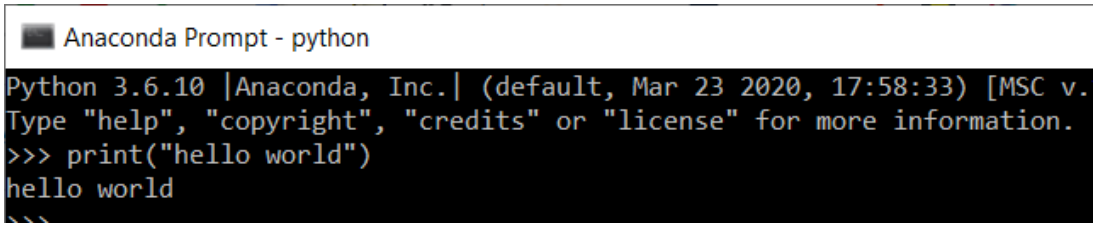
Machine Learning with Python

Intro

- Why Python?
- Running Python locally



- **`scripts.py`** vs. **`notebooks.ipynb`**
- Running Jupyter notebooks in the cloud, e.g. collaboratory
- GitHub

Schedule

#	date	epic	notebook	resouces, key learnings
1			10_Colab_intro.ipynb	
2	Tu. 19.05.		12_Logfile_challenge.ipynb	
3	13:00			
4				
5			15_Logfiles_w_NumPy.ipynb	15_NumPy.ipynb
6	We. 20.05.			16_Matplotlib.ipynb
7	13:00		17_Logfiles_w_Pandas.ipynb	17_Pandas.ipynb
8				(18_object_oriented_programming.ipynb)
9	Fr. 22.05.		21_machine_learning_intro.ipynb	generalization, overfitting
10	14:00			splitting test data, correlation
11	Fr. 29.05.		22_end2end_ml_project.ipynb	
12	14:00			
13	Tu. 02.06.			
14	14:00			
15	Fr. 05.06.			
16	14:00			

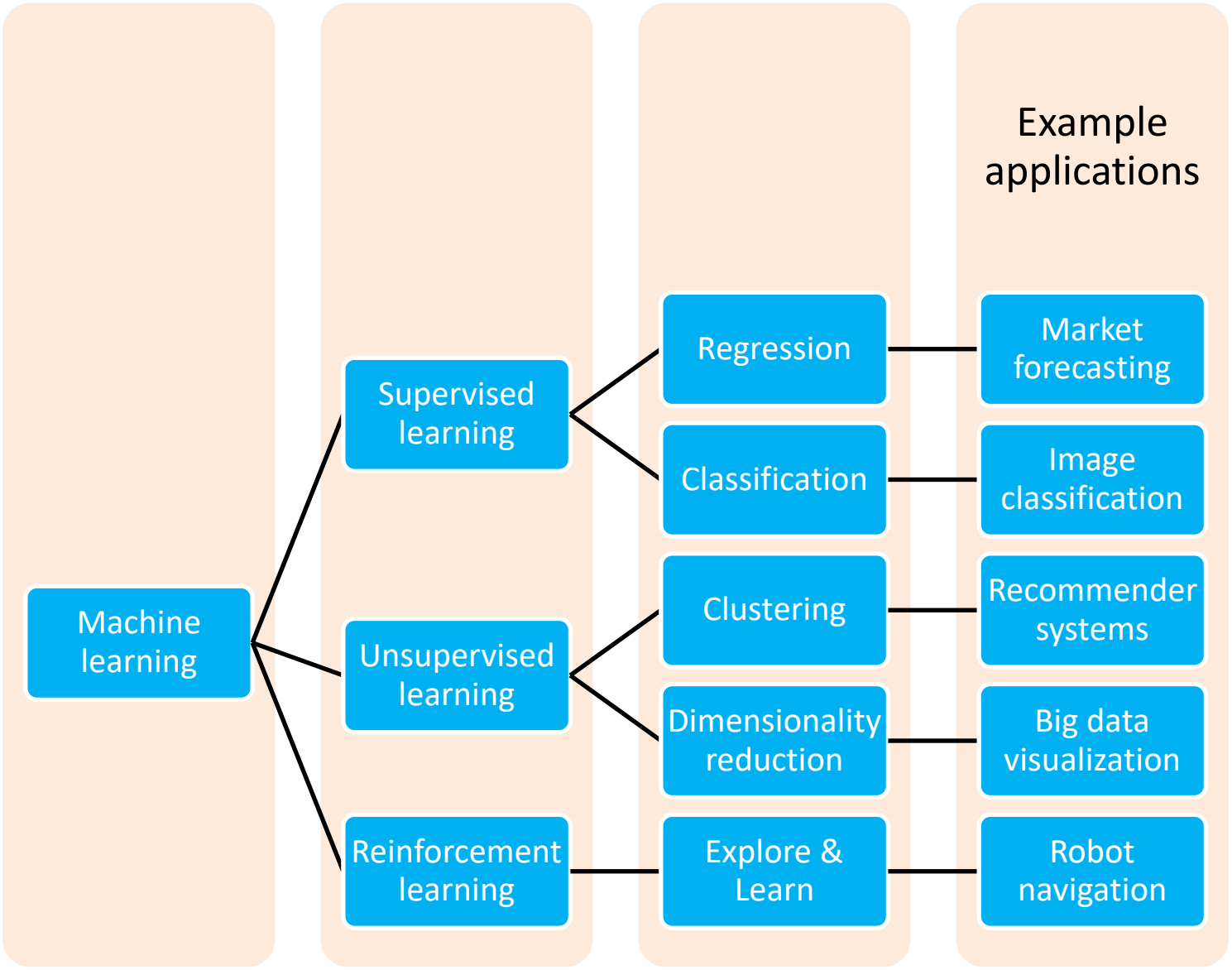
Tools and Notebooks

- Jupyter notebooks and datasets are available on GitHub: https://github.com/munich-ml/MLPy2020
- Naming conventions:
  - 12_Logfile_challenge**_blank**.ipynb — empty version to start with
  - 12_Logfile_challenge**_dirty**.ipynb — dirty version from life session
  - 12_Logfile_challenge.ipynb — clean version published later

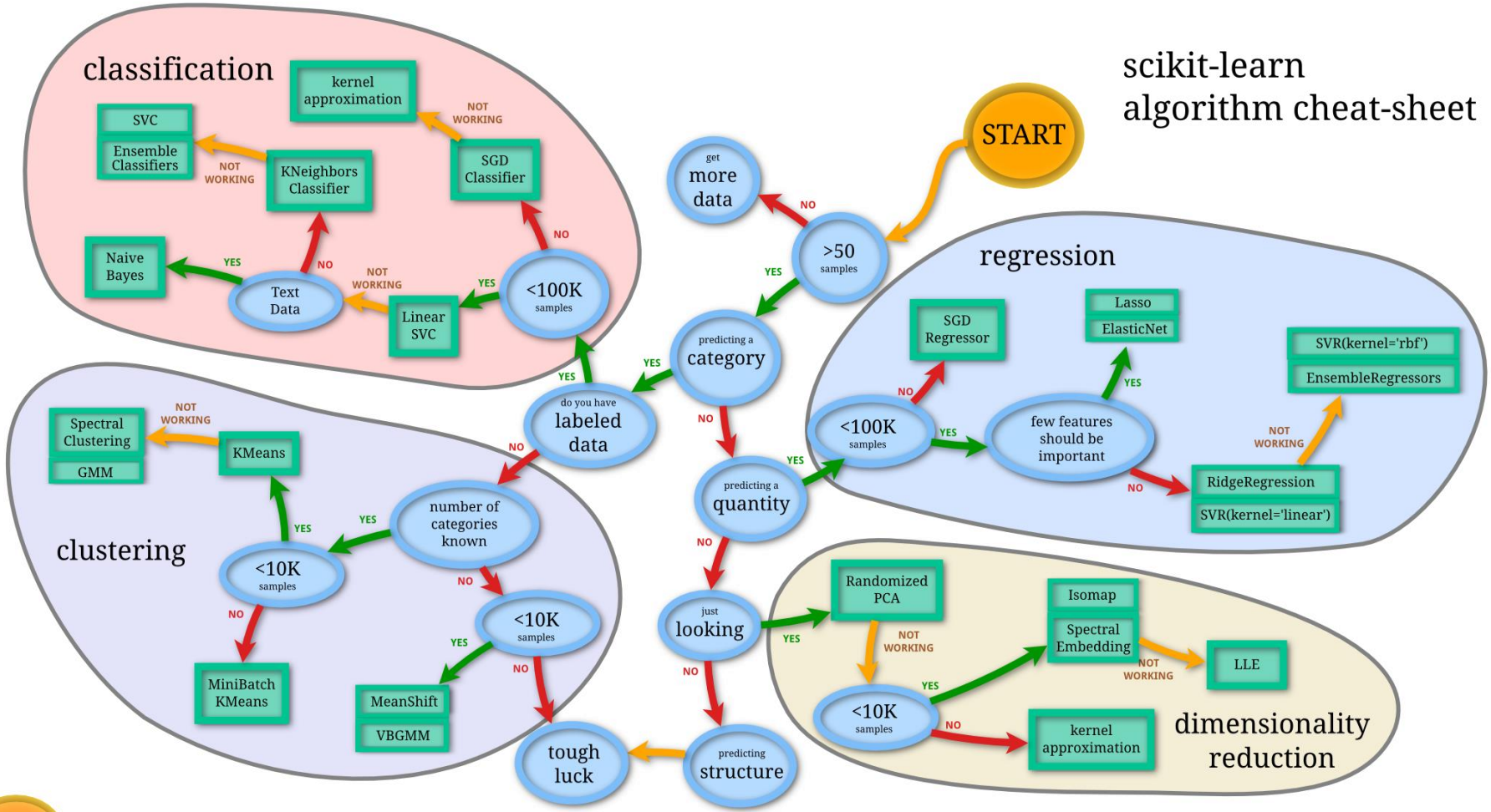
Colab notebook environment

- https://colab.research.google.com/
- Open notebook templates from GitHub:
  - File >> open notebook >> GitHub >> munich-ml/MLPy2020
- Save your own work on your Google Drive:
  - File >> Save a copy to Drive

Machine learning



Choosing the right estimator



Back

Source: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Neural networks
Computer vision with TensorFlow

Classification with neural networks

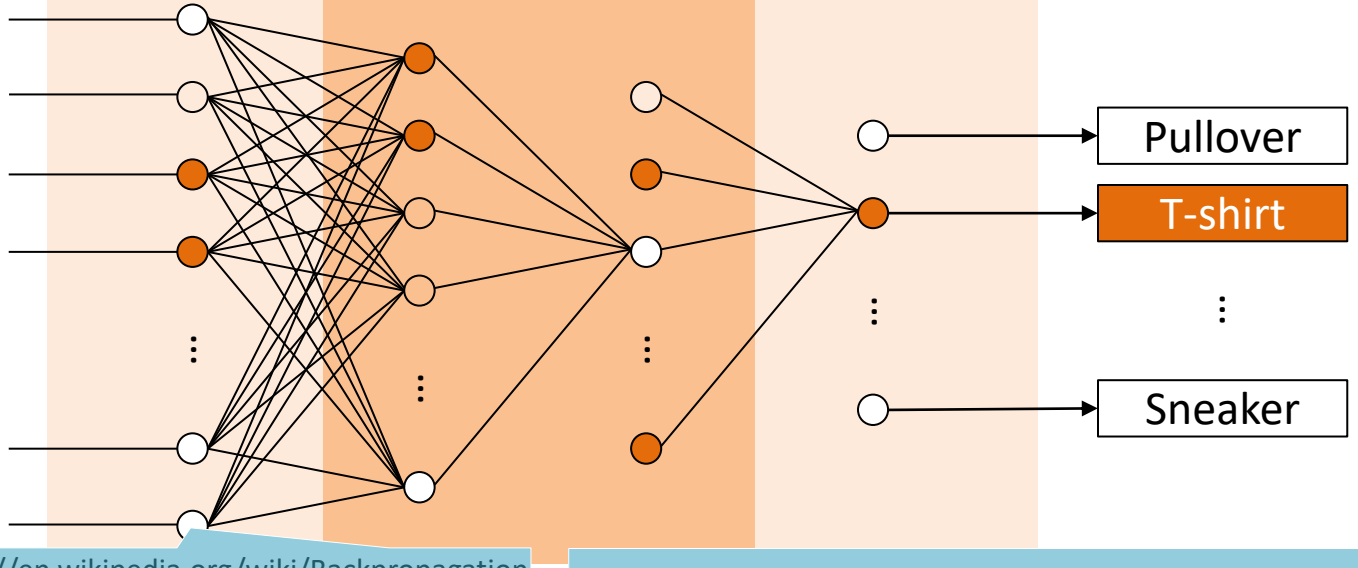
Input
Images
28 * 28 pixel
8 bit/pixel

Input layer
786 neurons
one per pixel

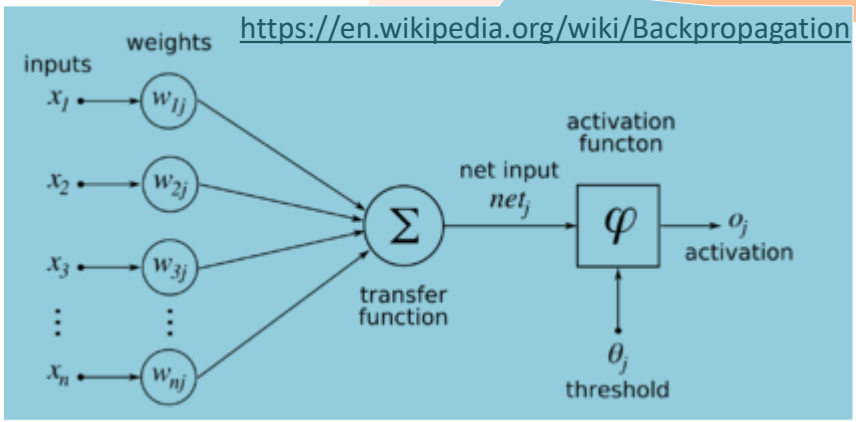
hidden layers
2 in this
example

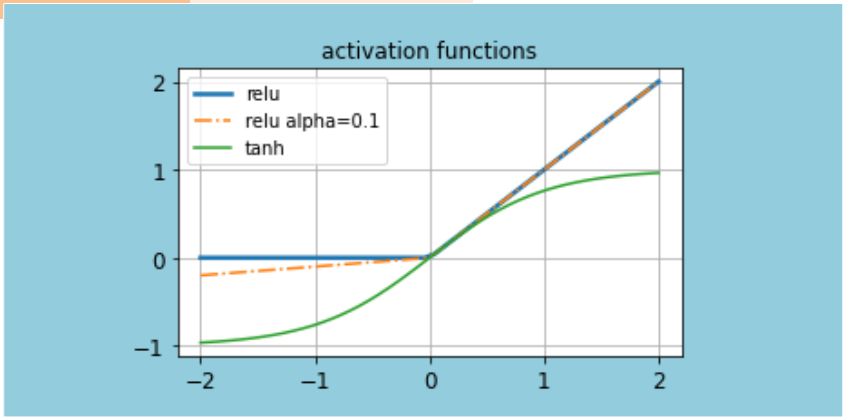
Output layer
10 neurons
one per class

Output
10 classes



https://en.wikipedia.org/wiki/Backpropagation





Training a neural network

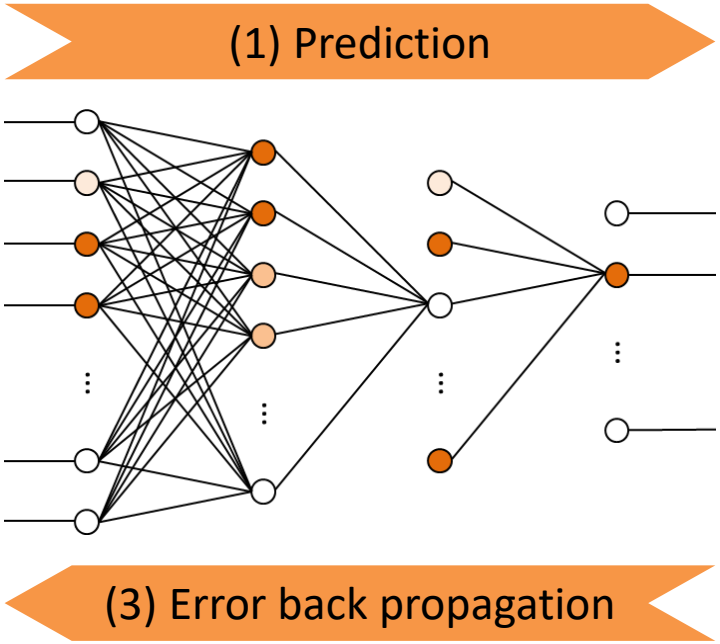
- Training: **Learning a function**/model that best maps the training dataset
- Neural net training: Finding weights & biases that minmize the cost function

Training algorithm: **Back propagation** in 3 steps:

(1) Feed forward a batch of example images through the model

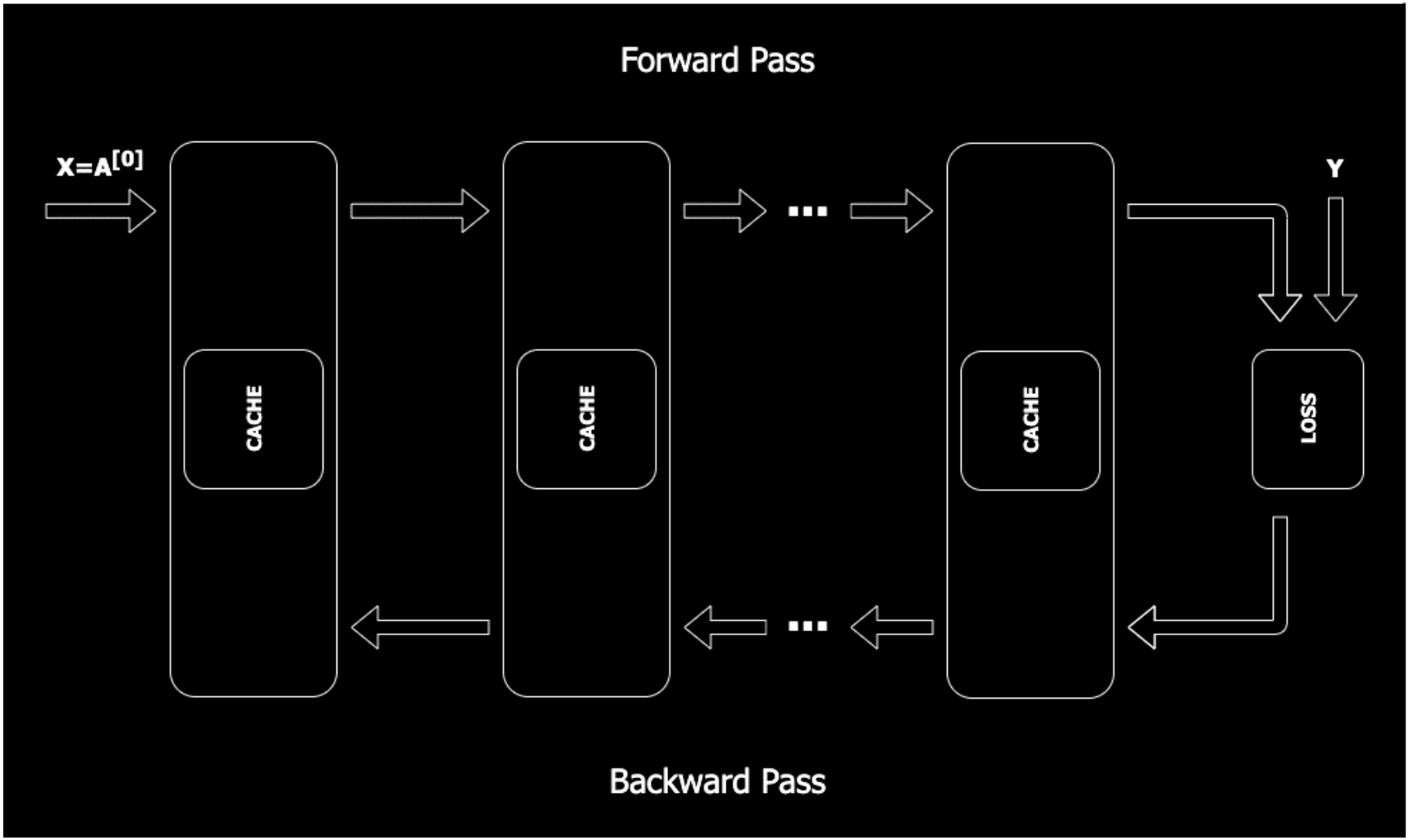
(2) Compute error from labels and predictions
- cost function $E = (y_{PRED} - y_{TRUE})^2$
- Effects of weights on the error `dE/dWeight`

(3) Propagate error through the model
- modify weights by their effect on the error `-lr * dE/dWeight`
- **Learning rate** `lr`



- **Epoch** is the training over the full dataset, executed in batches

Backpropagation animation

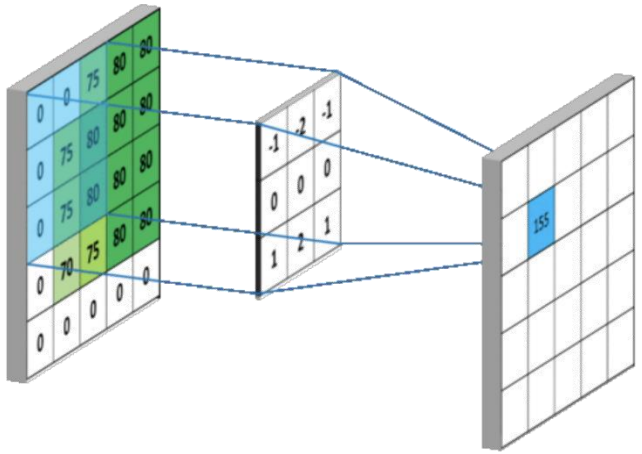


Source: https://towardsdatascience.com/lets-code-convolutional-neural-network-in-plain-numpy-ce48e732f5d5

Convolutional Neural Networks

Convolutional neural networks (CNN)

- **Disadvantages** of (fully connected) **neural nets (NN)** for image processing
  - Don‘t scale well (many features; prone to overfitting)
  - Disregard pixel distance (neighborhood have semantic meaning)
  - Objects of interest are detected at certain positions, only
- **Convolutional neural networks** (CNN) consist of 3 basic layer types:
  - **Convolutional layers**
    - Trainable filters measure the presence certain patterns / objects
    - The object may occur anywhere
  - **Pooling layers**
    - Downsize resolution
    - focus (on the maximum)
  - **Fully connected layers**
    - Aggregate information as higher represention



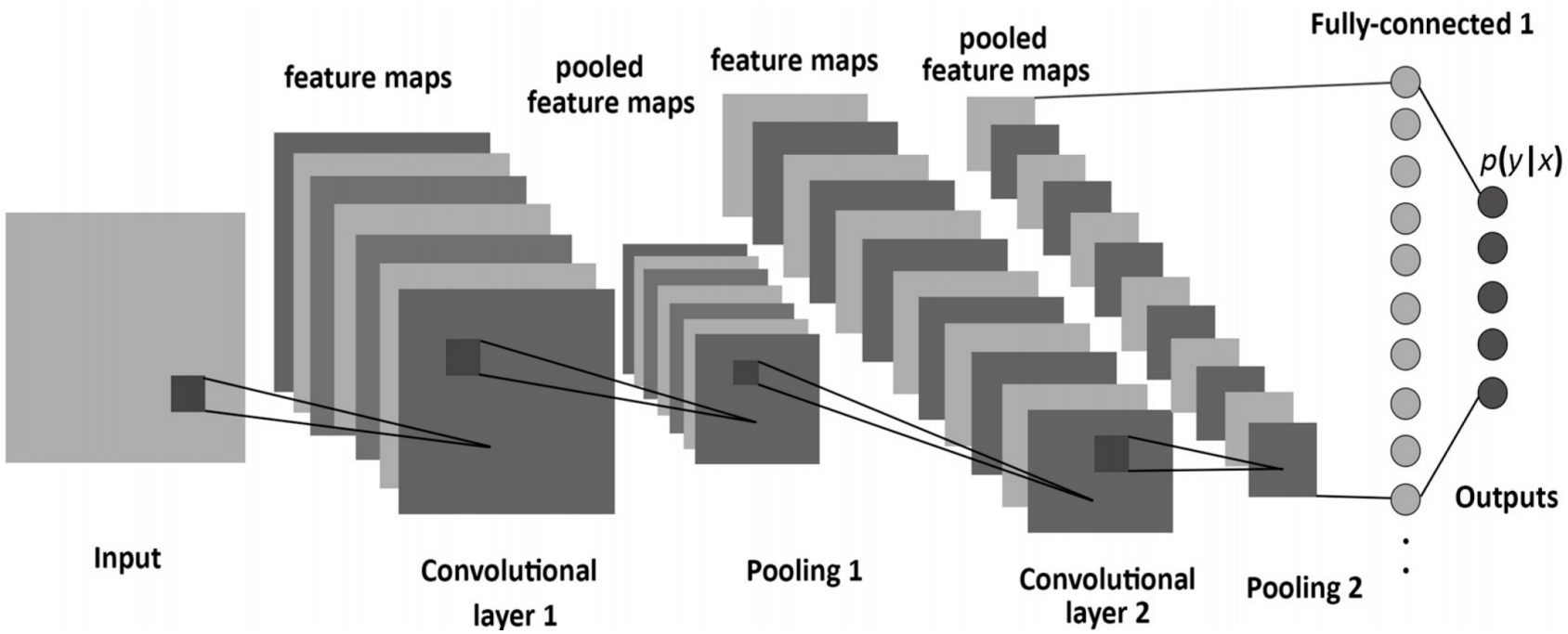
Source: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

Convolutional neural networks (CNN)

low conv layers
learn basic shapes like edges, corners, circles, textures,…

middle conv layers
learn parts of object like eyes, fingers, wheels, leafs,…

higher dense layers
learn to recognize full objects, in different shapes and positions



Interactive 3D CNN on MNIST: https://www.cs.ryerson.ca/~aharley/vis/conv/

Source: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac