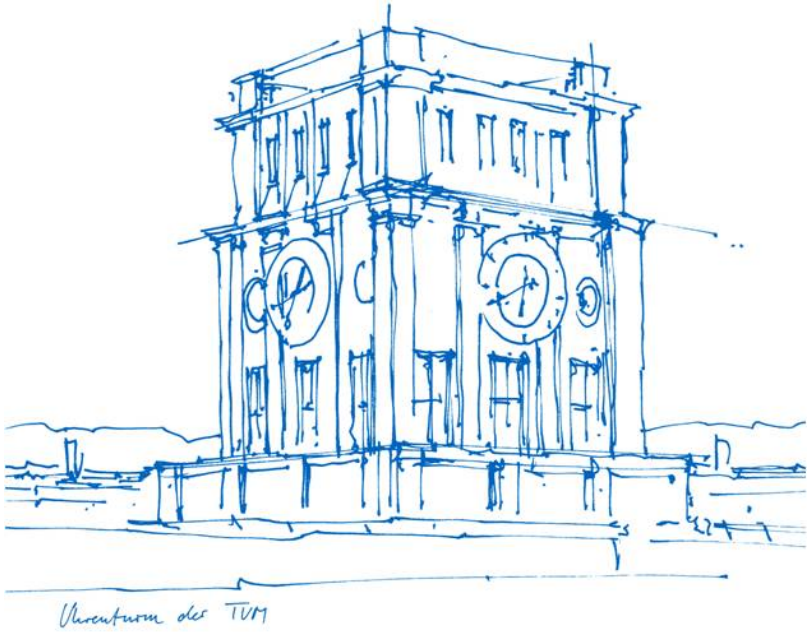
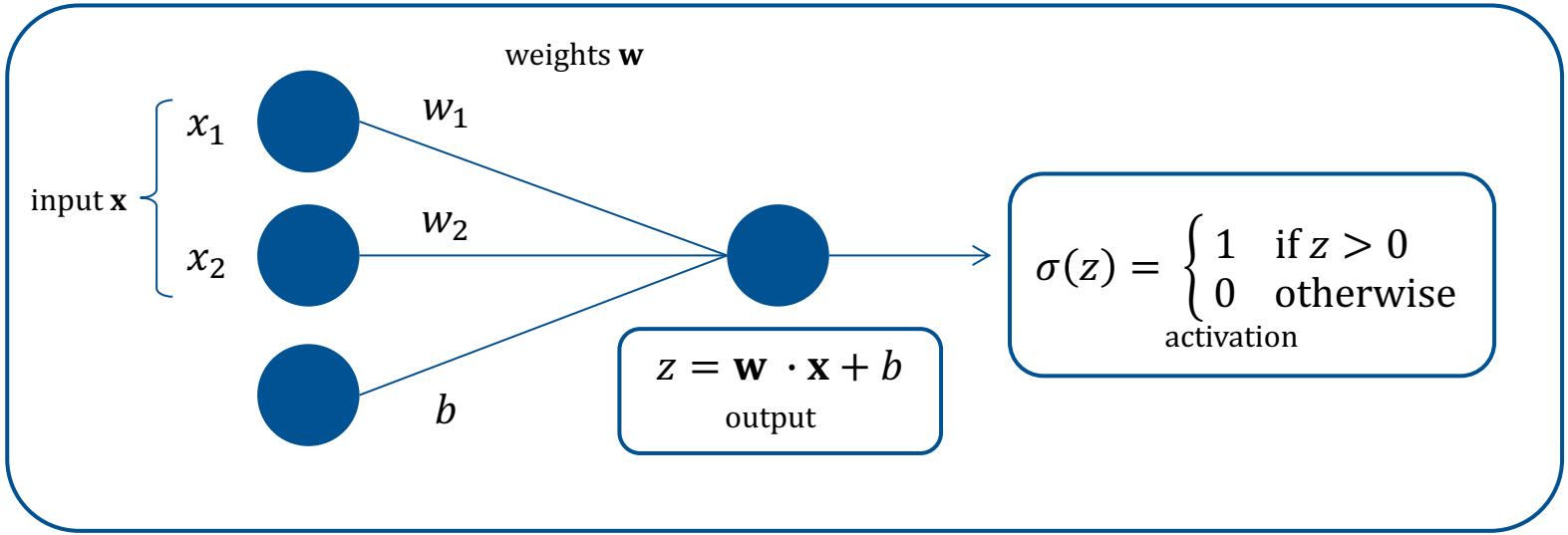
Neural Networks and Deep Learning



Denninger Maximilian

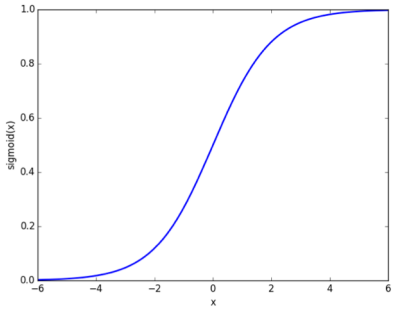
Perceptron

A binary classification algorithm invented in 1957 by F. Rosenblatt.



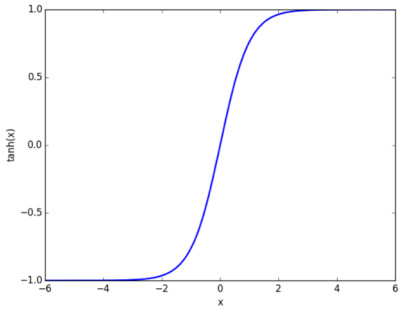
Activation functions

There are different common activation functions available.



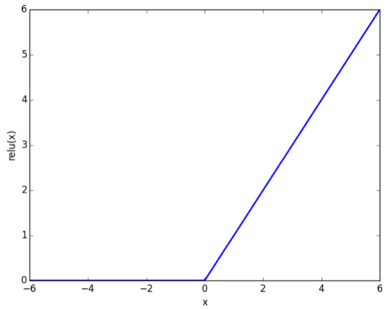
sigmoid

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



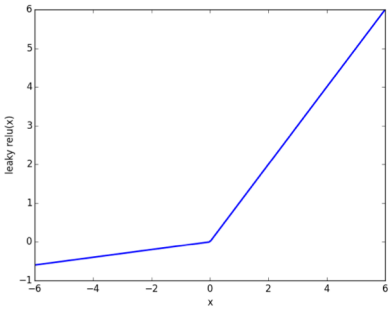
tanh

$$\sigma(x) = \tanh(x)$$



relu

$$\sigma(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

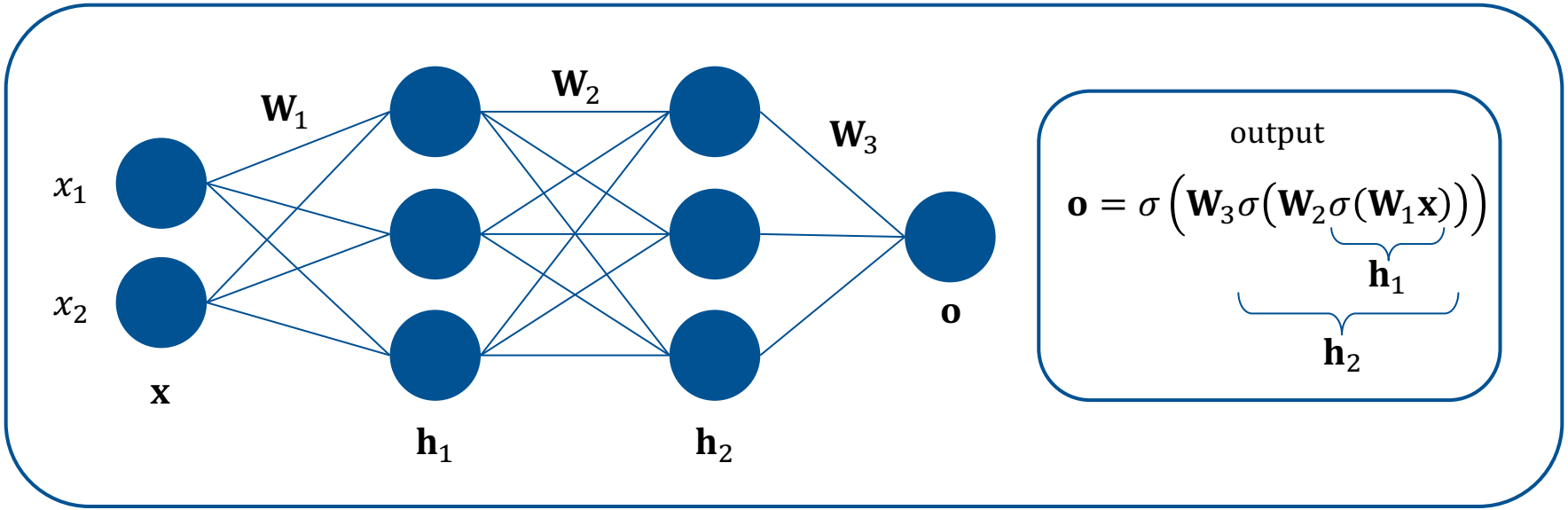


leaky relu

$$\sigma(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{else} \end{cases}$$

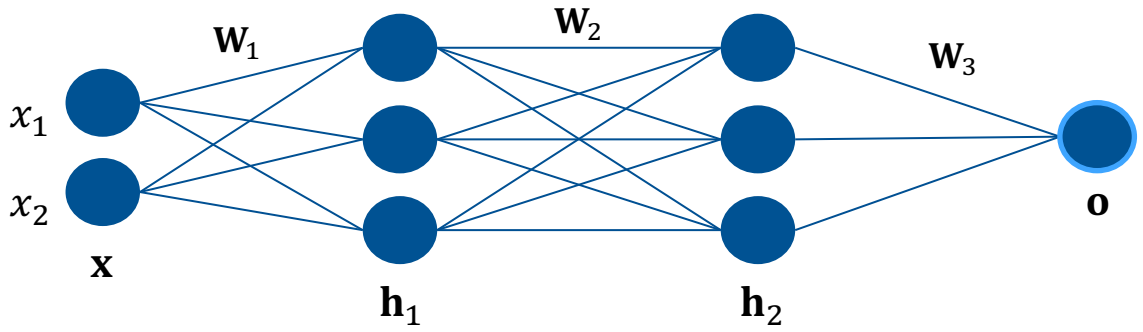
Neuronal Network

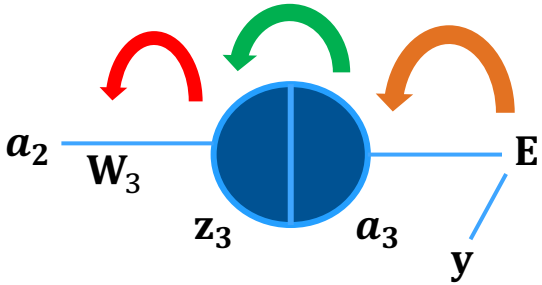
A simple neuronal network with two hidden layers, an input layer and an output layer.



Nice visualization of the activations: http://scs.ryerson.ca/~aharley/vis/fc/

Training with Backpropagation





Derivative of $\mathbf{E}$ in respect to $\mathbf{W}$:

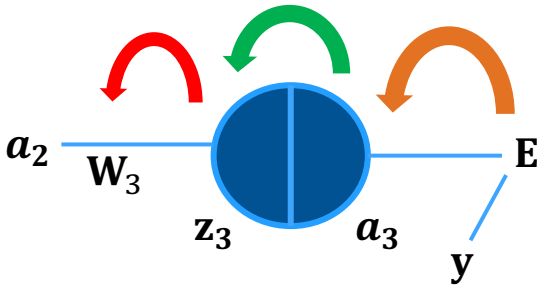
$$\frac{\delta \mathbf{E}}{\delta \mathbf{W}_3} = \underbrace{\frac{\delta \mathbf{z}_3}{\delta \mathbf{W}_3}\frac{\delta \boldsymbol{a}_3}{\delta \mathbf{z}_3}\frac{\delta \mathbf{E}}{\delta \boldsymbol{a}_3}}_{\text{Chainrule}}$$

Energy:
$$\mathbf{E} = 0.5(\mathbf{y} - \boldsymbol{a})^2$$
Activation:
$$\boldsymbol{a} = \delta(\mathbf{z})$$
Weighted sum:
$$\mathbf{z} = \mathbf{W}\boldsymbol{a}_{\text{last}}$$
Desired output: $\mathbf{y}$

Weight: $\mathbf{W}$

The same has to be done for a possible bias term $\mathbf{b}$.

Training with Backpropagation



Derivative of $\mathbf{C}$ in respect to $\mathbf{W}$:

$$\frac{\delta \mathbf{E}}{\delta \mathbf{W}_3} = \underbrace{\frac{\delta \mathbf{z}_3}{\delta \mathbf{W}_3} \frac{\delta \mathbf{a}_3}{\delta \mathbf{z}_3} \frac{\delta \mathbf{E}}{\delta \mathbf{a}_3}}_{\text{Chainrule}}$$

Energy to activation:

$$\frac{\delta \mathbf{E}}{\delta \mathbf{a}_3} = (\mathbf{y} - \mathbf{a}_3)$$

Activation to weighted sum:

$$\frac{\delta \mathbf{a}_3}{\delta \mathbf{z}_3} = \delta(\mathbf{x}) \cdot (1 - \delta(\mathbf{x}))$$

for the sigmoid

Weighted sum to weights:

$$\frac{\delta \mathbf{z}_3}{\delta \mathbf{W}_3} = \mathbf{a}_2$$

Energy:

$$\mathbf{E} = 0.5(\mathbf{y} - \boldsymbol{a})^2$$

Activation:

$$\boldsymbol{a} = \delta(\mathbf{z})$$

Weighted sum:

$$\mathbf{z} = \mathbf{W}\boldsymbol{a}_{\text{last}}$$

Desired output: $\mathbf{y}$

Weight: $\mathbf{W}$

The same has to be done for a possible bias term $\mathbf{b}$.

Updating of the weights

Update of the weights with the calculated derivative:

$$\Delta \mathbf{W}_3 = -\mu \frac{\delta \mathbf{E}}{\delta \mathbf{W}_3}$$

The parameter μ is the learning rate, it influence how much the weights are updated.

Why do we need a learning rate?

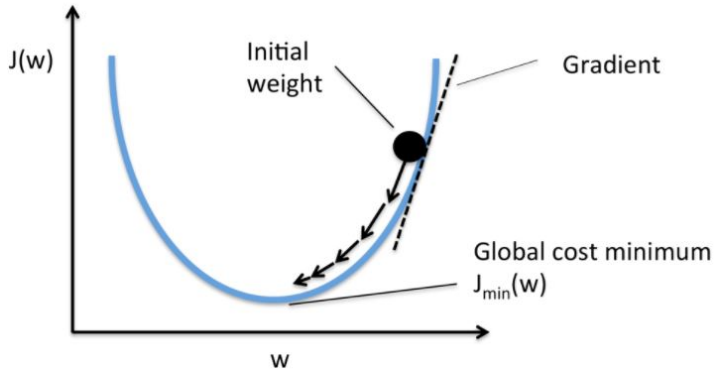
What is an optimizer?

Optimizer

An optimizer for a NN minimizes the error based on the gradient

The update steps are the same as in a gradient descent approach, we walk along the gradient to an optimal solution.

The learning rate μ determines the size of our steps.



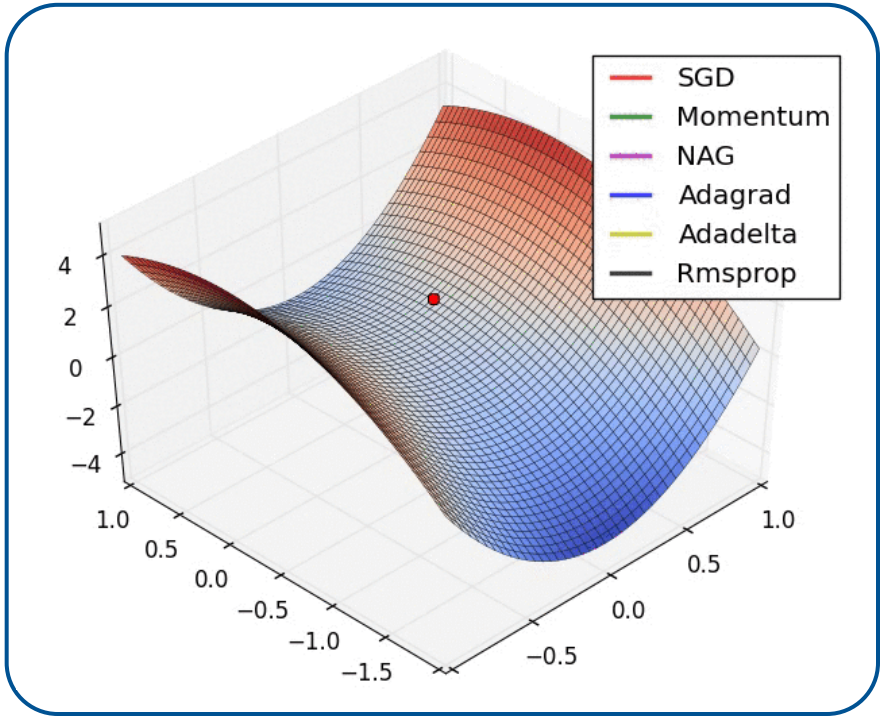
Stochastic gradient descent (SGD):

Instead of using a single data point, it uses a mini-batch and averages the gradients over all input points, to get a better approximation of the energy function, on which we optimize our weights.

Optimizer

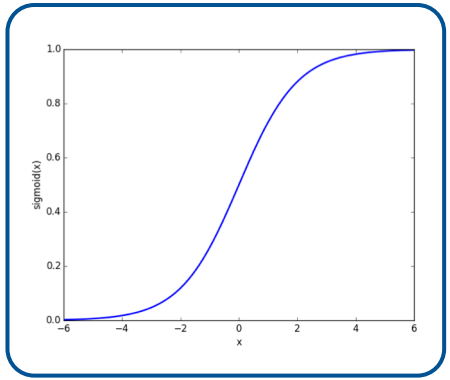
There are a lot of different optimizers:

For the most problems Adam (not shown), should perform fine, however this is not always the case!

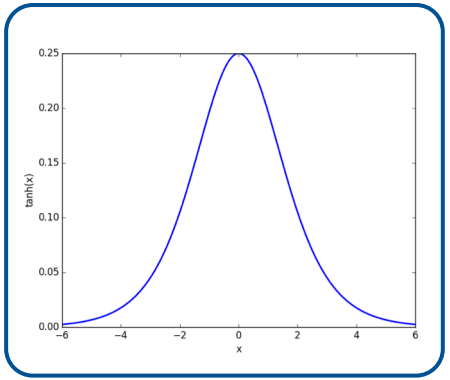


Vanishing Gradients

A common problem for deep neural networks (a lot of hidden layers > 5):



sigmoid



derivative of the sigmoid

In a deep neural network the errors are propagate back through the network depending on their activations. However if the activations are high the gradients gets smaller and smaller in each layer, which leads eventually to no gradient at all.

Use ReLu instead

Loss functions

There are different types of loss functions available.

- Classification:
  - Binary: use of a single neuron, with a sigmoid at the end
  - Multi: use of several neurons, which are using a softmax function:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The error term is defined by the cross entropy loss, for one sample:

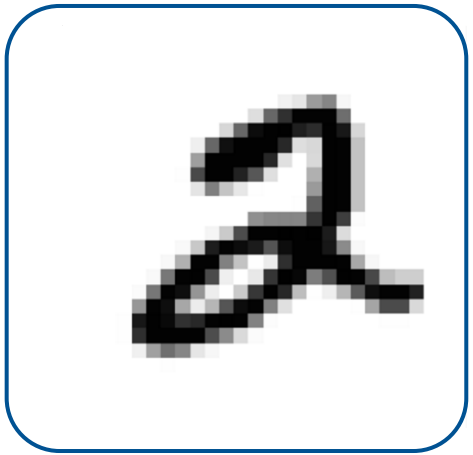
$$\mathbf{E} = -\mathbf{y} \cdot log(\boldsymbol{a})$$

- Regression:
  - It is sufficient to use a linear output unit with a least squared error

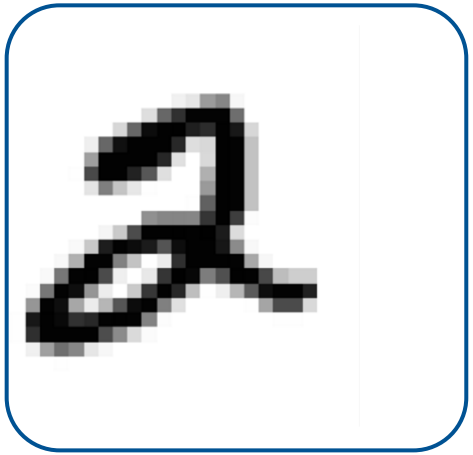
Deep Learning the mystery

Neural network for images

Moving of a picture by a few pixel leads to a complete different input for a normal NN



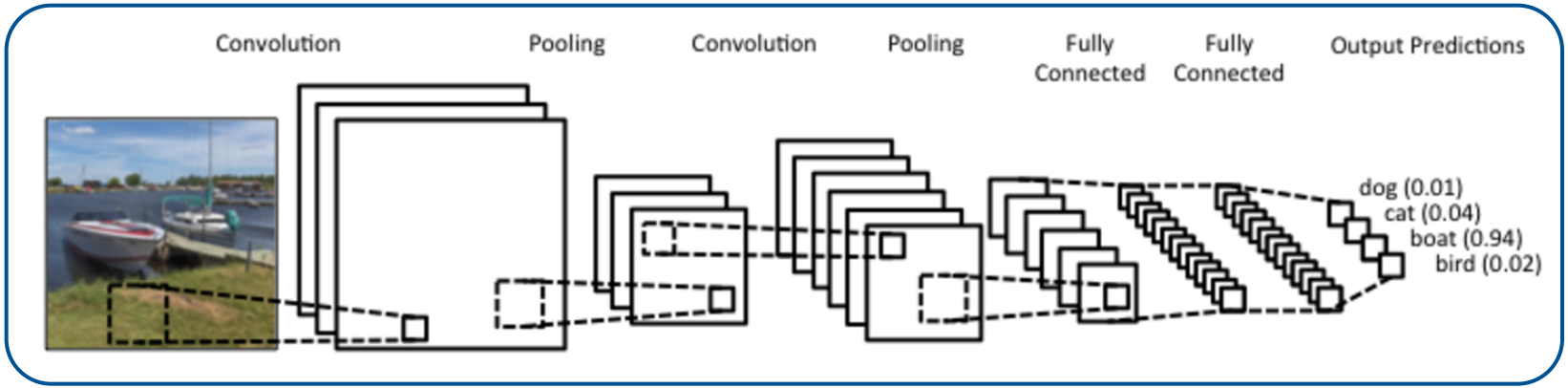
Prediction: 2



Prediction: 8

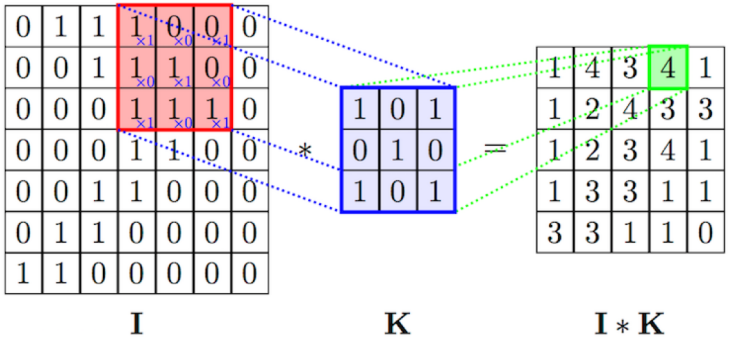
Convolutional Neural Network (CNN)

CNNs consist out of convolutional layers, pooling layers and fully connected layers



Convolutional Layer

Convolutions are simple filters, which are applied to every patch of the image and give a certain response.



In this image the filter K is applied on the 5th pixel in the 2nd row, each value of the sub-image is multiplied with the filter and then summed up. Afterwards an activation function is applied. This can be compared to a normal NN, by saying the filter K is our weight matrix and we only apply it to certain values from the sub-image.

Convolutional Layer

Convolutional Layer Video

Convolutional Layer

Types of padding:

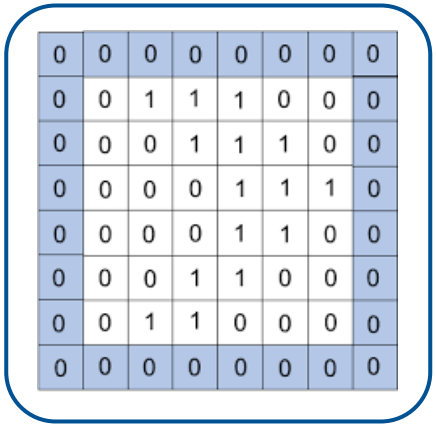
- No padding: no border is added to the filter responses and in each layer the size decreases
- Zero padding: fill the border with zeros
- Reflect padding: fill the borders with the same values in a reflected way
- Symmetric padding: fill the borders by repeating the same values

Stride:

Instead of applying the filter to every pixel a stride value can be set, which determines how big the distance between an application of the filter is.

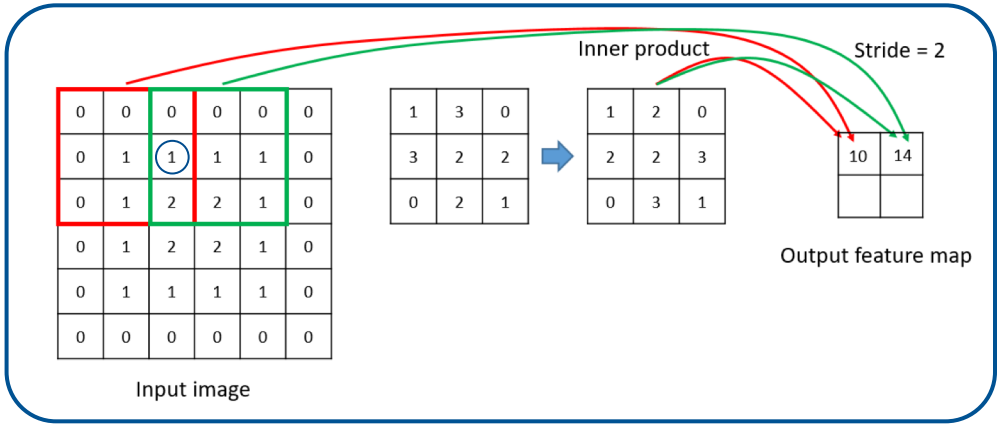
Convolutional Layer

Zero padding:



Fill the borders for the filters with zeros

Stride:

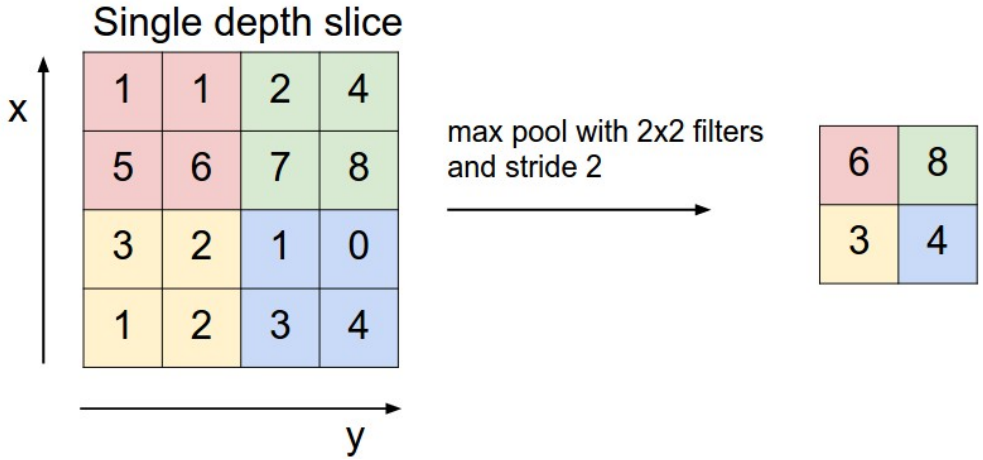


A stride of two, the circled pixel is skipped

Pooling Layer

Reducing the amount of weights, by combing/filtering them.

Max Pooling:



There are different approaches to pooling:

Max Pooling:

$$f(x) = \max(x)$$

Average Pooling:

$$f(x) = \frac{1}{\|x\|} \sum_i x_i$$

Pooling Layer

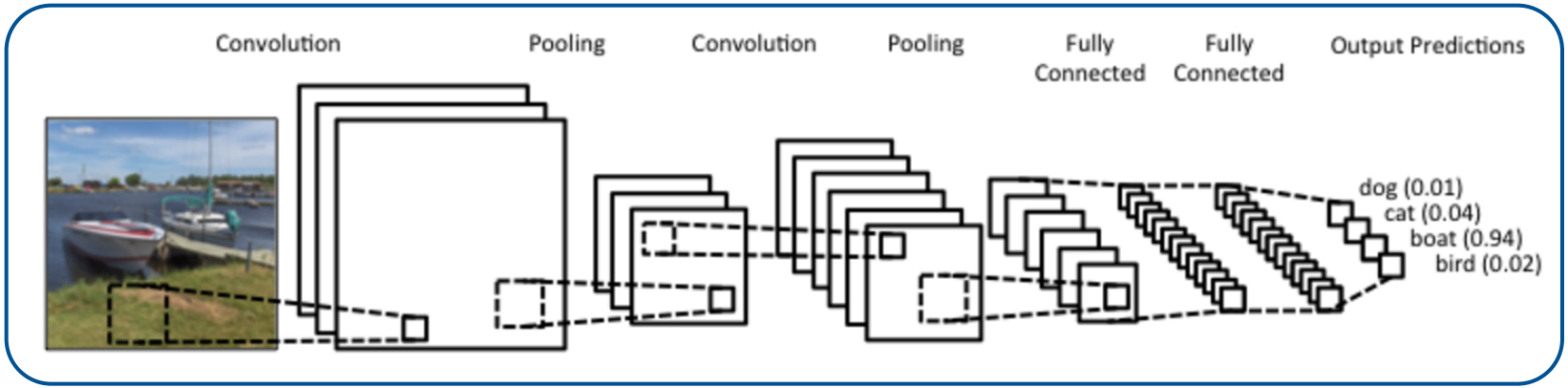
Reducing the amount of weights, by combing/filtering them.

Advantages:

- Reduces variance
- Reduces computation, by reducing complexity, without this reduction the most CNNs couldn’t be trained

Convolutional Neural Network (CNN)

CNNs consist out of convolutional layers, pooling layers and fully connected layers

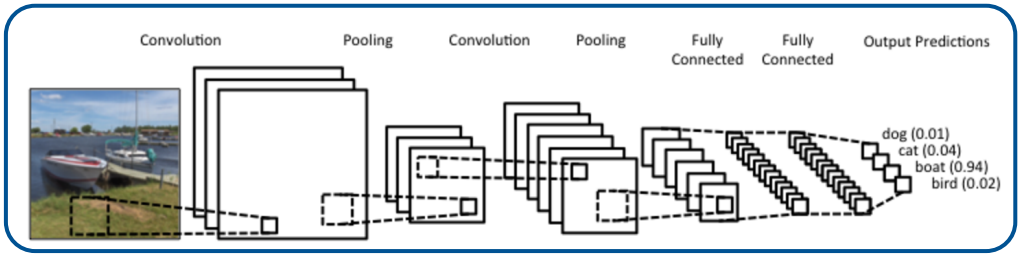


Nice visualization of the activations: http://scs.ryerson.ca/~aharley/vis/conv/

Convolutional Neural Network (CNN)

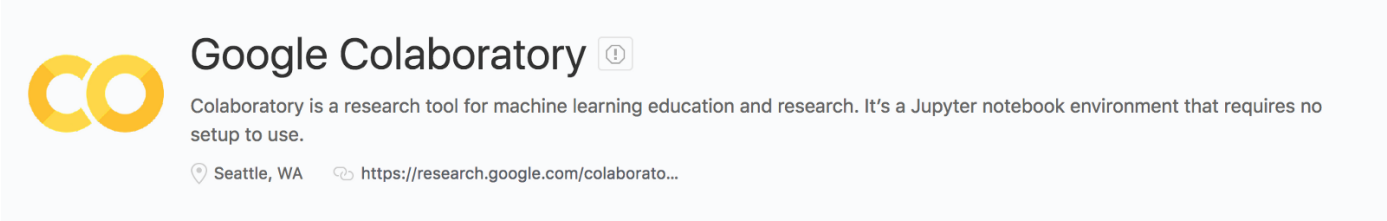
Advantages:

- Far less parameters necessary than for a neural network
- Translational invariance, because the filter is applied at every position in the picture
- Matrix multiplications can easily be done on a GPU
- Hierarchical Structure



Neural Networks and CNN Demo

Google Colab, a great tool for testing deep learning methods, if you don’t have a super strong GPU.



Summary

- Neural Networks use single Neurons to generate a larger connected network
- Backpropagation is used to train the weights used in a Neural Network
- There are different activation functions, which can be used in a NN
- The combination of convolutional layers, pooling layers and fully connected layers build up a CNN
- CNNs are great image classificators, because of their translational invariance and their hierarchical structure