

Deep Neural Networks

- Software and Material for our last Lecture on Deep Neural Networks

- The **zenon system** is setup for running various python3 based machine learning libraries using the anaconda3 framework. Anaconda3 is installed system wide and the setup is installed in `/local/anaconda3`

The environment setup for machine learning is called `myML` and uses python version 3.8 with the following packages `numpy matplotlib pandas iminuit scipy`

`scikit-learn tensorflow-gpu keras-gpu seaborn pytorch jupyter`

There is a very small graphic card, a Quadro P1000, which allows to use the machine learning library Tensorflow/Keras with GPU support.

- In a terminal start conda and enable our machine learning environment by executing:

```
source /local/kurs/SetPathforZenonAnaconda.sh
```

```
conda activate myML
```

```
conda info
```

you can now execute python3 scripts in your shell or start jupyter to run notebooks in a web browser.

- Credits: The talk contains material of a tutorial at the 3rd IML workshop 2019 at CERN by Yannik Rath and the Train-the-Trainer Workshop "Deep Learning Basics" in February 2023

Introduction

- ChatGPT's opinion on the relation of Artificial Intelligence (AI) Machine Learning (ML) and Deep Learning

AI (Artificial Intelligence) is a broad field that encompasses a range of techniques and approaches for developing intelligent systems. Machine Learning (ML) is a subset of AI that focuses on teaching machines to learn from data and make predictions or decisions without being explicitly programmed. Deep Learning (DL) is a subset of ML that uses artificial neural networks with many layers to learn increasingly abstract features from data, and has achieved significant breakthroughs in areas such as computer vision, natural language processing, and robotics. Therefore, DL is a specific type of ML, and ML is a specific type of AI.

Question: Where are you in this hierarchy?

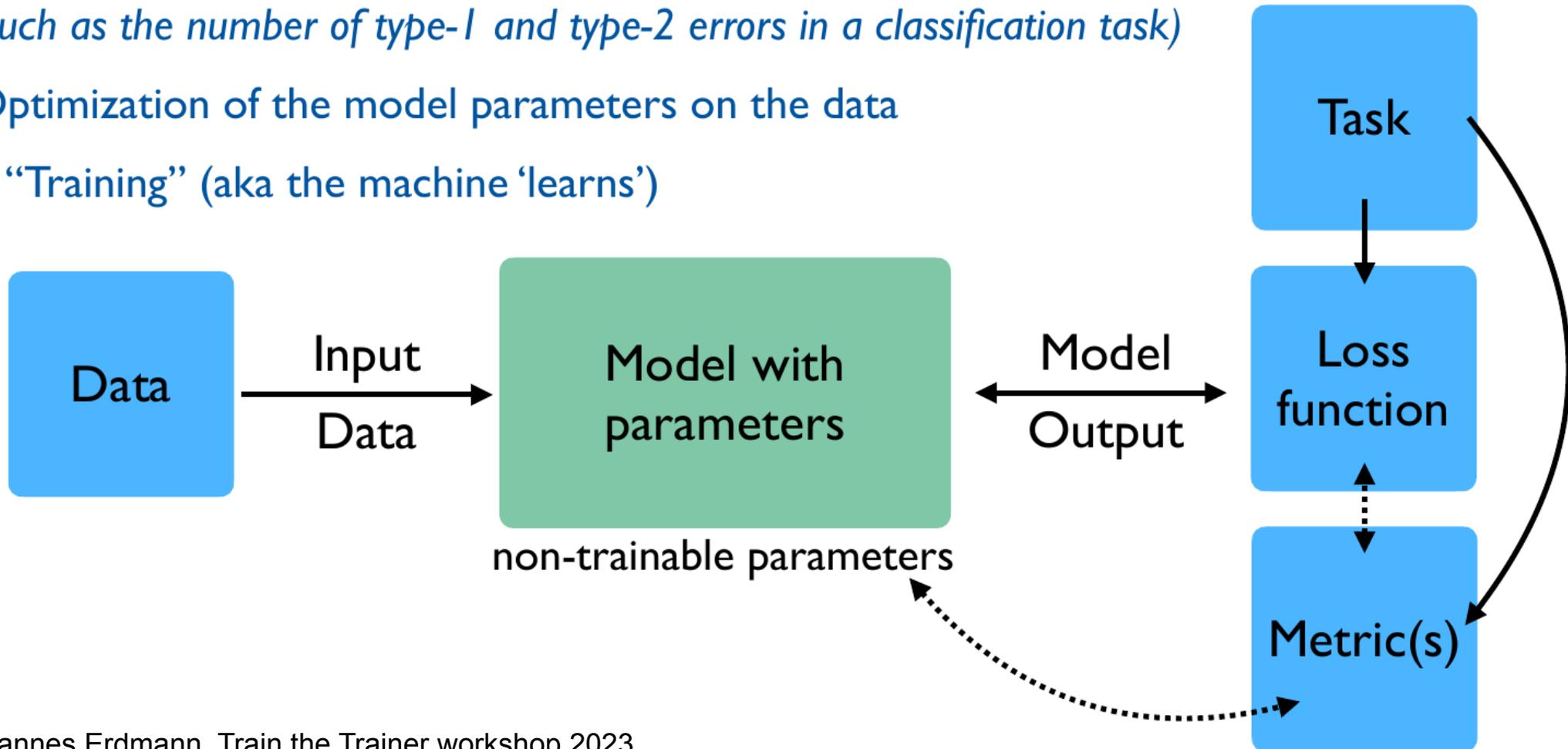
As an AI language model, I am built using various ML techniques, such as deep learning and natural language processing. Therefore, I am a subset of AI, specifically falling under the category of ML-based AI.

- Stable Diffusion is a latent text-to-image diffusion model capable of generating photo-realistic images given any text input



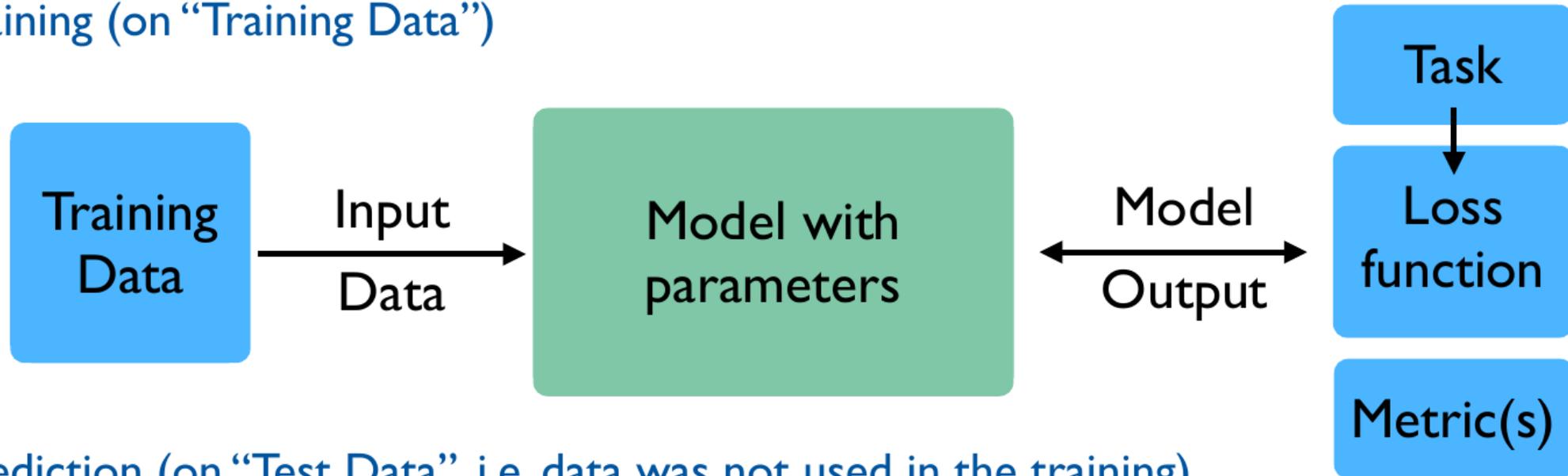
Introduction Machine Learning

- Data + Task + “a mathematical formulation of what means ‘better’ ” *
(such as the number of type-1 and type-2 errors in a classification task)
- Optimization of the model parameters on the data
= “Training” (aka the machine ‘learns’)



Introduction Machine Learning

- Training (on “Training Data”)



- Prediction (on “Test Data”, i.e. data was not used in the training)



Introduction to Deep Neural Networks

- Deep learning

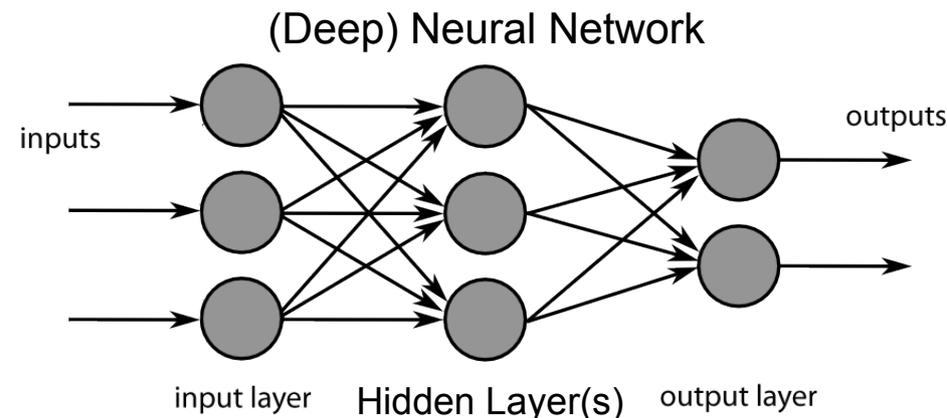
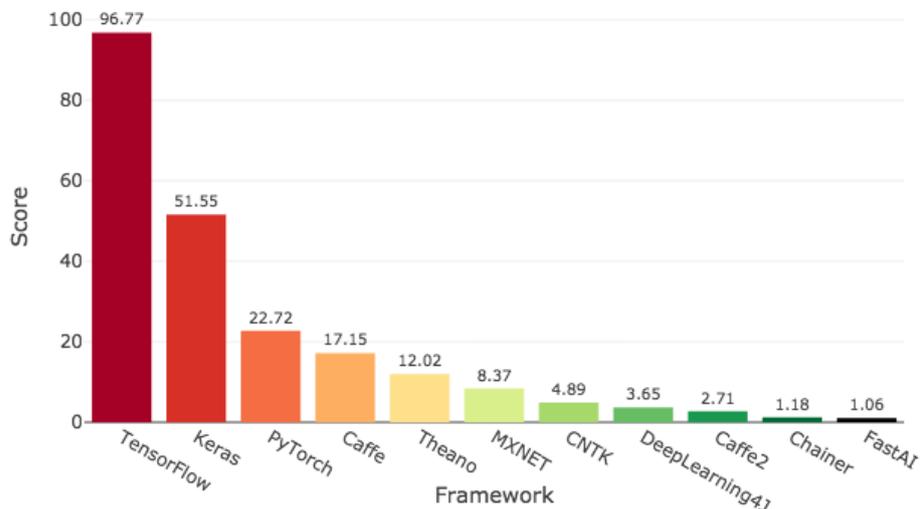
- Part of a broader family of machine learning methods based on artificial neural networks that use multiple layers to progressively extract higher level features from raw input

- Deep neural network

- Network with an input layer, at least a hidden layer and an output layer
- Each layer performs specific types of sorting and ordering in a process that some refer to as “feature hierarchy”
- Deal with unlabeled or unstructured data
- Algorithms are called **deep** if the input data is passed through a series of hidden layers with nonlinearities (**nonlinear transformations**) before it becomes output.

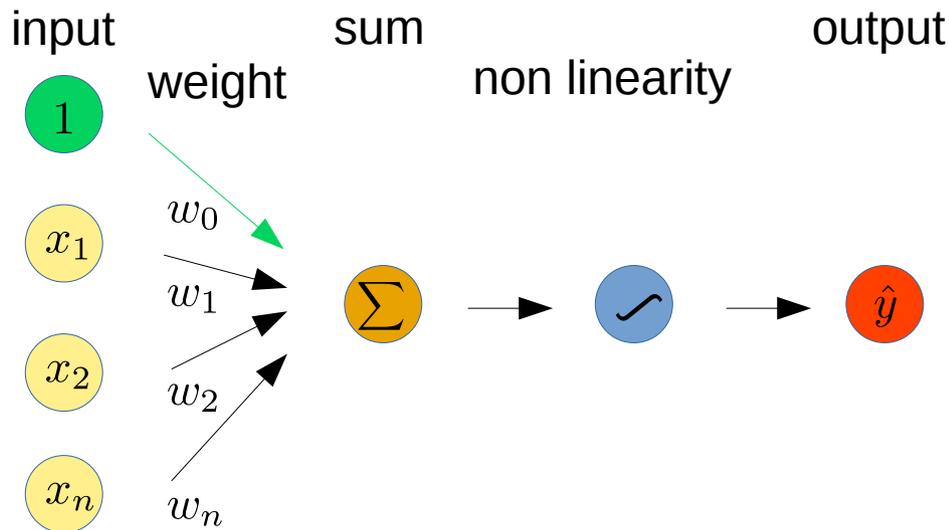
- Most Deep Learning frameworks (user interface) are based on Python

→ TensorFlow and Keras are one of the most popular frameworks



Introduction to Deep Neural Networks

- Forward propagating perceptron

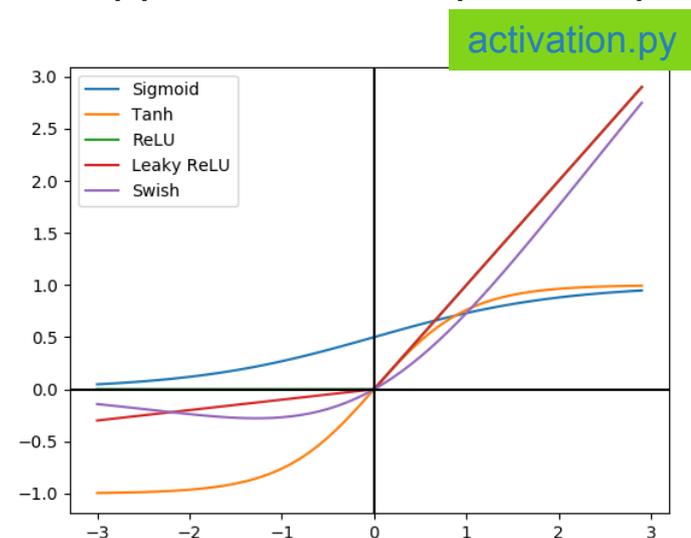


$$\hat{y} = \theta \cdot (w_0 + \vec{x}^T \vec{W})$$

- Activation function

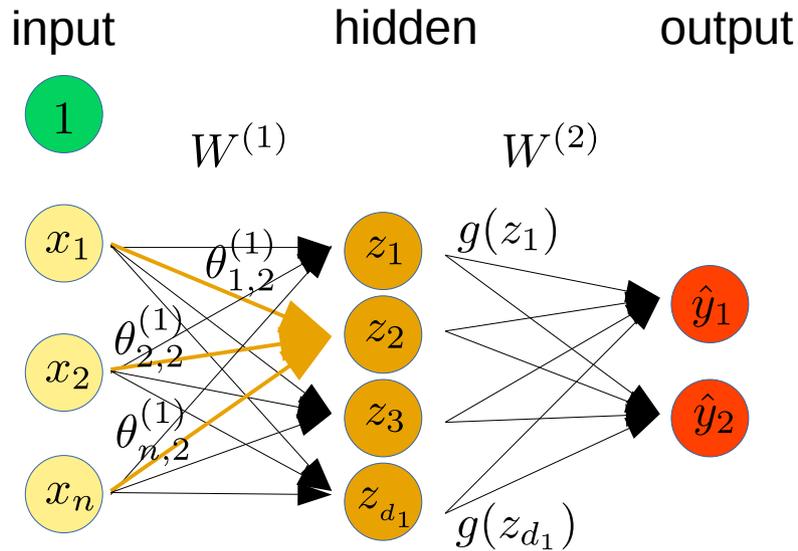
- Introduce non linearities into the network → allows to approximate complex shapes

ACTIVATION FUNCTION	EQUATION	RANGE
Linear Function	$f(x) = x$	$(-\infty, \infty)$
Step Function	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$\{0, 1\}$
Sigmoid Function	$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Hyperbolic Tanjant Function	$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Leaky ReLU	$f(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Swish Function	$f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 & \text{for } f(x) = x \\ \beta \rightarrow \infty & \text{for } f(x) = 2\max(0, x) \end{cases}$	$(-\infty, \infty)$



Introduction to Deep Neural Networks

- Single layer neural network



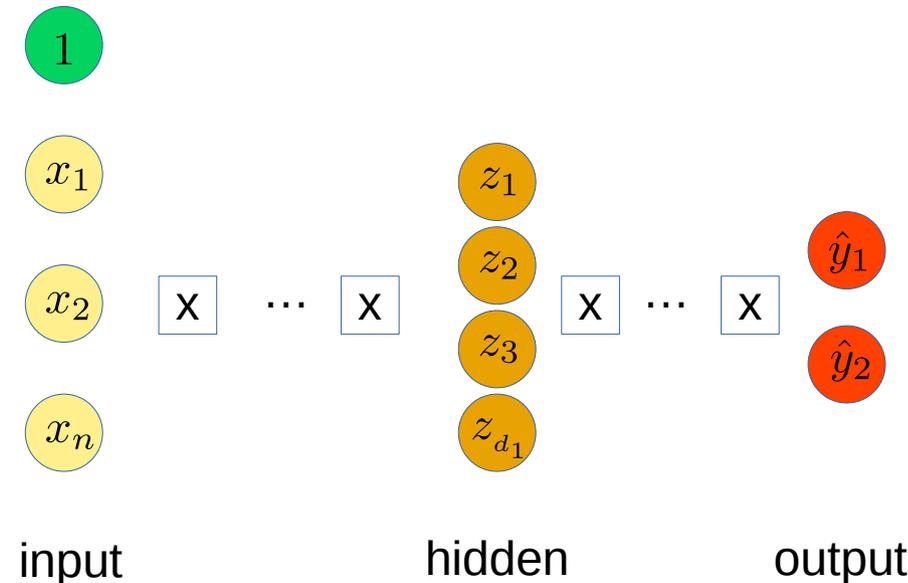
2nd element hidden layer 1 :

$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^n x_j w_{j,2}^{(1)}$$

ith output :

$$\hat{y}_i = g(w_{0,2}^{(2)} + \sum_{j=1}^{d_1} z_j w_{i,j}^{(2)})$$

- Deep neural network

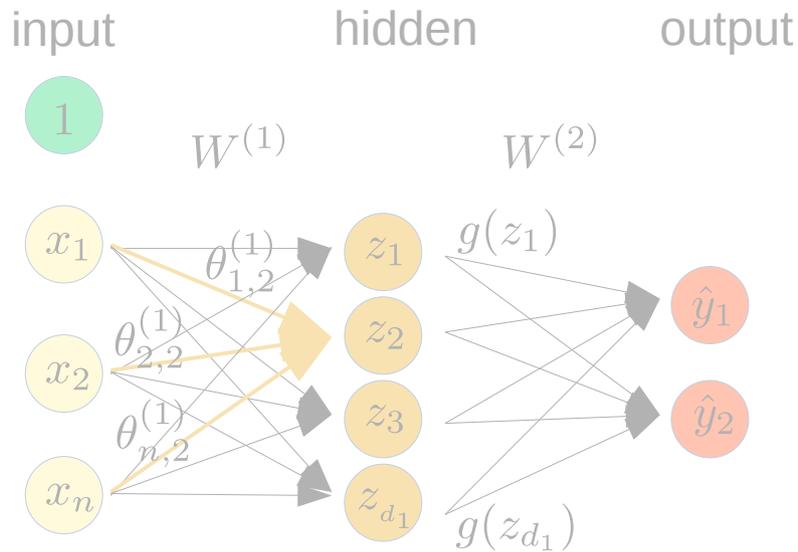


ith element hidden layer k :

$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j}) w_{i,j}^{(k)}$$

Introduction to Deep Neural Networks

- Single layer neural network



- Have to apply activation functions on nodes in each hidden layer and the final output layer

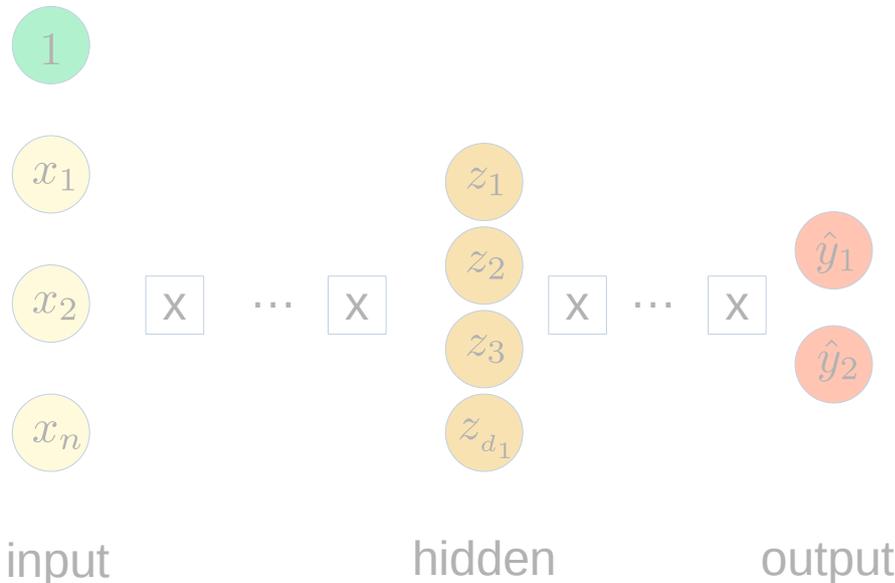
2nd element hidden layer 1 :

$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^n x_j w_{j,2}^{(1)}$$

$$z_i \rightarrow \theta(z_i)$$

Non-linear activation functions are really the key

- Deep neural network



ith element hidden layer k :

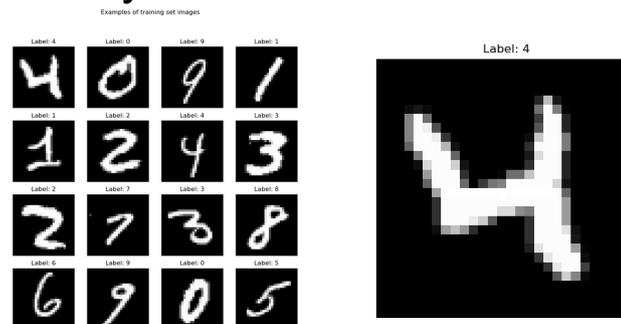
$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j}) w_{i,j}^{(k)}$$

Introduction to Deep Neural Networks

- Input data usually has to be pre processed
 - Needed for numerical stability
 - Want to have the mean around zero and the variance order of one
 - gaussian like distributed data → subtract mean and divide by standard deviation
 - data peaking at zero → apply logarithm and then subtract mean and divide by standard deviation
 - Example datasets in the MNIST database (Modified National Institute of Standards and Technology database) used for machine learning and practicing techniques
- The data can be loaded directly in Tensorflow

- MNIST handwriting
60000 images

`display_HandWrt.py`



28 x 28 array with greyscale values, 0 – 255
→ preprocessing: divide by 255

use pixel array and label

- Fashion-MNIST dataset
60000 images

`display_Clothing.py`



28 x 28 array with greyscale values, 0 – 255 and labels

- Input data needs often reshaping

Introduction to Deep Neural Networks

- Metric

Machine learning datasets contain truth information to allow an independent quality measure of the neural network. Metrics are typically defined in terms of the neural network model's predictions and the true labels. They quantify how well the model is doing at its task.

- accuracy: number of correctly classified data points over the total number.
- precision: ratio of true positive (TP) predictions to the total number of positive (TP + false positive) predictions.
- recall: ratio of true positives to the sum of true positives and false negatives
- F1 score: $F1 \text{ score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$

- Splitting the dataset in 2 or 3 parts

- Training dataset – Do all the parameter determination (training) of the neural network.
- Validation dataset – Used to evaluate the model during training and to tune its hyperparameters (e.g. handled by tensorflow during minimization)
- Test dataset – Use it for evaluating the optimized parameters and the final performance of the model after it has been fully trained.

Splitting: Training:Validation:Test → 60%:20%:20%

Introduction to Deep Neural Networks

- Find the network weights such that the loss function is minimal

$$W_{min} = \underset{w}{\operatorname{argmin}} J(W) = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; W), y^{(i)})$$

Repeat until convergence

- Initialize weights randomly
- Loop until convergence:

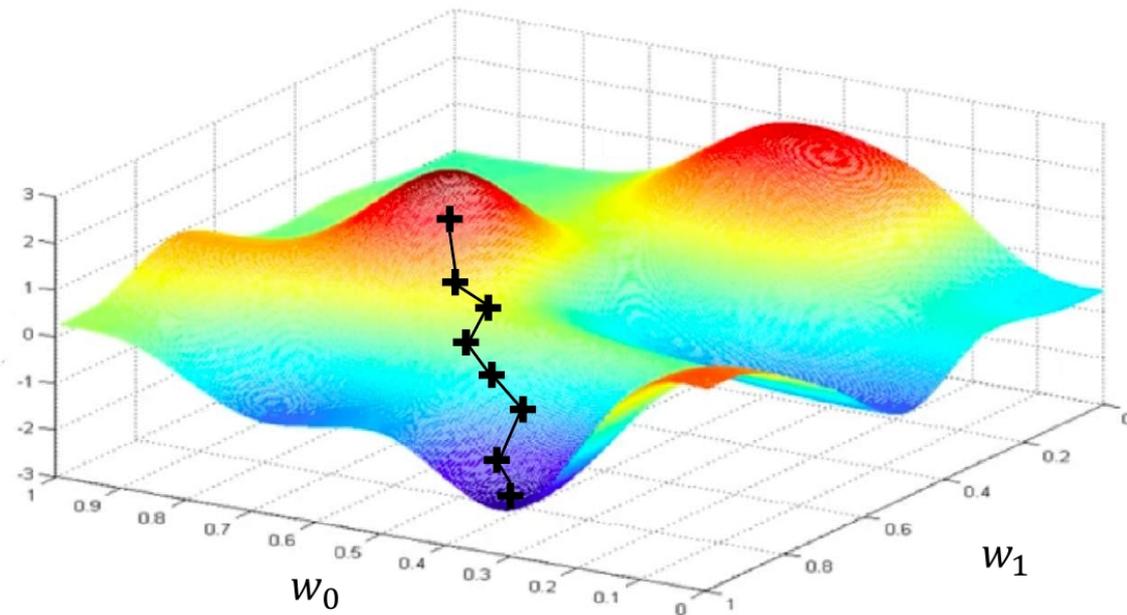
compute $\frac{\partial J(W)}{\partial W}$

backpropagation

update weights

$$W \leftarrow W - \eta \cdot \frac{\partial J(W)}{\partial W}$$

- return weights
- derivative calculation with chain rule



Introduction to Deep Neural Networks

- Find the network weights such that the loss function is minimal

$$W_{min} = \underset{w}{\operatorname{argmin}} J(W) = \operatorname{argmin} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; W), y^{(i)})$$

- Initialize weights randomly
- Loop until convergence:

compute $\frac{\partial J(W)}{\partial W}$

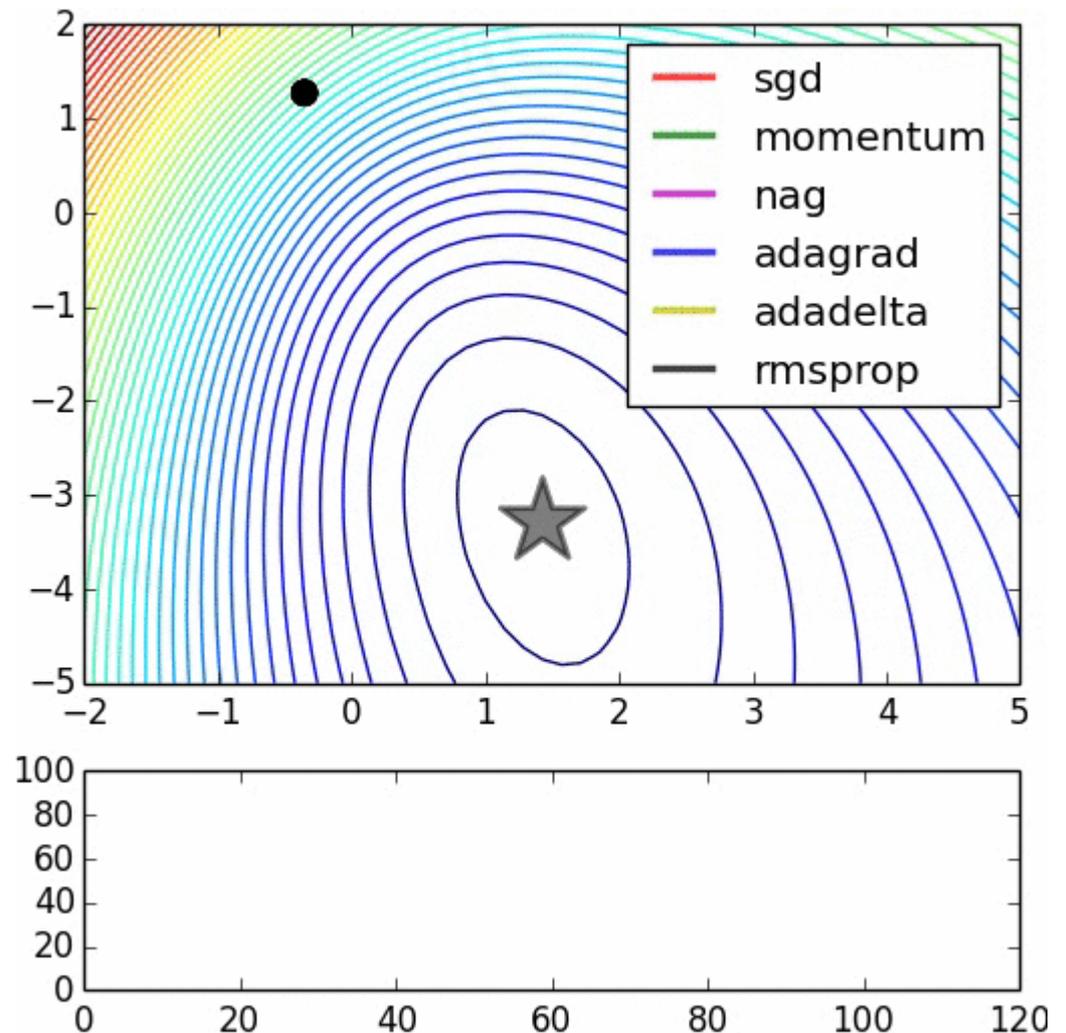
update weights backpropagation

$$W \leftarrow W - \eta \cdot \frac{\partial J(W)}{\partial W}$$

- return weights
- derivative calculation with chain rule

- Example: Minimizer usage in TensorFlow

`linearRegression.py`



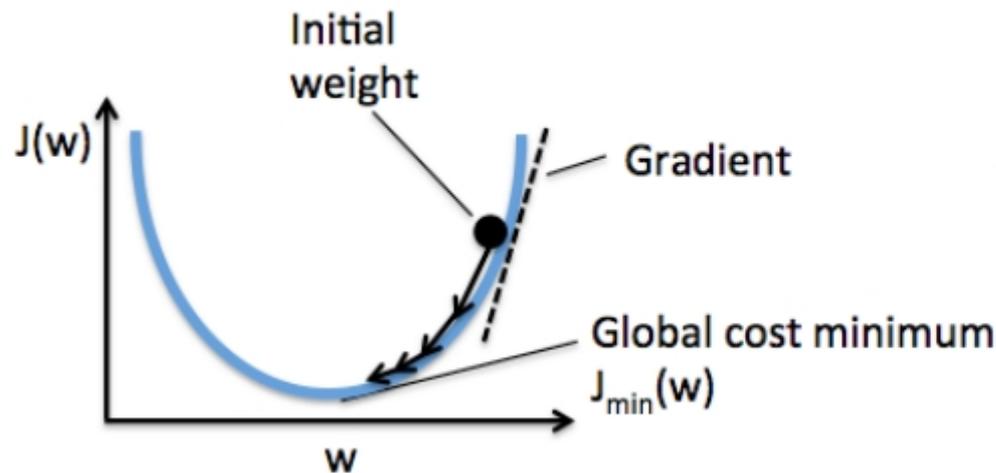
Introduction to Deep Neural Networks

- Training strategies

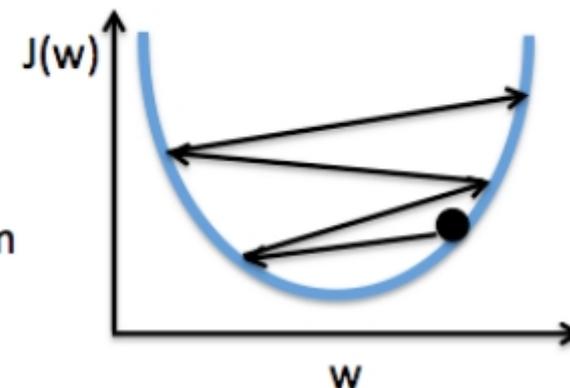
- Minimization: difficult to converge to a global minimum with large number of parameters.
- Learning rate

$$W \rightarrow W - \alpha \cdot \frac{\partial J(W)}{\partial W}$$

α is called learning rate



choosing the right learning rate is important for convergence



learning rate is too large

[https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/ch02/images/02_12.png, MIT License]

- Adaptive moment estimation (Adam) uses adaptive learning rates which reduces α during minimization for each parameter separately

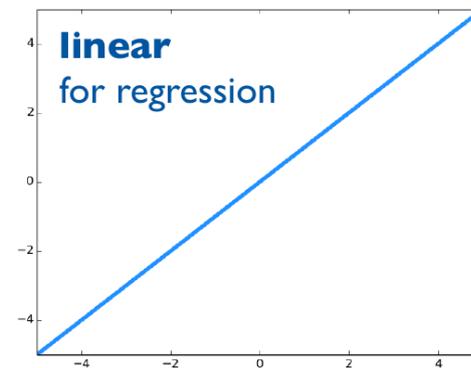
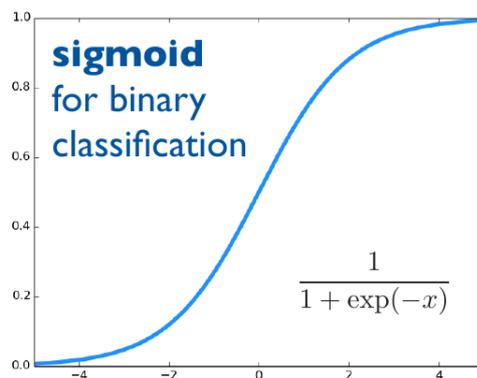
$$W \rightarrow W - (1 - \beta) \cdot \alpha \cdot \frac{\partial J(W)}{\partial W}$$

β change in W from previous step

Introduction to Deep Neural Networks

- Output to the last node(s)

- The output function(s) of the neural network (nodes in the final layer) are determined by activation functions. Their choice depends on the problem to be solved.
- For binary classification problems the sigmoid function is used and interpreted as probability.
- For regression problems a linear activation function or no activation function is used.



- For multi-class classification problems the softmax function is mostly in use and gives a probability distribution over the classes. Softmax assigns decimal probabilities to each class i of a multi class problem with its N multiclass output values:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^N e^{z_j}}$$

Examples - Deep Neural Networks

- We need start values for the network

- Initialize randomly, a range of values is needed, suitable values depend on the details of the network, like layer size and activation functions

- In general:

$$\text{var(input)} \approx \text{var(output)} \quad \text{with} \quad \text{var} \approx 2 / (N_{\text{input nodes}} + N_{\text{output nodes}})$$

draw from gaussian or uniform distributions within a range $\pm\sqrt{3\text{var}}$

- Usually input range differs largely

- transform to mean 0 and variance 1

- perform de-correlation of input data

- Simple example using TensorFlow (TF)

tf_intro.py

- Generate toy sample with 2 normalized gaussian distributions with mean (-1,-1) and (1,1)

- Each sample gets a label and then they are combined to a training set

- The data is plotted and colored according to their labels

- In TensorFlow 's a model is set up specifying the optimizer, loss function and metric

- Use AdamOptimizer to find the minimum and accuracy as metric

- The data and labels are given to the fit method, also the training iterations and the batch size. The model doesn't process the entire dataset at once, but rather in batches.

- Loss and accuracy are plotted as number of iterations

- Display classification results for sample points together with labeled data points

Examples - Deep Neural Networks

- Using the handwriting MNIST example with TensorFlow (TF)
 - Read in the dataset with a pre defined function in TF. There are 10 categories.
 - Each sample has a label and the 28 x 28 pixel grey scale data array.
 - The data is normalized and reshaped to a 1D array.
 - In TensorFlow 's a model is set up specifying a first dense hidden layer with 512 nodes and a ReLU activation function. Dense means all nodes are fully connected. A second layer also with 512 nodes and ReLU activation. The 3rd layer is the output layer with 10 nodes and Softmax is used as activation.
 - Use AdamOptimizer to find the minimum and accuracy as metric
 - The data and labels are given to the fit method, also the training iterations and the batch size. The model doesn't process the entire dataset at once, but rather in batches.
 - Loss and Accuracy are plotted as number of iterations
There is a difference between the training and the test data → Overtraining, the net learns features of the data which are not existing. To prevent this reduce the number of nodes or refine the model
 - Test accuracy = 0.98
 - The data is plotted and true and predicted label are written [MLP_MNIST_digits_tf.py](#)

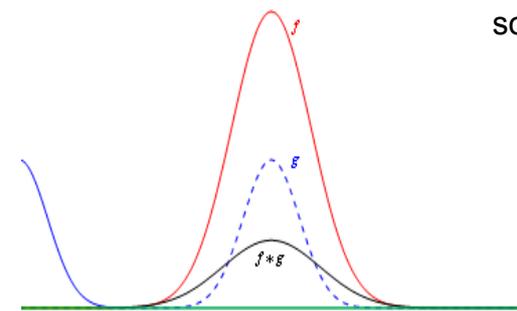
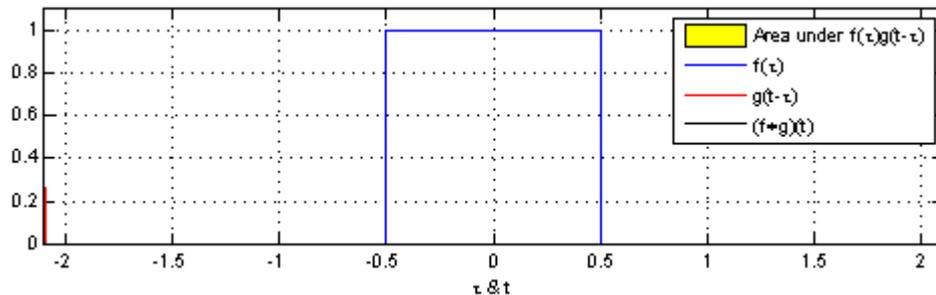
Convolutional Neural Networks

- A Convolutional Neural Network (CNN) learns features directly from data without specifying the features explicitly. CNNs are useful for pattern recognition in images and computer vision, but also for classification tasks in audio data and signal processing.
- Learning the features happens via convolution filter, which are activated by scanning over the pixel of an image. Edges or structure and color changes are transported to another filter layer. A Rectified Linear Unit (ReLU) is applied as activation function. Via a pooling technique the number of parameters is reduced from layer to layer.

- Convolution
$$(f * g)(x) = \int_{\mathbb{R}^n} f(\tau)g(x - \tau)d\tau$$

$f(x)$: input distribution $g(x - \tau)$: kernel or filter function

The function value of the weight function f at the position τ tells us how strong the contribution of $g(x - \tau)$ in f at the position x is.



source Wikipedia

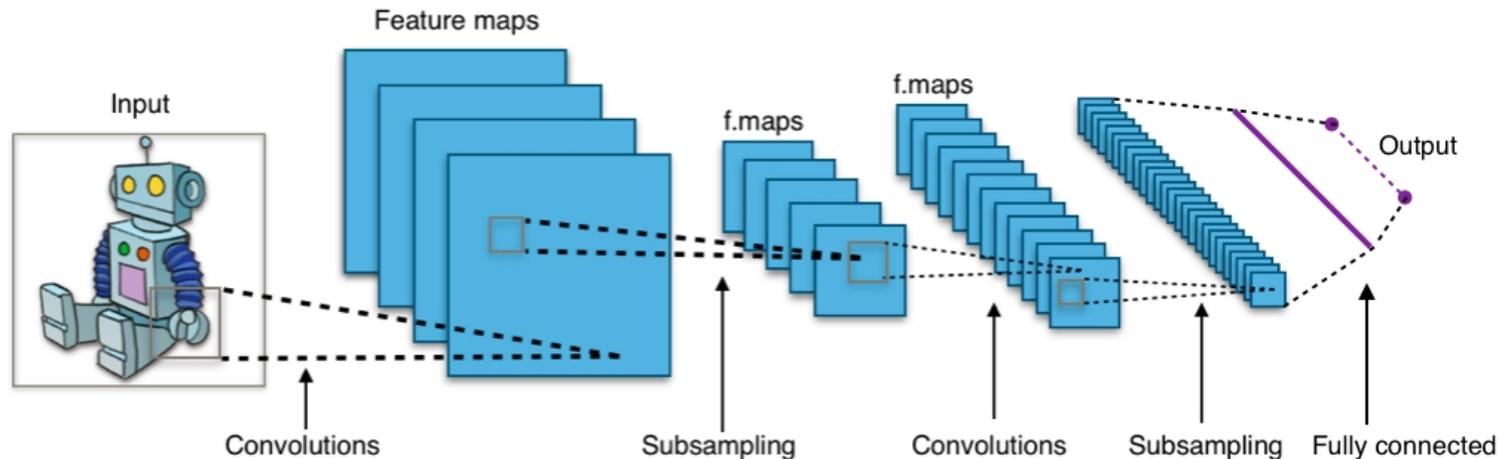
- Discrete convolution
$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)$$

Play with images and various kernel for image processing: <https://setosa.io/ev/image-kernels/>

Convolutional Neural Networks

More information can be found in the Stanford course <https://cs231n.github.io/convolutional-networks/>

- Structure of a typical CNN used in image classification



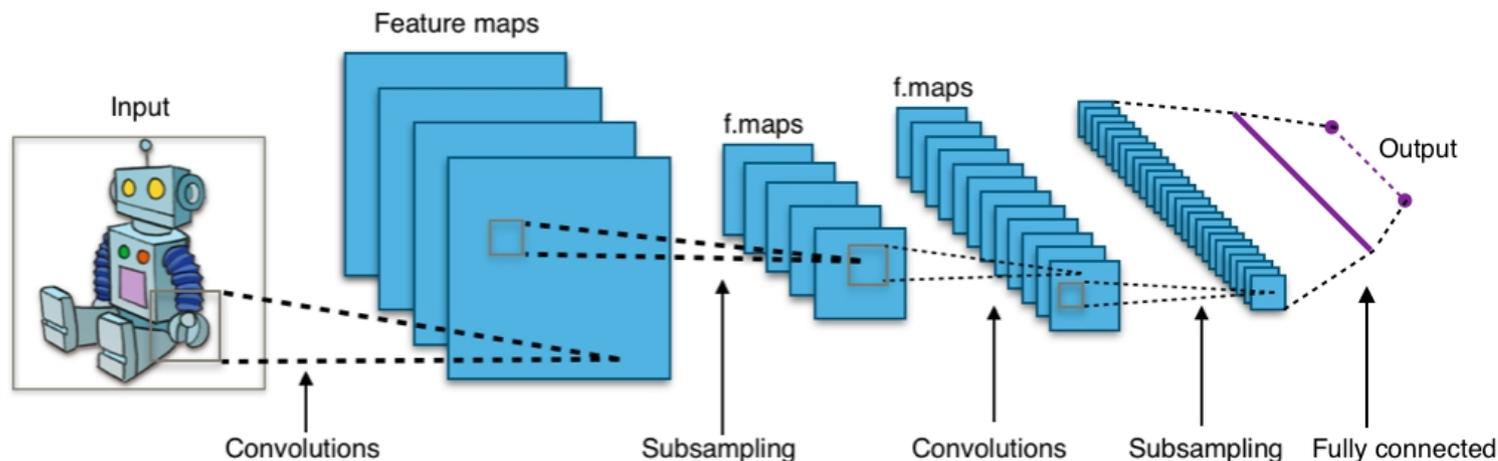
- Main idea is to extract particular localized features of data, eg. an image, using a convolution as filter mechanism. The input is a data structure $[xPix, yPix, 3]$ of raw pixel values with three color channels R,G,B. Each color channel is treated independently.
- There are 3 main building blocks:

1) Convolutional layer

- Obtain a weight matrix by computing the dot product of the weight matrix and a small sub region of the input data and scanning over the whole image. This results in a weight matrix which transports certain features of the image to further layers.
- To the weights activation functions like ReLU are applied. The convolution operation (weight matrix \cdot sub input structure) behaves like a filter.
- The weight matrix is determined by a loss function.
- Multiple convolutional layers extract with increasing depth more and more complex features

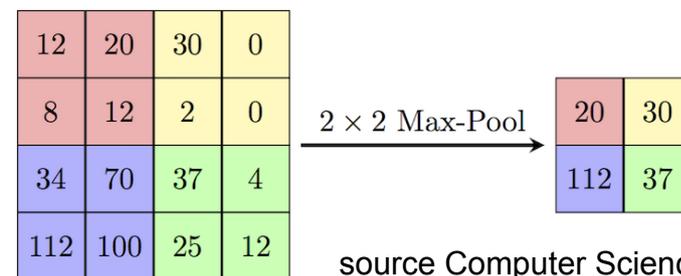
Convolutional Neural Networks

- Structure of a typical CNN used in image classification



II) Pooling layer

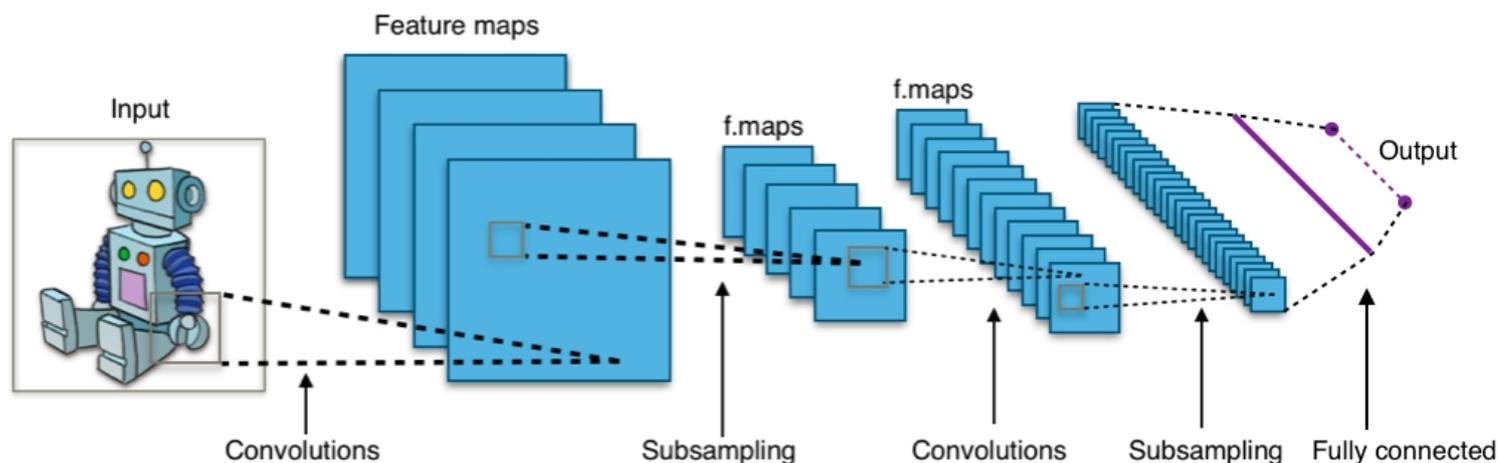
- Several neighbouring pixel are pooled together by averaging or by taking their maximum.
- Max Pooling algorithm: Take a 4x4 input (W), step (S) with a 2x2 filter (F) over the input, take the maximum entry
- Zero Padding: to the edges of the picture pixel are added (P) and filled with zeros



- Output size OP :
$$OP_{x,y} = \frac{W - F_{x,y} + 2P}{S_{x,y}} + 1$$

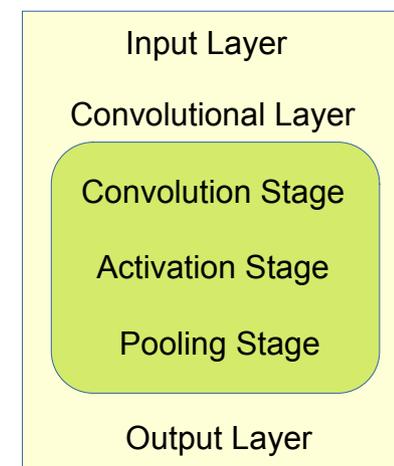
Convolutional Neural Networks

- Structure of a typical CNN used in image classification



III) Output layer

- The final stage is a fully connected layer which works as MLP to generate an output
- The very last layer is equal to the number of output classes. In order to obtain probabilities for each class SoftMax is applied
- A loss function determines all the weights



- As CNN example the MNIST handwriting dst is used which over comes overfitting

Examples - Deep Neural Networks

- Two extreme cases of training results

- If the model does not reflect the data content or the training is insufficient
→ bad network performance
- If the model allows for too much complexity it learns features of the training data sample
→ network can be applied to other samples (overtraining effect)
test overtraining in the example clothing dataset by including the method Dropout in TF or changing the number of nodes in the hidden layer

- Other classification examples with Tensorflow and Keras

- uses the Fashion MNIST dataset of Zalando, which contains 60,000 grey scale images in 10 categories each showing low resolution clothing pictures.
- sequential model with two dense layers → significant overtraining with test accuracy = 87.7%
`clothingSequential.py`
- sequential model with two dense layers adding Dropout and softmax activation
→ no overtraining with test accuracy = 88.2 %
`clothingSequentialDropout.py`
- CNN model with MaxPooling2D and 3x3 Filters and ReLu activation. The final output is 10 nodes with Softmax activation → overtraining with test accuracy = 90.6 %
`clothingCNN.py`
- CNN model where the input data is shaped to a 4D tensor of shape (samples, height, width, channels). MaxPooling2D and 3x3 Filters and ReLu activation and the Dropout feature is used → no overtraining with test accuracy = 91.4 %
`clothingCNN_4D.py`