Programowanie w Pythonie

Łukasz Mioduszewski, UKSW 2022

Wstęp do sieci neuronowych

Sieci Neuronowe - historia

- 1933 - Edward Thorndike: human learning consists in the strengthening of some (then unknown) property of neurons
- 1949 - Donald Hebb: it is specifically a strengthening of *connections* between neurons in the brain that accounts for learning

„When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

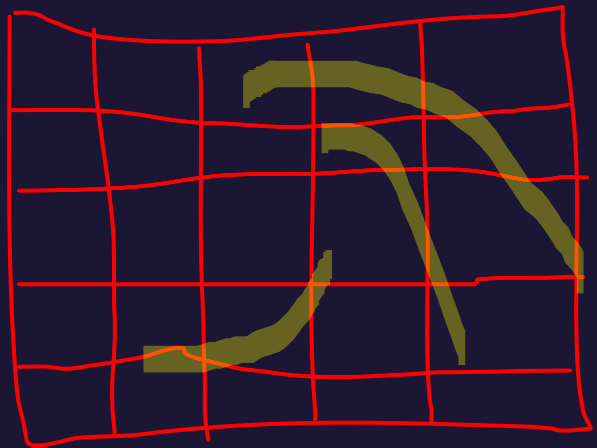
D. Hebb, The Organization of Behavior, 1949

"Neurons that fire together wire together."

(Hebb's Law)

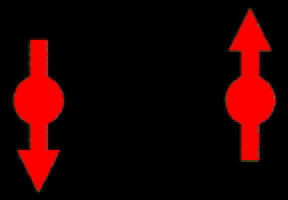
Sieć Hopfielda

- Pomysł Hopfielda (1982): jeden „neuron” odpowiada jednemu pikselowi obrazu (czarno-białego, a więc ma wartość +1 albo -1)
- Każdy neuron może być „połączony” z każdym przy użyciu macierzy wagi **w**
- Dążymy do minimalizacji energii



$L=5$ $\quad M=L\times L$

$x_i = \pm 1$



$$E = -\frac{1}{2}\sum_{ij} w_{ij} x_i x_j$$

Uzupełnienie matematyczne

- Iloczyn zewnętrzny

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

- Macierz jednostkowa

$$\mathbb{1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Sieć Hopfielda

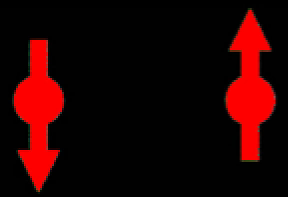
- „uczymy” sieć nowych wzorów, może ich być n:

"Neurons that fire together, wire together. Neurons that fire out of sync, fail to link".

$$\mathbf{w} = \frac{1}{n}\sum_{\mu=1}^{n}\left(\mathbf{x}^{\mu} \otimes \mathbf{x}^{\mu} - \mathbf{1}\right)$$

To są macierze!

$$x_i = \pm 1$$



$$E = -\frac{1}{2}\sum_{ij} w_{ij} x_i x_j$$

- Takie same wartości dadzą dodatnie w, czyli ujemną energię

Sieć Hopfielda

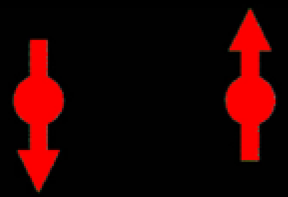
- energia i-tego spinu:

$$E_i = -\frac{1}{2} x_i \left(\sum_j w_{ij} x_j \right) = -\frac{1}{2} x_i h(i)$$

- Ewolucja w czasie:

$$\mathbf{x}(t+1) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x}(t))$$

$$x_i = \pm 1$$



$$E = -\frac{1}{2} \sum_{ij} w_{ij} x_i x_j$$

Sieć Hopfielda w praktyce

- Zadanie: prosty OCR
  (Optical Character Recognition)

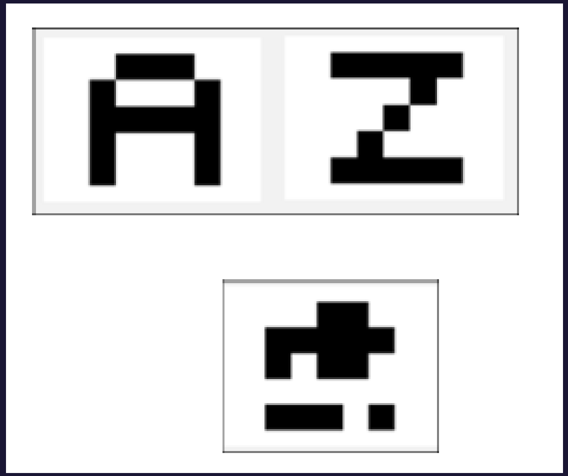
- Wprowadzamy dwa wzory do nauczenia się: litery A i Z

- Reprezentujemy je jako obrazki 5x5 pikseli

- Zaczynamy z czegoś zdeformowanego

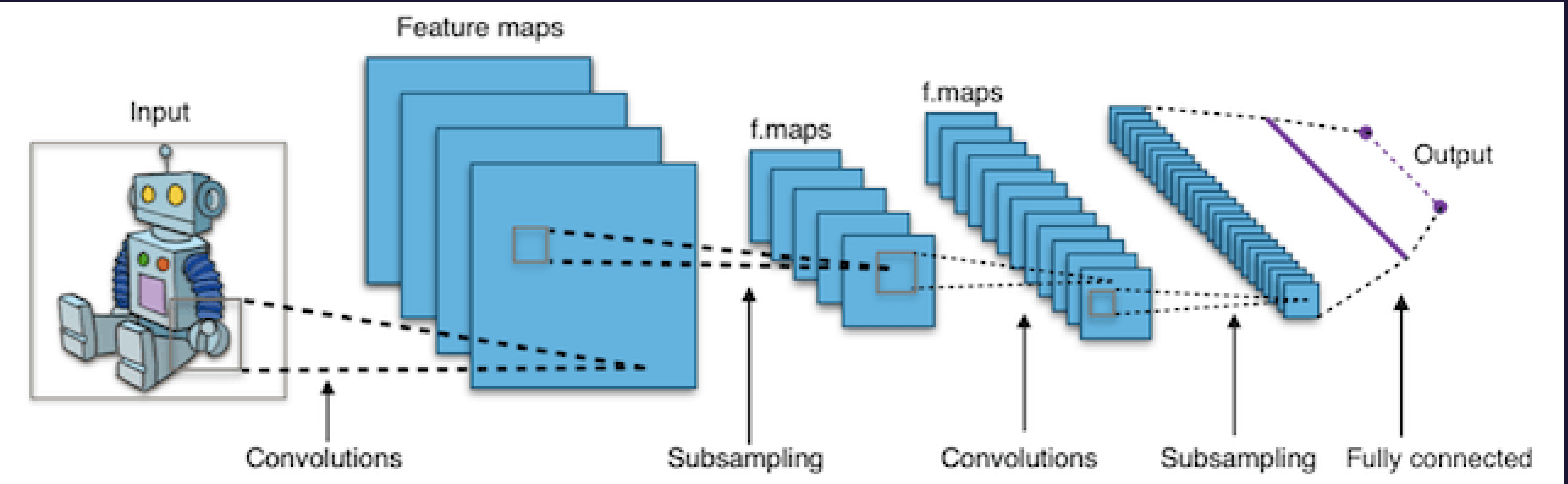
```
A = """
OXXXO
XOOOX
XXXXX
XOOOX
XOOOX
"""
```

```
Z = """
XXXXX
OOOXO
OOXOO
OXOOO
XXXXX
"""
```



Sieci neuronowe dziś

- Każdy neuron pobiera **dane** (liczby), każdą daną mnoży przez **wagę**, czego wynikiem jest liczba, która staje się daną dla kolejnej warstwy neuronów
- **Deep** learning – na tyle wiele warstw że nie wiemy co się dzieje



Biblioteka Keras

- Wysokopoziomowy "frontend" do TensorFlow od Google
- Alternatywa: FastAI ("frontend" do PyTorch od Facebook/Meta)
- Oba są open source

Biblioteka Keras

- Wymagania:
  - Python 3+
  - SciPy z NumPy
  - Matplotlib (tylko do rysowania, niekonieczny)
- Instalacja przez pip:

```
pip install tensorflow # zwykle wystarczy
pip install keras           # jeśli nie, to jeszcze to
```

- Instalacja przez Anacondę: https://elitedatascience.com/keras-tutorial-deep-learning-in-python

Początek skryptu

keras_CNN_example.py

```
import numpy as np
np.random.seed(123)  # for reproducibility
from keras.models import Sequential # sequential neural network
from keras.layers import Dense, Dropout, Activation, Flatten # usual layers
from keras.layers import Convolution2D, MaxPooling2D # to train on image data
from keras.utils import np_utils # some utilities
```

Ładowanie bazy danych do uczenia sieci

keras_CNN_example.py

```
import numpy as np
np.random.seed(123)  # for reproducibility
from keras.models import Sequential # sequential neural network
from keras.layers import Dense, Dropout, Activation, Flatten # usual layers
from keras.layers import Convolution2D, MaxPooling2D # to train on image data
from keras.utils import np_utils # some utilities

from keras.datasets import mnist
# Load pre-shuffled MNIST data into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Ładowanie bazy danych do uczenia sieci

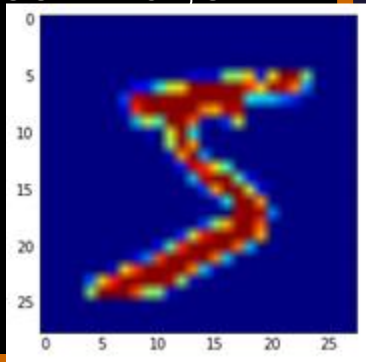
keras_CNN_example.py

```
import numpy as np
np.random.seed(123)  # for reproducibility
from keras.models import Sequential # sequential neural network
from keras.layers import Dense, Dropout, Activation, Flatten # usual layers
from keras.layers import Convolution2D, MaxPooling2D # to train on image data
from keras.utils import np_utils # some utilities

from keras.datasets import mnist
# Load pre-shuffled MNIST data into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print( X_train.shape ) # 60k samples, each is a 28x28 black-and-white image
# (60000, 28, 28)

# X_train to dane do treningu, y_train to poprawne wyniki dla tych danych
# X_test to dane testujące sieć, y_test to pożądany wynik
```

Ładowanie bazy danych do uczenia sieci

keras_CNN_example.py

```
import numpy as np
np.random.seed(123)  # for reproducibility
from keras.models import Sequential # sequential neural network
from keras.layers import Dense, Dropout, Activation, Flatten # usual layers
from keras.layers import Convolution2D, MaxPooling2D # to train on image data
from keras.utils import np_utils # some utilities

from keras.datasets import mnist
# Load pre-shuffled MNIST data into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print( X_train.shape ) # 60k samples, each is a 28x28 black-and-white image
# (60000, 28, 28)
from matplotlib import pyplot as plt
plt.imshow(X_train[0])
```



Preprocessing inputu

keras_CNN_example.py

```
# tensorflow must know how many channes there are, so the last dimension
should be the number of channels (in this case, one)
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
print( X_train.shape )
# (60000, 28, 28, 1)
X_train = X_train.astype('float32') # input data must be float32
X_test = X_test.astype('float32')
X_train /= 255  # input data must be in range from 0 to 1
X_test /= 255
```

Preprocessing outputu

keras_CNN_example.py

```
# tensorflow must know how many channes there are, so the last dimension
should be the number of channels (in this case, one)
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
print( X_train.shape )
# (60000, 28, 28, 1)
X_train = X_train.astype('float32') # input data must be float32
X_test = X_test.astype('float32')
X_train /= 255  # input data must be in range from 0 to 1
X_test /= 255

print( y_train[:10] )
# [5 0 4 1 9 2 1 3 1 4] # y_train tells which digit is showed in every picture
# Convert 1-dimensional class arrays to 10-dimensional class matrices
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
print( Y_train.shape )  # (60000, 10)
```

Konwolucja

- Konwolucja pomiędzy obrazkiem a filtrem/jądrem konwolucji (convolution kernel/filter, tak jak te okna Hanninga albo Blackmana)

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy)$$

- g to wyjściowy obraz, f to wejściowy obraz, w to filtr

	0	1	2	3
0	25	80	110	5
1	80	$\boxed{100}$	120	7
2	92	60	30	25
3	93	97	23	6

Kernel (sharpen)

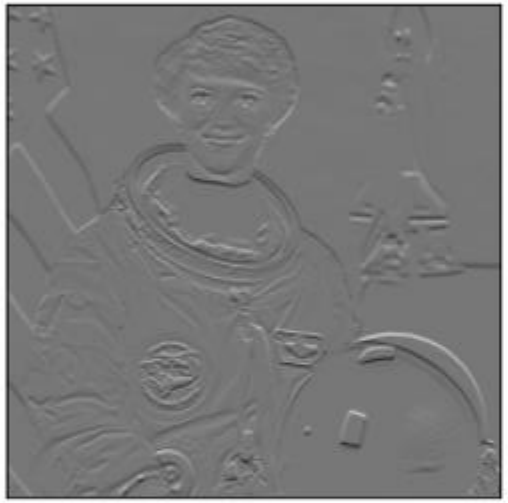
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

	0	1	2	3
0				
1		$\boxed{160}$		
2				
3				

Konwolucja

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



Name	Kernel	Image Result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Mean Blur	$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$	
Laplacian	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
Gaussian Blur	$\begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$	

Architektura sieci

keras_CNN_example.py

```
model = Sequential()

# defining input layer. input_shape should be the shape of 1 sample.
# here it’s (28, 28, 1) = (width, height, channels) of each digit image.
model.add(Convolution2D(32, (3,3), activation='relu', input_shape=(28,28,1)))
# 32 convolution filters, each of them is a 3x3 matrix
print( model.output_shape )
# (None, 26, 26, 32) corresponds to (samples, new_rows, new_cols, filters)
# model will output all samples, convoluted into 26x26 array using 32 filters

# adding more layers
model.add(Convolution2D(32,(3,3),activation='relu')) # regularization max(x,0)

model.add(MaxPooling2D(pool_size=(2,2))) # way to reduce number of parameters
in our model by sliding a 2×2 pooling filter across the previous layer and
taking the max of the 4 values in the 2×2 filter
model.add(Dropout(0.25)) # killing some neurons
```

Architektura sieci – idziemy głębiej

keras_CNN_example.py

```
model.add(Flatten()) # flattens the input (1 channel)
model.add(Dense(128, activation='relu')) # 128 is the output size
# keras automatically matches layer input/output sizes
model.add(Dropout(0.5)) # killing excessive neurons again
model.add(Dense(10, activation='softmax')) # normalize the output
# to a probability distribution over predicted output classes
```

Kompilacja modelu

keras_CNN_example.py

```
model.add(Flatten()) # flattens the input (1 channel)
model.add(Dense(128, activation='relu')) # 128 is the output size
# keras automatically matches layer input/output sizes
model.add(Dropout(0.5)) # killing excessive neurons again
model.add(Dense(10, activation='softmax')) # normalize the output
# to a probability distribution over predicted output classes

model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])
 # loss function defines the "cost" associated with good/bad image recognition
```

Trenujemy nasz model

keras_CNN_example.py

```
model.add(Flatten()) # flattens the input (1 channel)
model.add(Dense(128, activation='relu')) # 128 is the output size
# keras automatically matches layer input/output sizes
model.add(Dropout(0.5)) # killing excessive neurons again
model.add(Dense(10, activation='softmax')) # normalize the output
# to a probability distribution over predicted output classes

model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])
 # loss function defines the "cost" associated with good/bad image recognition

model.fit(X_train, Y_train,
          batch_size=32, epochs=10, verbose=1)
# Epoch 1/10
# 7744/60000 [==>.................] - ETA: 96s - loss: 0.5806 - acc: 0.816

```

Testujemy nasz model

keras_CNN_example.py

```
model.add(Flatten()) # flattens the input (1 channel)
model.add(Dense(128, activation='relu')) # 128 is the output size
# keras automatically matches layer input/output sizes
model.add(Dropout(0.5)) # killing excessive neurons again
model.add(Dense(10, activation='softmax')) # normalize the output
# to a probability distribution over predicted output classes

model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])
 # loss function defines the "cost" associated with good/bad image recognition

model.fit(X_train, Y_train,
          batch_size=32, epochs=10, verbose=1)
# Epoch 10/10

score = model.evaluate(X_test, Y_test, verbose=0)
```

A jak potem tego używać?

- y_krm = model.predict(x)
- Można też używać metody call(), ale są pewne subtelne różnice: https://stackoverflow.com/questions/60837962/confusion-about-keras-model-call-vs-call-vs-predict-methods?fbclid=IwAR0wAIRHfSa2o6-qriOgLp64lfXhNtLr8kk8f9jF9Q67_hdgJVvjzM5MBW8
- Aby zapisać model:
  ```
  model = ...  # Get model (Sequential, Functional Model, or Model subclass)
  model.save('path/to/location')
  ```
- Aby odczytać model:
  ```
  from tensorflow import keras
  model = keras.models.load_model('path/to/location')
  ```

Bibliografia

- https://elitedatascience.com/keras-tutorial-deep-learning-in-python
- https://www.knowledgehut.com/blog/data-science/pytorch-vs-tensorflow
- https://medium.com/@ianormy/convolution-filters-4971820e851f
- https://www.activestate.com/resources/quick-reads/what-is-a-keras-model/