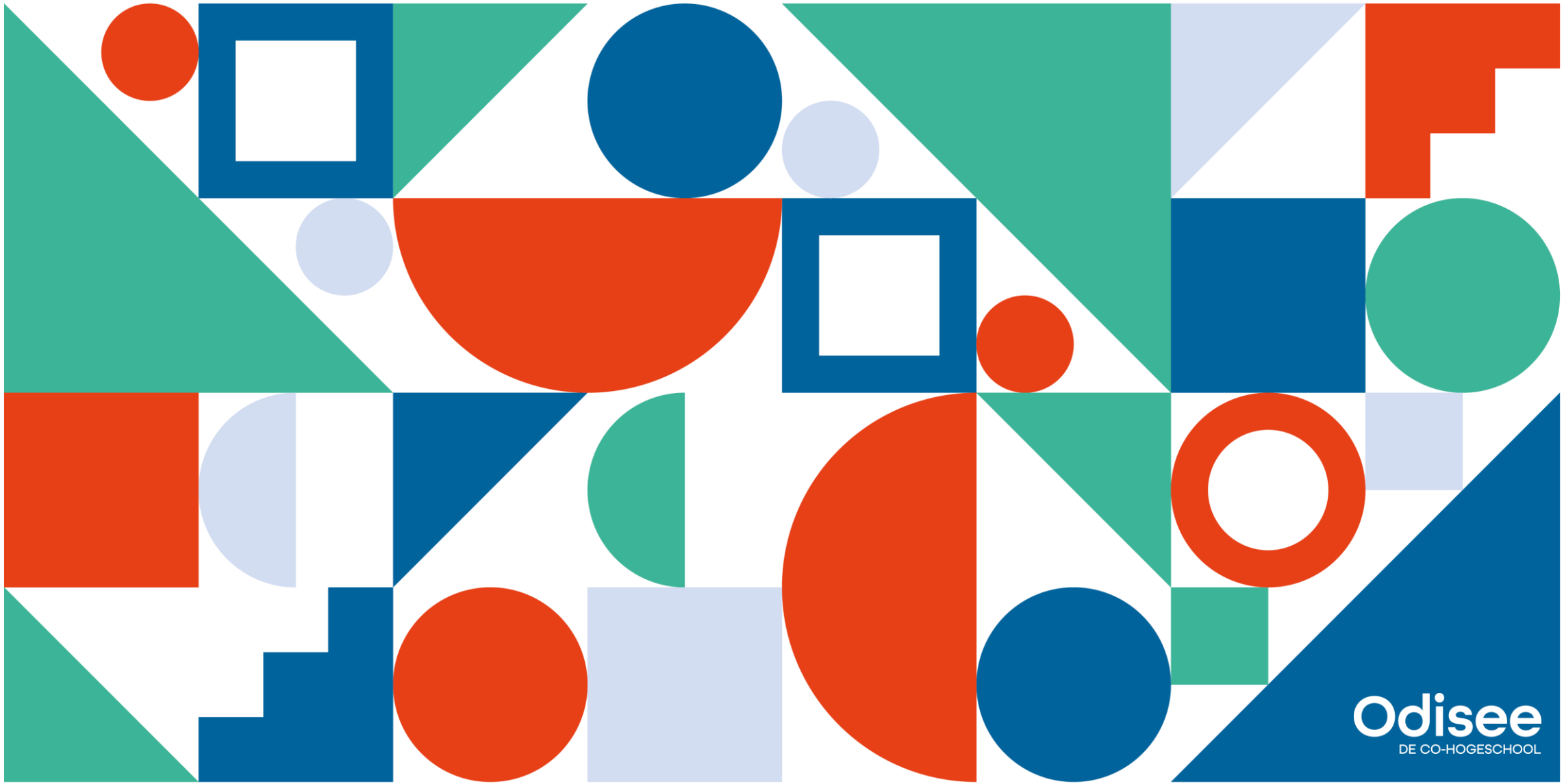

Odisee
DE CO-HOGESCHOOL

# Neurale netwerken

Jens Baetens

# Basic ML with Tensorflow

# Machine learning

- Wat waren jullie vragen over de tensorflow basics?

# Tensorboard

- Visualisatie en tooling voor experimenteren met machine learning
- Bijhouden en visualiseren van verscheidene metrieken
  - Zoals accuraatheid
- Visualiseren gebruikte model
- Visualiseren van histogrammen van data en weights
- Tonen van de data
- Profiling/Performantie bepalen van tensorflow programma's
- …

# Tensorboard

```
import tensorflow as tf
from datetime import datetime

%load_ext tensorboard
```

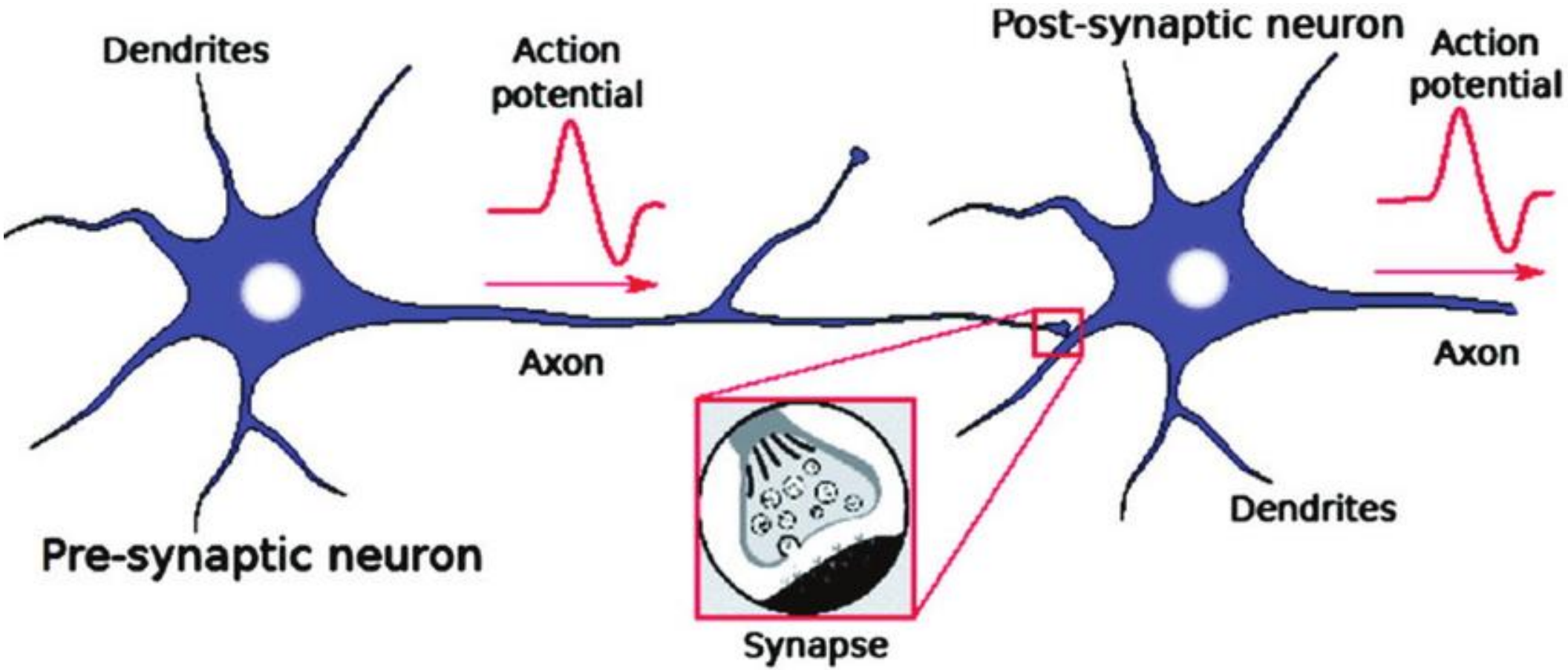
```
#docs_infra: no_execute
%tensorboard --logdir logs/func
```

```
# Set up logging.
stamp = datetime.now().strftime("%Y%m%d-%H%M%S")
logdir = "logs/func/%s" % stamp
writer = tf.summary.create_file_writer(logdir)

# Create a new model to get a fresh trace
# Otherwise the summary will not see the graph.
new_model = MySequentialModule()

# Bracket the function call with
# tf.summary.trace_on() and tf.summary.trace_export().
tf.summary.trace_on(graph=True)
tf.profiler.experimental.start(logdir)
# Call only one tf.function when tracing.
z = print(new_model(tf.constant([[2.0, 2.0, 2.0]])))
with writer.as_default():
  tf.summary.trace_export(
      name="my_func_trace",
      step=0,
      profiler_outdir=logdir)
```

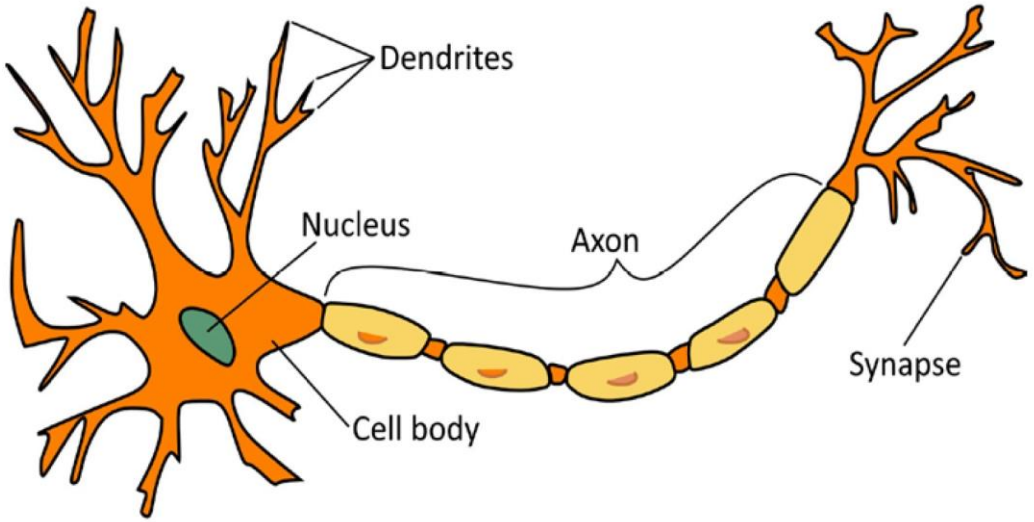
# Training loop

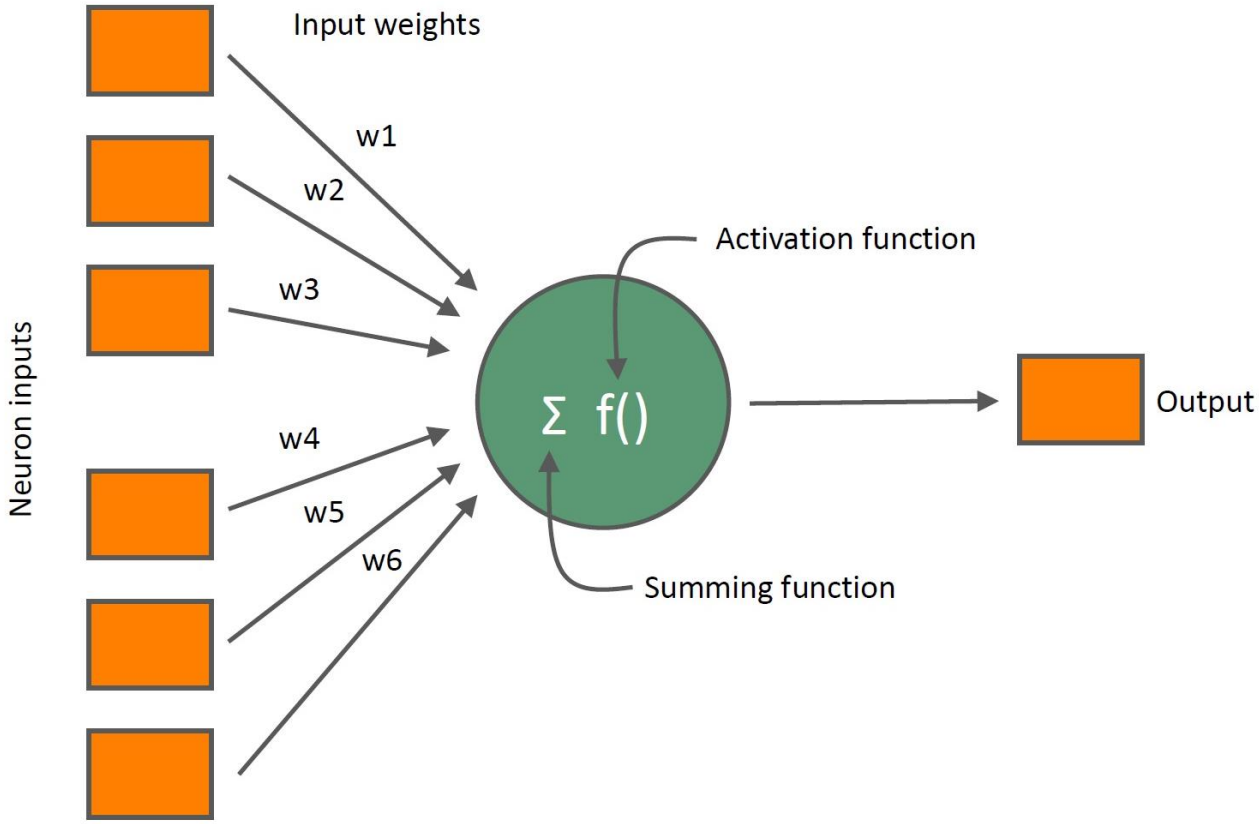
- Verzamel trainingsdata
- Definieer een model
- Kies een loss functie
  - Hoe ver zit het model ernaast als het een output genereert
- Maak voorspelling van de trainingsdata en bepaal alle fouten
- Bereken de afgeleide (gradient) voor de fouten
- Optimaliseer de variabelen op basis van deze gradient
- Evalueer de resultaten met testdata

# Neurale netwerken


Dendrites
Action potential
Post-synaptic neuron
Action potential
Axon
Axon
Pre-synaptic neuron
Dendrites
Synapse

# Biologisch naar artificieel neuron



# Een artificieel neuron

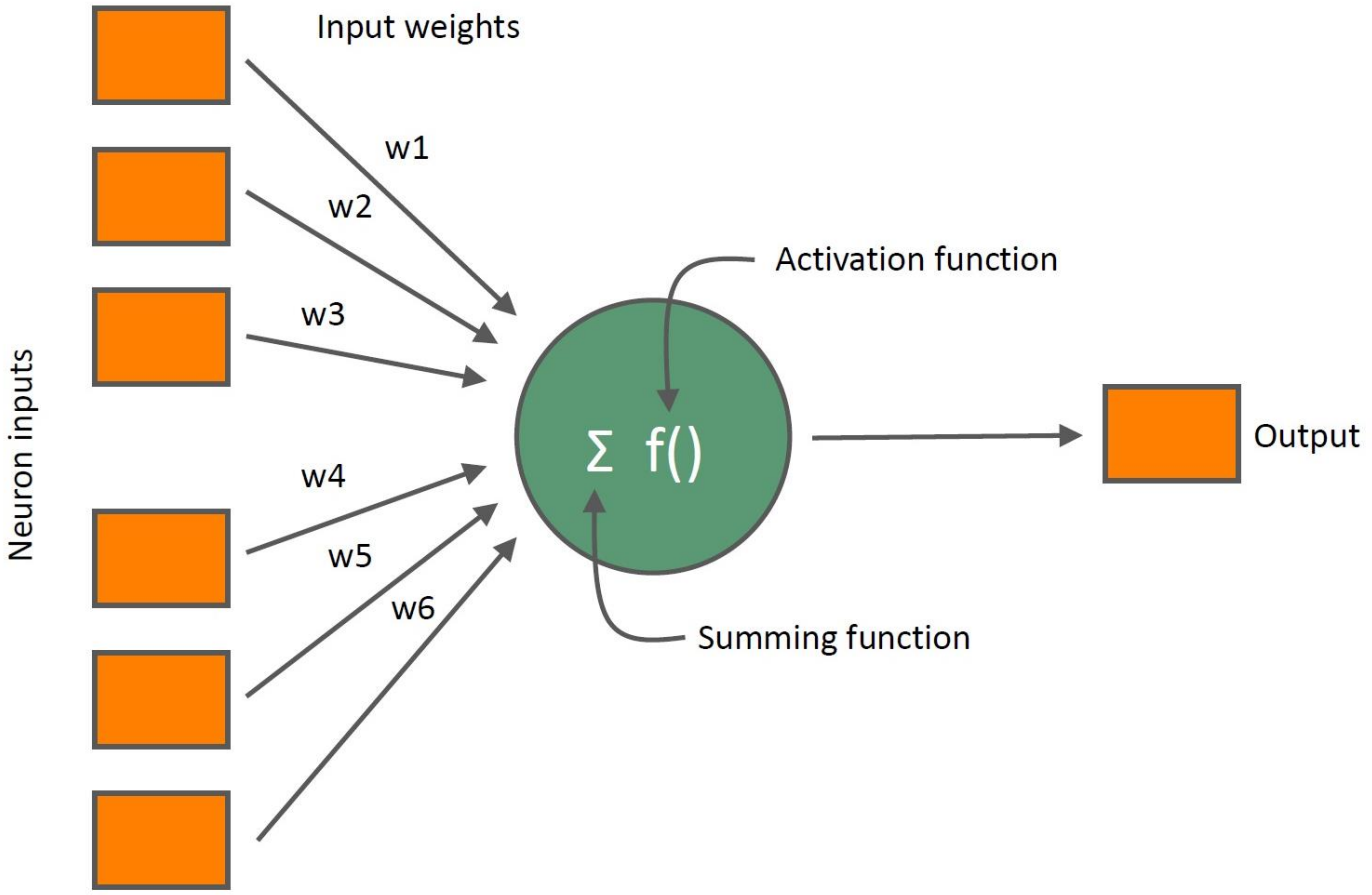
- Summing function

$$\mu = w_1 * x_1 + w_2 * x_2 + \cdots w_6 * x_6$$

- Activation function

$$y = f(\mu) = \frac{1}{1-e^{-\mu}}$$

- De gewichten zijn parameters
  - Elk neuron zijn eigen gewichten
  - Deze moeten getrained worden
- Activatiefunctie is een hyperparameter



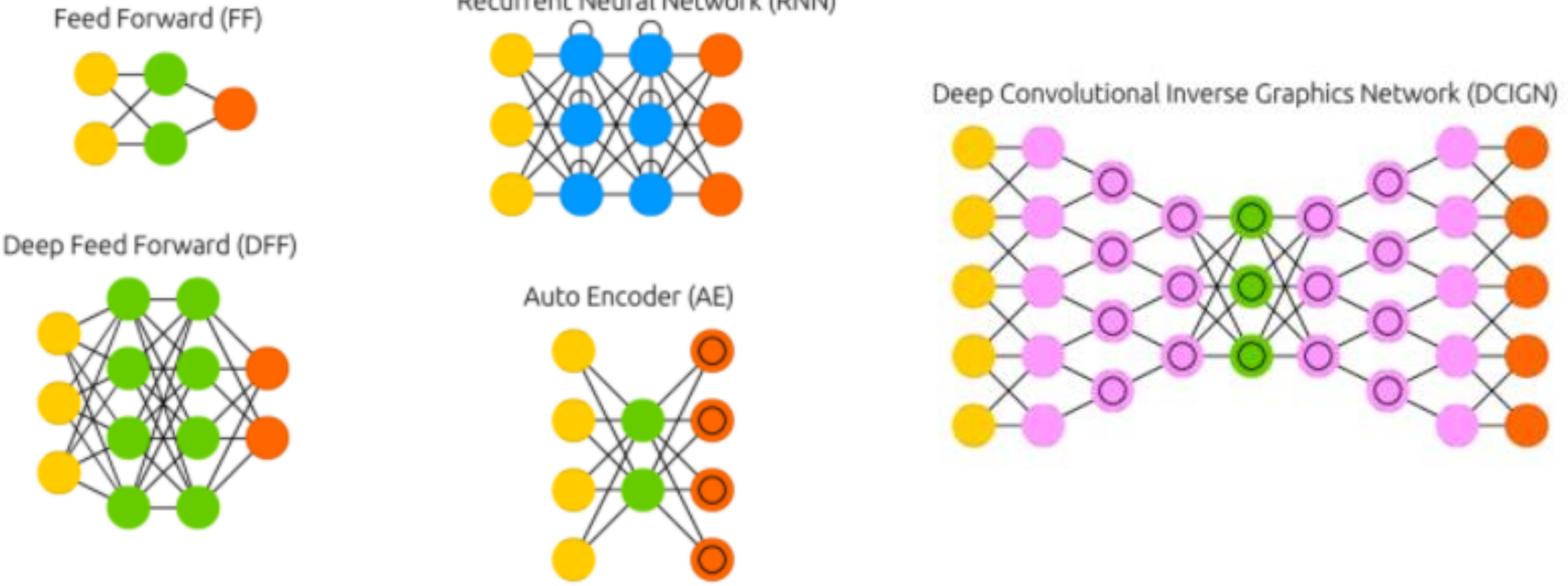
# Analogie

- Lineaire regressie
  - 1 neuron met lineaire activatiefunctie
- Logistische regressie
  - 1 neuron met logistische activatiefunctie

- Neuraal netwerk
  - Soort van ensemble techniek

# Netwerken van neurons

- https://www.asimovinstitute.org/neural-network-zoo/

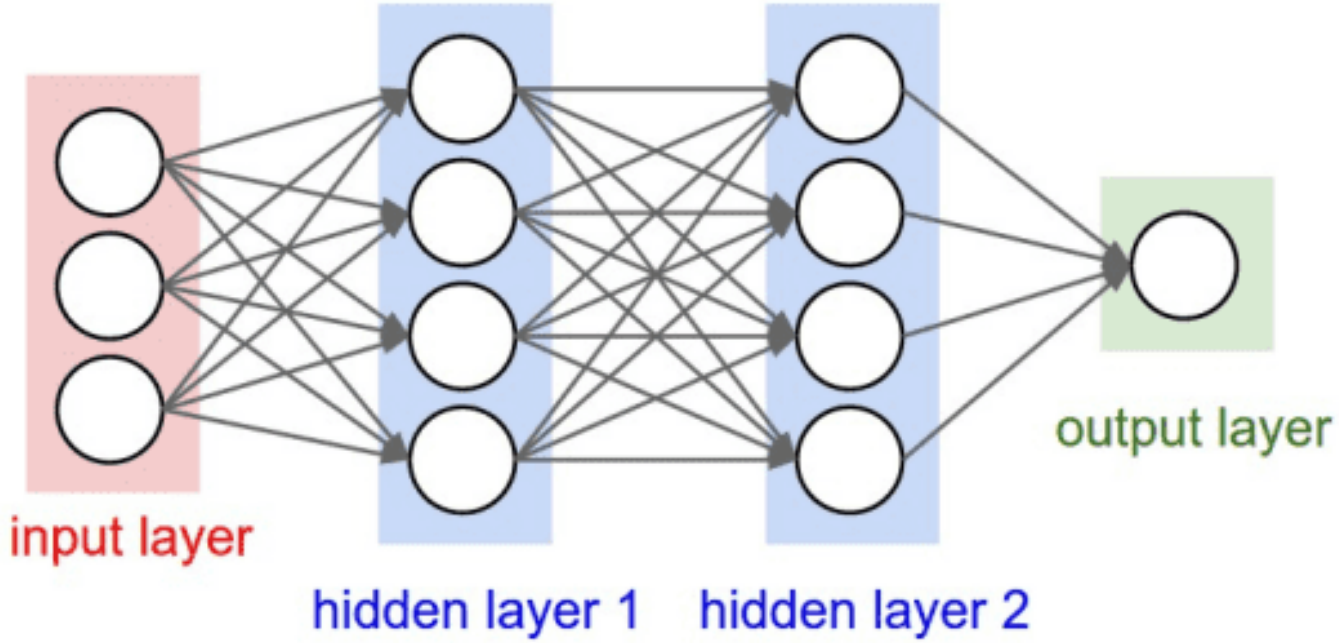


# Zijn de volgende termen parameters of hyperparameters

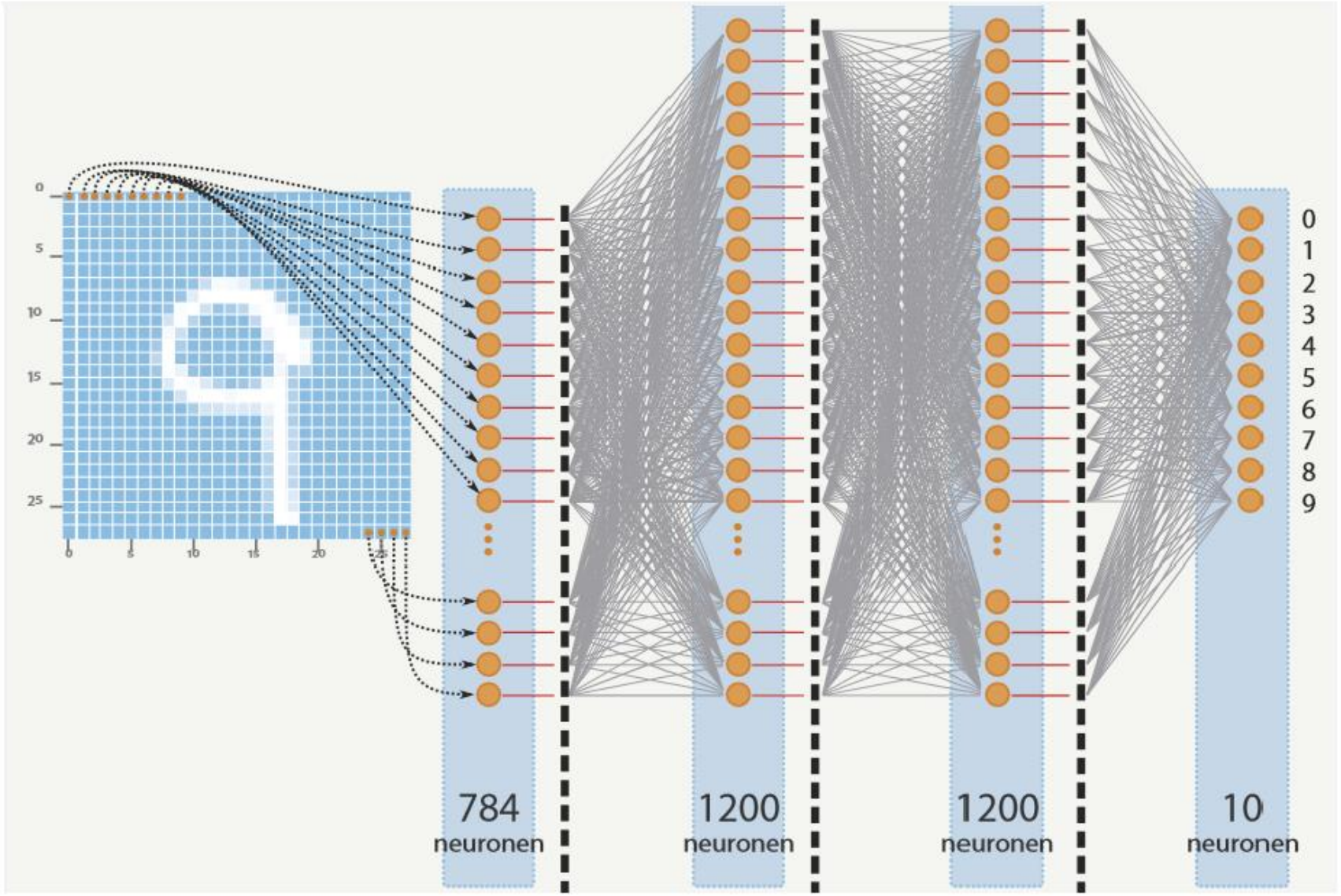
- De activatie functie
- Aantal lagen
- De gewichten van de neuronen
- Aantal neuronen per laag

# Feedforward neuraal netwerk

- Hoeveel gewichten?
- Informatie van links naar rechts

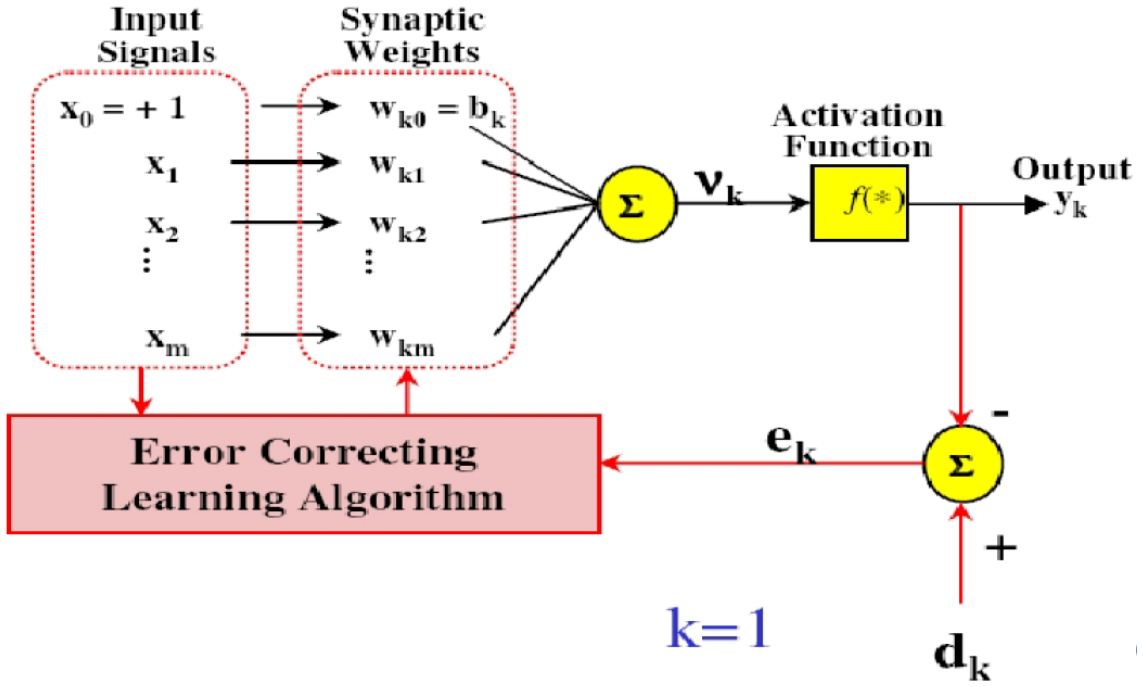


- Aantal gewichten?
- Output is de kans van elke klasse
- One-hot encoding
- Veel parameters
  - Meer data nodig



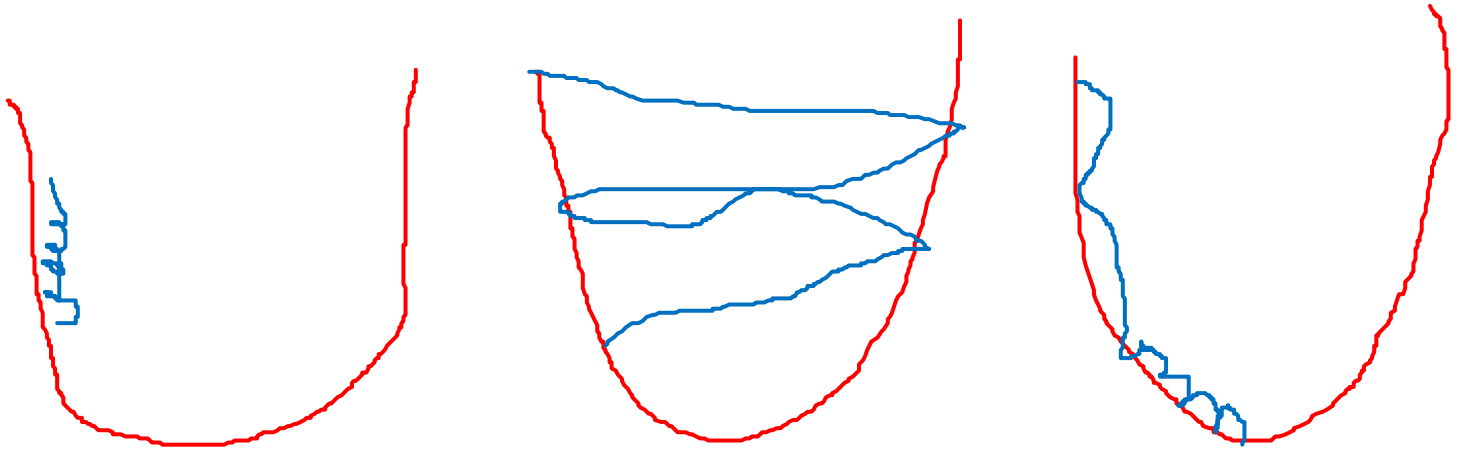
# Hoe leert een neuraal netwerk?

- Wiskunde vrij complex
- Omgekeerde beweging van feed forward
- Via Gradient Descent

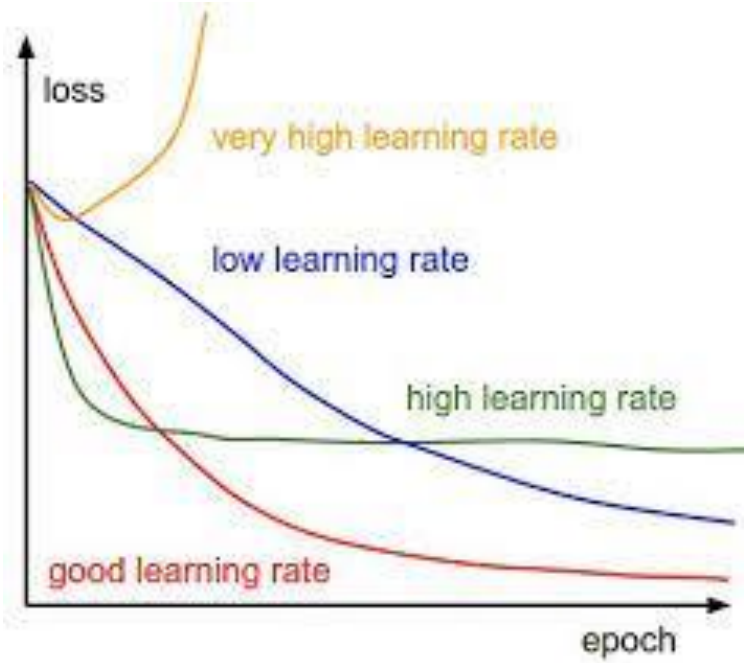


# Back propagation

# Backpropagation

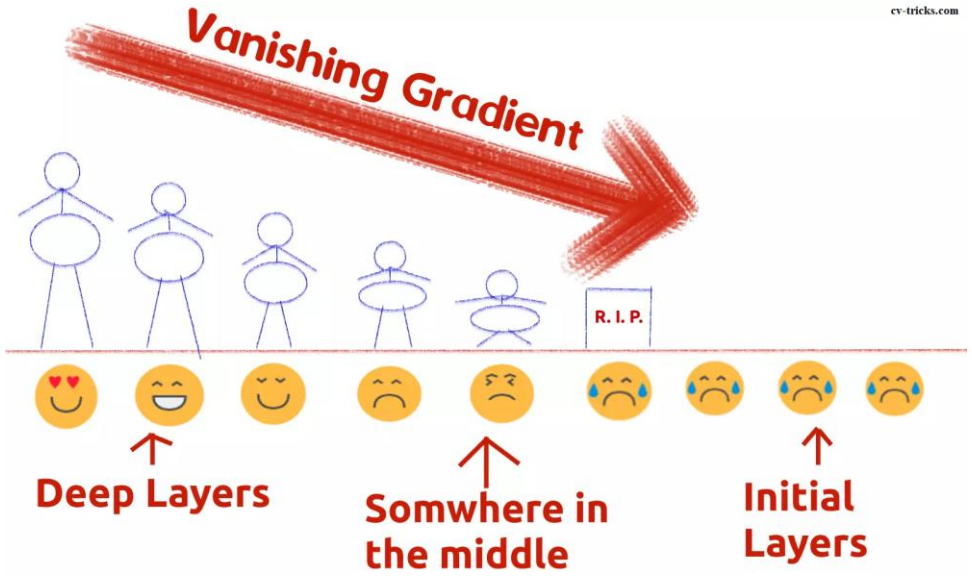


- Error functie
  - Afhankelijk van een learning rate
    - Standaard algoritme hiervoor is Adam
  - Epochs = aantal keer dat de volledige dataset getoond is aan het netwerk



# Backpropagation

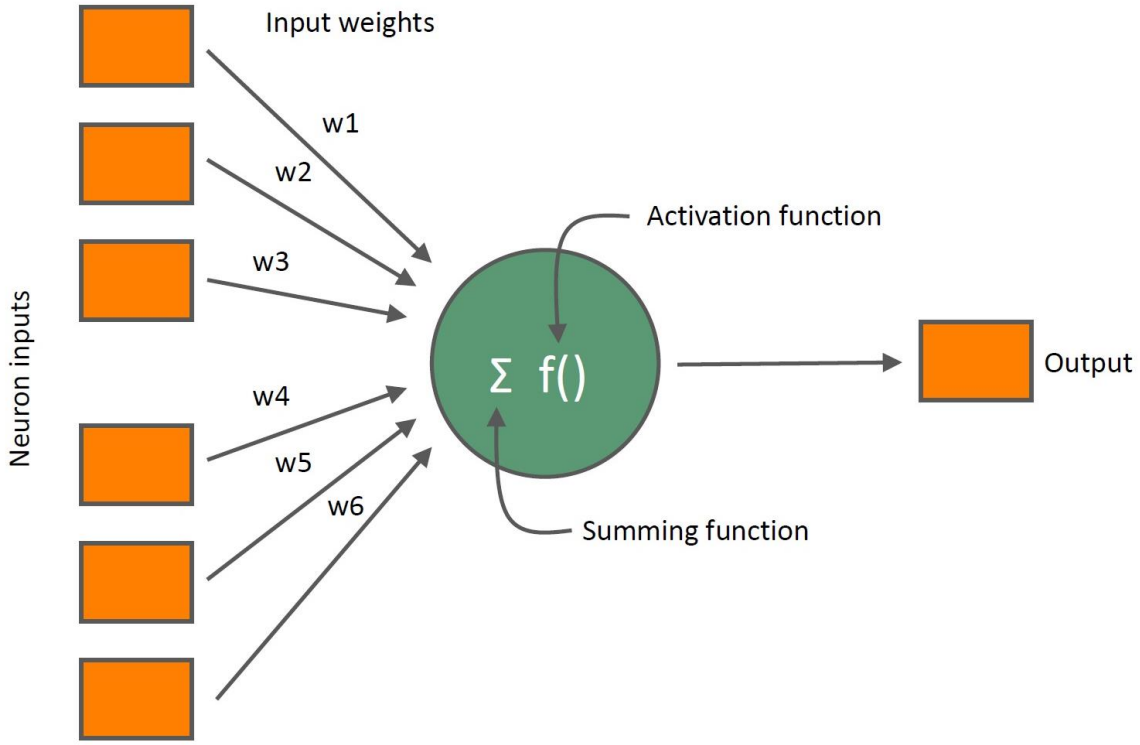
- Vanishing gradient problem
  - Met meerdere lagen wordt de gradient zo goed als 0
    - Door afgeleiden te nemen en herhaaldelijk te vermenigvuldigen met iets dat kleiner is dan 1
  - Zorgt ervoor dat het heel moeilijk te trainen is



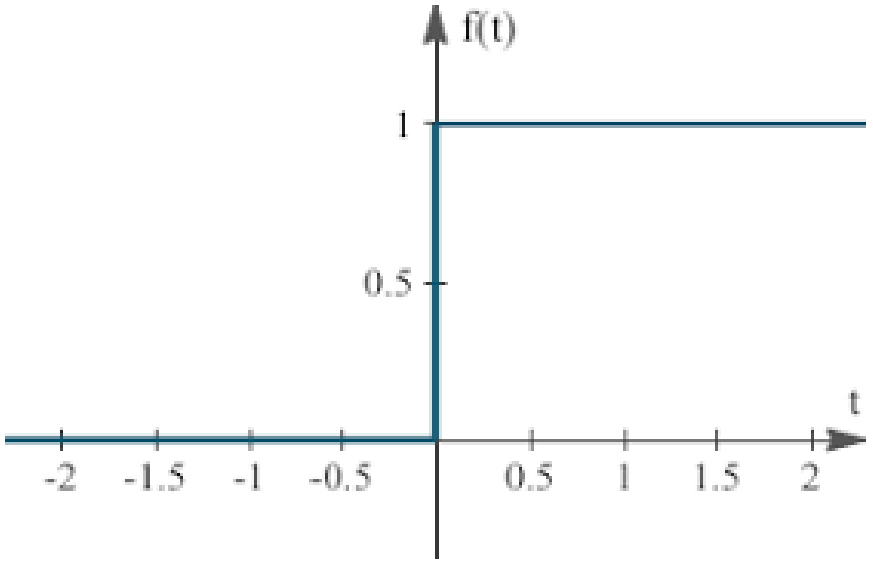
# Activation function

- Functie dat de som van alle inputs omzet naar een bepaald getal

- Moet niet-lineair zijn
  - Anders kan een neuron niet leren
  - Aantal lagen heeft geen impact
  - Is het gewoon lineaire regressie



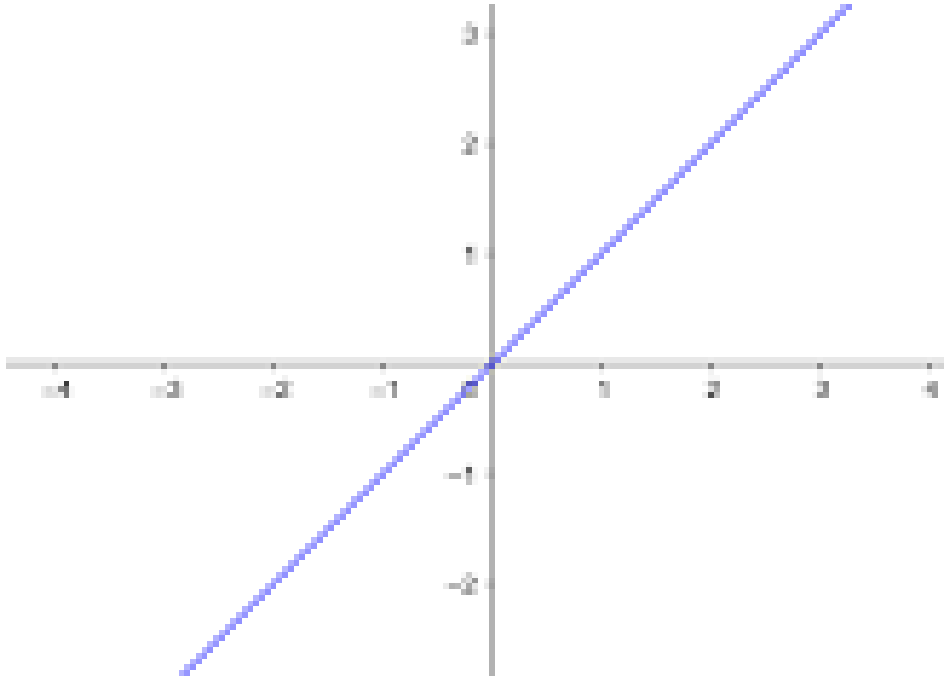
# Activation function: step function




- X>0 -> output = 1
- X<0 -> output = 0
- Nadelen
  - Enkel ja of nee
  - Geen indicatie hoe verkeerd het neuron is
  - Backpropagation werkt niet omdat afgeleide 0 is
  - Meerdere klassen die op 1 staan -> welke output is het dan?

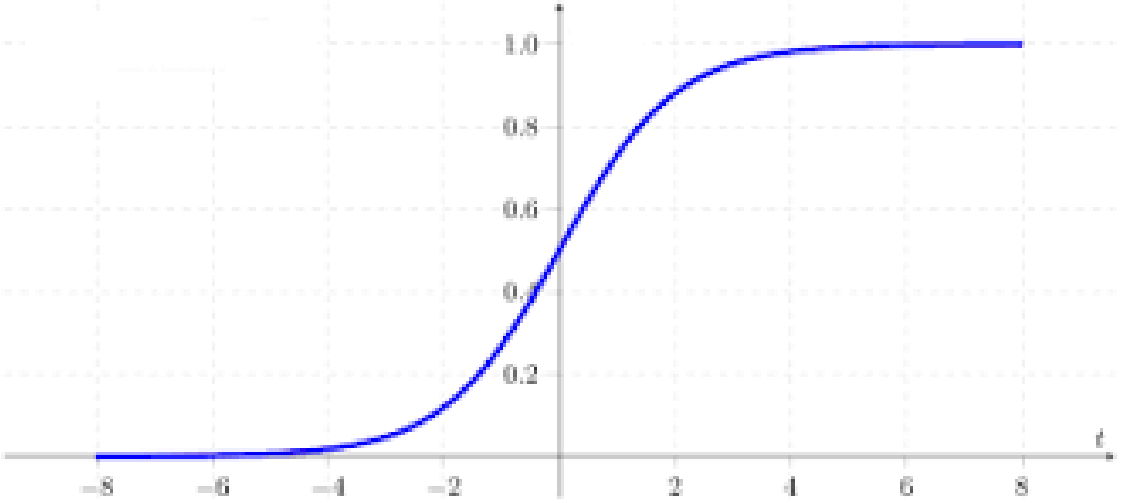
# Activation function: lineaire functie

- Output = X*a
- Nadelen
  - Enkel lineaire scheidingen mogelijk
  - Afgeleide
- Gebruikt voor
  - Input layer
  - Output layer voor regressie



# Activation function: sigmoid function

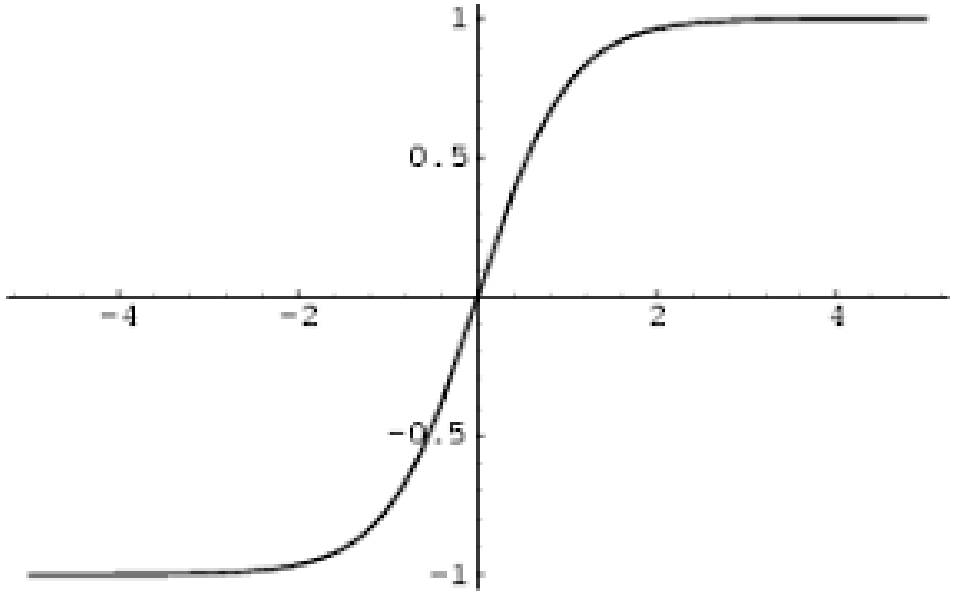
$$f(x) = \frac{1}{1+e^{-x}}$$




- Nadelen
  - Vanishing gradient problem
    - Afgeleidde gaat naar 0 als x groot is.
  - Rekenintensief
- Gebruikt  voor:
  - Output layer voor classificatie

# Activation function: Hyperbolic tangent

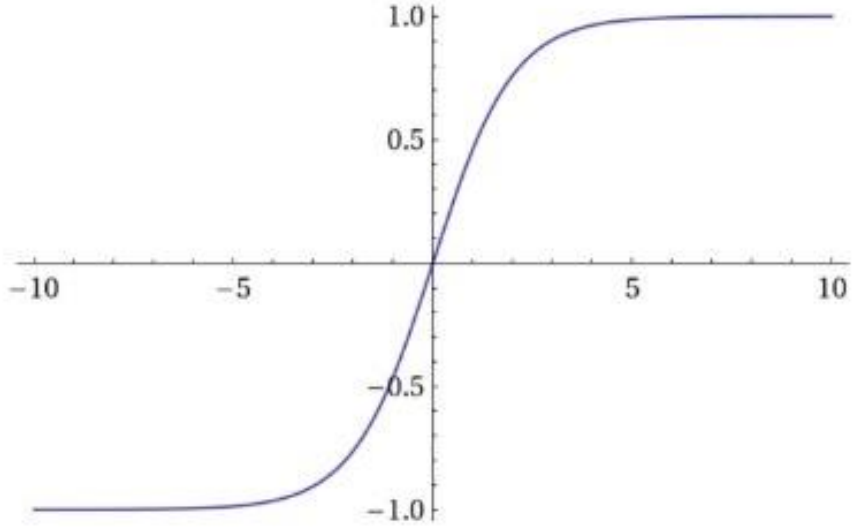
$$f(x) = \tanh(x)$$




- Nadelen
  - Vanishing gradient problem
  - Rekenintensief
- Gebruikt voor
  - Zelden maar kan voor output in classificatie

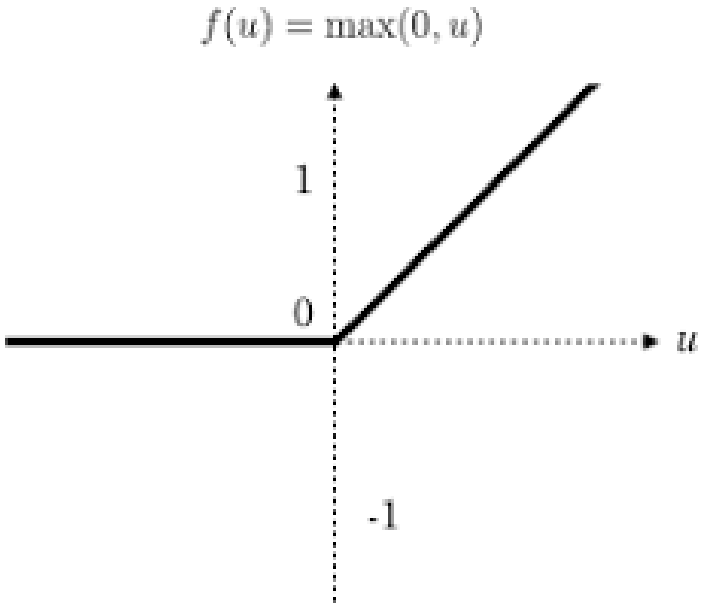
# Activation function: softmax function

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



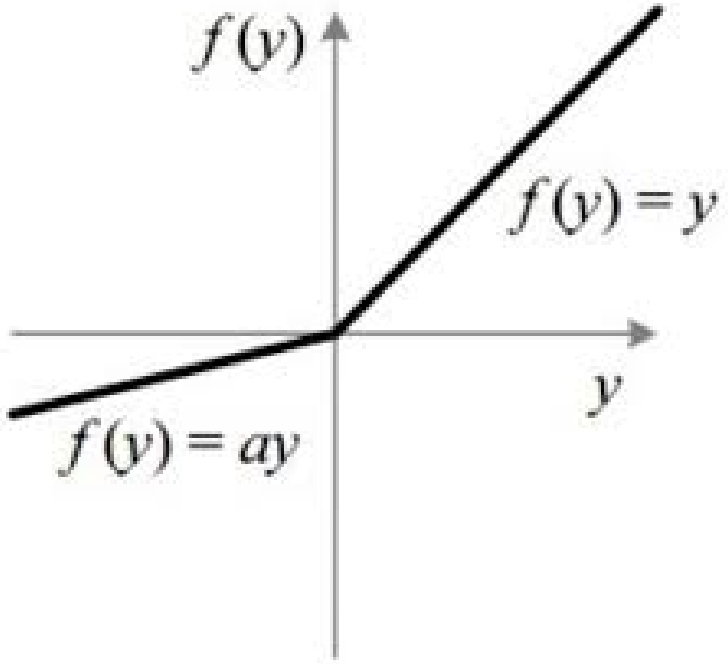

- Generalisatie van sigmoid
- Gebruikt voor
  - Output layer voor classificatie
  - Kan meerdere klassen tegelijkertijd aangeven (multi-label)

# Activation function: Rectified linear unit




- Output = max(0,x)
- Voordelen
  - Kan alle functies benaderen
  - Rekenefficient
  - Sparse activation
- Nadelen
  - Dode neurons blijven dood
- Gebruikt voor hidden layers

# Activation function: Leaky Relu



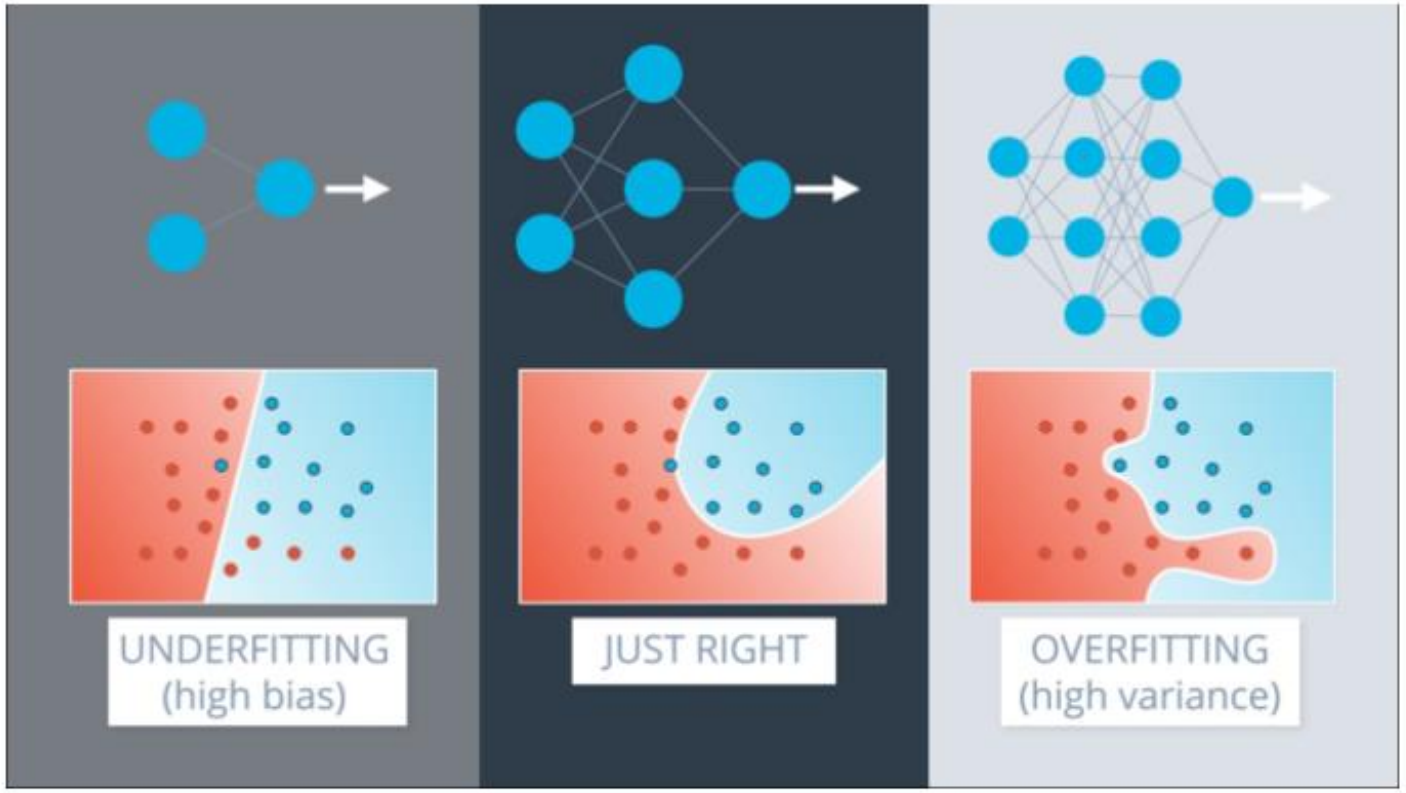

- Voordelen
  - Zelfde als Relu functie
  - Neurons gaan niet dood
- Nadelen
  - Meer parameters om te trainen
- Gebruikt voor hidden layers

# Activation functions - samenvatting

- Hidden layers
  - Relu eerste keuze, probeer erna leaky relu
  - Geen sigmoid of tanh
- Output layer
  - Regressie -> lineaire activatie functie
  - Classificatie
    - Input kan maar tot 1 klasse behoren -> sigmoid
    - Input kan tot meerdere klassen horen -> softmax

# Underfitting vs overfitting

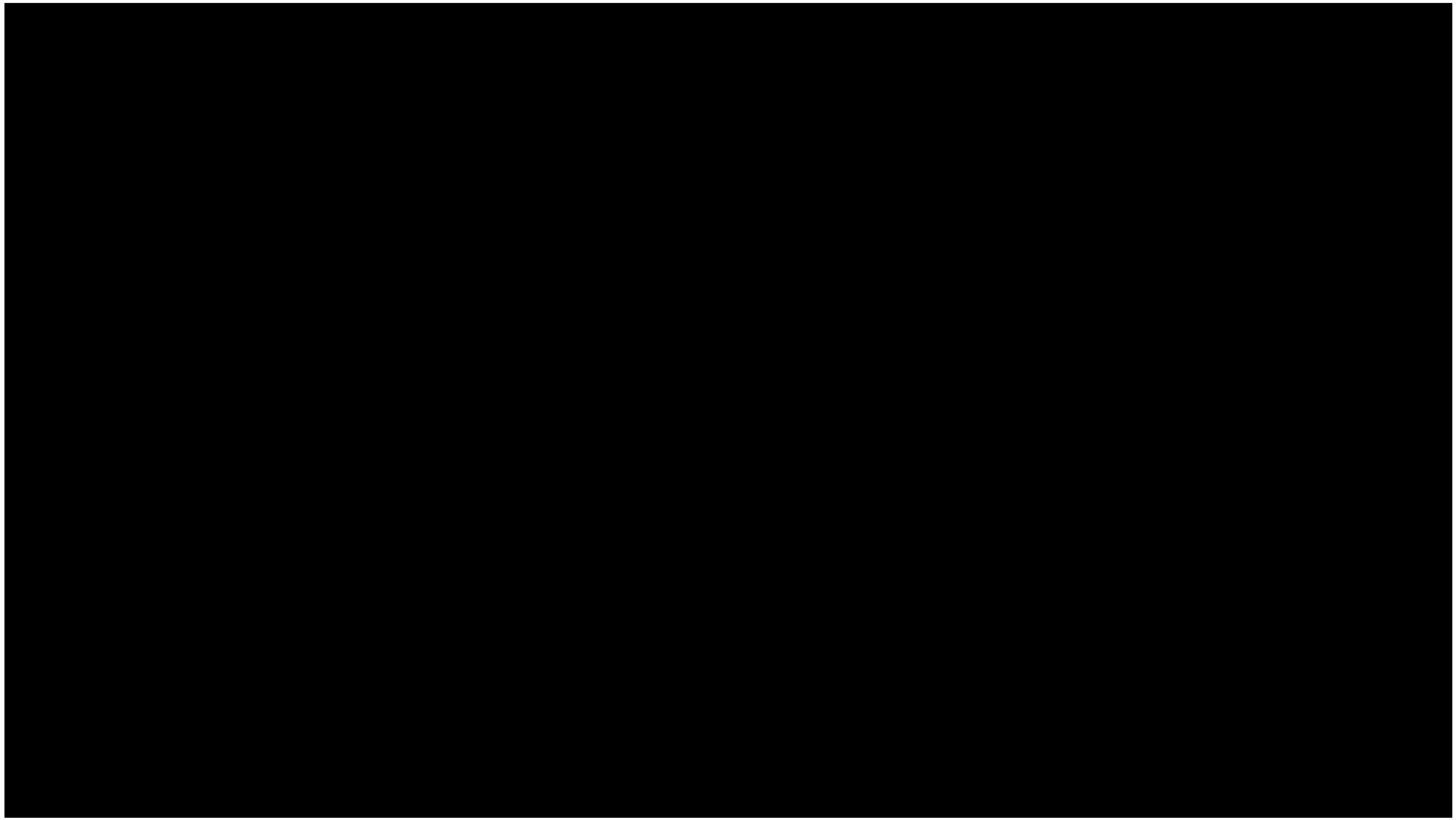
- Overfitting
  - Leert trainingsdata te sterk
  - Testdata niet voldoende
  - Netwerk te groot of onvoldoende data
- Underfitting
  - Kan input niet goed modelleren
  - Te klein netwerk



# Hoe overfitting tegengaan?

- Weight regularisation
  - Zorg ervoor dat de gewichten zo klein mogelijk zijn
  - Hierdoor focust het netwerk op inputs die belangrijk zijn
  - Extra kost in de loss/error functie op basis van de L2 of L1 norm
    - L2-norm = som van de kwadraten van de gewichten
    - L1-norm = som van de absolute waarden van de gewichten

# Hoe overfitting tegengaan?



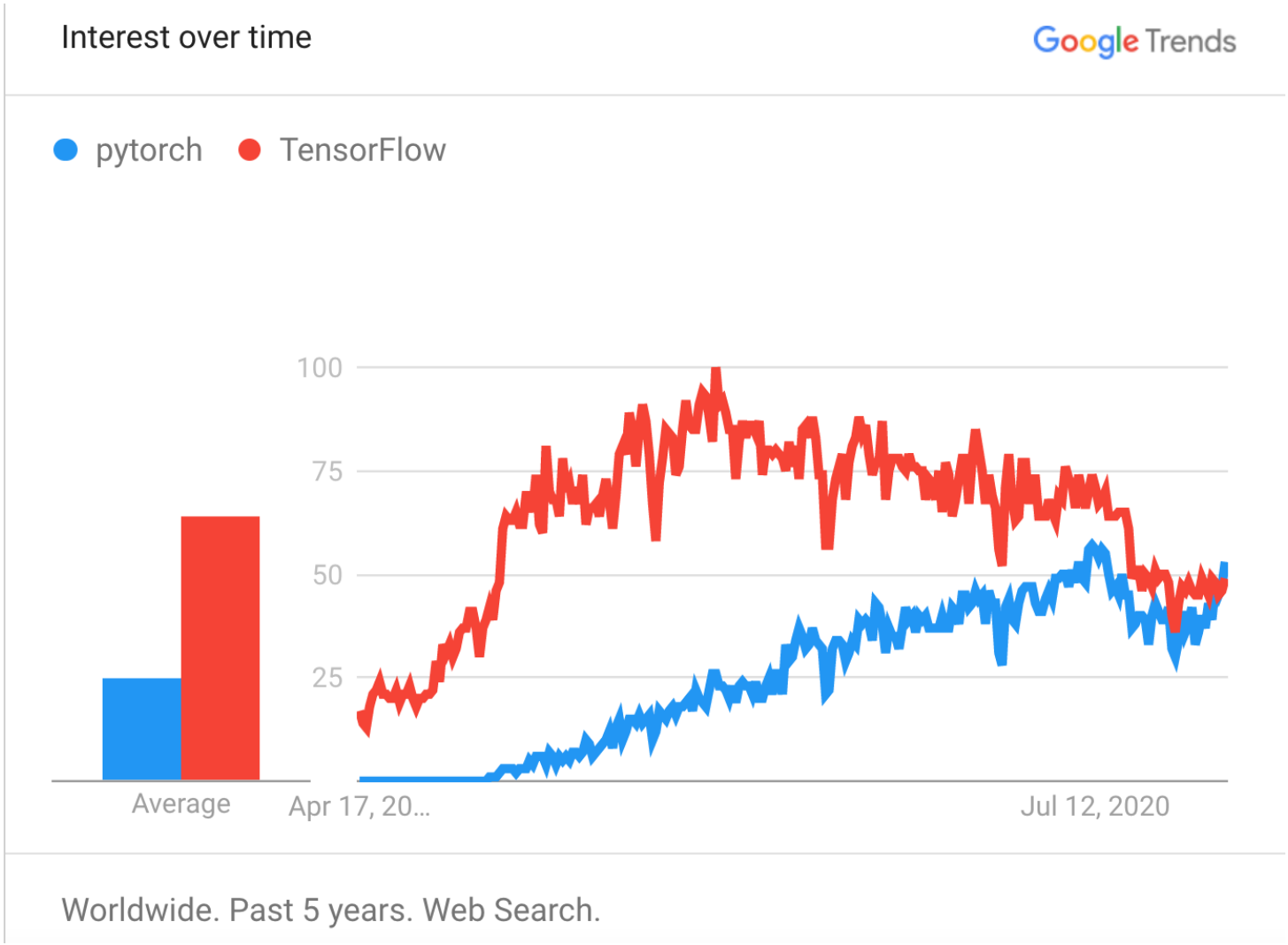
https://www.youtube.com/watch?v=NhZVe50QwPM&t

# Hoe overfitting tegengaan

- Techniek om overfitting te voorkomen
- Willekeurig neurons uitschakelen bij training
  - Andere neurons moeten inspringen om een goed resultaat te bekomen
  - Vermijd dat andere neurons afsterven

- Bootst een ensemble van netwerken na in 1 netwerk
  - Robuster en accurater model

# Hoe neuraal netwerk implementeren?



# Neurale netwerken met Tensorflow

# The sequential model

- Stack of layers
  - Achter elkaar (sequence) uitgevoerd
  - Elke laag heeft 1 input en output tensor (niet  1 waarde)

```
# Define Sequential model with 3 layers
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
# Call model on a test input
x = tf.ones((3, 3))
y = model(x)
```

# The sequential model

- Dit model kan getrained worden waarbij alle lagen getrained worden
- Kan geëvalueerd worden
- Kan bewaard worden op en ingeladen worden van de harde schijf
- Werking kan versneld worden door gebruik van eventuele GPU’s.

# The sequential model

- Feature extraction
  - Extract de output van 1 of meerdere hidden layers

```
initial_model = keras.Sequential(
    [
        keras.Input(shape=(250, 250, 3)),
        layers.Conv2D(32, 5, strides=2, activation="relu"),
        layers.Conv2D(32, 3, activation="relu", name="my_intermediate_layer"),
        layers.Conv2D(32, 3, activation="relu"),
    ]
)
feature_extractor = keras.Model(
    inputs=initial_model.inputs,
    outputs=initial_model.get_layer(name="my_intermediate_layer").output,
)
# Call feature extractor on test input.
x = tf.ones((1, 250, 250, 3))
features = feature_extractor(x)
```

# The sequential model

- Transfer learning
  - Veel gebruikt voor pre-trained models aan te passen
  - Bevries de eerste lagen van het model en train enkel de laatste laag/lagen
  - Gebruik een generiek model dat reeds getrained is en specialiseer het voor je doel

```
model = keras.Sequential([
    keras.Input(shape=(784)),
    layers.Dense(32, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(10),
])

# Presumably you would want to first load pre-trained weights.
model.load_weights(...)

# Freeze all layers except the last one.
for layer in model.layers[:-1]:
  layer.trainable = False

# Recompile and train (this will only update the weights of the last layer).
model.compile(...)
model.fit(...)
```

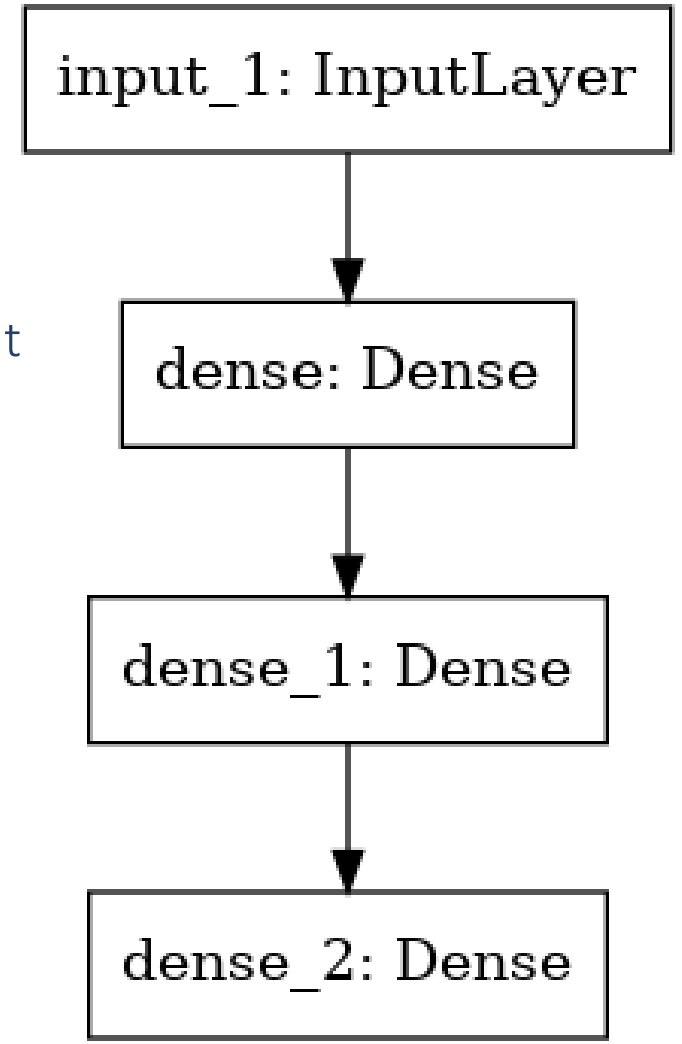
# Sequential model

- Kan niet gebruikt worden wanneer
  - Lagen gedeeld worden
  - Niet-lineaire topology
  - Het model of een laag heeft meerdere input/output tensors

# Visualiseren van een model

```
keras.utils.plot_model(model, "my_first_model.png")
```

- Optionele parameter
  - show_shape=True
    - Toont dimensies van in- en output



# Training

- Default training loop voor supervised learning
  - Kan aangepast worden door eigen implementaties

- Te volgens stappen:
  - Bouw je model
  - Compileer het en bepaal loss-functie, metrieken en learning rate optimizers
  - Fit het met de .fit functie
  - Evalueer het met de .evaluate functie

# Bewaren van een model

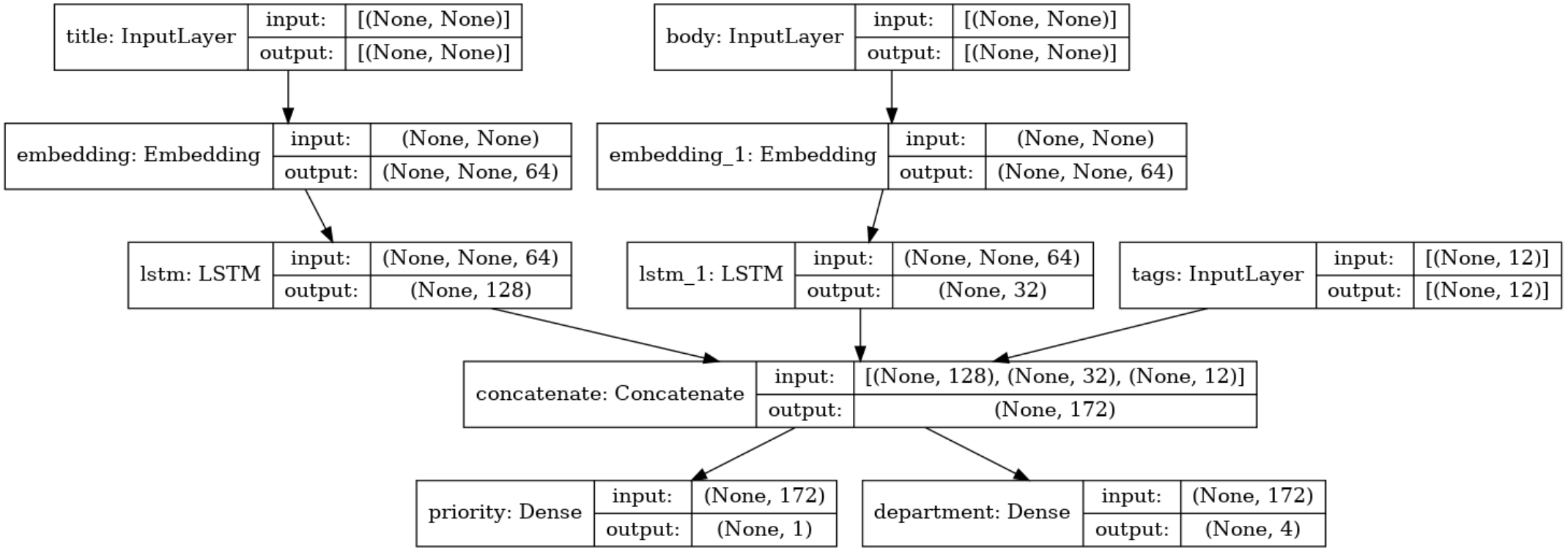
- Bij bewaren van een model worden de volgende gegevens opgeslagen
  - De architectuur van het model
  - De gewichten die getraind zijn
  - De gebruikte training configuratie
  - De gebruikt optimizer en zijn state

- De laatste twee worden gebruikt om verder te trainen indien nodig

```
model.save("path_to_my_model")
del model
# Recreate the exact same model purely from the file:
model = keras.models.load_model("path_to_my_model")
```

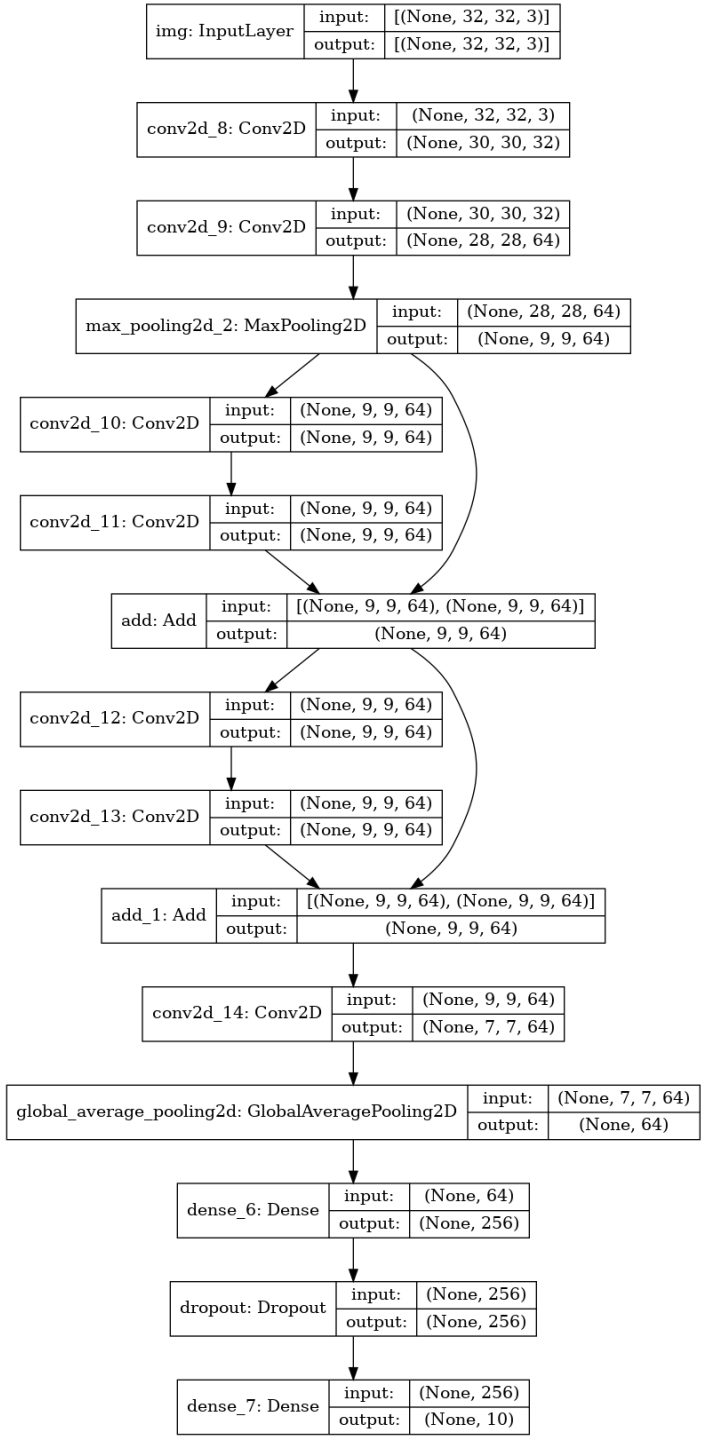
# Functional API

- Voor complexere zaken
  - Zoals meerdere in/outputs
  - Anders behandelen van sommige features



# Functional API

- Voor complexere zaken
  - Zoals meerdere in/outputs
  - Geen sequentiële uitvoering



# Layers

- Standaard lagen: Sparse, Dense (deze les)
- Convolutionele lagen: Conv1d, Conv2d, Conv3d, Conv2dTranspose
  - Computervisie
- Pooling layers: MaxPooling1d, MaxPooling2d, MaxPooling3d, AveragePooling, …
  - Computervisie
- Recurrente lagen: GRU, LSTM, ConvLSTM2d, ..
  - Geheugen in het network
- Andere: BatchNormalization, Dropout, Embedding, …

# Functional API vs Subclassing Model

- Functional API:
  - Higher level code, eenvoudiger, en veiliger
  - Kortere code
  - Validatie model bij het opbouwen van de graaf (dimensies komen overeen)
  - Visueler
  - Porteerbaar
- Subclassing model
  - Flexibeler
  - Ook modellen te bouwen die niet voorgesteld kunnen worden als een acyclische graaf (Tree-RNN)

# Training en evaluation

# Built-in function

- Model.compile()
- Model.fit()
  - Returned de geschiedenis van de loss values en metric values
- Model.evaluate()
- Model.predict()

# Model.compile()

- Standaard
  - Optimizers: SGD(), RMSprop(), Adam(), …
  - Loss-functions: MeanSquaredError(), KLDivergence(), …
  - Metrics: AUC(), Precision(), Recall(), …

- Custom versies zijn ook mogelijk door eigen functies/klassen te maken
  - Zie documentatie: https://www.tensorflow.org/guide/keras/train_and_evaluate#custom_metrics

# Validation holdout set

- Validation
  - Houd deel van trainingsdata apart om overfitting te detecteren
  - validation_split parameter van de fit-functie
    - Werkt enkel als er gewerkt wordt met numpy-data
    - Werk dan met validation_data om zelf een tuple (x,y) mee te geven
- Steeds de laatste samples worden overgehouden
  - Dus geen random samples, pas dus op indien je dataset niet random is

# Tf.data

- API met een set van functies voor inladen en preprocessing data
  - https://www.tensorflow.org/guide/data

- Werkt met tf.data.Dataset objecten
  - Kan meegegeven worden aan de functies voor fit, evaluatie en predict
  - Kan ook gebruikt worden als validation_data argument

# Unbalanced datasets

- Standaard wordt het gewicht van een sample bepaald door hoeveel keer het voorkomt in de dataset
- Class weights
  - Balance classes zonder resampling of zwaarder gewicht aan een bepaalde klasse
  - Als je bijvoorbeeld geen kanker over het hoofd wil zien
- Sample weight
  - Geef meer gewicht/belang aan bepaalde samples
  - Voor zeldzame klassen of outliers
  - Kan ook gebruikt worden om samples te negeren door het gewicht op 0 te zetten

# Callbacks

- ModelCheckpoint: Sla het model op
- EarlyStopping: Stop training wanneer validatie-metrieken niet meer verbeteren
- Tensorboard: Sla de logs af en toe op voor visualisatie in Tensorboard
- CSVlogger: Sla loss en metriek data op in een csv file
- ...
  - Zie: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/
  - Custom callbacks: https://www.tensorflow.org/guide/keras/train_and_evaluate#writing_your_own_callback