

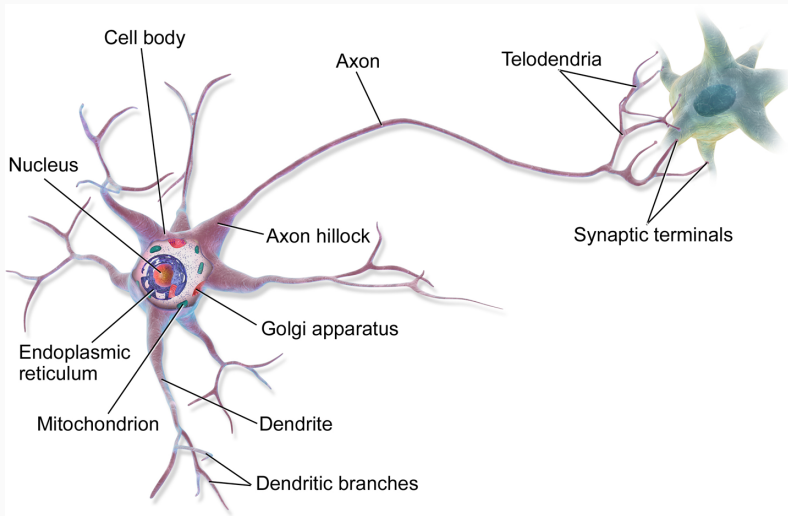
Wprowadzenie do sieci neuronowych

dr inż. Sebastian Ernst

Przedmiot: Uczenie Maszynowe

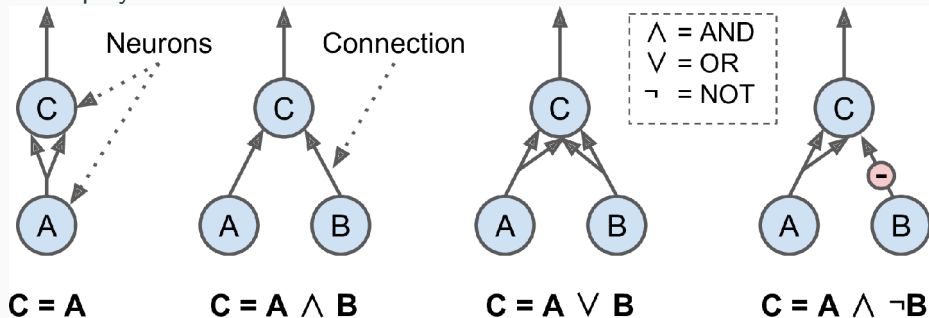
Sieci neuronowe

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:

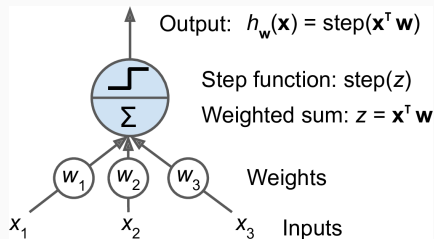


- Sieć połączonych ze sobą neuronów:
 - biologicznych (BNN)
 - sztucznych (ANN)
- Może modelować złożone procesy logiczne przy pomocy prostych co do zasady działania elementów

Perceptron

Perceptron

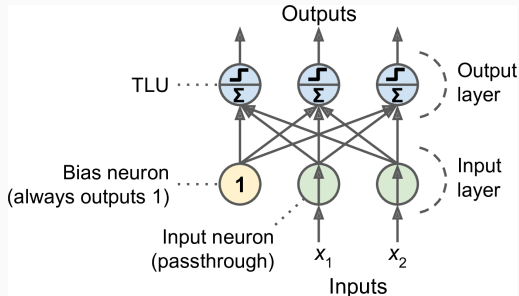
- Jedna z najprostszych architektur ANN
- Zaproponowana w 1957 przez Franka Rosenblatta
- Neuron zwany TLU (*threshold logic unit*) lub LTU (*linear threshold unit*)
- Wejścia są *liczbami*, a każde wejście posiada także *wagę*
- Sposób działania:
 1. oblicz sumę wejść:
$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n = \mathbf{x}^T \mathbf{w}$$
 2. zastosuj funkcję schodkową: $h_w(\mathbf{x}) = \text{step}(z)$



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

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

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:

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

- 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

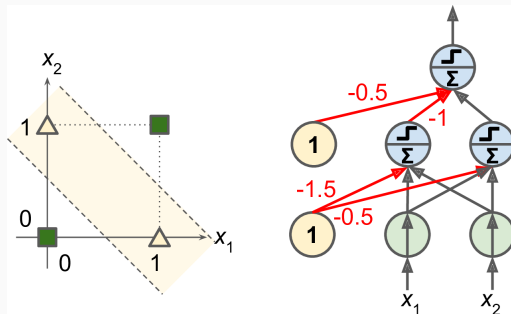
iris = load_iris()
X = iris.data[:, (2, 3)]  # petal length, petal width
y = (iris.target == 0).astype(int)

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

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

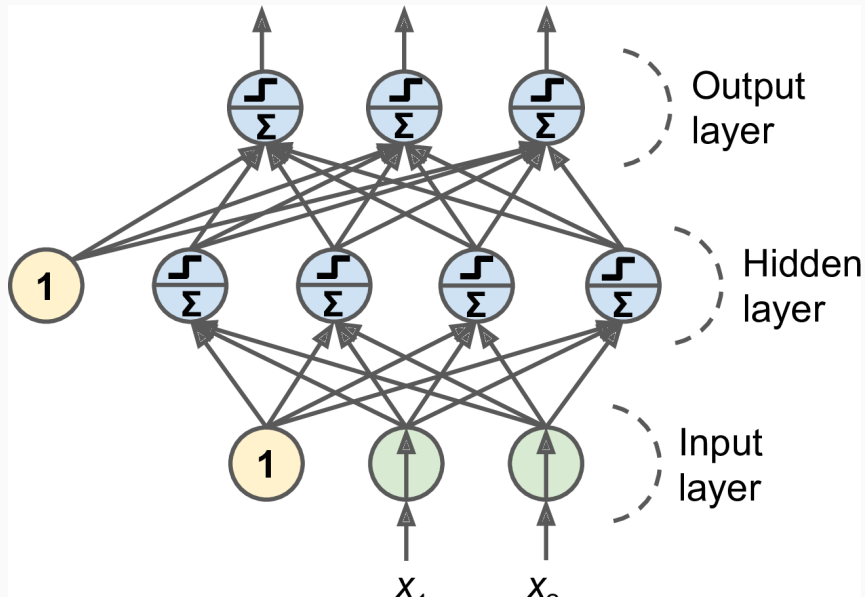
Ograniczenia perceptronu

- 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)



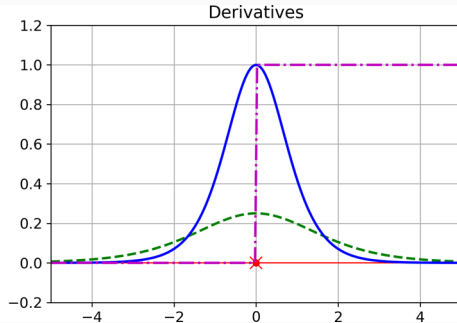
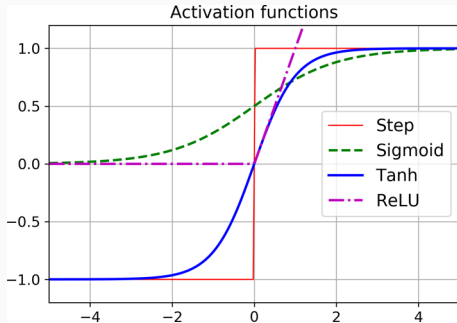
- Przez wiele lat opracowanie efektywnego algorytmu uczenia stanowiło problem
- W 1986 Rumelhart, Hinton i Williams zaproponowali algorytm *propagacji wstecznej* (*backpropagation*) – używany do dziś
- Przechodzi przez sieć raz do przodu (przeprowadzenie predykcji) i raz wstecz
- Oblicza gradient błędu 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

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

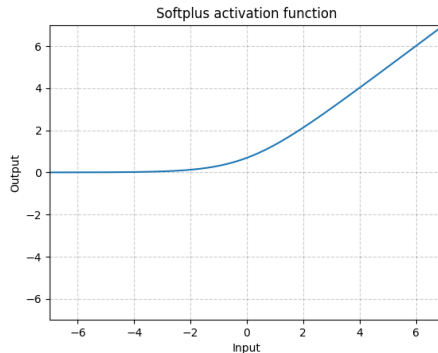
$$\tanh(z) = 2\sigma(2z) - 1$$

$$\text{ReLU}(z) = \max(0, z)$$



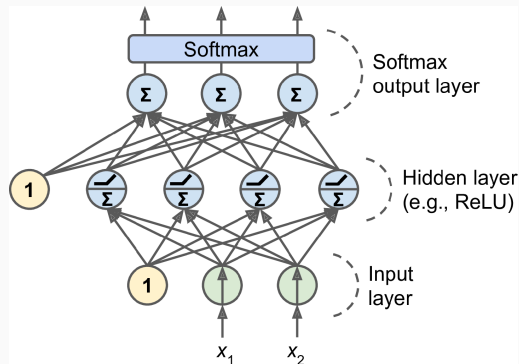
Regresja przy pomocy MLP

- 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**

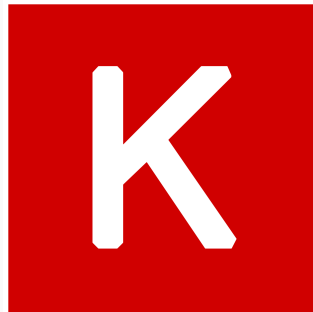


Keras + TensorFlow

- Interfejs Python do wygodnego modelowania sieci neuronowych
- Licencja Apache 2.0
- Za przetwarzanie odpowiedzialny jest backend:
 - do wersji 2.3 – TensorFlow, MS Cognitive Toolkit, Theano, PlaidML:

```
import tensorflow as tf
import keras
```
 - od wersji 2.4 – tylko TensorFlow:

```
import tensorflow as tf
from tensorflow import keras
```



- Biblioteka udostępniająca szereg narzędzi uczenia maszynowego, ale ukierunkowana na budowanie głębokich sieci neuronowych
- Stworzona przez Google
- Wsparcie dla GPU, dostępne wersje zoptymalizowane dla określonych środowisk sprzętowych (np. [Apple Metal](#))



TensorFlow

Keras Sequential API

- Zakłada, że warstwy połączone są w sekwencję (szeregowo)

- Tworzenie modelu:

```
model = keras.models.Sequential()
```

- Warstwy dodajemy jako instancje odpowiednich klas z pakietu `keras.layers`:

```
model.add(keras.layers.Flatten(input_shape=[28, 28]))  
model.add(keras.layers.Dense(300, activation="relu"))  
model.add(keras.layers.Dense(100, activation="relu"))  
model.add(keras.layers.Dense(10, activation="softmax"))
```

- Parametry często przekazujemy jako ciągi znaków, jest to zapis uproszczony: zamiast np. "relu" możemy przekazać `keras.activations.relu`.
- Normalizację robimy sami (wcześniej), lub używamy [warstwy Normalization](#):

```
normalizer = keras.layers.Normalization(  
    input_shape=[1,], axis=None)  
normalizer.adapt(y)  
# ...  
model.add(normalizer)
```

Analiza modelu

```
>>> model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 300)	235500
dense_1 (Dense)	(None, 100)	30100
dense_2 (Dense)	(None, 10)	1010

```
Total params: 266,610
```

```
Trainable params: 266,610
```

```
Non-trainable params: 0
```

```
model.compile(loss="sparse_categorical_crossentropy",  
              optimizer="sgd",  
              metrics=["accuracy"])
```

- Tu znów zamiast "sgd" moglibyśmy podać `keras.losses.sparse_categorical_crossentropy`.
- W przypadku metryki, Keras zachowa się „inteligentnie” i w oparciu o "accuracy" wybierze `keras.metrics.sparse_categorical_accuracy`.


```
history = model.fit(X_train, y_train, epochs=30,  
                    validation_data=(X_valid, y_valid))
```

- Zamiast `validation_data` moglibyśmy podać `validation_split` (np. 0.2); wtedy zbiór walidacyjny zostałby wydzielony automatycznie ze zbioru uczącego
- Jeżeli klasy nie są reprezentowane równomiernie w zbiorze danych, można zmienić ich wagi przy pomocy argumentu `class_weight`
- Podobnie, możemy „faworyzować” określone próbki przy pomocy `sample_weight`