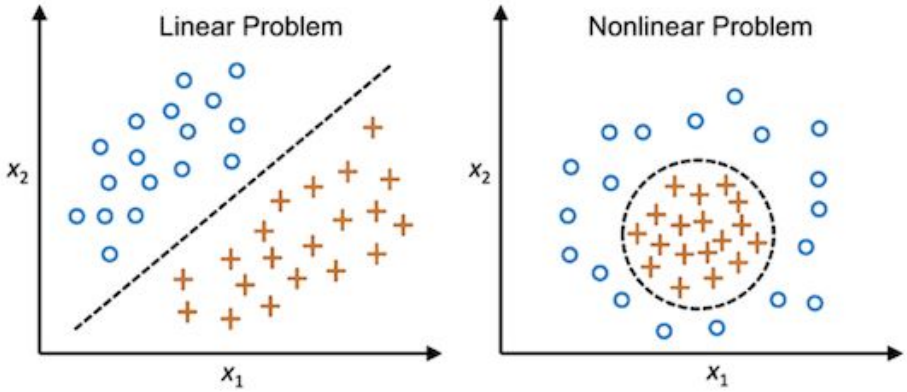
- 1 neuron
- "bez aktywacji" (liniowa)

Klasyfikacja:

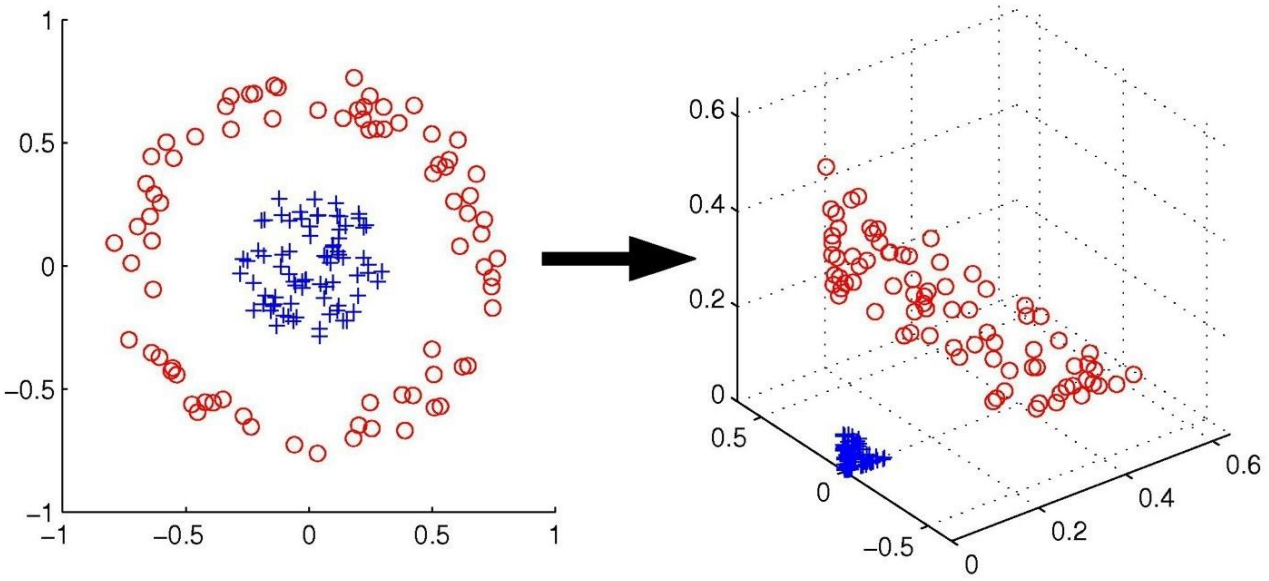
- K neuronów dla K klas
- wyjście przed aktywacją to **logity (logits)**, wartości rzeczywiste dla każdej klasy
- aktywacja **funkcją softmax** - zamienia logity na rozkład prawdopodobieństw

Co to tak naprawdę robi?

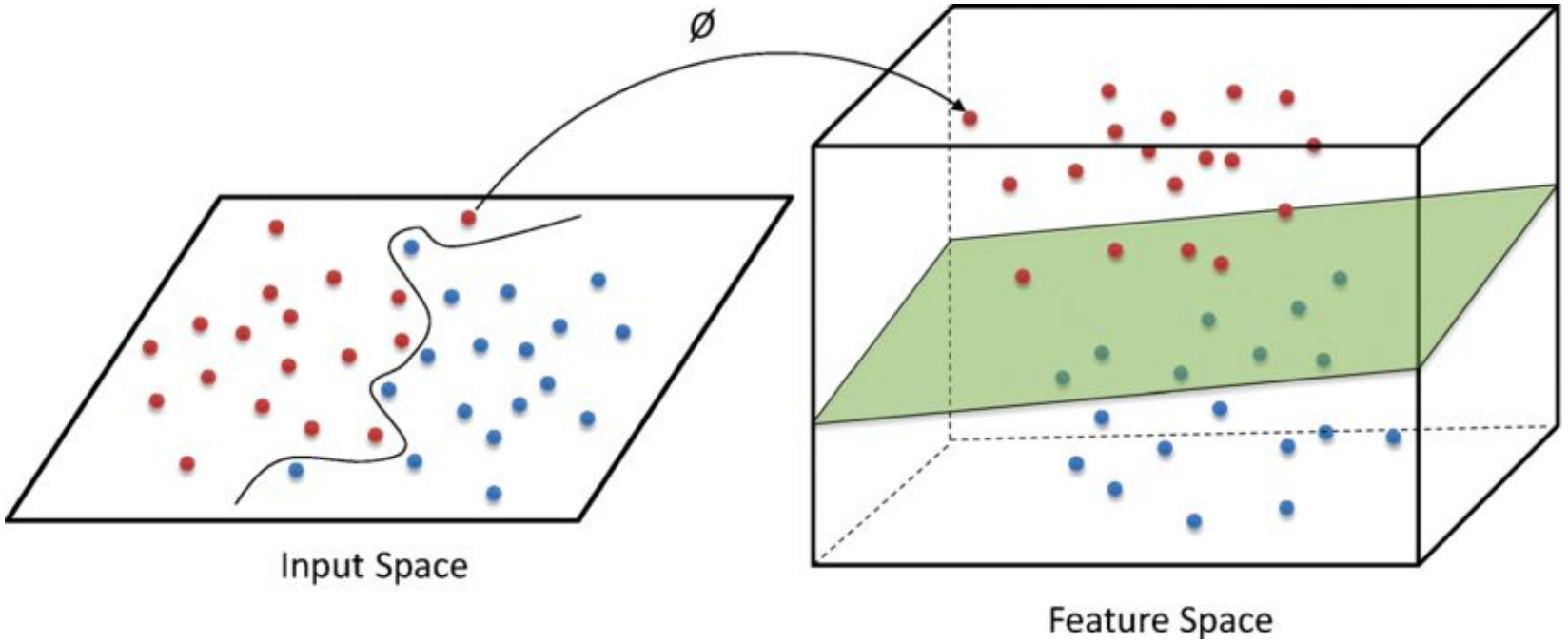
- ostatnia warstwa sieci to **tak naprawdę** regresja liniowa lub logistyczna / softmax!
- zakładamy w niej, że klasy są **liniowo separowalne (linearly separable)**
- poprzednie warstwy to tak naprawdę **transformatory**, które przekształcają przestrzeń tak, żeby na końcu było jak najłatwiej



Nieliniowe przekształcenia przestrzeni



Nieliniowe przekształcenia przestrzeni



Trening sieci neuronowych

Trening sieci neuronowych

- **przejście w przód (forward pass)** - wejście przechodzi przez sieć i dostajemy predykcję
- obliczamy różnicę między predykcją a prawdą z użyciem **funkcji kosztu (loss / cost function)**
- **cel:** chcemy wiedzieć, jak zmienić poszczególne wagi (parametry sieci), aby uzyskać mniejszy koszt

Trening sieci neuronowych

- **obserwacja 1:** pochodna wyznacza, jak zmienia się funkcja
- **obserwacja 2:** dla funkcji wielu zmiennych pochodne cząstkowe pozwalają wyznaczyć, jak duży wpływ na wartość mają poszczególne zmienne
- **obserwacja 3:** pochodne cząstkowe można łatwo liczyć **regułą łańcuchową (chain rule)**

- **idea:** użyjmy pochodnych i spadku wzdłuż gradientu, żeby optymalizować wagi sieci

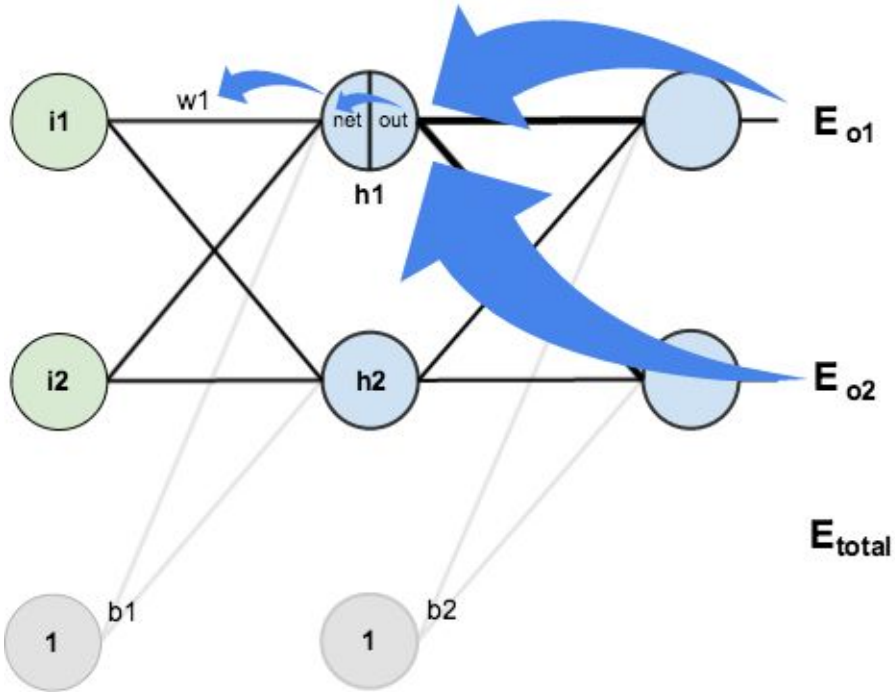
Propagacja wsteczna

- liczymy **pochodną funkcji kosztu** i jej wartość
- za pomocą reguły łańcuchowej liczymy pochodne cząstkowe - pochodne po neuronach z **ostatniej warstwy**
- aktualizujemy zgodnie ze spadkiem wzdłuż gradientu: $w_i = w_i - \alpha \frac{\partial C(W)}{\partial w_i}$
- cofamy się do poprzedniej warstwy, powtarzamy

Propagacja wsteczna

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$$E_{total} = E_{o1} + E_{o2}$$

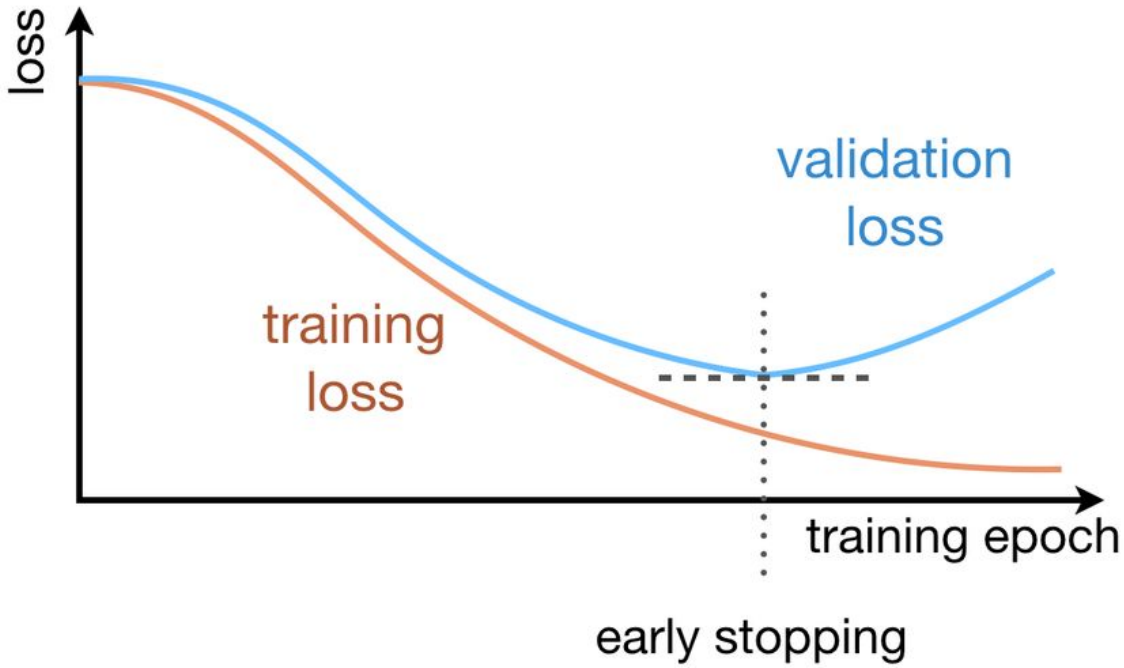
Jak liczyć pochodne?

- obliczenie dokładnej pochodnej zakłada użycie całych danych - **gradient descent (GD)**, lub full-batch GD
- **problem:** nie wystarczy nam na to pamięci, trzeba by za każdym razem przetwarzać cały zbiór danych
- **idea:** przybliżmy pochodną, używając mniej danych; mniejsze kroki, mniej dokładne, ale szybciej i więcej
- **stochastyczny GD (stochastic GD, SGD)** - uczymy 1 próbką naraz
- **minibatch GD** - złoty środek, używany w praktyce

Czas treningu

- przejście przez cały zbiór treningowy (wszystkimi minibatchami) to **epoka (epoch)** treningu
- im dłużej uczymy, tym lepiej dostosowujemy się do zbioru treningowego - overfitting
- można przerwać wcześniej - metoda **wczesnego stopu (early stopping)**
- co epokę (albo rzadziej) sprawdzamy wynik na zbiorze walidacyjnym
- jeżeli nie dostaliśmy lepszego wyniku od dotychczasowego najlepszego przez liczbę epok równą **cierpliwości (patience)**, to przerywamy trening

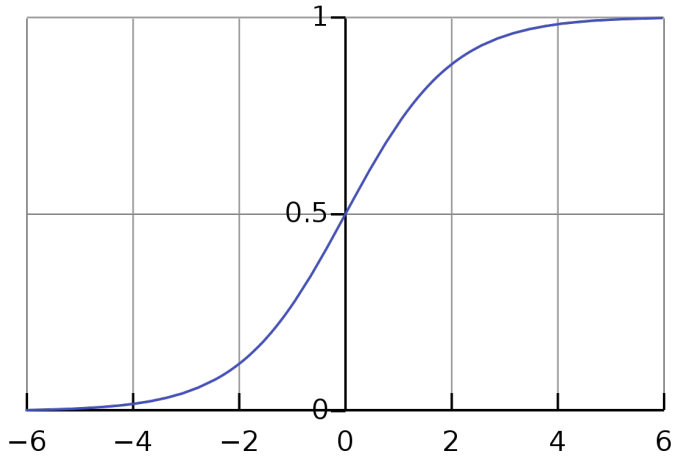
Training-validation loss curve



Funkcje aktywacji

Sigmoida

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

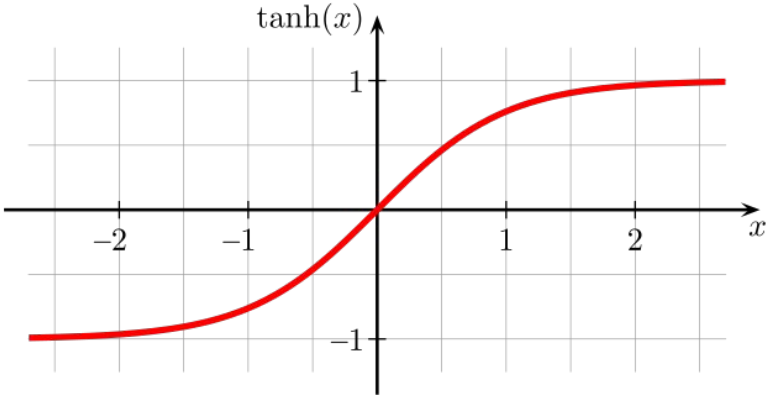


- nieużywane wewnątrz sieci, bo ma płaskie końce
- płaskie = gradienty bliskie 0 = sieć przestaje się uczyć - problem **nasyconych neuronów (saturated neurons)**
- czasem używane na końcu sieci w klasyfikacji binarnej

Tangens hiperboliczny

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

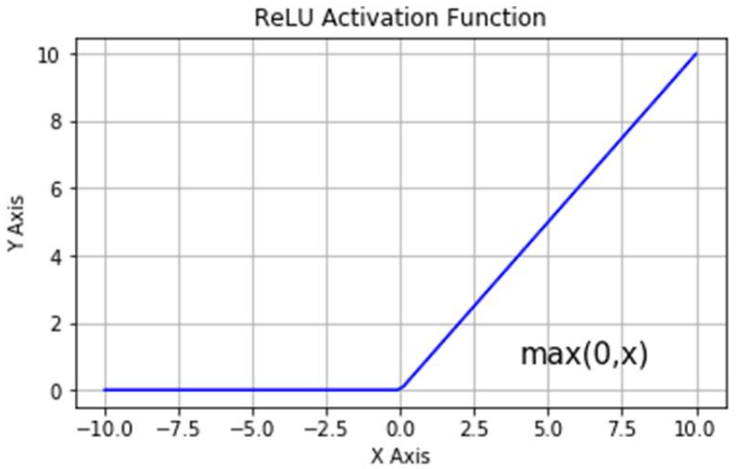
- kiedyś używana, żeby rozwiązać problem nasyconych neuronów z sigmoidy
- dalej ma ten problem
- obecnie nieużywana



Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(x) = \max(0, x)$$

- przełom w uczeniu głębokich sieci neuronowych
- prosta, szybka, dobrze działa
- nieróżniczkowalna w 0 - robimy if/else
- pochodna w znacznej części to 0, więc neurony się nie uczą, jeżeli wpadną w ten obszar - problem **martwych neuronów (dead neurons)**

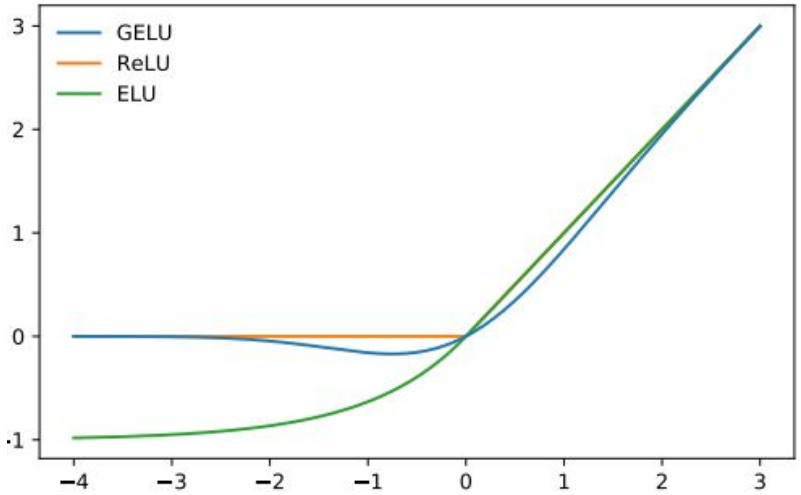


Nowe funkcje aktywacji

- różniczkowalne przybliżenia ReLU
- szczególnie przydatne przy bardzo głębokich i bardzo dużych sieciach
- różne podejścia, koszt i dodatkowe parametry
- ciężko wybrać, GELU lub ELU to zwykle dobry wybór

$$\mathrm{GELU}(x) = x \cdot \Phi(x)$$

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ a(e^x - 1) & \text{otherwise} \end{cases}$$



Softmax

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{K} e^{x_i}}$$

- zamienia wektor na rozkład prawdopodobieństw
- używane na logitach na wyjściu sieci

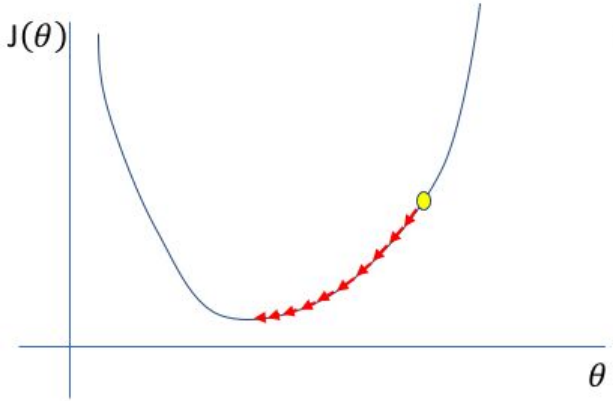
Ulepszenia i hiperparametry

Stała ucząca

- **wyznacza wielkość kroków**
- duża:
  - uczymy szybko
  - przeskakujemy minima lokalne
  - ciężko wbić się w minimum
- mała:
  - uczymy powoli
  - precyzyjnie znajdujemy minima
  - możemy wpaść w minimum lokalne
- **typowe wartości:**
  - domyślnie: 1e-3 (0.001)
  - zakres: od 1e-5 do 1e-2
- może być **adaptacyjna (adaptive):**
  - zależna od momentu treningu
  - różna dla różnych zmiennych
  - dostosowywać się do dynamiki treningu

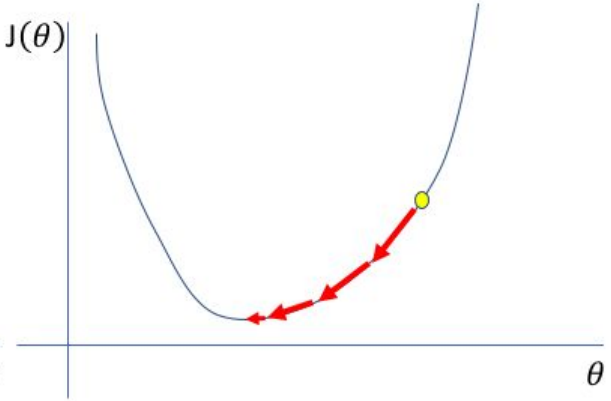
Stała ucząca

Too low



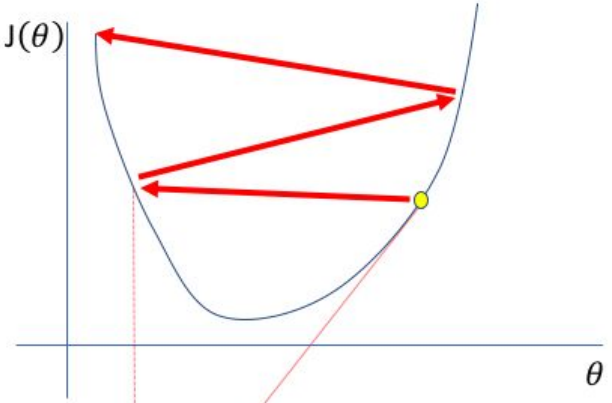
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

Regularyzacja

- **L2** - tak jak przy regresji liniowej, penalizujemy duże wagi
- **dropout:**
  - losowo wyłączamy neurony w trakcie treningu (zerujemy wagi)
  - zapobiega nadmiernej specjalizacji neuronów - muszą być gotowe, że inne przestaną istnieć
  - często bardzo ostry, np. 25% czy 50%
  - realizuje **ensemble learning** - w każdej iteracji uczymy nieco inną sieć
- **wczesny stop**

Hiperparametry

Architektura sieci:

- liczba warstw
- rozmiary warstw
- funkcja aktywacji

Trening:

- stała ucząca
- siła regularyzacji L2
- prawdopodobieństwo dropoutu
- cierpliwość wczesnego stopu

Pytania?