# Simple Neural Networks in PyTorch vs Tensorflow

## Fast intro for beginners

Krzysztof Podlaski

Seminarium
Katedry Systemów Inteligentnych
Uniwersytet Łódzki
31 Marca 2023

W have two main Machine Learning tools in python:

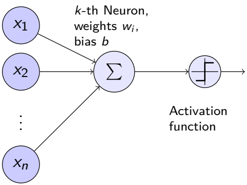
- PyTorch – library for advanced programmers, more control, but also more details and coding.
- Tensorflow – much nicer library for the start.

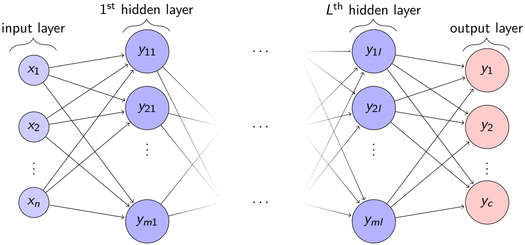
Both support GPU computing, PyTorch is said to do better in distributed GPU systems.

# Dense network

Dense networks are the most classical architectures. MLP – Multilayer Perceptron.
We have set of neurons, each have inputs and weights, sums them up, add bias and activation function.

$$y = f(\sum_i x_i w_i + b)$$

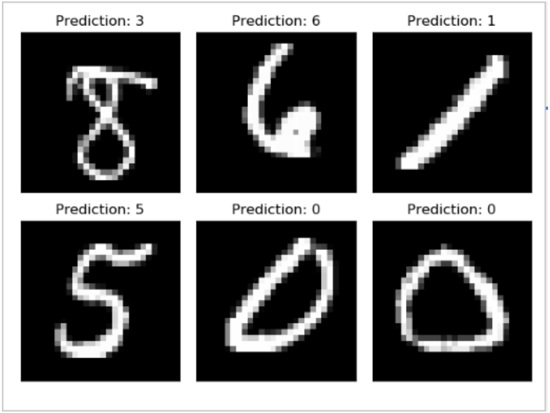




pictures based on https://tex.stackexchange.com/questions/104334/tikz-diagram-of-a-perceptron and https://tikz.net/neural_networks/

# Dataset – MNIST

- contains images of digits
- each image 28x28 gray scale
- all have assigned labels
- incorporated with PyTorch and Tensorflow



# Basic info about implementation

- All codes are on github:
  `https://github.com/kpodlaski/NeuralNetworksIntro`
- Dense network are in directory: examples → DenseNetwork
- Tensorflow is in: examples → DenseNetwork → tensorflow
- PyTorch is in: examples → DenseNetwork → pytorch
  and common → pytorch
- Task: digit recognition (classification)

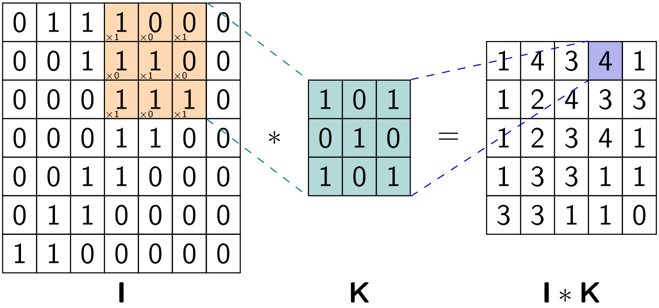
# Dense network architecture (MLP)

Implemented in both libraries

- input layer – 794 signals (28 × 28)
- 1st layer – 320 neurons tanh activation
- 2nd layer – 240 neurons sigmoid activation
- 3rd layer – 120 neurons relu activation
- out layer – 10 neurons softmax activation

# Convolution Layer

$$(I * K)_m, n = \sum_i \sum_k I_{j,k} K_{m-j,n-k} \tag{1}$$



**I**

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |

**K**

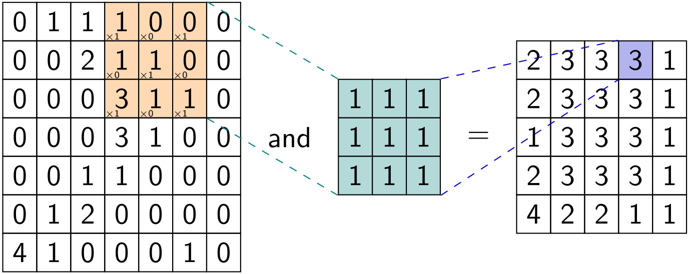
| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

**I * K**

| | | | | |
|---|---|---|---|---|
| 1 | 4 | 3 | 4 | 1 |
| 1 | 2 | 4 | 3 | 3 |
| 1 | 2 | 3 | 4 | 1 |
| 1 | 3 | 3 | 1 | 1 |
| 3 | 3 | 1 | 1 | 0 |

picture based on `https://tikz.net/conv2d/`

# Pooling layer



|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 ×1 | 0 ×0 | 0 ×1 | 0 |
| 0 | 0 | 2 | 1 ×0 | 1 ×1 | 0 ×0 | 0 |
| 0 | 0 | 0 | 3 ×1 | 1 ×0 | 1 ×1 | 0 |
| 0 | 0 | 0 | 3 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 2 | 0 | 0 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 1 | 0 |

$\mathbf{I}$

and

|  |  |  |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

MaxPool$(3, 3)$

$=$

|  |  |  |  |  |
|---|---|---|---|---|
| 2 | 3 | 3 | 3 | 1 |
| 2 | 3 | 3 | 3 | 1 |
| 1 | 3 | 3 | 3 | 1 |
| 2 | 3 | 3 | 3 | 1 |
| 4 | 2 | 2 | 1 | 1 |

MaxPool$(3, 3)\mathbf{I}$

# Convolutional network architecture (CNN)

Implemented in both libraries

- input layer – 794 signals ($28 \times 28$)
- 1st layer – 10 filters 5x5, stride 1,1, tanh
- 2nd layer – MaxPool 2x2, stride 2,2
- 3rd layer – 20 filters 5x5, stride 1,1, tanh
- 4th layer – MaxPool 2x2, stride 2,2
- 5th layer – Dense, 50 neurons, tanh
- out layer – 10 neurons softmax activation

# 1D convolution

We need to change dataset, one dimensional or time-series data is better in this case.

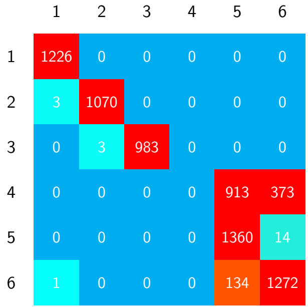
- We use UCI HAR Dataset
- It contain mobile sensors data, allow recision behaviour
- walking, walking upstairs, walking downstairs, sitting, standing, laying
- we use only global acceleration and as a total i.e a $= \sqrt{a_x^2 + a_y^2 + a_z^2}$
  - its not the best approach, but serves for this tutorial
  - we should join three last classes as they should not be distinguished by acceleration
  - code connected with data preparation is in file:
    examples → Conv1D → read_dataset.py
- for PyTorch we prepare data for DataLoader
- for Tensorflow we use OneHotEncoding

Implemented in both libraries

- input layer – 128 signals (128 acc measurements)
- 1st layer – Conv 1D, 64 filters 10x1, relu
- 2nd layer – Conv 1D, 64 filters 10x1, relu
- 3rd layer – Dropout (.15)
- 4th layer – MaxPool1D 2, stride 2
- 5th layer – Dense, 100 neurons, tanh
- out layer – 6 neurons softmax activation

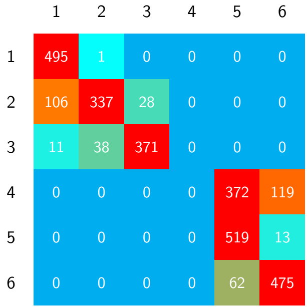
# Classification results analysis

Confusion matrix is the easiest way to analyse the effects of our ML system



|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1226 | 0 | 0 | 0 | 0 | 0 |
| 2 | 3 | 1070 | 0 | 0 | 0 | 0 |
| 3 | 0 | 3 | 983 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 913 | 373 |
| 5 | 0 | 0 | 0 | 0 | 1360 | 14 |
| 6 | 1 | 0 | 0 | 0 | 134 | 1272 |

(a) Confusion matrix for train set, accuracy: 5911/7352(80%)



|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 495 | 1 | 0 | 0 | 0 | 0 |
| 2 | 106 | 337 | 28 | 0 | 0 | 0 |
| 3 | 11 | 38 | 371 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 372 | 119 |
| 5 | 0 | 0 | 0 | 0 | 519 | 13 |
| 6 | 0 | 0 | 0 | 0 | 62 | 475 |

(b) Confusion matrix for test set, accuracy: 2197/2947(75%)

# Advanced tasks

This is shown only in PyTorch, but can be done (probaby) in Tensorflow. Advanced analysis of the network, activations etc.

- We can always get weight and biases for our layers
- We can watch "live" activations during feed forward pass
  - Hook a layer,
  - Hook can be added to layer or layer after activation function is applied