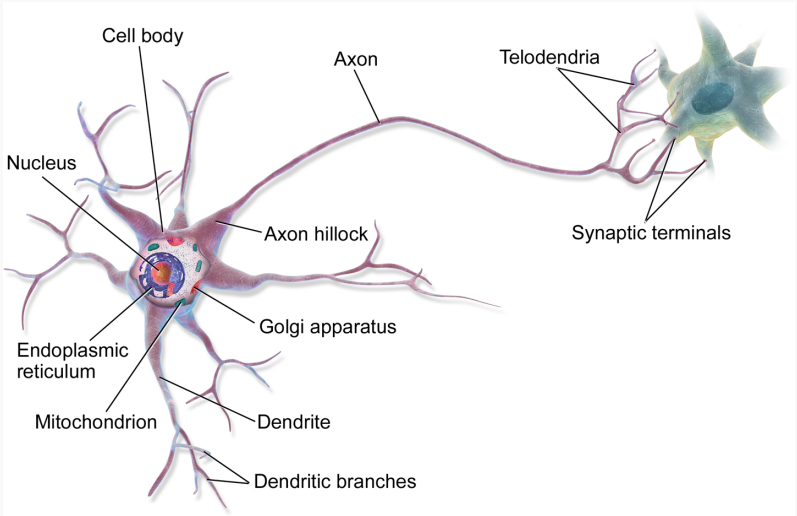
Wprowadzenie do sieci neuronowych

dr inż. Sebastian Ernst

Przedmiot: Uczenie Maszynowe

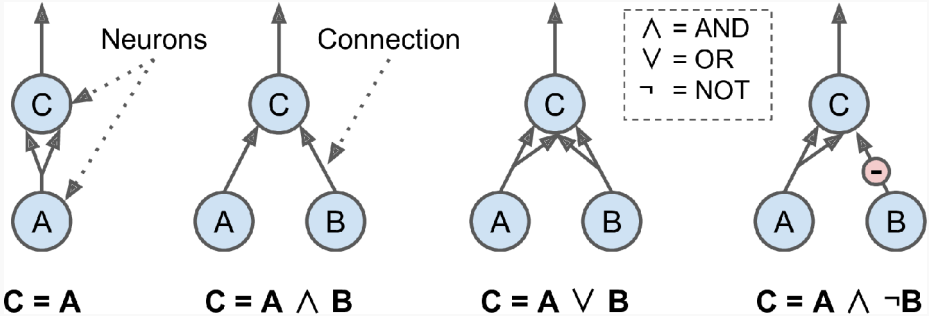
Sieci neuronowe

Neuron (biologiczny)



Neuron (sztuczny)

- Po raz pierwszy zaproponowany w 1943 przez Warrena S. McCullocha i Waltera Pittsa, w pracy pt. *A logical calculus of the ideas immanent in nervous activity*
- Założenie: neuron ulega aktywacji gdy co najmniej dwa jego wejścia są aktywne.
- Kilka przykładów:



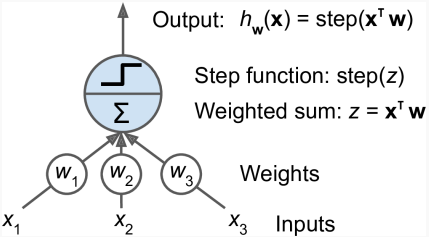
Sieć neuronowa

- Sieć połączonych ze sobą neuronów:
  - biologicznych (BNN)
  - sztucznych (ANN)
- Może modelować złożone procesy logiczne przy pomocy prostych co do zasady działania elementów

Perceptron

Perceptron

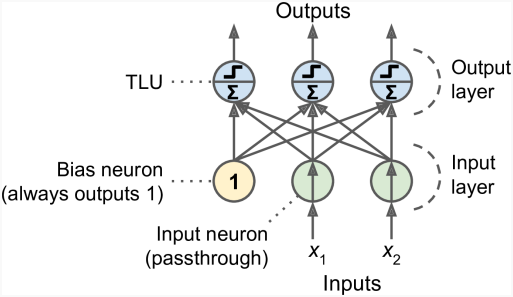
- Jedna z najprostszych architektur ANN
- Zaproponowana w 1957 przez Franka Rosenblatta
- Neuron zwany TLU (*threshold logic unit*) lub LTU (*linear threshold unit*)
- Wejścia są *liczbami*, a każde wejście posiada także *wagę*
- Sposób działania:
  1. oblicz sumę wejść:
     $z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \mathbf{x}^T \mathbf{w}$
  2. zastosuj funkcję schodkową: $h_w(\mathbf{x}) = \text{step}(z)$



$$\text{heaviside}(z) = \begin{cases} 0 & \text{dla } z < 0 \\ 1 & \text{dla } z \geqslant 0 \end{cases}$$

$$\text{sgn}(z) = \begin{cases} -1 & \text{dla } z < 0 \\ 0 & \text{dla } z = 0 \\ 1 & \text{dla } x > 0 \end{cases}$$

Perceptron



- Jeżeli każdy neuron w warstwie jest połączony ze wszystkimi neuronami w warstwie poprzedzającej, warstwa jest *w pełni połączona* (ang. *fully connected*) lub *gęsta* (ang. *dense*)
- Obliczenie wyjścia warstwy gęstej:

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = \phi(\mathbf{XW} + \mathbf{b})$$

Uczenie perceptronu

- Algorytm Rosenblatta był zainspirowany *zasadą Hebba* (Donald Hebb, *The Organization of Behavior*, 1949)
- Siegrid Löwel: *„Cells that fire together, wire together"*
- Aktualizacja wag w każdym kroku:

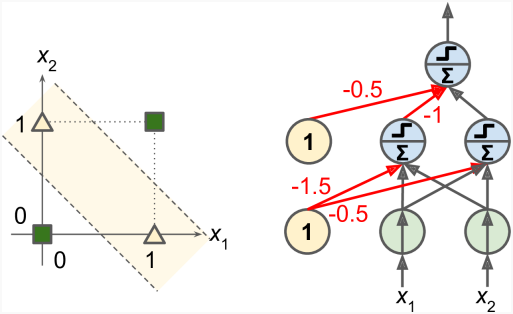
$$w'_{i,j} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$$

Perceptron w scikit-learn

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)]  # petal length, petal width
y = (iris.target == 0).astype(int)

per_clf = Perceptron()
per_clf.fit(X, y)

y_pred = per_clf.predict([[2, 0.5]])
```

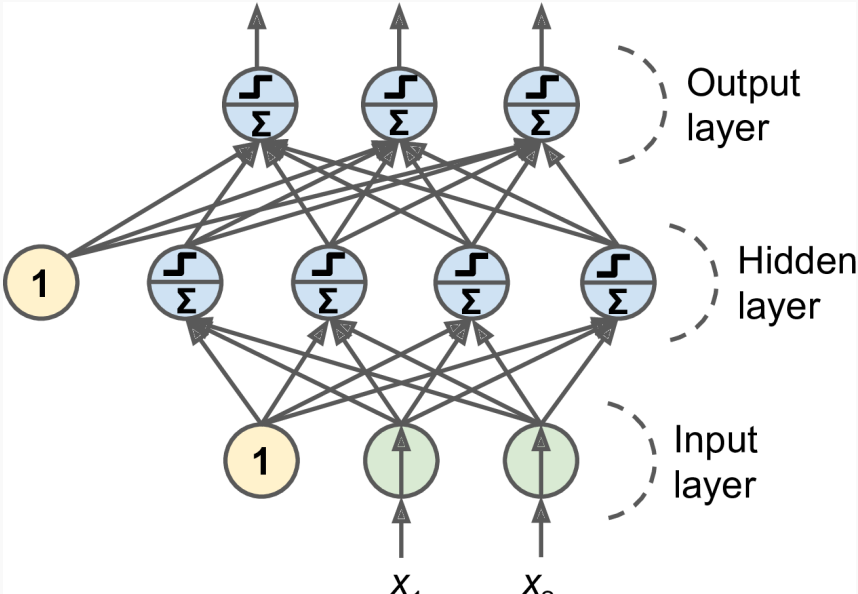
Ograniczenia perceptronu

- Nie potrafią rozwiązywać pewnych trywialnych problemów, jak np. klasyfikacja XOR
- Ale problem ten można rozwiązać poprzez dodanie dodatkowej warstwy



Perceptron wielowarstwowy (MLP)

Perceptron wielowarstwowy (MLP)

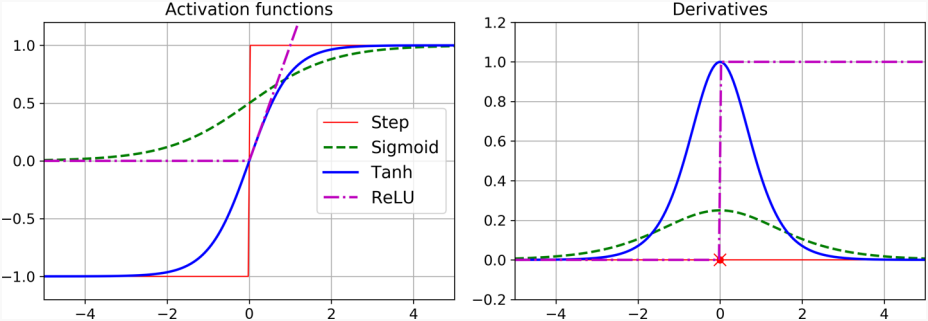


Uczenie MLP

- Przez wiele lat opracowanie efektywnego algorytmu uczenia stanowiło problem
- W 1986 Rumelhart, Hinton i Williams zaproponowali algorytm *propagacji wstecznej* (*backpropagation*) – używany do dziś
- Przechodzi przez sieć raz do przodu (przeprowadzenie predykcji) i raz wstecz
- Oblicza gradient błędu w odniesieniu do każdego parametru i modyfikuje wagi (a'la *gradient descent*)
- Aby pojawił się gradient, zastępujemy funkcję schodkową inną funkcją.

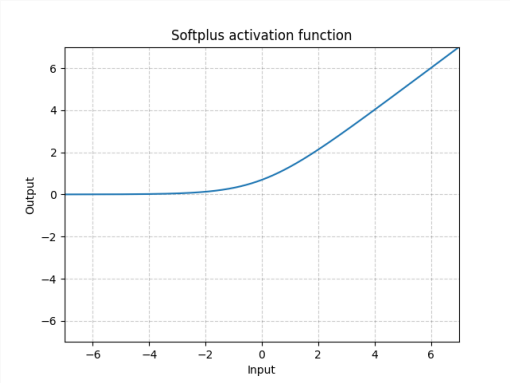
Funkcje aktywacji dla propagacji wstecznej

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \tanh(z) = 2\sigma(2z) - 1 \qquad \mathrm{ReLU}(z) = \max(0, z)$$



Regresja przy pomocy MLP

- Tyle neuronów wyjściowych, ile predykowanych wymiarów
- Funkcja aktywacji warstwy wyjściowej:
  - żadna
  - ReLU – aby wyjście było dodatnie
  - $\text{softplus}(z) = \log(1 + e^z)$ – gładki wariant ReLU
  - sigmoid lub tanh – jeżeli wartości mają być w zadanym przedziale
- Funkcja straty: MSE, MAE lub Hubera



Klasyfikacja przy pomocy MLP

- Działa podobnie jak regresja, z drobnymi różnicami
- Klasyfikacja binarna:
  - 1 neuron wyjściowy
  - aktywacja funkcją logistyczną
- Klasyfikacja wieloetykietowa:
  - 1 neuron/etykieta
  - aktywacja funkcją logistyczną
- Klasyfikacja wieloklasowa:
  - 1 neuron/klasa
  - aktywacja funkcją softmax
- Funkcja straty: log loss