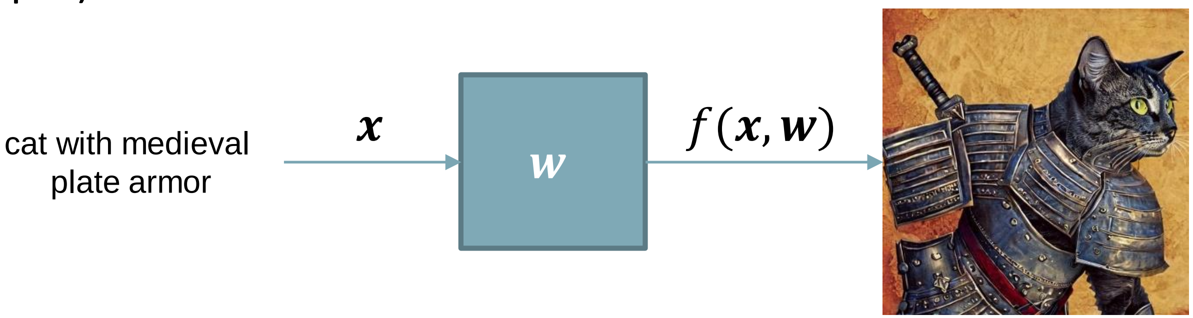
Neural networks

Neural networks

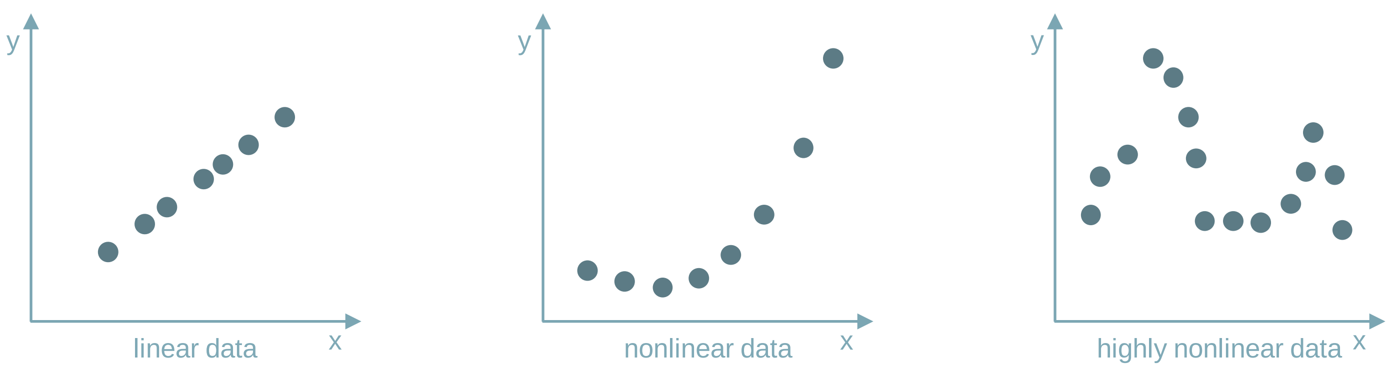
Neural networks

- Are very powerful function approximators (e.g. with images or sentences as input/output)



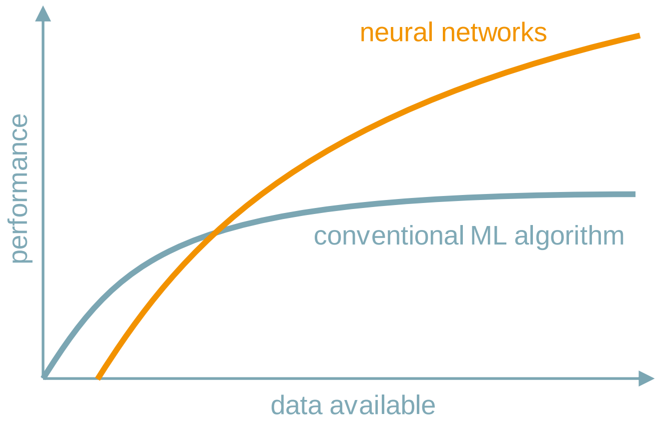
https://stablediffusionweb.com/#demo

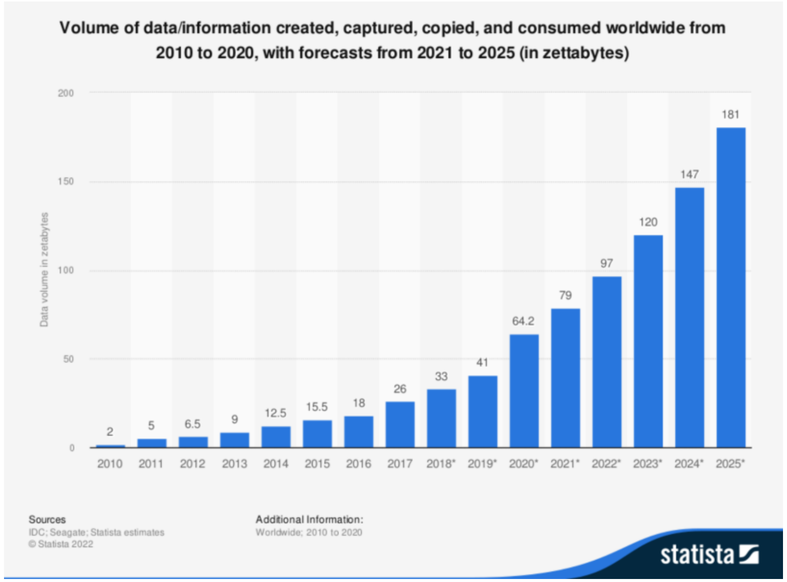
- Other interpretation: Can approximate highly nonlinear data



Neural networks

- Can deal with large amounts of data better than other, more conventional machine learning algorithms





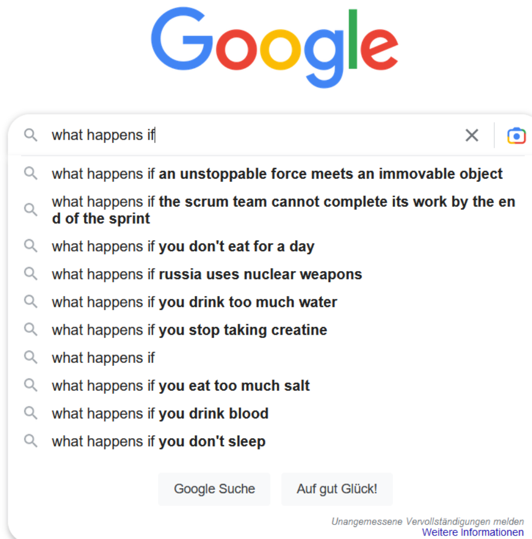
Neural networks

- Are applicable for a wide variety of problems

panoptic segmentation

https://149695847.v2.pressablecdn.com/wp-content/uploads/2021/02/panoptic-output-1024x683.png

autocompletion



https://www.google.com/

deep fakes

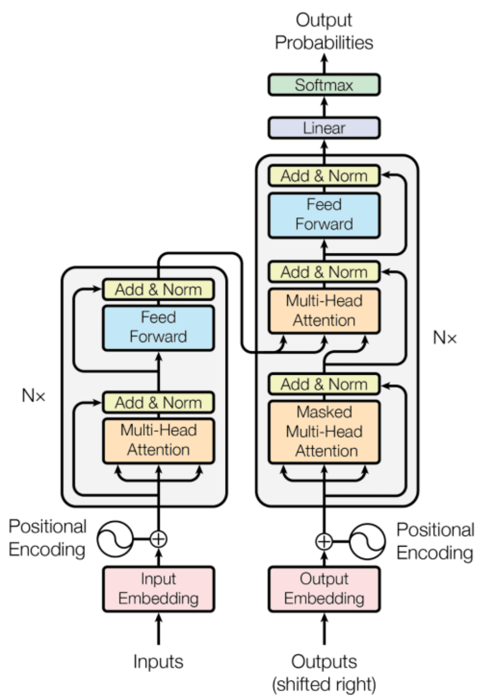
https://www.youtube.com/watch?v=TgCsJfLypZYhttps://www.youtube.com/shorts/LDpSsutr5wo

- Are all trained through gradient descent (**deep learning**)

Neural networks

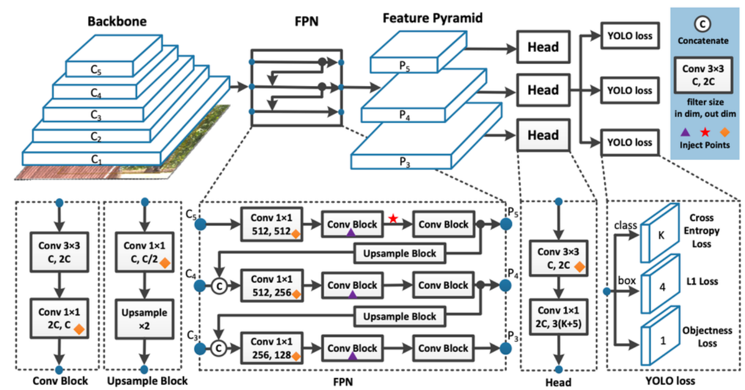
Note that there exist myriads of different architectures, we will only consider the most fundamental one, named **multilayer perceptron**

transformer



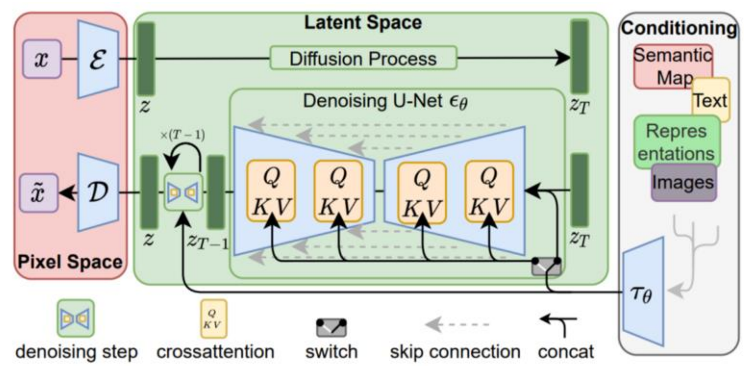
https://machinelearningmastery.com/wp-content/uploads/2021/08/attention_research_1.png

YOLO



https://blog.roboflow.com/yolov7-breakdown/

stable diffusion



https://www.louisbouchard.ai/latent-diffusion-models/

Neural networks

Neural networks are inspired by nature

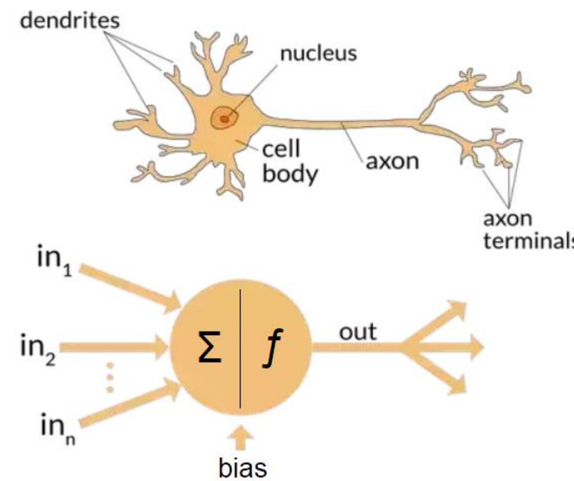
- Neurons receive **input signals** x_i with **weight** w_i from other neurons through dendrites
- Those signals are accumulated within the cell body (with **bias** b)

  $$\sum_i w_i x_i + b$$

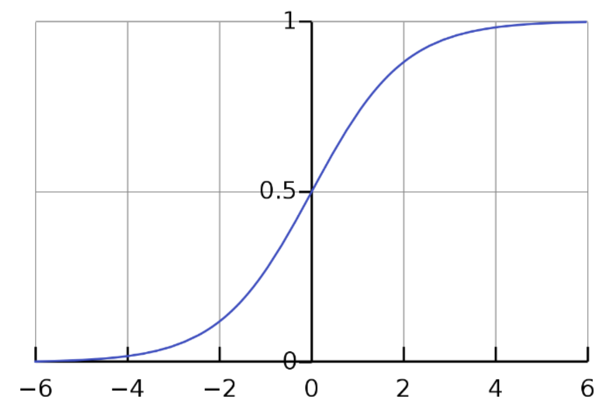
- Once the sum exceeds a certain threshold, the neuron emits a signal through its axon (fires)

  $$f\left(\sum_i w_i x_i + b\right)$$

- The nonlinear function f (**activation function**) can e.g. be a sigmoid function
- An **artificial neuron** calculating $f\left(\sum_i w_i x_i + b\right)$ constitutes the basic building block of artificial neural networks



https://miro.medium.com/max/610/1*SJPacPhP4KDEB1AdhOFy_Q.png

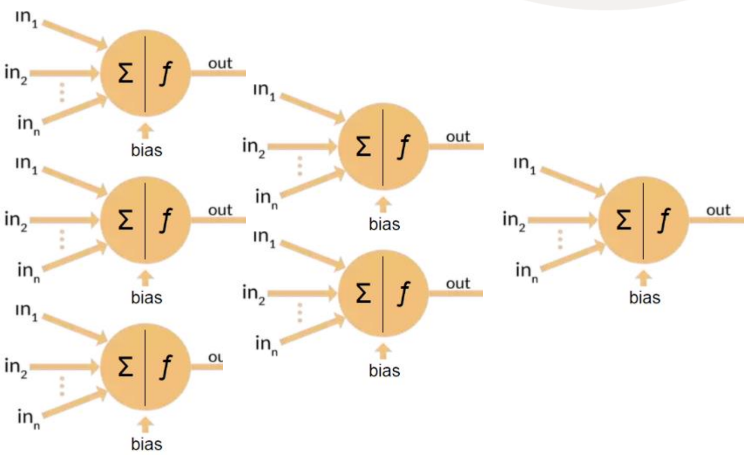


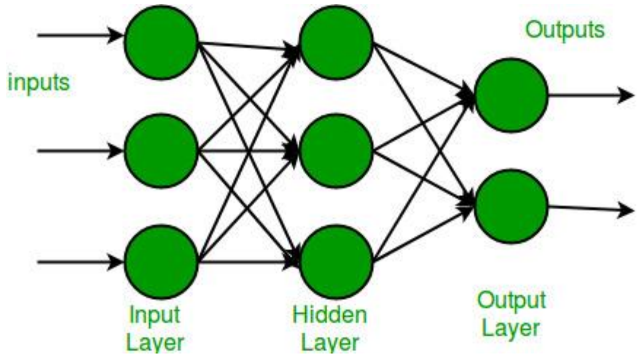
https://en.wikipedia.org/wiki/Sigmoid_function

Neural networks

Artificial neural networks (ANN)

- are the combination of multiple artificial neurons
- If the artificial neurons are aligned in a layered structure, it is called a **multilayer perceptron** (**MLP**)
- A single layer of this type of artificial neurons is called a **fully connected / linear / dense layer**
- The first layer is called **input layer** (dummy), the last layer is called **output layer**. All intermediate layers are called **hidden layers**



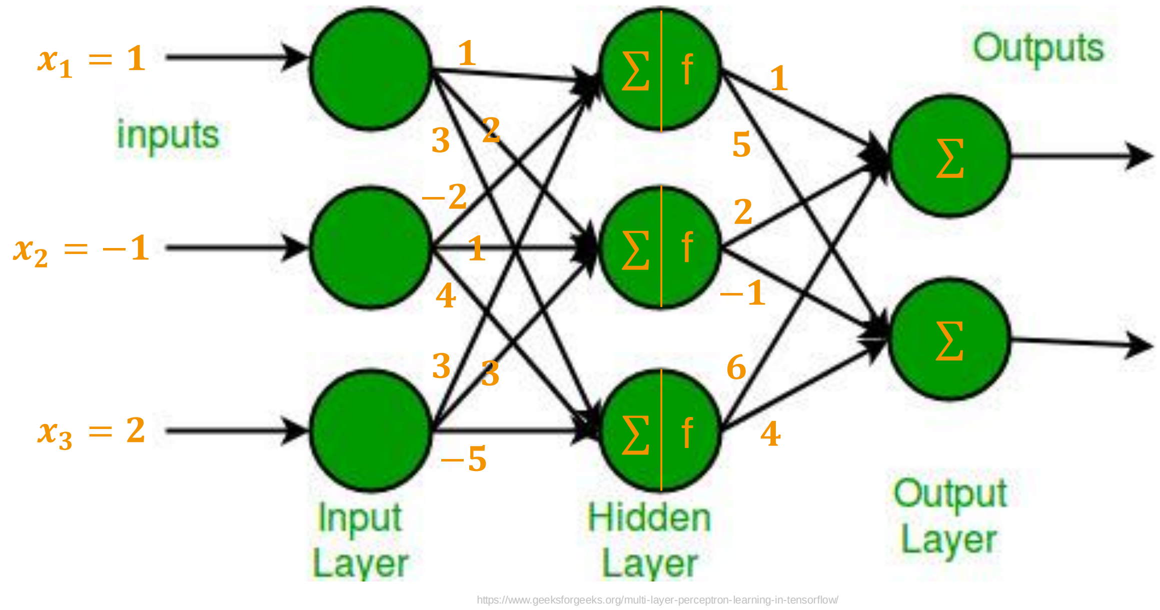


https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/

Neural networks

Task: Calculate the outputs of the displayed neural network

- all biases = 0
- ReLU activation function $f(x) = \max(0, x)$ in hidden layer (no other activation functions)
- weights w_i as shown



Task: Why is there no activation function in the output layer?

Neural networks

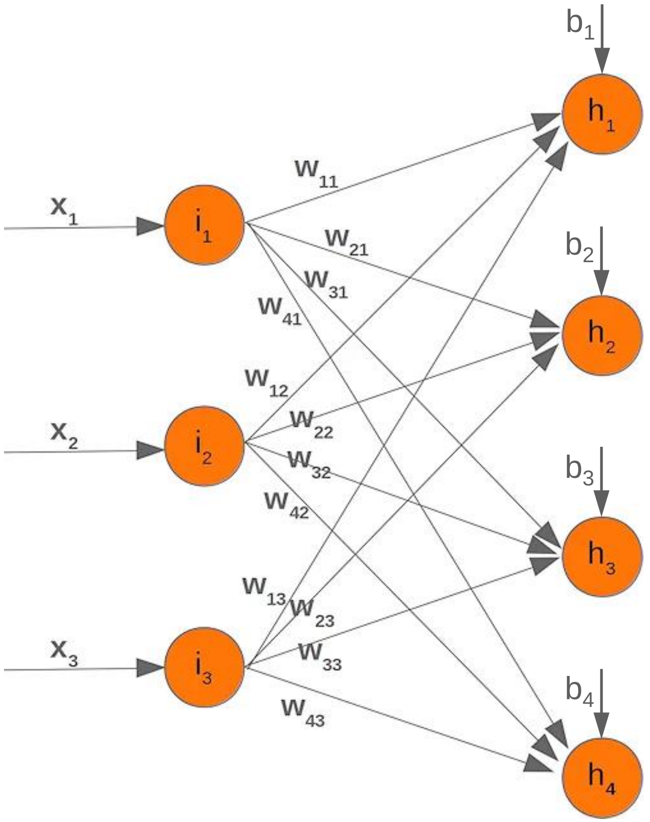
Matrix representation

- The computation of a dense layer can
  be simplified using matrix-vector multiplication
  $$f(W\boldsymbol{x} + \boldsymbol{b})$$
- GPUs are extremely fast in calculating $W\boldsymbol{x} + \boldsymbol{b}$

Example: Matrix-vector representation

- The calculation $f(\sum_i w_i x_i + b)$ of every neuron within a dense layer can be represented as (for the example on the right)

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = f\left(\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \\ w_{14} & w_{24} & w_{34} & w_{44} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}\right)$$



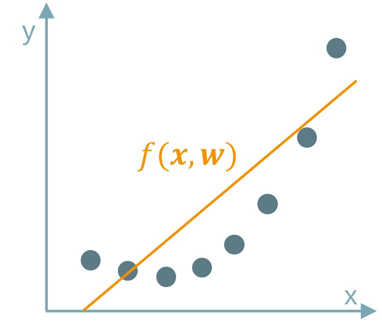
https://python-course.eu/machine-learning/neural-networks-structure-weights-and-matrices.php

Neural networks

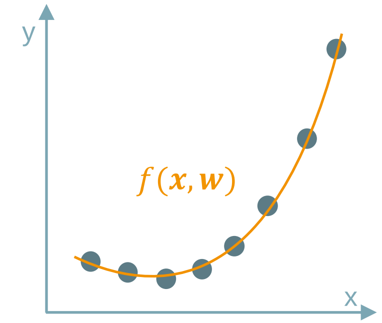
Activation function

- The nonlinear activation function f is an central part of any MLP and must not be omitted
- Without nonlinear activation function, any number of dense layers acts like one single dense layer, e.g. it is for a MLP with two dense layers

$$
\begin{aligned}
\boldsymbol{x}_1 &= W_1\boldsymbol{x} + \boldsymbol{b}_1 \\
\hat{\boldsymbol{y}} &= W_2\boldsymbol{x}_1 + \boldsymbol{b}_2 \\
&= W_2(W_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2 \\
&= (W_2W_1)\boldsymbol{x} + (\boldsymbol{b}_1 + \boldsymbol{b}_2) \\
&= W\boldsymbol{x} + \boldsymbol{b}
\end{aligned}
$$



without activation function



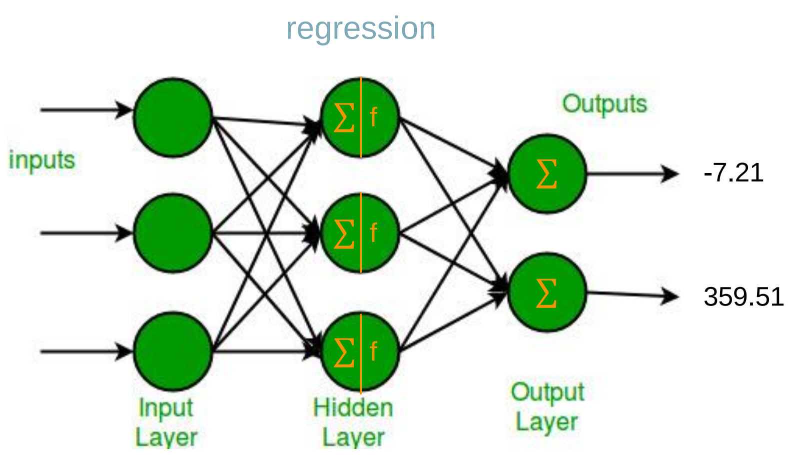
with activation function

- Without a nonlinear activation function the neural network is only a linear function approximator
- There exist many different activation functions, the most common ones are sigmoid, tanh and ReLU

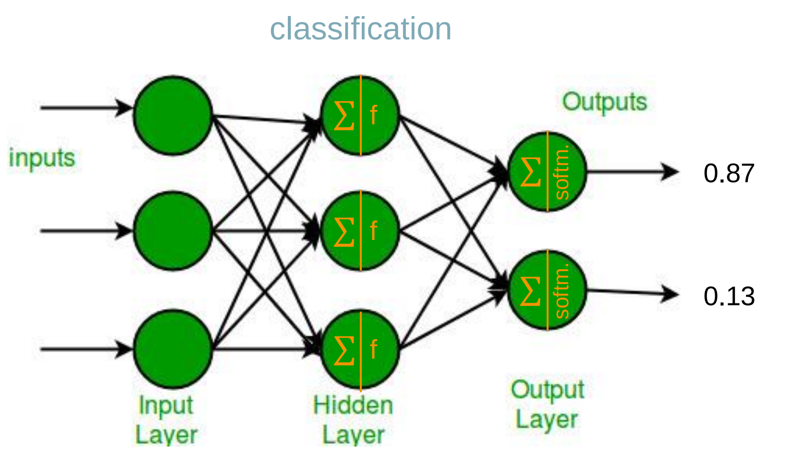
Neural networks

Regression vs. classification

- A MLP for regression tasks does not have an activation function in its output layer. Every output represents one dimension of the regression task.
  → e.g. Q-function



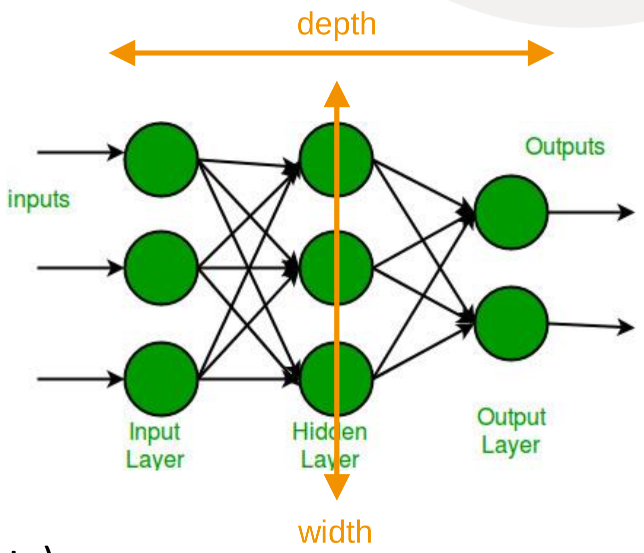
- A MLP for classification tasks has a softmax activation function in its output layer, limiting every output to the $[0, 1]$ range and enforcing $\sum outputs = 1$. This allows the outputs to be interpreted as probabilities.
  → e.g. policy for discrete actions



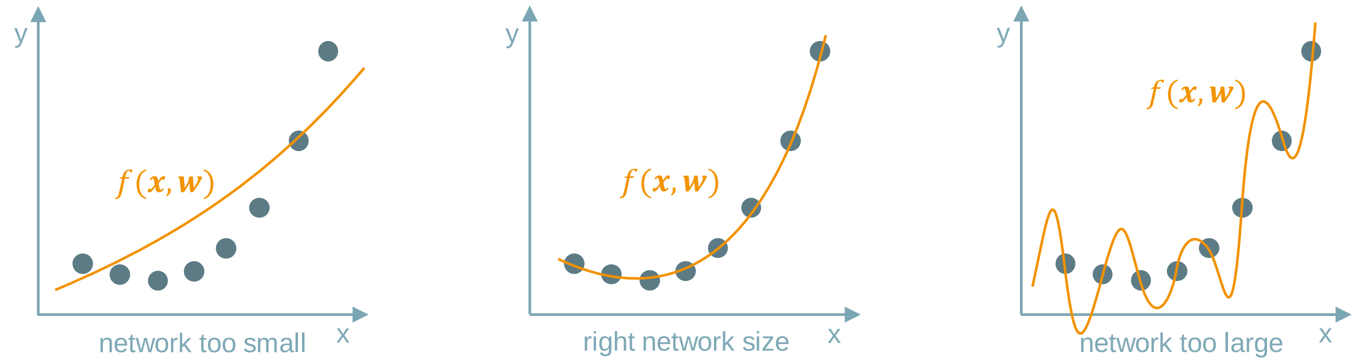
Neural networks

Interpretation

- The **depth** of an ANN (number of layers) represents the number of subsequent calculations
- The **width** of an ANN (number of neurons per layer) represents the number of parallel calculations per layer
- The deeper/wider (larger) a network is, the better it can approximate highly nonlinear data (more weights)
- BUT: If a network is too large, overfitting can occur (described later)



https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/



Neural networks

Task: Draw the structure of a neural network with the following specifications

- Two input dimensions
- One output dimension
- Regression task
- First hidden layer with three neurons
- Second hidden layer with four neurons

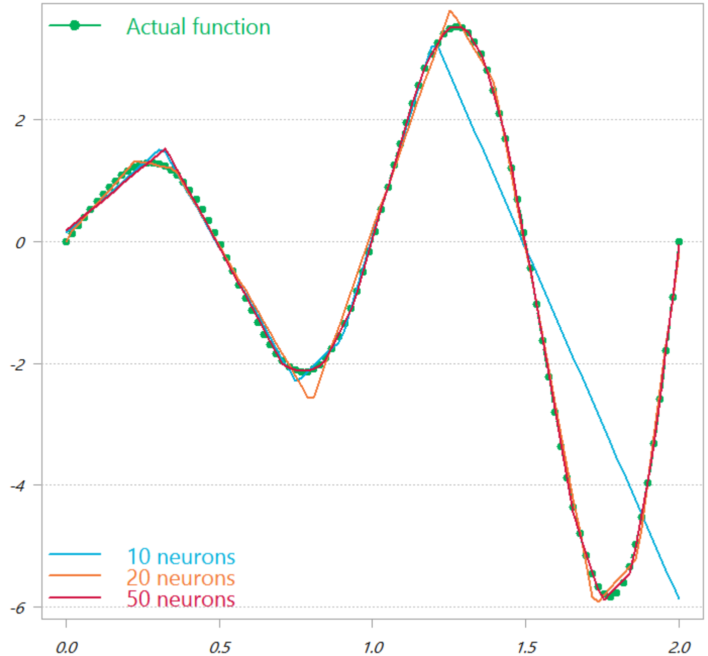
Neural networks

Universal approximation theorem

- Any function can be approximated with arbitrary precision (though not exactly) with a single hidden layer with finite width

→ In theory sufficient to use only one (very wide) layer

→ In practice tradeoff between width and depth



https://eranraviv.com/wp-content/uploads/2021/03/Universal-approximation-theorem.png

Neural networks

Different layer types

- This lecture only deals with very simple neural networks (MLPs)
- There are many more layers which can be used to build more complex / highly specialized neural networks (e.g. convolution layers for image tasks) → deep learning lecture

TORCH.NN

These are the basic building blocks for graphs:

torch.nn

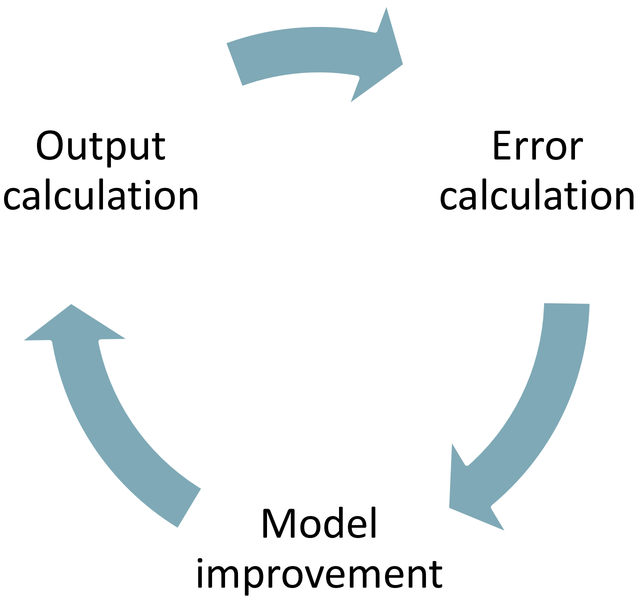
- Containers
- Convolution Layers
- Pooling layers
- Padding Layers
- Non-linear Activations (weighted sum, nonlinearity)
- Non-linear Activations (other)
- Normalization Layers
- Recurrent Layers
- Transformer Layers
- Linear Layers
- Dropout Layers
- Sparse Layers
- Distance Functions
- Loss Functions
- Vision Layers
- Shuffle Layers
- DataParallel Layers (multi-GPU, distributed)
- Utilities
- Quantized Functions
- Lazy Modules Initialization

https://pytorch.org/docs/stable/nn.html

Neural networks

Neural network training

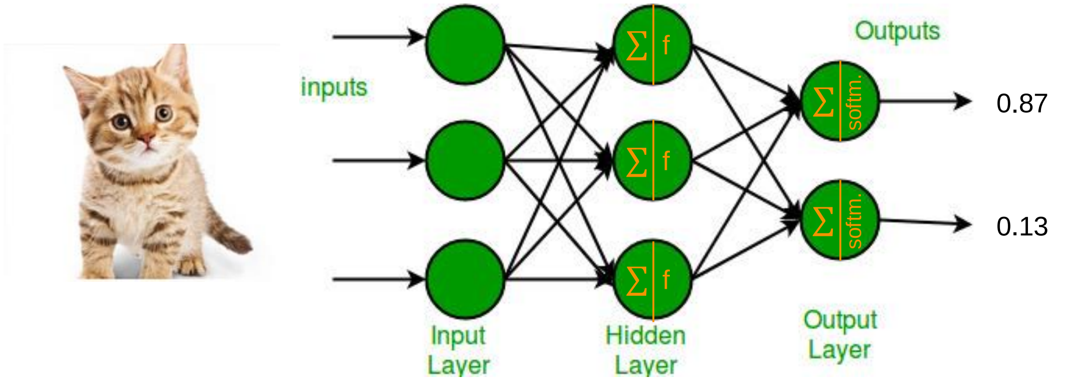
- Training a neural network is a tripartite process which is repeated multiple times
  - Output calculation
  - Error calculation
  - Model improvement



Neural networks

Output calculation

- Calculate the output of a neural network for a given input
- Was already done manually before



p(cat) = 0.87

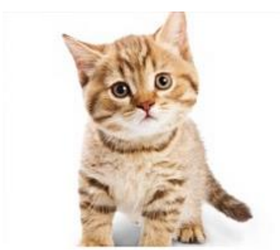
p(dog) = 0.13

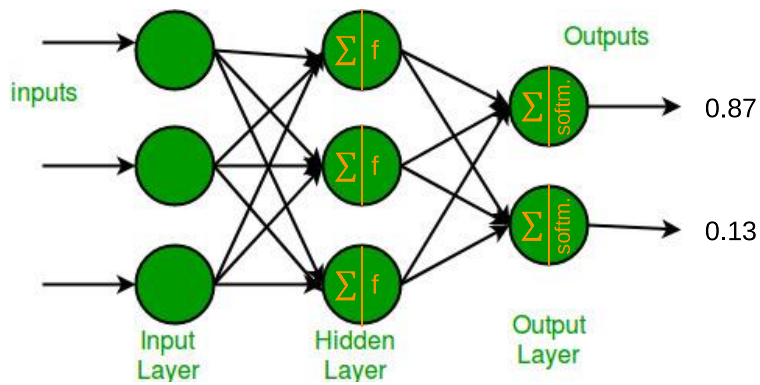
Neural networks

Error calculation

- Calculate the error (**loss**) L between output $\hat{\boldsymbol{y}}$ of the neural network (**prediction**) and the desired output $\boldsymbol{y}$ (**label**)
- A proper loss function is 0 for $\hat{\boldsymbol{y}} = \boldsymbol{y}$ and > 0, the larger the difference between $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$ is

$$L = (0.87 - 1)^2 + (0.13 - 0)^2 = 0.0338$$





class	$\hat{y}$	y
cat	0.87	1.00
dog	0.13	0.00

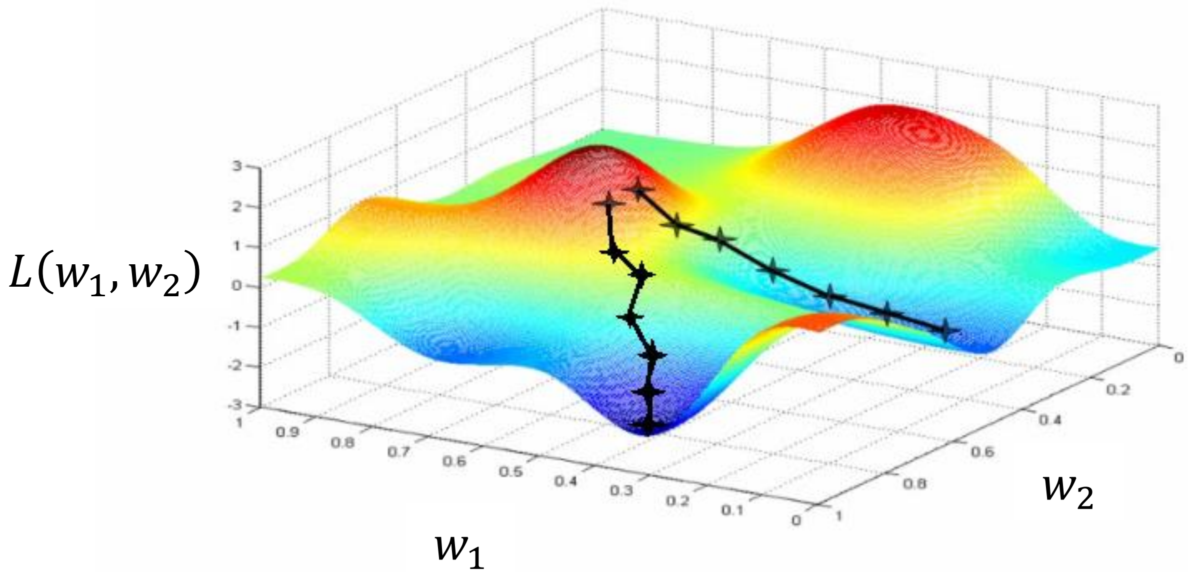
Neural networks

Model improvement

- Modify the network weights $\mathbf{w}$ (= weights W_k and biases $\boldsymbol{b}_k$ of all dense layers) to reduce the error → gradient descent

$$\boldsymbol{w}_k = \boldsymbol{w}_{\mathrm{k-1}} - \eta \cdot \nabla_{\mathbf{w}} L$$

- Problem: How to calculate the gradient
  → backpropagation



https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/

Neural networks

Backpropagation

- is the name of the method to calculate the gradient $\nabla_{\mathbf{w}} L$ for neural networks
- is a fancy name for the chain rule of differentiation you know from high school

Example: Derivative of

$$f(x) = \log(\sin(x^2)) = a\Big(b\big(c(x)\big)\Big)$$

with

$$a(b) = \log(b), \; b(c) = \sin(c), \; c(x) = x^2$$

$$\begin{aligned} \frac{\partial f}{\partial x} &= \frac{\partial a}{\partial b} \frac{\partial b}{\partial c} \frac{\partial c}{\partial x} \\ &= \frac{1}{b} \cos(c)\, 2x \\ &= \frac{1}{\sin(x^2)} \cos(x^2)\, 2x \end{aligned}$$

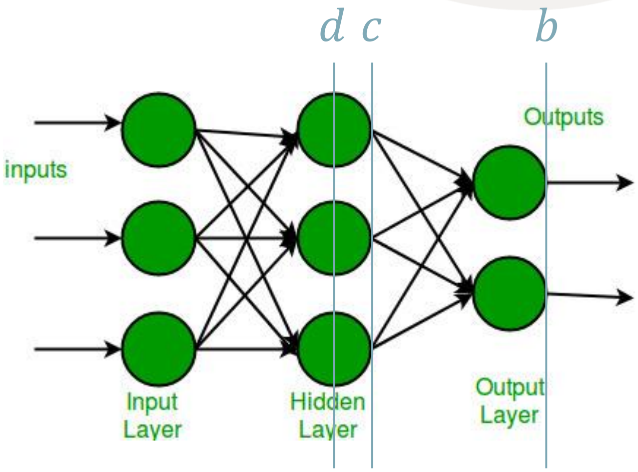
Neural networks

Example: Derivative of

$$L = a\left(\boldsymbol{b}\left(\boldsymbol{c}\left(\boldsymbol{d}(W_1)\right)\right)\right)$$

with

$$a(\boldsymbol{b}) = \sum_i \left((f(\boldsymbol{b})_i - \boldsymbol{y}_i)^T \cdot (f(\boldsymbol{b})_i - \boldsymbol{y}_i)\right)$$
$$\boldsymbol{b}(\boldsymbol{c}) = W_2\boldsymbol{c} + \boldsymbol{b}_2$$
$$\boldsymbol{c}(\boldsymbol{d}) = \mathrm{ReLU}(\boldsymbol{d})$$
$$\boldsymbol{d}(W_1) = W_1\boldsymbol{x} + \boldsymbol{b}_1$$

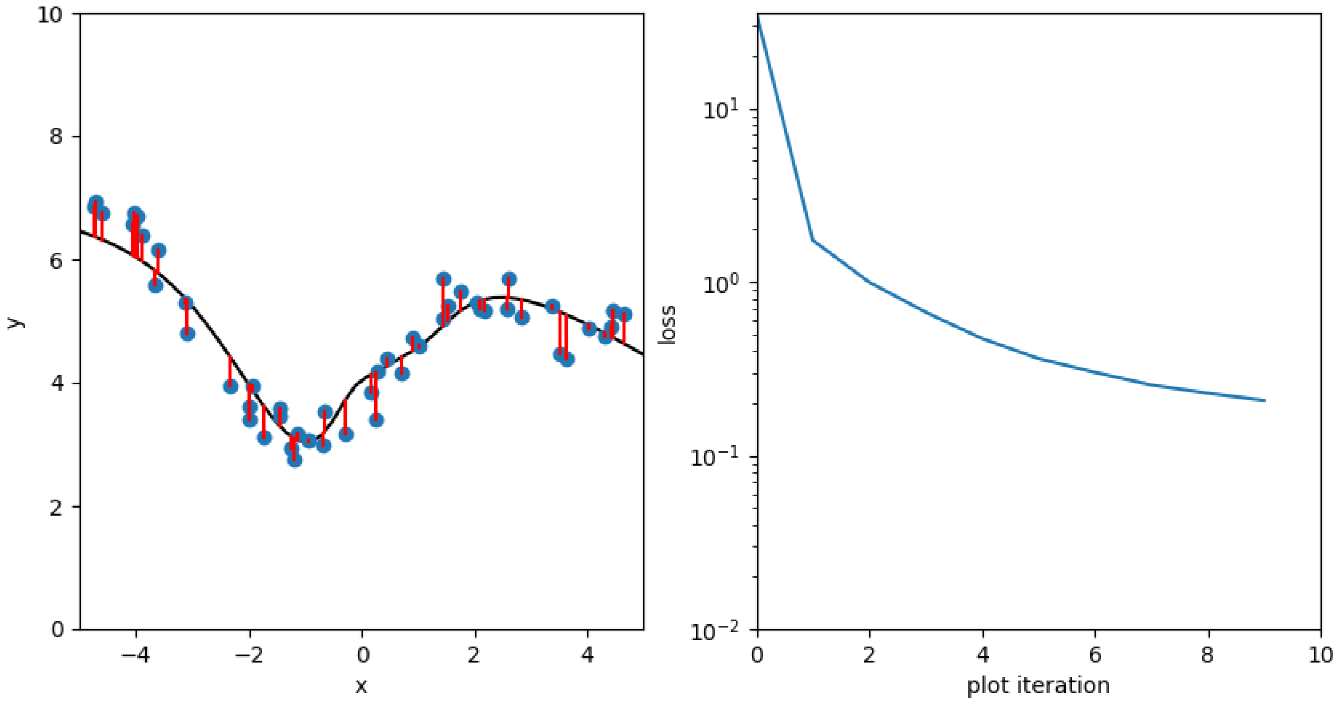


$$\frac{\partial L}{\partial W_1} = \frac{\partial a}{\partial \boldsymbol{b}} \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{c}} \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{d}} \frac{\partial \boldsymbol{d}}{\partial W_1}$$

- Calculating $\frac{\partial L}{\partial W_1}$ is not easy, but doable (→ deep learning lecture)
- Also have to calculate $\frac{\partial L}{\partial W_2}$, $\frac{\partial L}{\partial \boldsymbol{b}_1}$ and $\frac{\partial L}{\partial \boldsymbol{b}_2}$ to obtain the gradient $\nabla_{\mathbf{w}} L$

Neural networks

Example: Code



Neural networks

PyTorch / Tensorflow / MXNet / …

- are software frameworks specifically designed to calculate the gradient $\nabla_{\mathbf{w}} L$ and gradient descent
- no need to worry, all the complex gradient calculations will be done by software

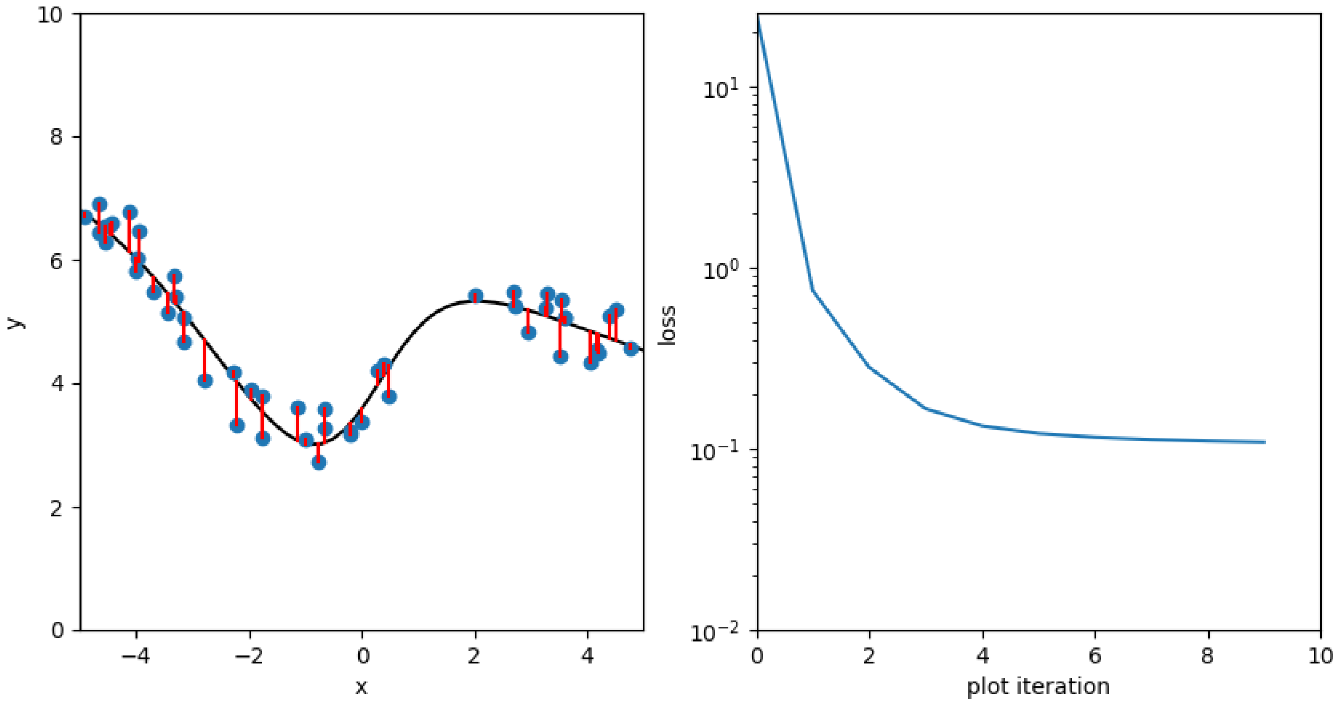
Example: Pytorch (simplified)

```
for i in range (n_iter):

    # making predictions with forward pass
    Y_pred = forward(X)

    # calculating the loss between original and predicted data points
    loss = criterion(Y_pred, Y)

    # backward pass for computing the gradients of the loss w.r.t to learnable parameters
    loss.backward()

    # update the parameters based on the gradient
    w.data = w.data - step_size * w.grad.data
```

https://machinelearningmastery.com/implementing-gradient-descent-in-pytorch/

Neural networks

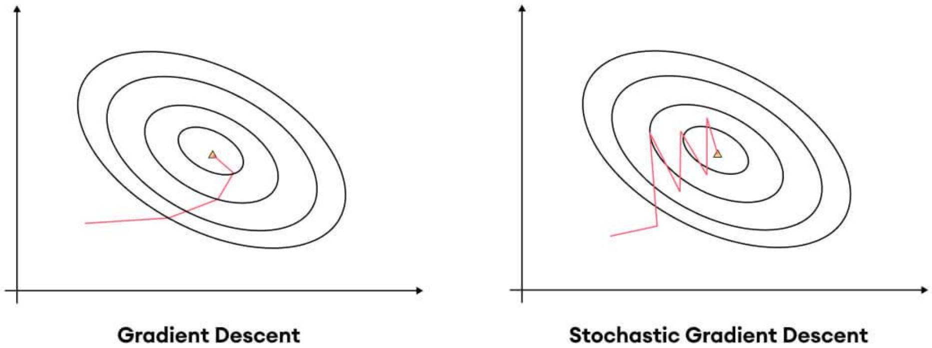
Example: Code



Neural networks

Stochastic gradient descent (SGD)

- Problem: If a dataset contains too many samples, calculating the gradient $\nabla_{\mathbf{w}} L$ based on all samples at once will require a lot of time
- Solution: Calculate the gradient only for a subset of all samples (stochastic gradient descent)
- The samples of the subset should change for each gradient descent step
- The number of samples within the subset is called **batch size**
- An **epoch** is defined as $\frac{\text{dataset size}}{\text{batch size}}$



effect of SGD:
noisy approximation of
the true gradient (which
is based on all samples)

https://uploads-ssl.webflow.com/614c82ed388d53640613982e/635b9f1c245a9873d0c77353_6320786c39548e9df8f5b0a6_traditional-and-stochastic-gradient-descent-1.jpeg

Neural networks

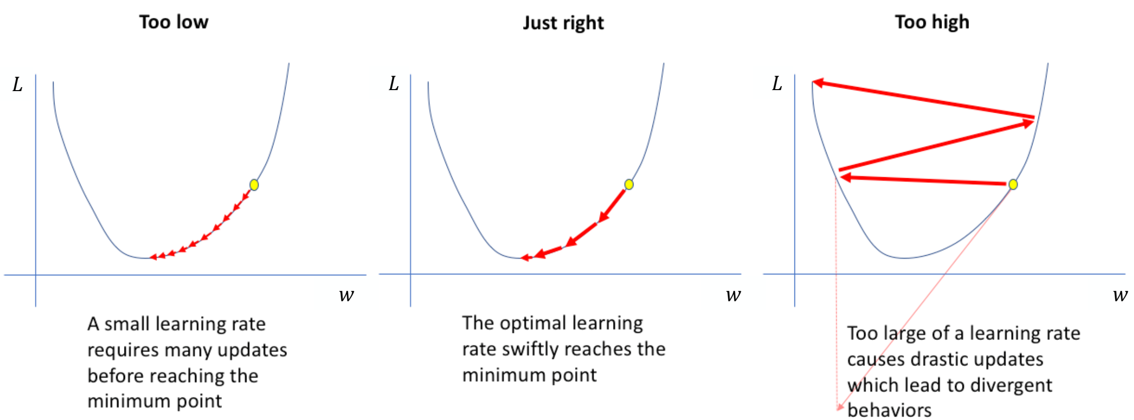
Problem with all gradient descent approaches: Finding the correct learning rate η in

$$\boldsymbol{w}_k = \boldsymbol{w}_{\mathrm{k-1}} - \eta \cdot \nabla_{\mathbf{w}} L$$

- If the learning rate is too small, training takes long
- If the learning rate is too large, the updates diverge

Solution: Adapt learning rate during training (next slides) through

- Learning rate scheduler
- Optimizer



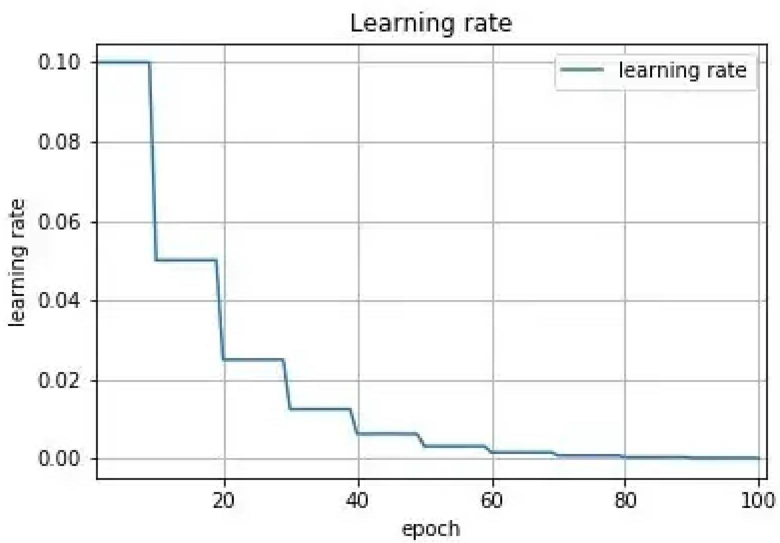
https://www.jeremyjordan.me/nn-learning-rate/

Neural networks

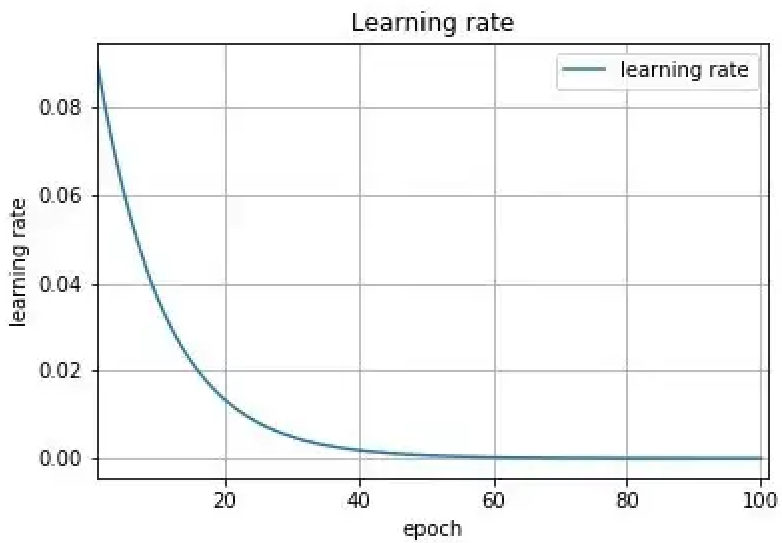
Learning rate scheduler

- decrease the learning rate during training according to a predefined schedule

Step decay scheduler



Exponential decay scheduler



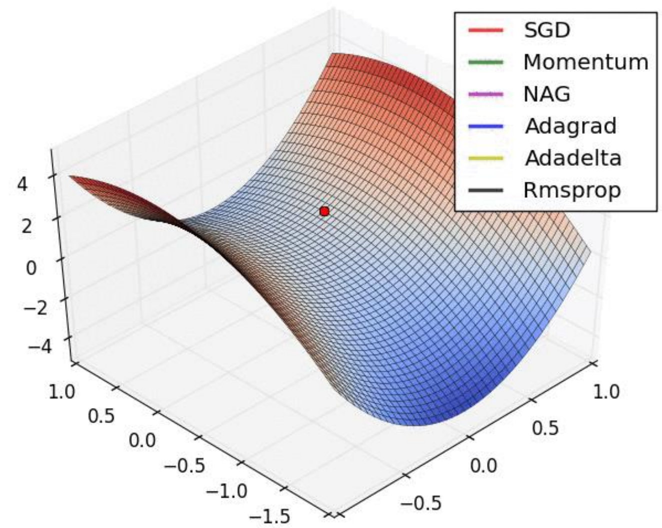
https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

Neural networks

Ostbayerische Technische Hochschule Amberg-Weiden

Optimizer

- Update the weights not only based on the gradient but also
  - based on the last update steps
  - dynamically along every dimensions
- There exist many different optimizer which often accelerate training but may fail for specific ANNs



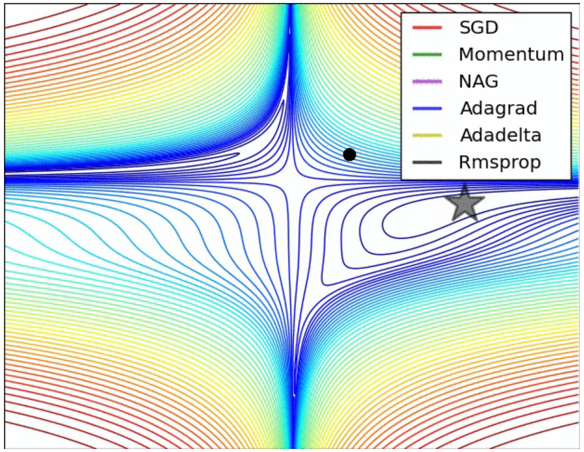
Example

- Adagrad

$$w_k = w_{k-1} - \frac{\eta}{\sqrt{\epsilon + diag(G_k)}} \cdot \nabla_{\mathbf{w}} L$$

compared to SGD

$$w_k = w_{k-1} - \eta \cdot \nabla_{\mathbf{w}} L$$

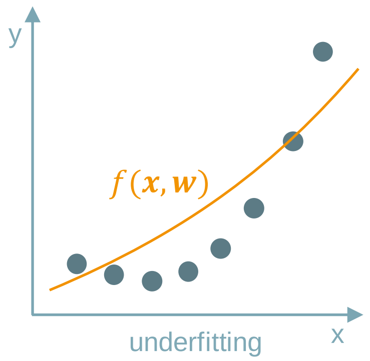


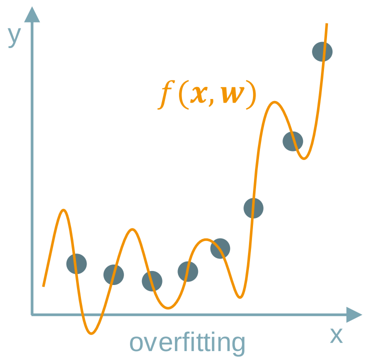
https://ruder.io/optimizing-gradient-descent/

Neural networks

Overfitting/underfitting

- If a neural network is too deep/wide, overfitting occurs
- If a neural network is not deep/wide enough, underfitting occurs
- Both over- and underfitting are undesired properties and can be countered e.g. through proper dimensioning of the ANN
  (more techniques in the deep learning lecture)

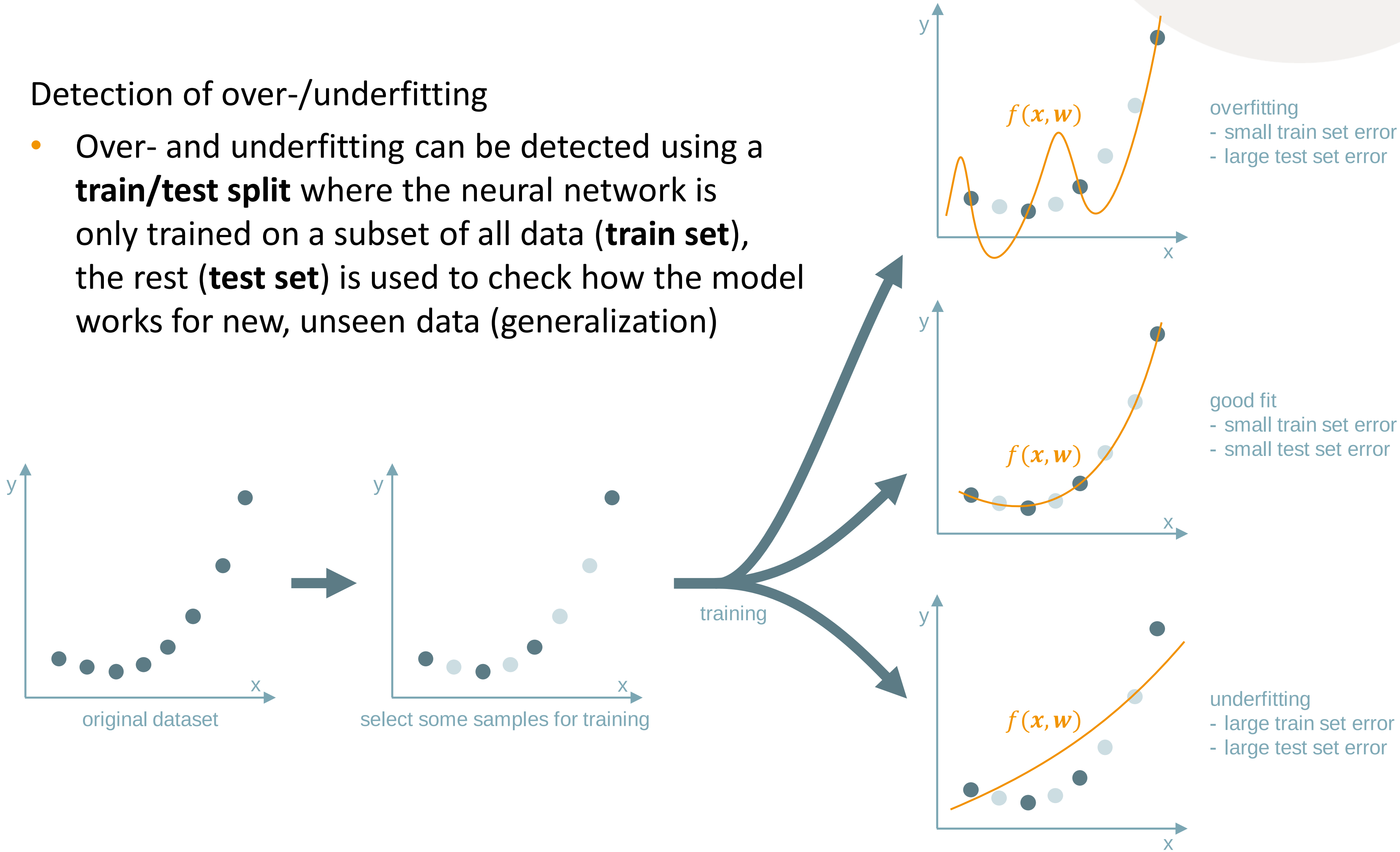




Neural networks

Detection of over-/underfitting

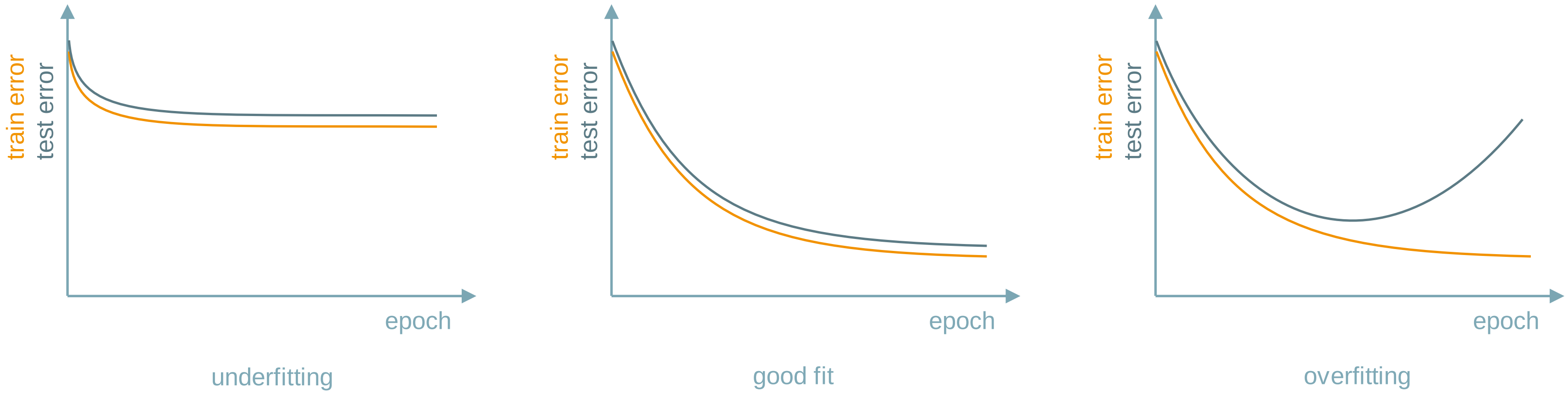
- Over- and underfitting can be detected using a **train/test split** where the neural network is only trained on a subset of all data (**train set**), the rest (**test set**) is used to check how the model works for new, unseen data (generalization)



Neural networks

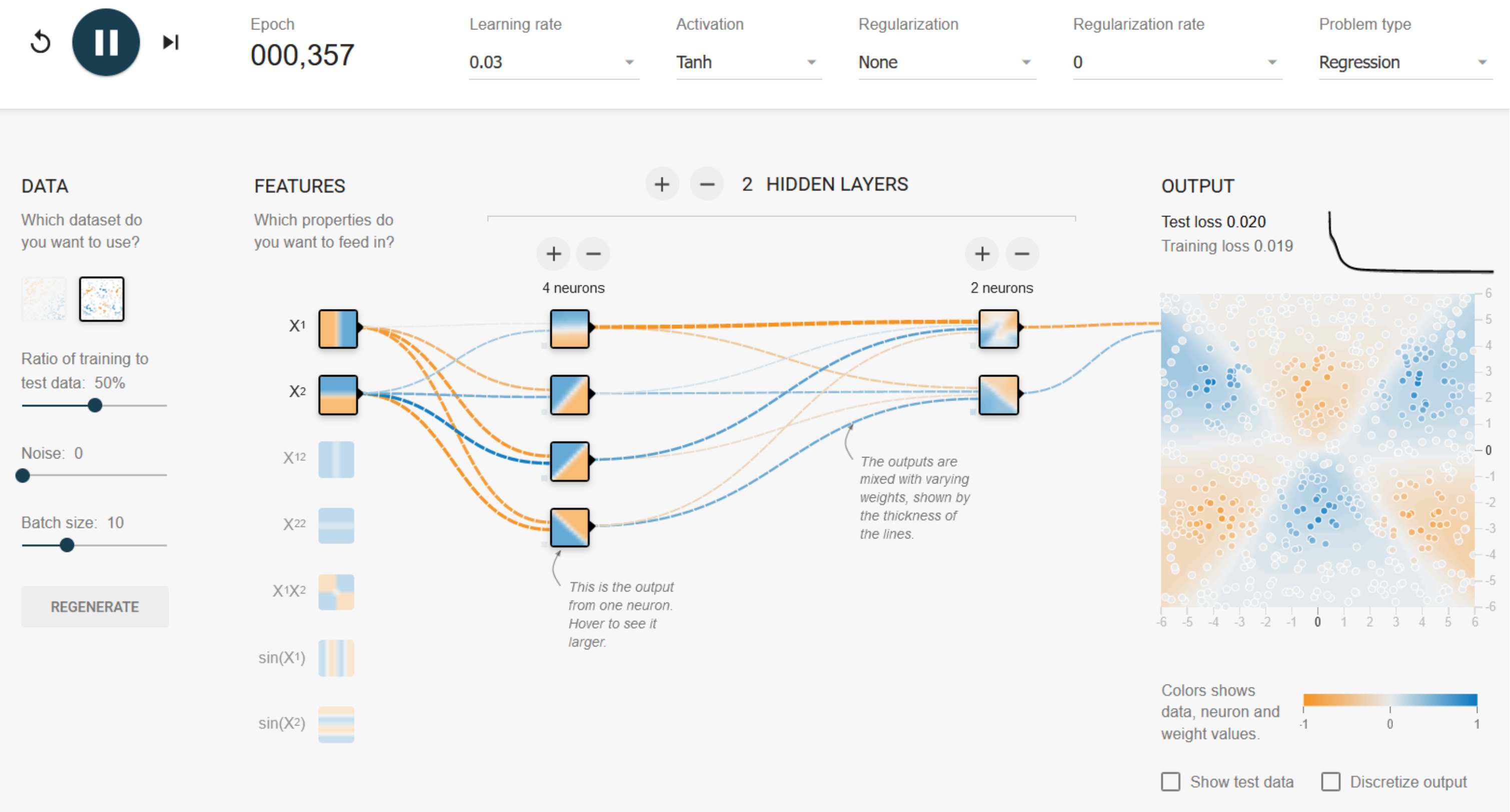
Learning curve

- A learning curve visualizes the train/test set error over gradient descent steps / epochs
- It is used to detect over-/underfitting through "manual inspection"



Neural networks

Example: Tensorflow playground



https://playground.tensorflow.org

Kahoot!

Neural networks

Homework:

Learn the basics of PyTorch: https://pytorch.org/tutorials/beginner/basics/intro.html

- at least read all linked guides on the page
- ideally install PyTorch locally on your PC and run the code

https://cdn2.psychologytoday.com/assets/styles/manual_crop_1_91_1_1528x800/public/field_blog_entry_teaser_image/2018-11/depositphotos_51277329_s-2015.jpg?itok=guo89dbR