Neural Networks

Data Science 2 / Data & AI 3

KdG Karel de Grote Hogeschool

Agenda

1. What are Neural Networks?
2. Training a Neural Network
3. Neural Network - Classification
4. Neural Network - Regression

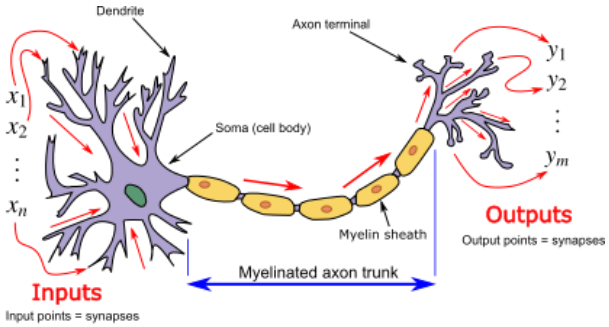
Cursus / Boek

canvas Oefeningen

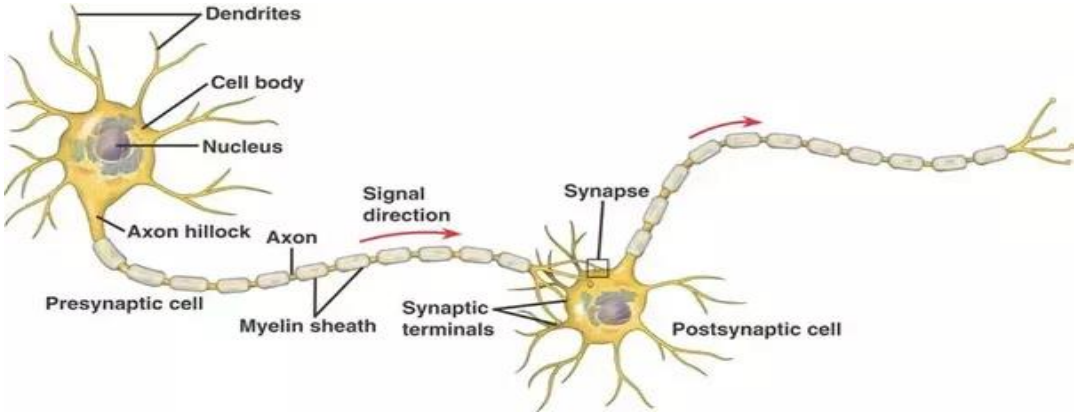
What are Neural Networks?

A biological neural network

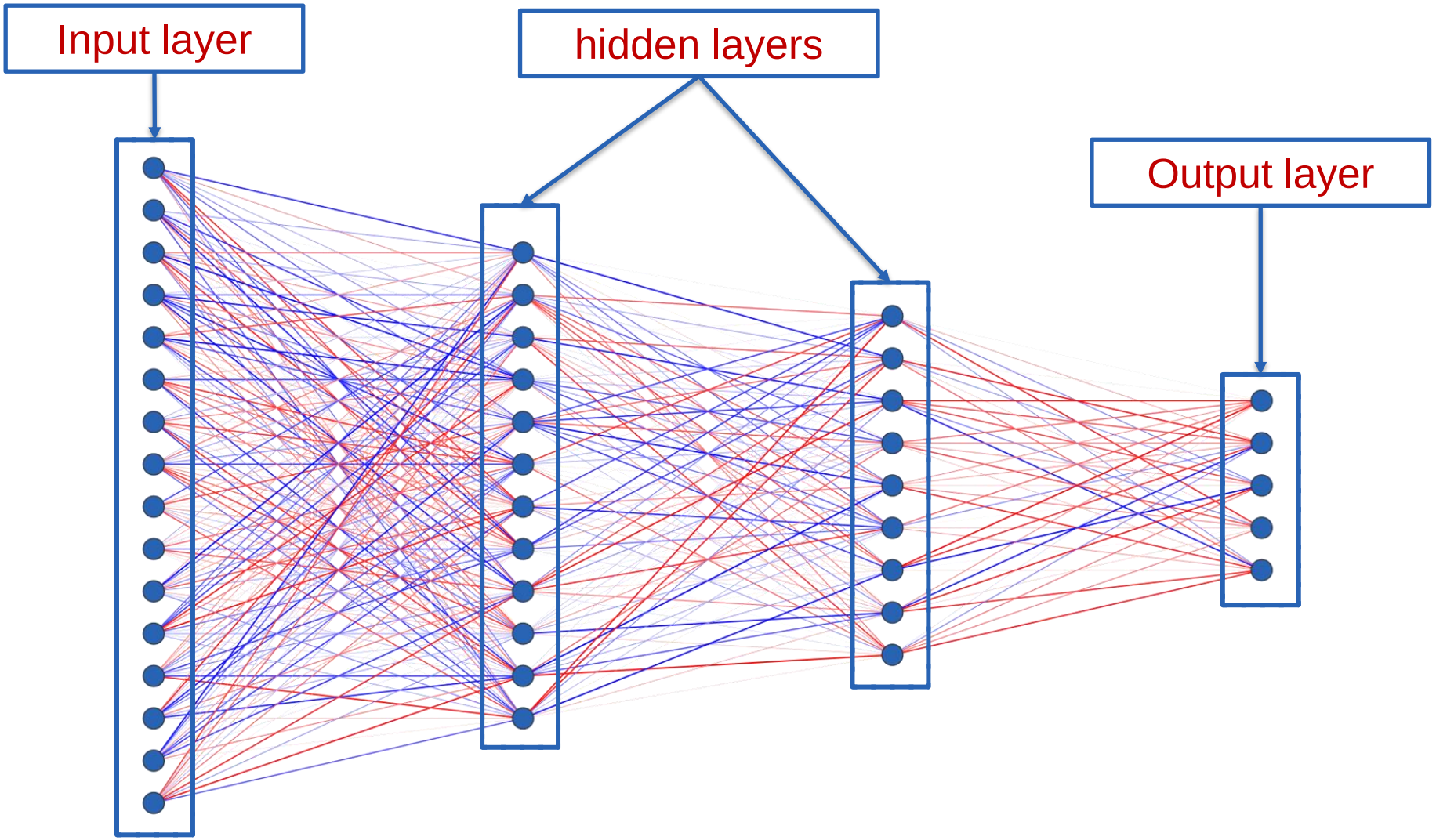
The human brain has about 86,100,000,000 neurons. Input signals are combined and strengthened or weakened (non-linear).



Neurons are connected to each other to form a neural network with 100,000,000,000,000 connections.

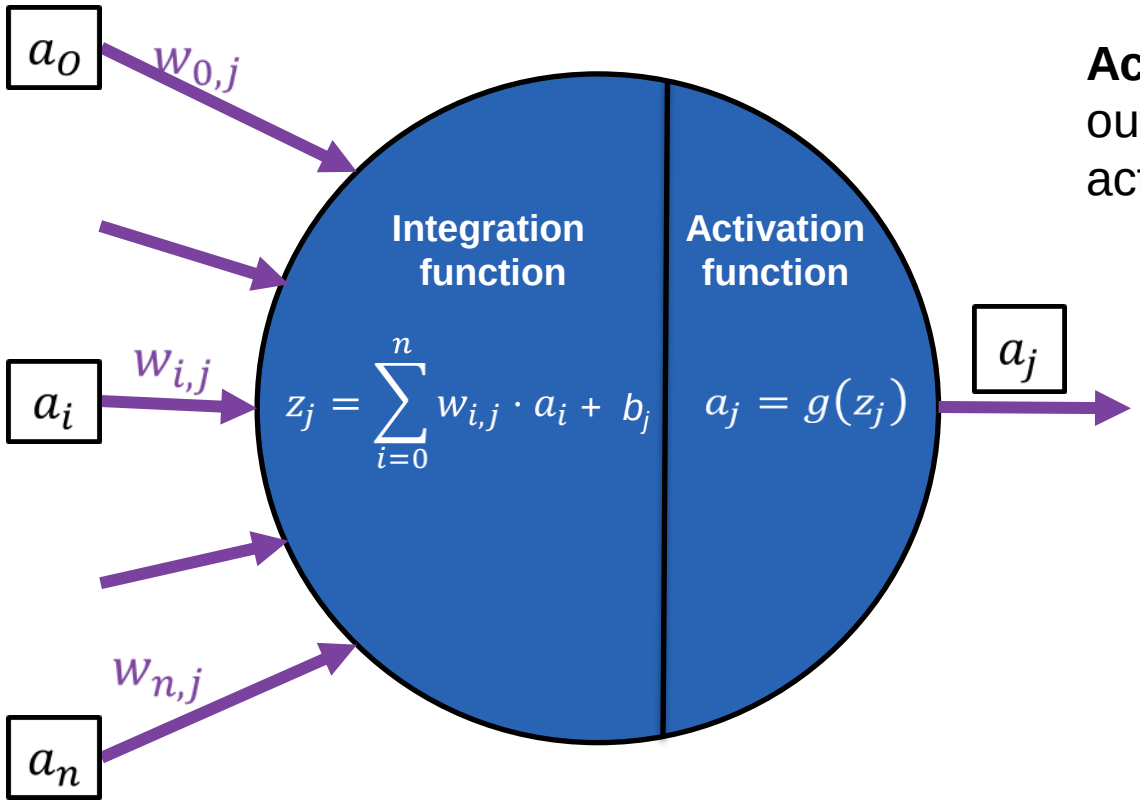


Artificial neural network (ANN or NN)



When the network has more than 1 hidden layer, we talk about Deep Learning

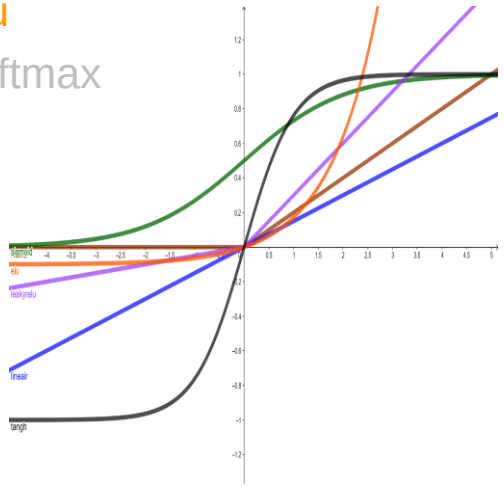
Artificial Neuron



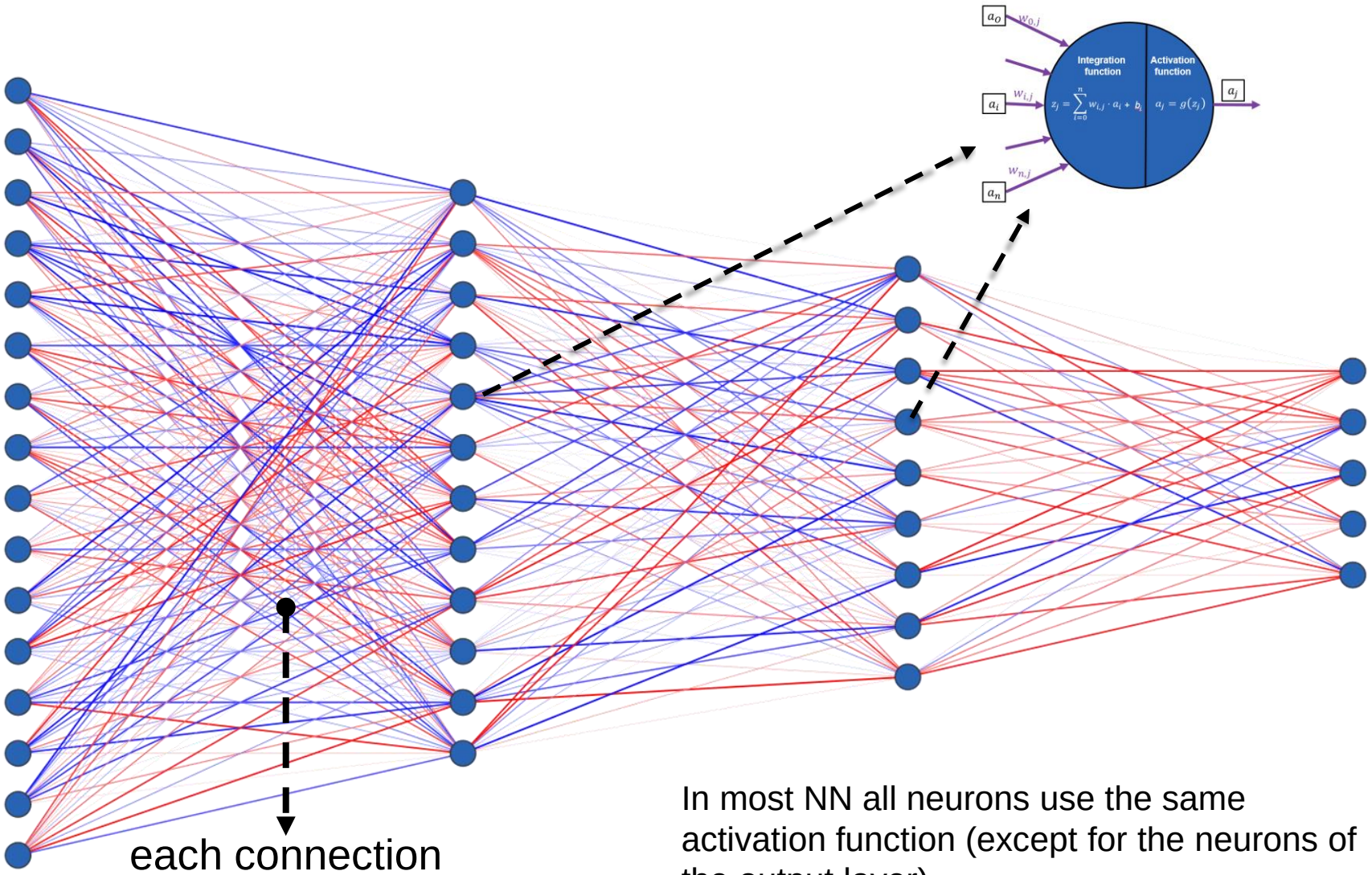
Integration function:
weighted average (weights $w_{i,j}$) plus bias b_j

Activation function: produces output of the neuron. Several activation functions exist:

- linear
- sigmoid
- relu
- leaky relu
- tanh
- elu
- softmax
- …



Neural Network



each connection

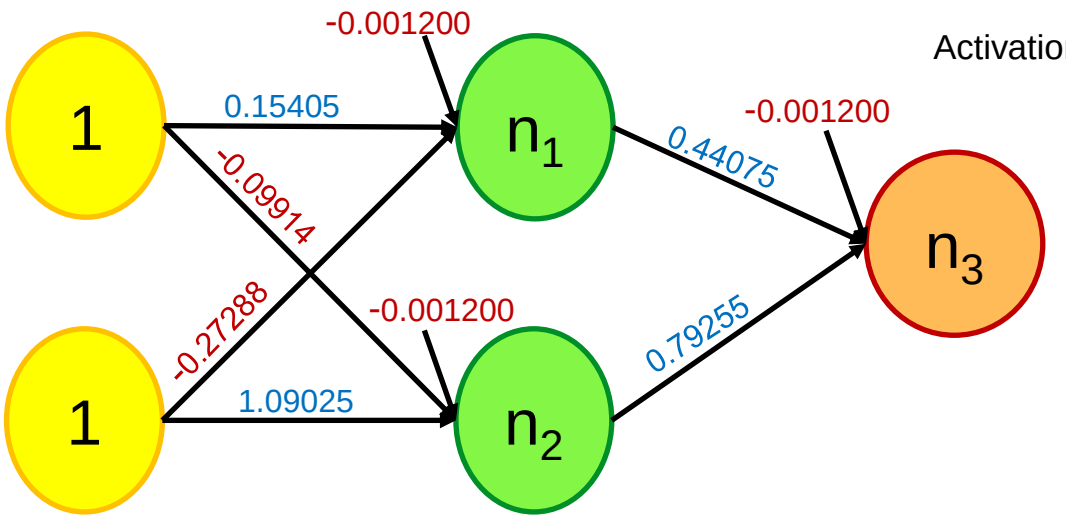
=

weight

In most NN all neurons use the same activation function (except for the neurons of the output layer)

Example

- Input layer: 2 nodes
- One hidden layer: 2 nodes
- Output layer: 1 node
- Activation function: sigmoid $g(z_j) = \frac{1}{1+e^{-z_j}}$ for all nodes
- Calculate output with weights and biases as in figure and both inputs equal to 1



n_1:

Integration function: z_j = 0.15405 x **1** -0.27288 x **1** – 0.00200
= -0.12082

Activation function : $a_j = \frac{1}{1+e^{-z_j}} = \frac{1}{1+e^{0.12082}}$ = **0.46983**

n_2:

Integration function: z_j = -0.09914 x **1** + 1.09025 x **1** - 0.00200
= 0.98912

Activation function: $a_j = \frac{1}{1+e^{-z_j}} = \frac{1}{1+e^{-0.98912}}$ = **0.72891**

n_3:

Integration function: z_j = 0.44075 x **0.46983** +
0.79255 x **0.72891** – 0.00200
= **0.782781**

Activation function: $a_j = \frac{1}{1+e^{-z_j}} = \frac{1}{1+e^{-0.782781}}$ = **0.68628**

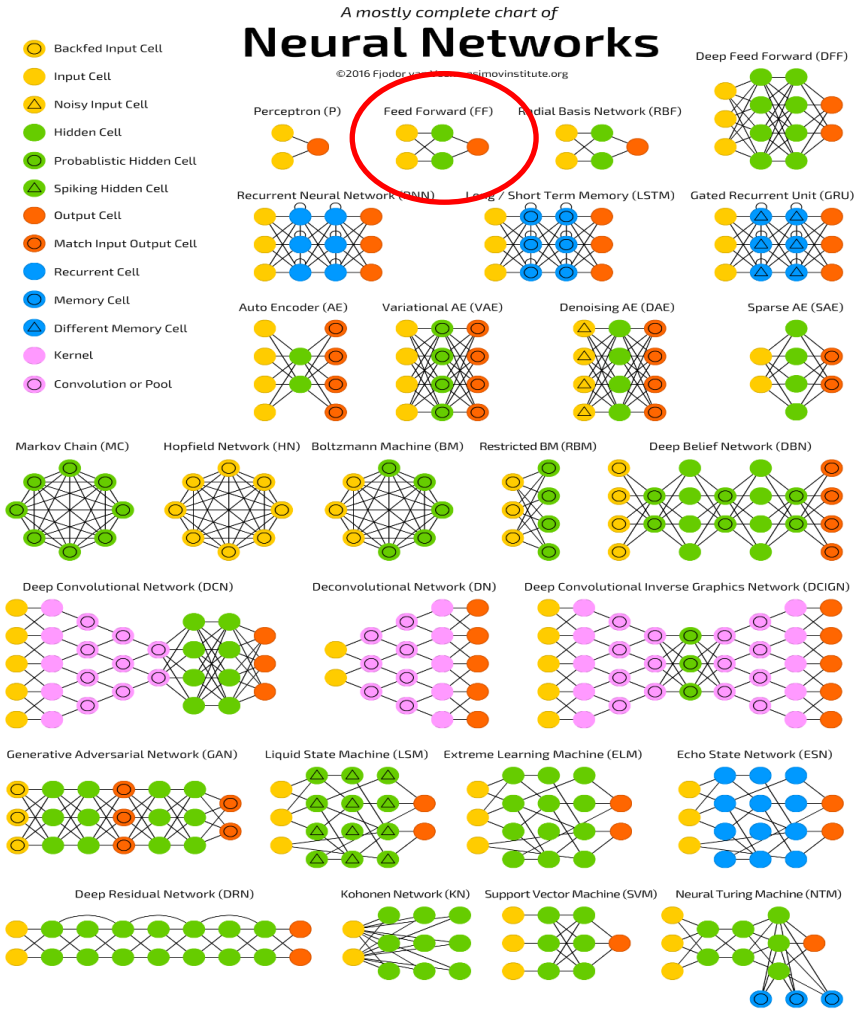
ANN

A lot of different types of layers:

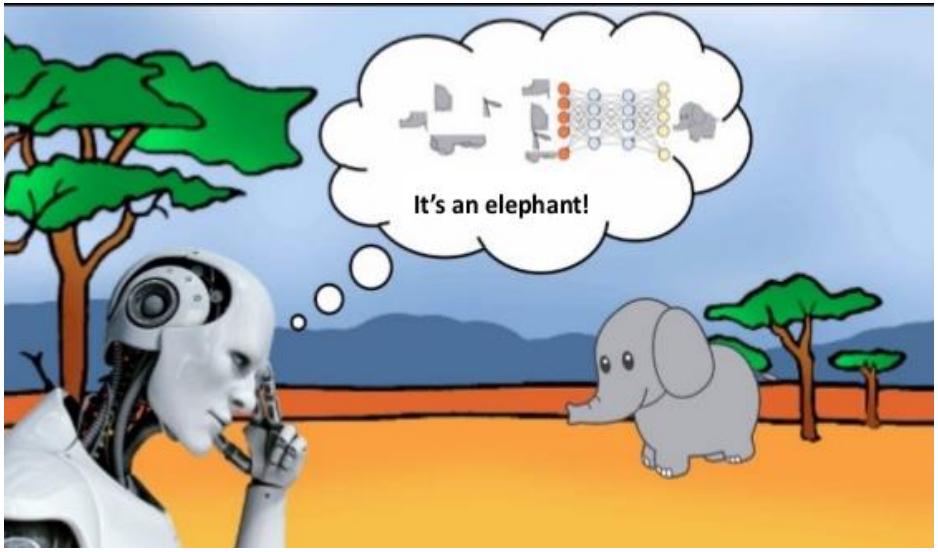
- Fully connected (dense)
- Convolutional
- LSTM
- GRU
- Attention
- …

A lot of different topologies:

- Feed Forward NN
- Recurrent NN
- Autoencoders
- GAN
- Graph NN
- …



This course: Feed Forward NN with fully connected (dense) layers

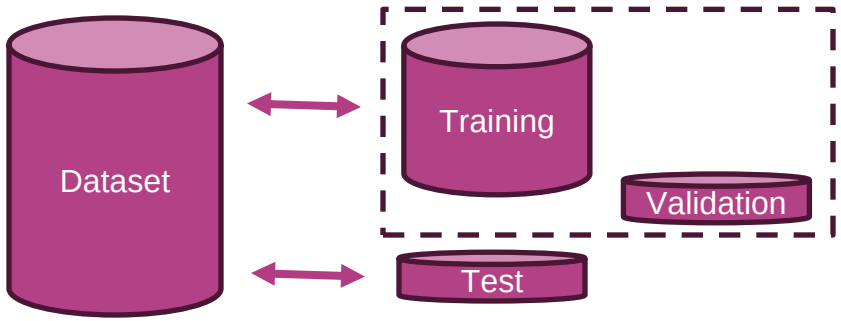


Training a Neural Network

Dataset

Split dataset in training, validation and test data

1. Training data
   - Used to train the model
   - Good representation of your data, e.g. all classes are equally represented (in case of a classification model)
   - Size: typical 80-90% of your dataset
2. Validation data
   - Used during training to follow up how well the training is doing
   - To check whether there is no overfitting
   - Classical models, like decision trees do not use validation data, but NN do need it
   - Size: typical 5-10% of your dataset
3. Test data
   - Used after training to evaluate how good to model is doing
   - Size: typical 5-10% of your dataset



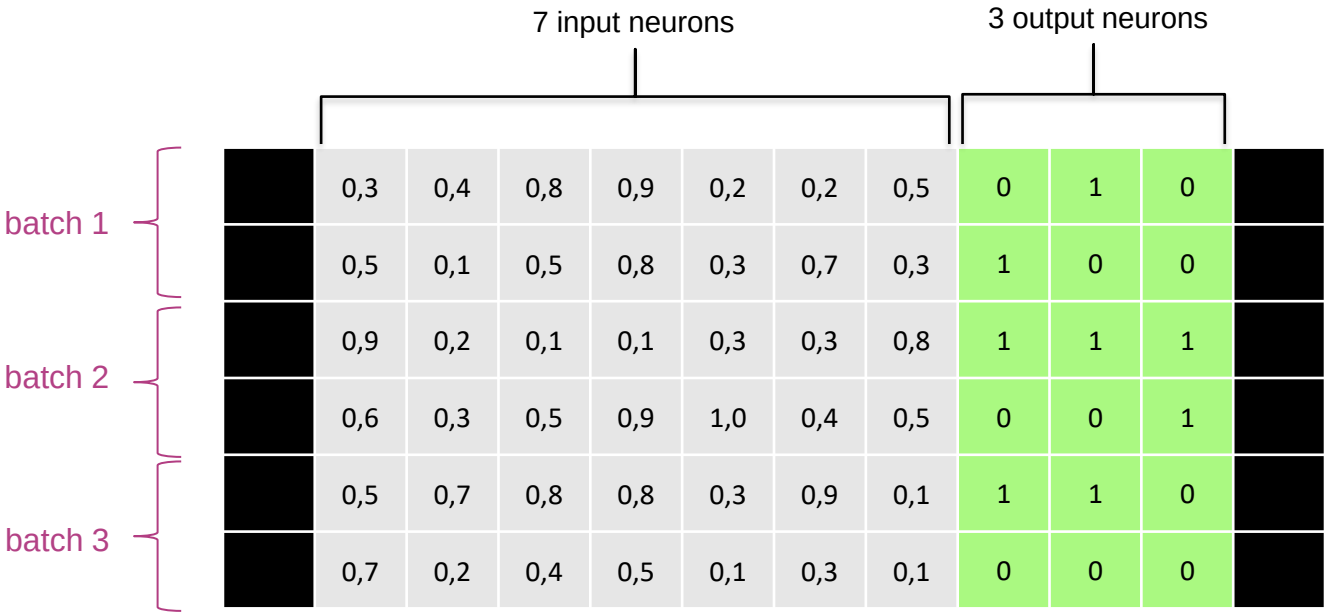
Training a NN

Very simple example:

- Model with 7 inputs and 3 outputs
- Training set size = 6
- Batch size = 2

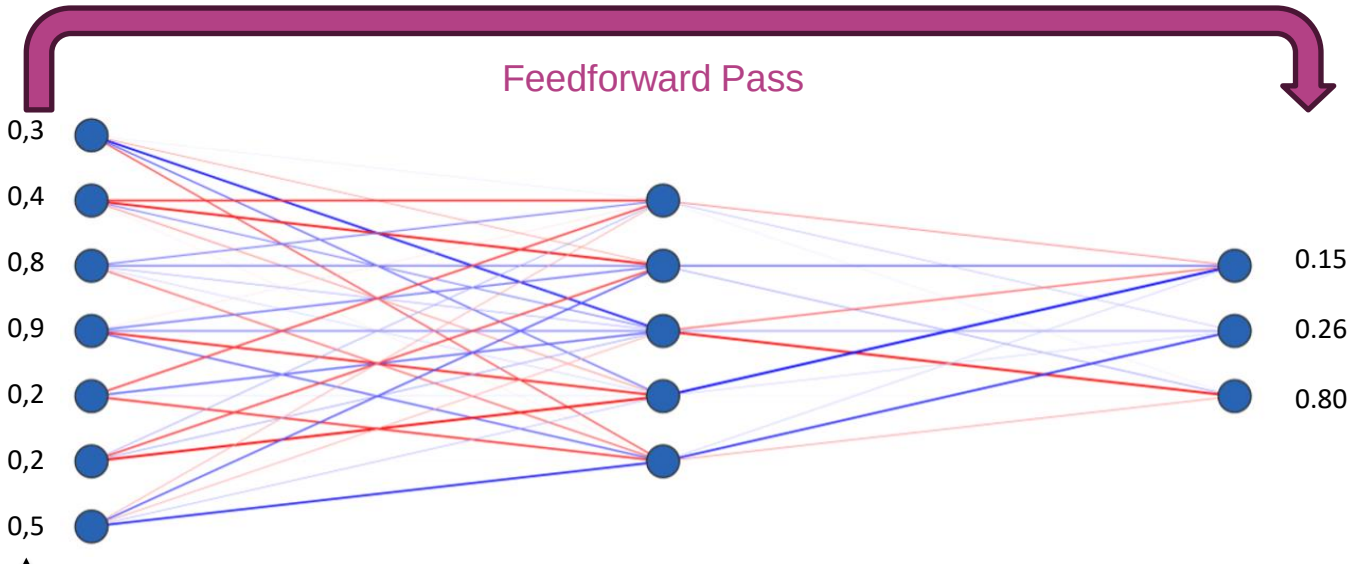
Supervised learning:

- Tell the model what the output should be for a certain input



Training a NN – feedforward

Start with random weights and biases and calculated output

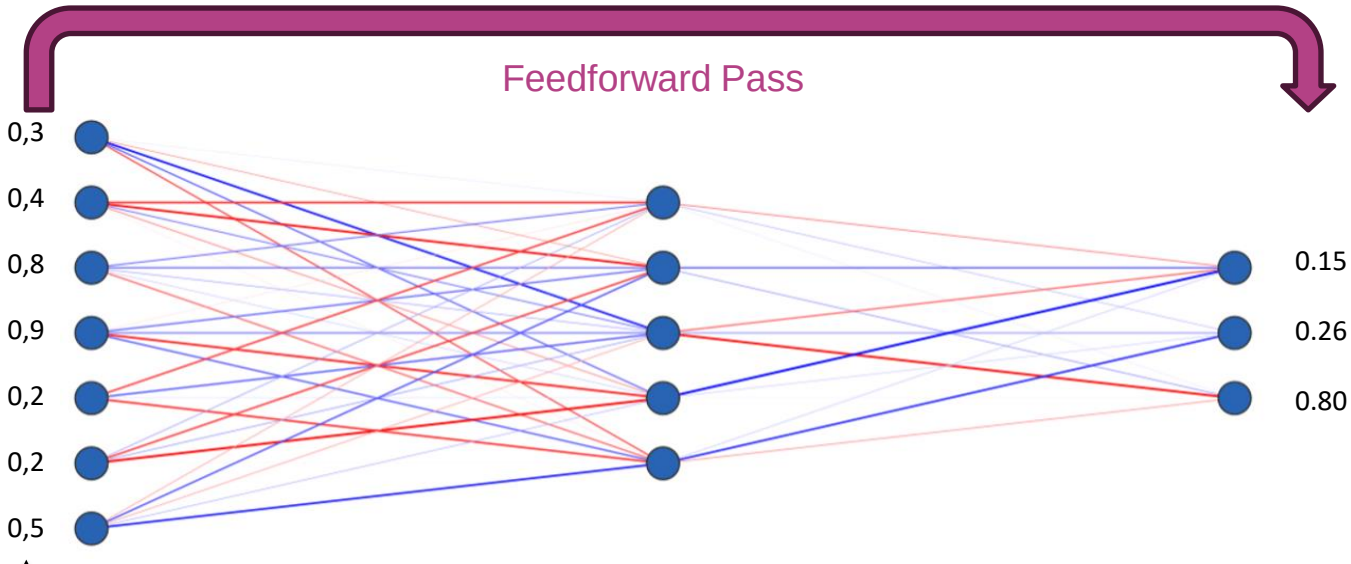


7 input neurons | 3 output neurons

batch 1	0,3	0,4	0,8	0,9	0,2	0,2	0,5	0	1	0
	0,5	0,1	0,5	0,8	0,3	0,7	0,3	1	0	0
batch 2	0,9	0,2	0,1	0,1	0,3	0,3	0,8	1	1	1
	0,6	0,3	0,5	0,9	1,0	0,4	0,5	0	0	1
batch 3	0,5	0,7	0,8	0,8	0,3	0,9	0,1	1	1	0
	0,7	0,2	0,4	0,5	0,1	0,3	0,1	0	0	0

Training a NN – feedforward

Compair with what the output should be

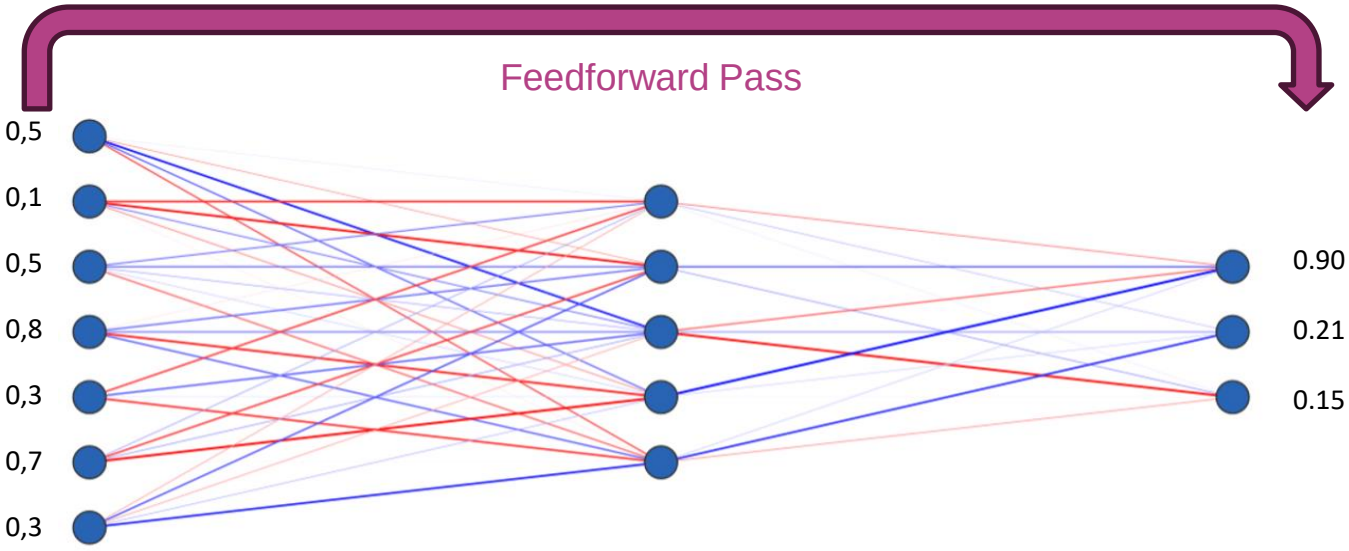


7 input neurons | 3 output neurons

batch 1	0,3	0,4	0,8	0,9	0,2	0,2	0,5	0	1	0
	0,5	0,1	0,5	0,8	0,3	0,7	0,3	1	0	0
batch 2	0,9	0,2	0,1	0,1	0,3	0,3	0,8	1	1	1
	0,6	0,3	0,5	0,9	1,0	0,4	0,5	0	0	1
batch 3	0,5	0,7	0,8	0,8	0,3	0,9	0,1	1	1	0
	0,7	0,2	0,4	0,5	0,1	0,3	0,1	0	0	0

Training a NN – feedforward

Repeat for other samples in the batch



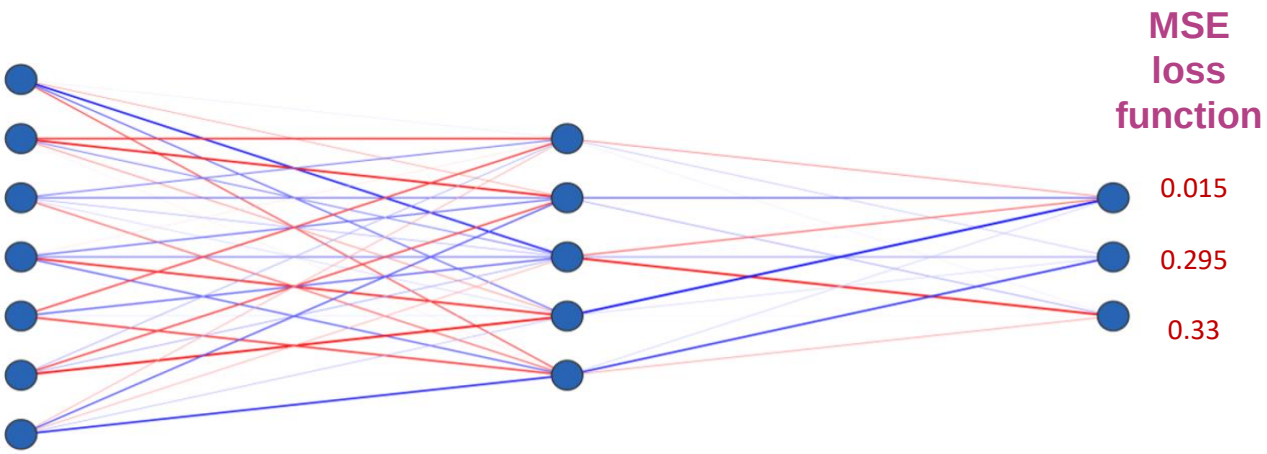
	7 input neurons							3 output neurons		
batch 1	0,3	0,4	0,8	0,9	0,2	0,2	0,5	0	1	0
	0,5	0,1	0,5	0,8	0,3	0,7	0,3	1	0	0
batch 2	0,9	0,2	0,1	0,1	0,3	0,3	0,8	1	1	1
	0,6	0,3	0,5	0,9	1,0	0,4	0,5	0	0	1
batch 3	0,5	0,7	0,8	0,8	0,3	0,9	0,1	1	1	0
	0,7	0,2	0,4	0,5	0,1	0,3	0,1	0	0	0

Training a NN – loss function

The loss function is calculating the mean error over the batch. Several loss functions exist:

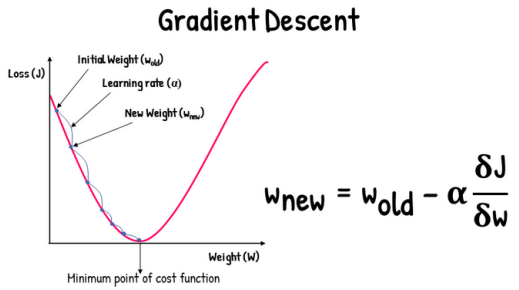
- Mean squared error (MSE) loss (used in this example)
- Mean absolute error (MAE) loss
- Binary cross entropy loss
- Categorical cross entropy loss
- …

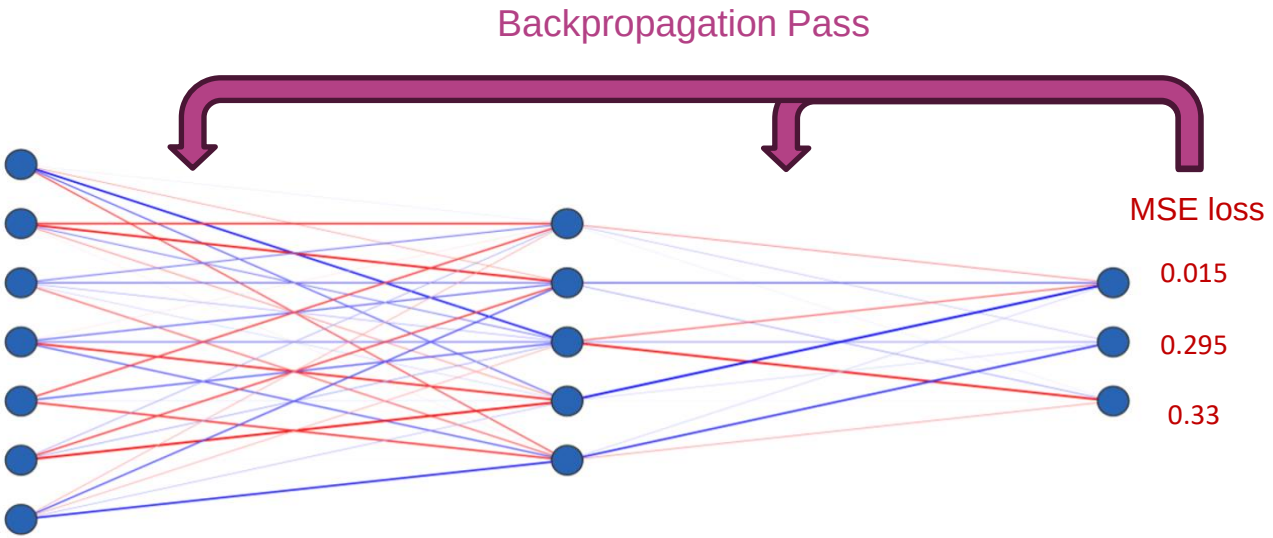
The loss function measures how good the model is performing (the lower, the better). The goal of the training is to minimize the loss function.



Training a NN – backpropagation

Backpropagation is adapting the weights and biases based on the output of the loss function. This is done by an optimizer algorithm called gradient descent. Today a variant of this algorithm called adam is mostly used.

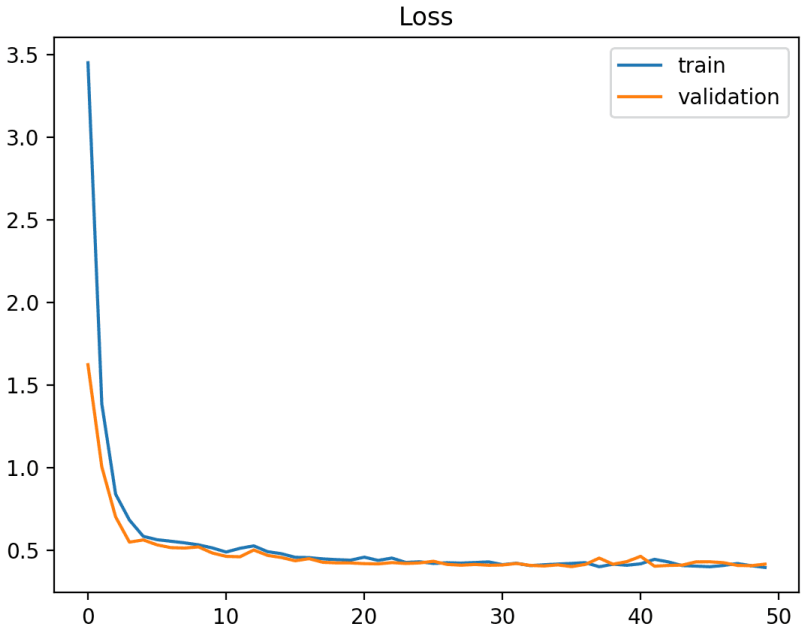




Backpropagation is done after every batch. So in our very simple example, the feedforward/backpropagation is executes 3 times when going ones over the training dataset (1 epoch).

Training a NN – epochs

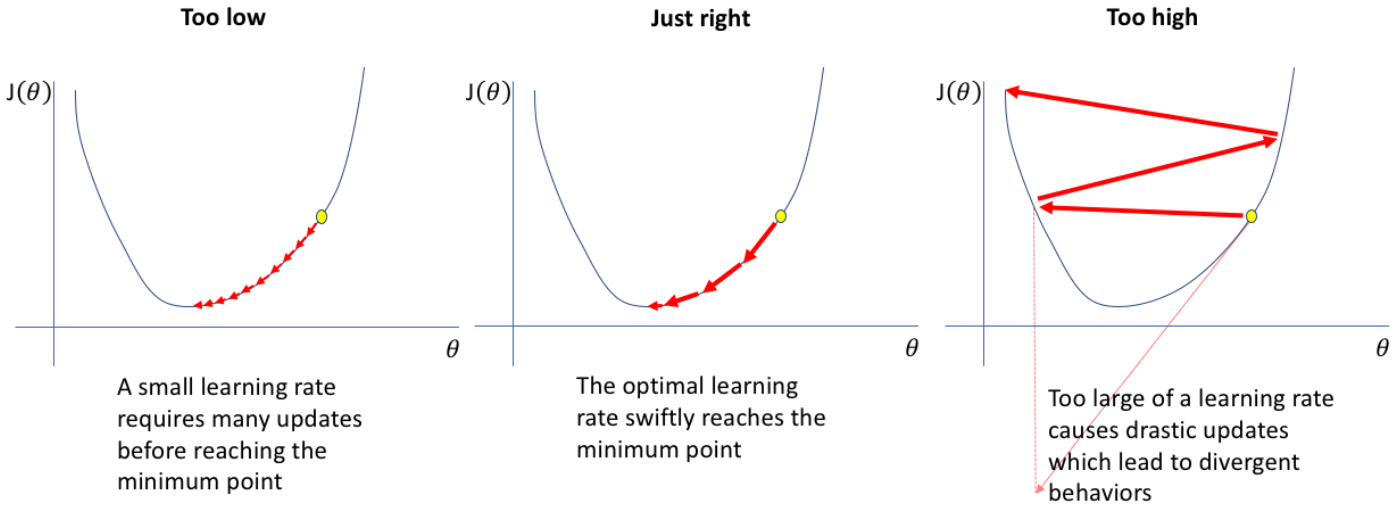
Training is done for a number of epochs (1 epoch is 1 pass over the training dataset). After every epoch, the loss function is also calculated for the validation data set. The number of epochs used for the training is a hyperparameter you have to define. It is also possible to automatically stop the training if the loss function is not going down anymore.



Training a NN – learning rate

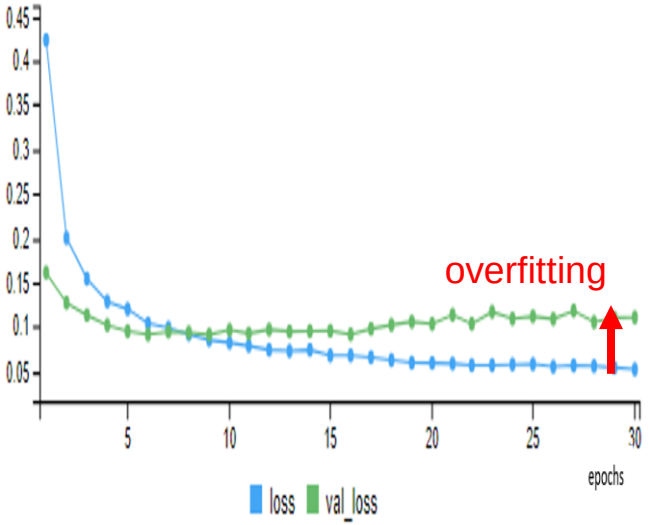
The learning rate is the step size in the gradient descent or adam optimizer

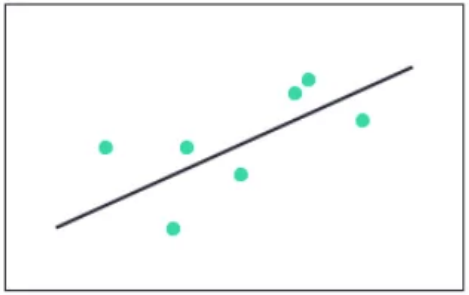
- A small learning rate will be slow and requires a lot of epochs
- A too large learning rate will not find the minimum of the loss function
- Typical learning rate are between 0.01 and 0.0001
- Sometimes a learning rate schedule is defined, starting with a bigger value and decreasing the step after a number of epochs



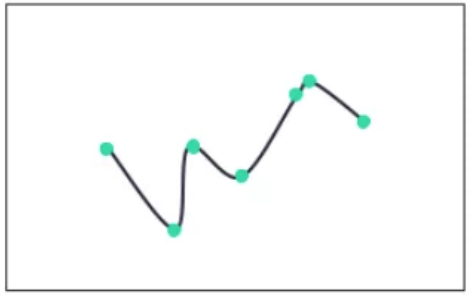
Training a NN – overfitting

Overfitting is happening when the model is performing well on the training dataset, but performing worse on the validation dataset (validation loss is higher than training loss). Most of the time, this is coming from the fact that the dataset is too small to train the model. Sometime data augmentation techniques are used to create more samples (e.g. create more pictures by rotating the existing ones a bit)





well generalized



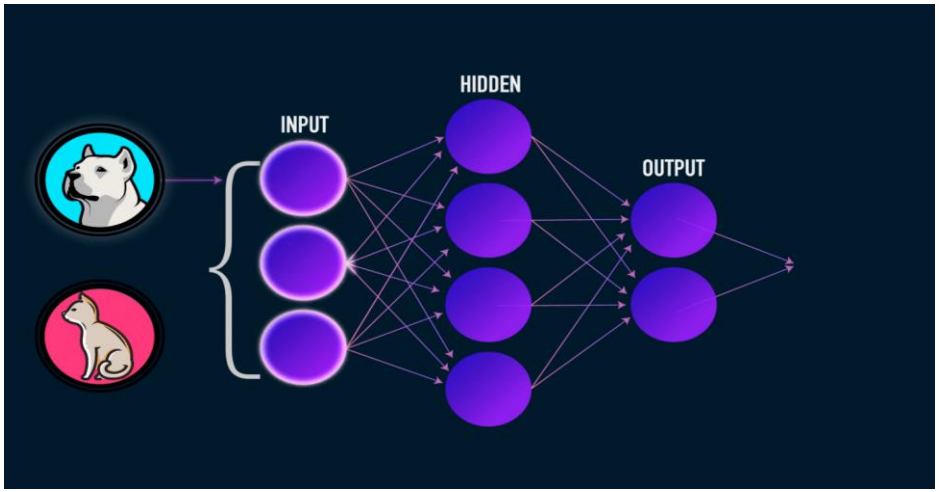
overfitting

Training a NN – hyperparameter tuning

A lot of hyperparameters need to be defined for the NN model

- Number of layers
- Number of nodes in the layers
- Number of epochs
- Batch size
- Learning rate
- Type of activation function
- Type of loss function
- …

In reality several model with different hyperparameters are training and the model giving the best results for the test dataset (as this one is not used during training) is chosen. This process is called hyperparameter tuning.

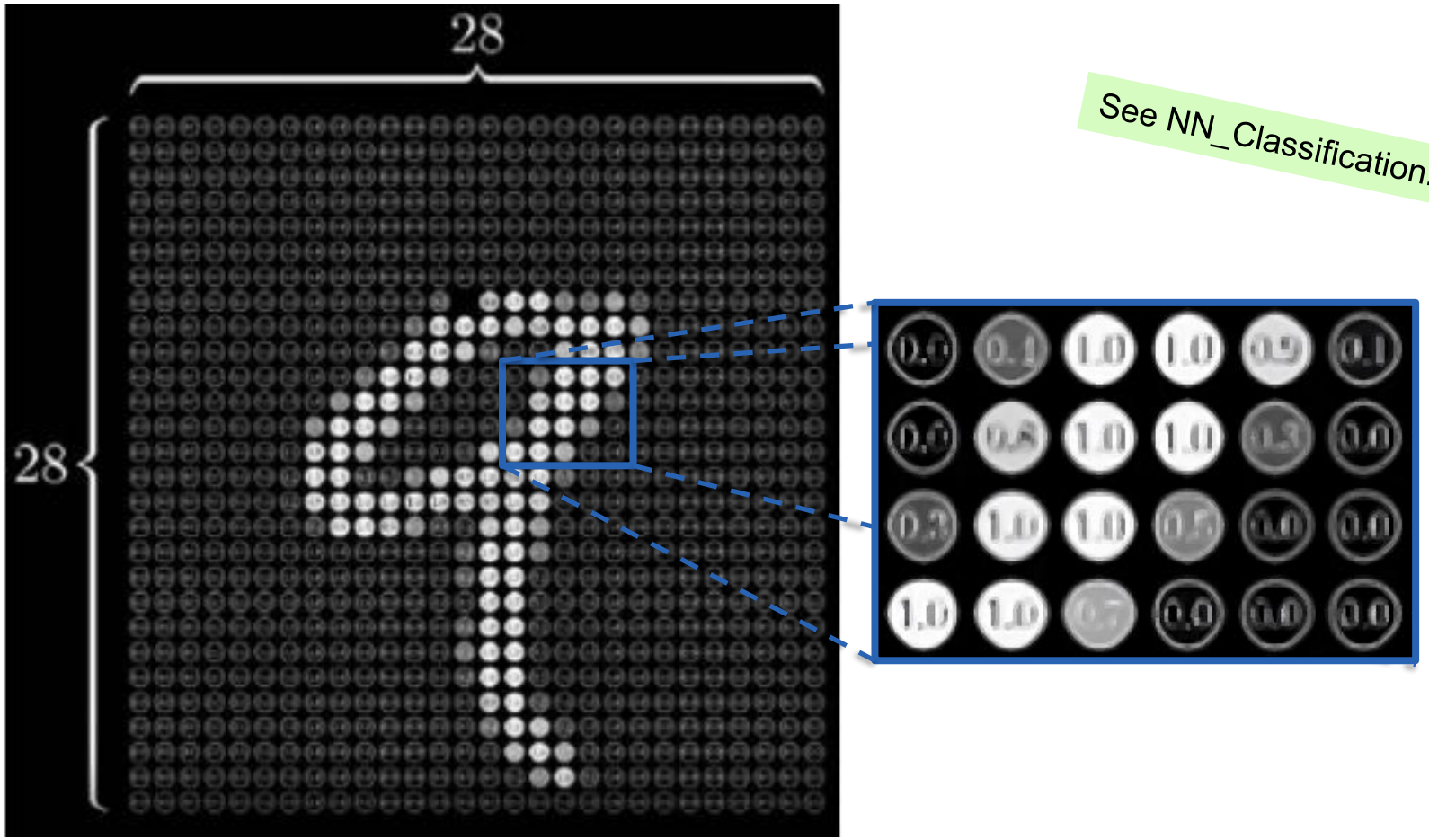


Neural Network – Classification

Classification

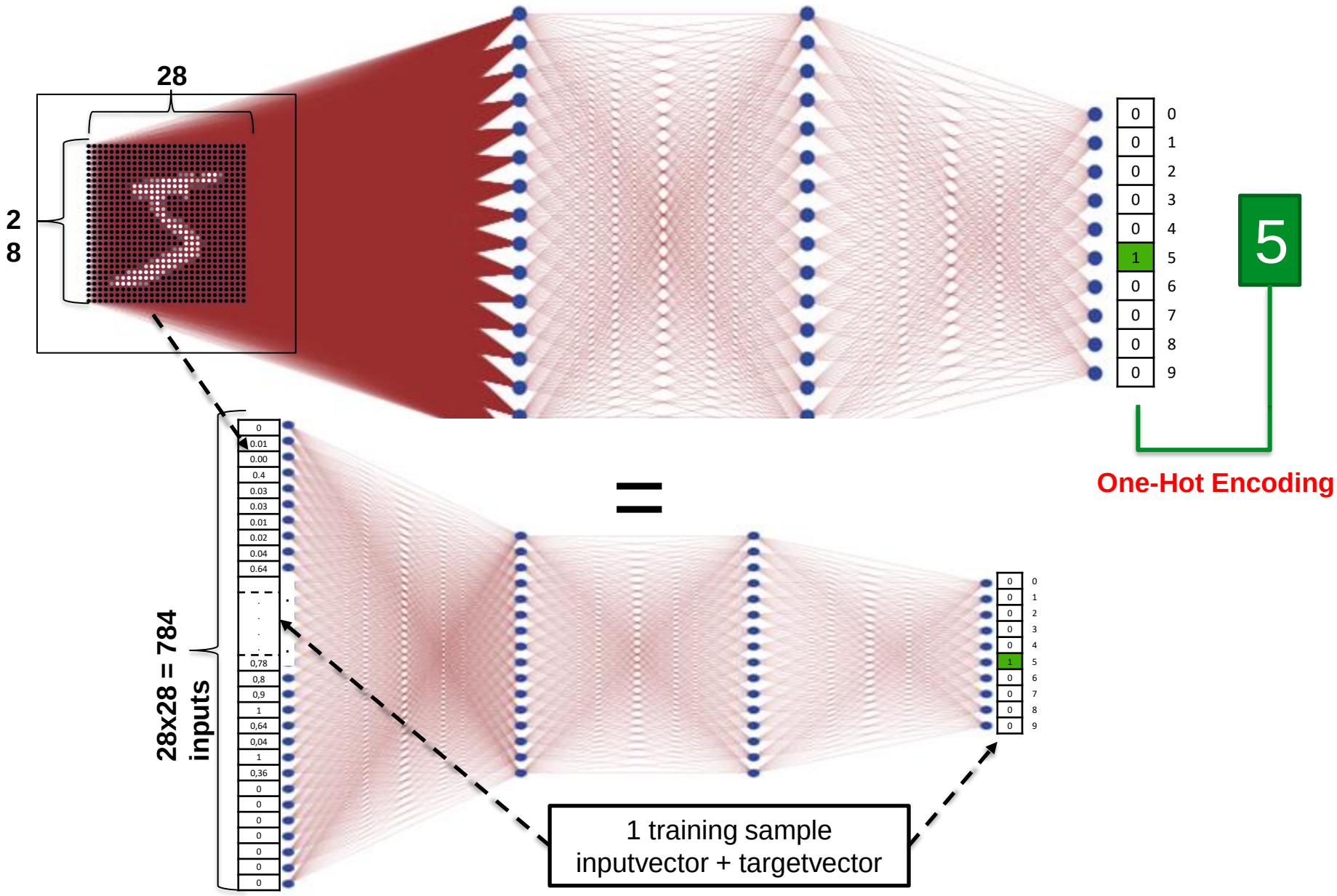
Classification models are used to predict discrete categories or classes.

Example: Make a model that can recognize handwritten digits.
MNIST dataset: The training dataset has 60000 images, the test dataset 10000. Images are grayscale and have a size of 28x28 pixels.



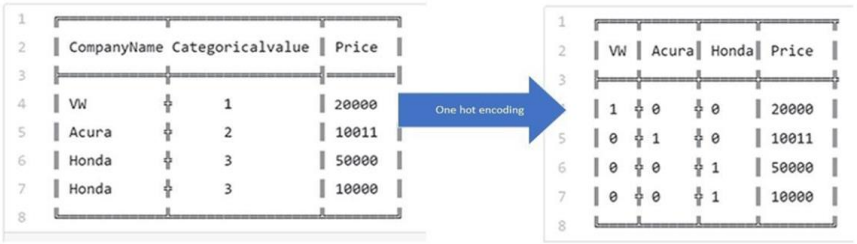
See NN_Classification.ipynb

Classification

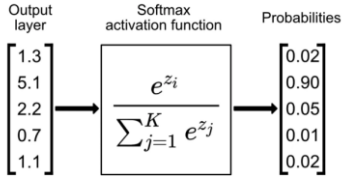


Classification

One-Hot Encoding: transforms categorical variable into binary columns



When the classification model predict a class, each output node will give the probability for the corresponding class. The sum of all the output nodes (sum of all the probabilities) is equal to 1. To enforce this, the softmax activation function is used for the output layer together with a categorical_crossentropy loss.



For one-hot encoding the to_categorical function from tensorflow.keras.utils can be used. In case the dataset is a pandas dateframe, the pd.get_dummies function can be used.

Remark: in case of a binary classification ("positive"/"negative"), it is also possible to have 1 output node (so not using one-hot encoding), giving the probability of "positive". In this case a sigmoid activation is used (giving a value between 0 and 1) together with a binary_crossentropy loss function.

Define the model

```
# input layer: 784 nodes
inputs = Input(shape=(784,))
# 2 hidden layers: 128 and 64 nodes and relu activation
x = Dense(128, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
# output layer: 10 nodes and softmax activation
outputs = Dense(10, activation='softmax')(x)
# construct the model
model = Model(inputs, outputs, name='MNIST')
# print a summary
model.summary()
# compile the model
# categorical_crossentropy loss
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

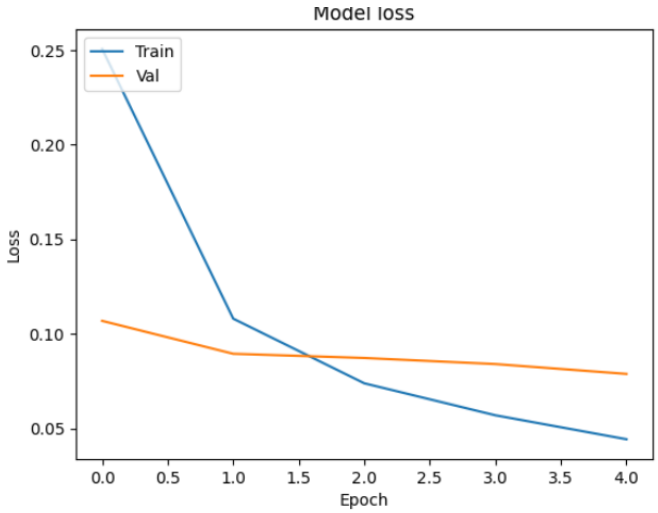
```
Model: "MNIST"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 109386 (427.29 KB)
Trainable params: 109386 (427.29 KB)
Non-trainable params: 0 (0.00 Byte)
```

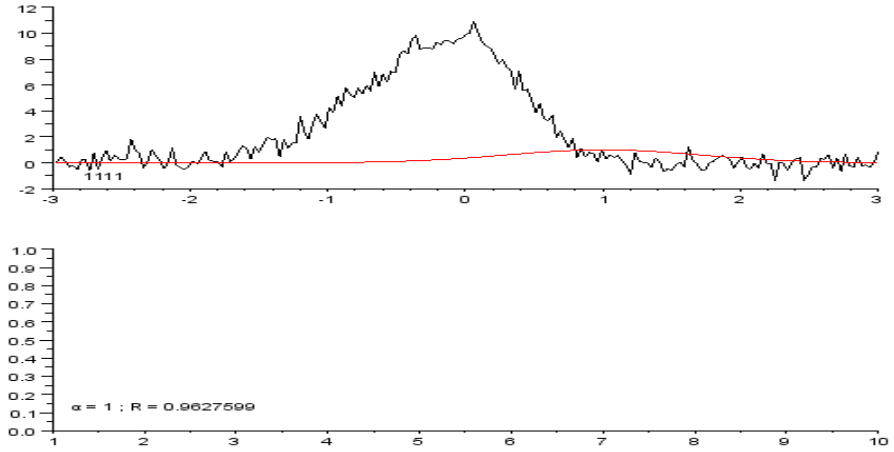
Train, evaluate and predict

```
# train the model
history = model.fit(
    x_train_norm,           # training input
    y_train_onehot,         # training targets
    epochs=5,
    batch_size=32,
    validation_split=0.1,   # 10% of training data used for validation
)
```

```
# evaluate
model.evaluate(x_test_norm,y_test_onehot)
```

```
# predict
predicted = model.predict(x_test_norm)
```





Neural Network – Regression

4

Regression

See NN_Regression.ipynb

Regression models are used to predict continuous values

- For regression usually a linear activation is used for the output node.
- The loss function can be an mean squared error (MSE) or mean absolute error (MAE) loss
- As metric, mean absolute percentage error (MAPE) can be used

	output activation	loss	metrics
regression	linear	mean_squared_error	mean_absolute_percentage_error
		mean_absolute_error	mean_absolute_percentage_error
classificatie one-hot encoding	softmax	categorical_crossentropy	accuracy
binary classification	sigmoid	binary_crossentropy	accuracy

Exercise time!

See NN_Exercises.ipynb