



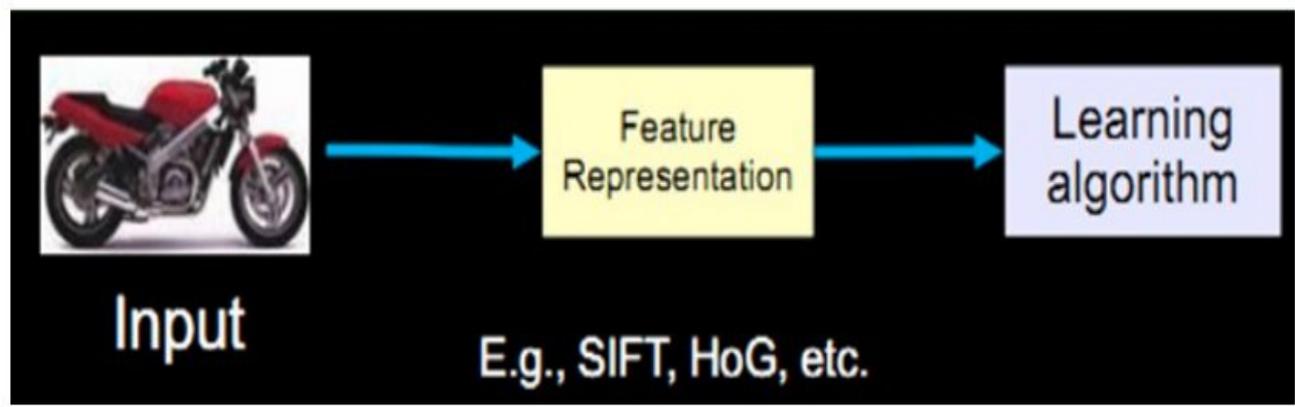
DSHL Summer School 2022

Day 2: Deep Learning

Martin Antenreiter

Conventional Machine Learning

- Limited in their ability to process natural data in their raw form. We need features!
 - Creating features is difficult, time-consuming, requires expert knowledge.



Deep Learning

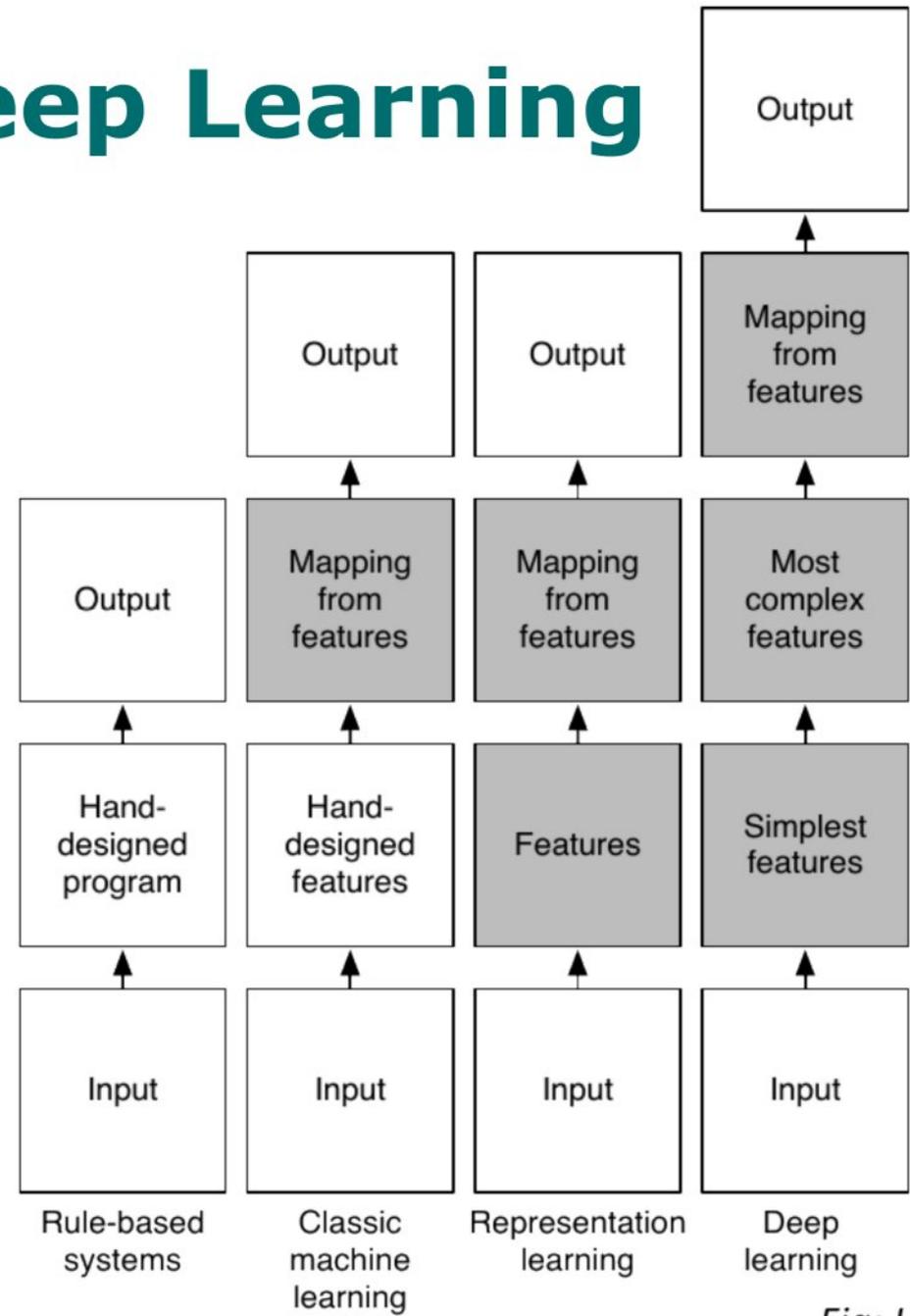
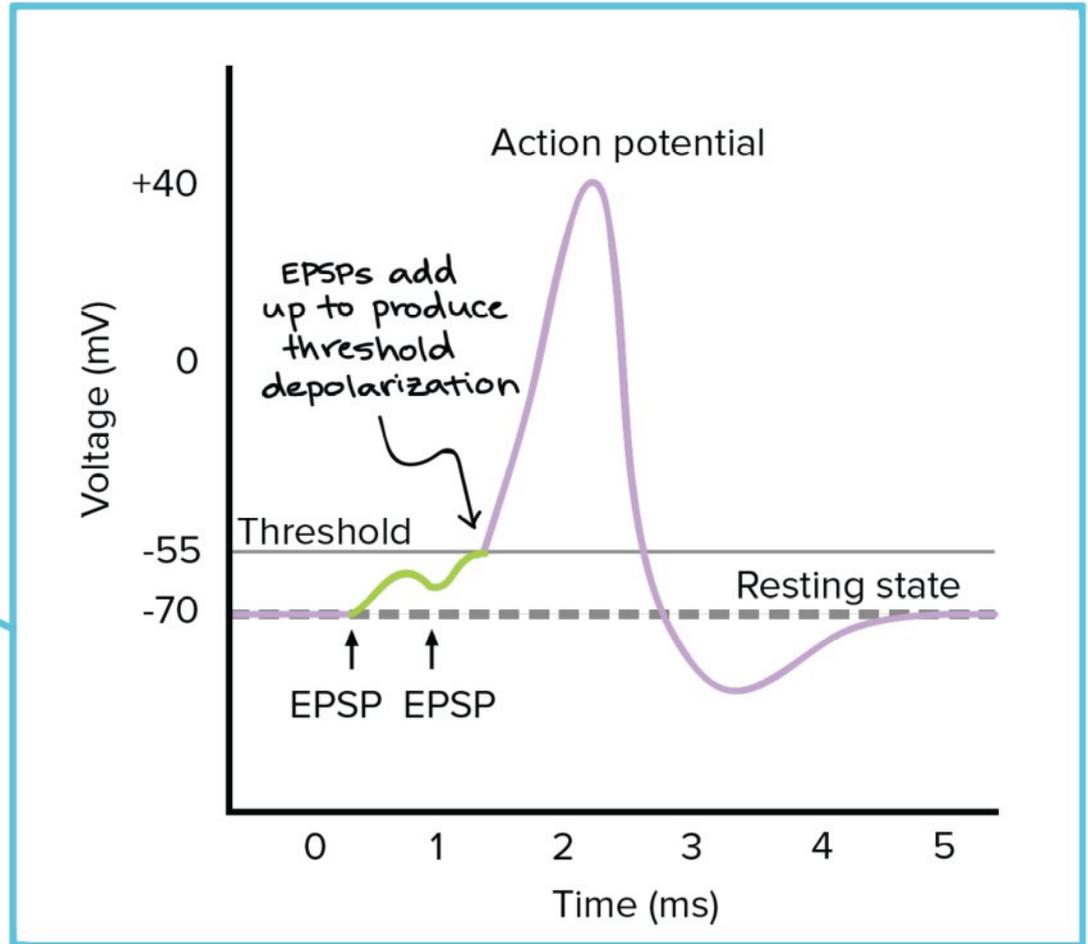
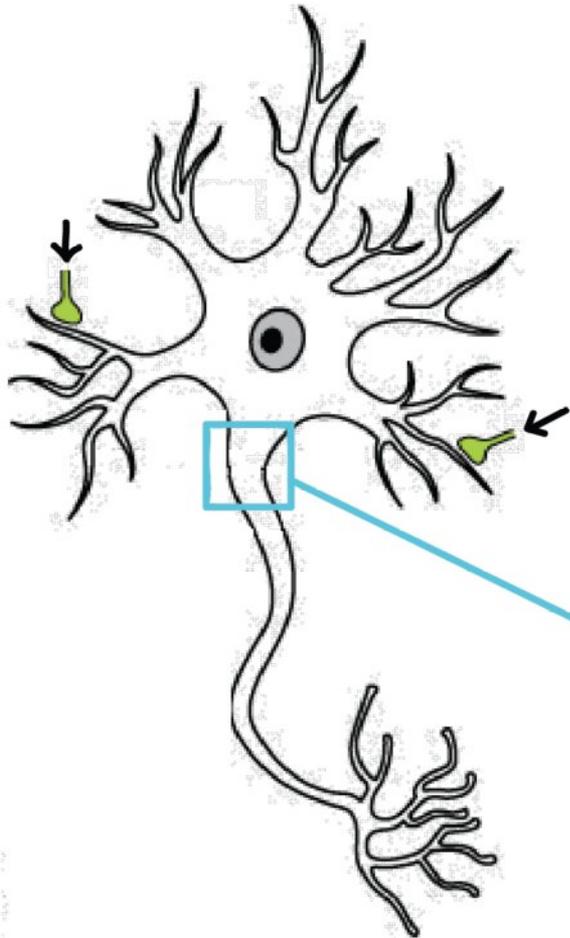


Fig: 1. Goodfellow

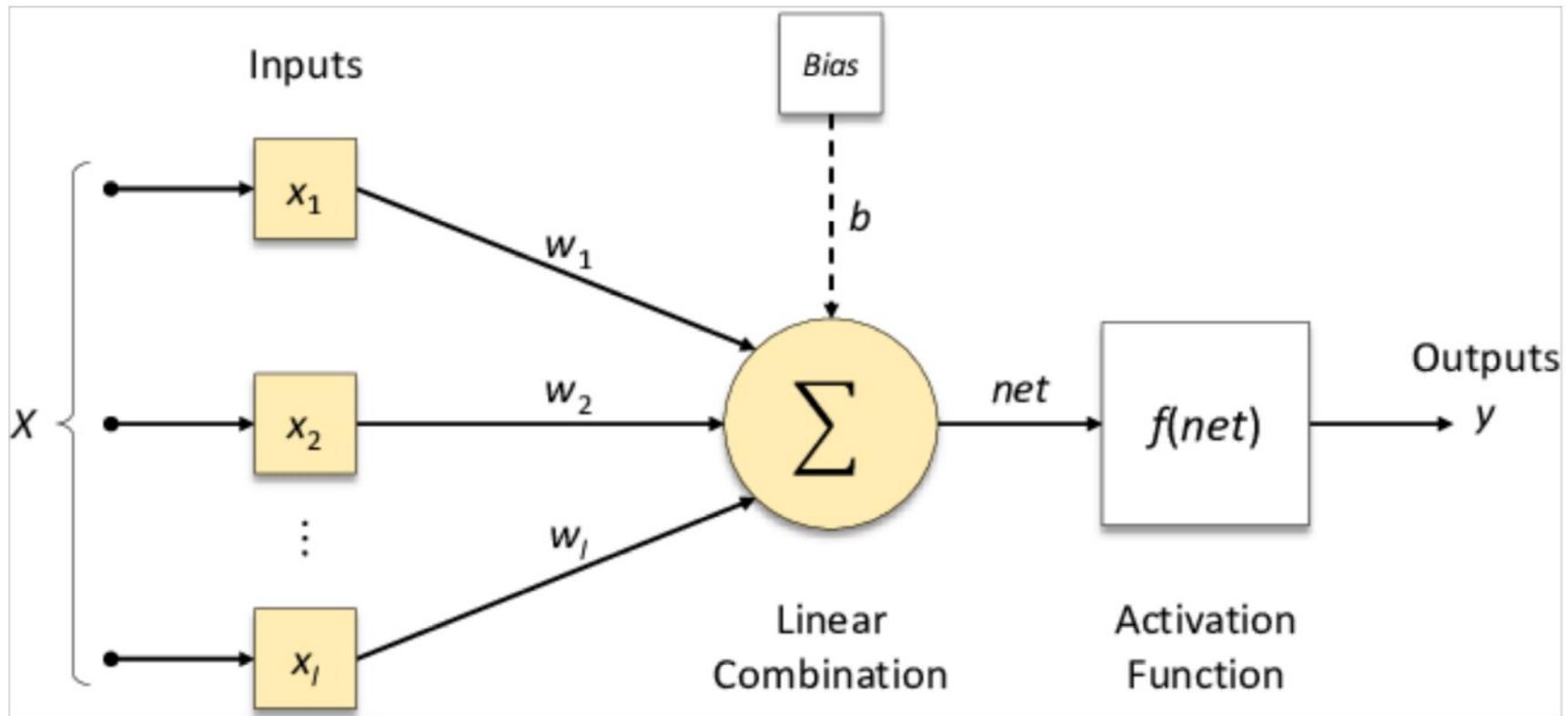
History

- 1958 - Frank Rosenblatt creates the Perceptron.
- 1959 - Hubel and Wiesel elaborate cells in Visual Cortex
- 1975 - Paul J. Werbos develops the Backpropagation Algorithm
- 1980 - Neocognitron, a hierarchical multilayered ANN.
- 1990 - Convolutional Neural Networks

Model: Biological Neuron



Artificial Neuron - Perceptron

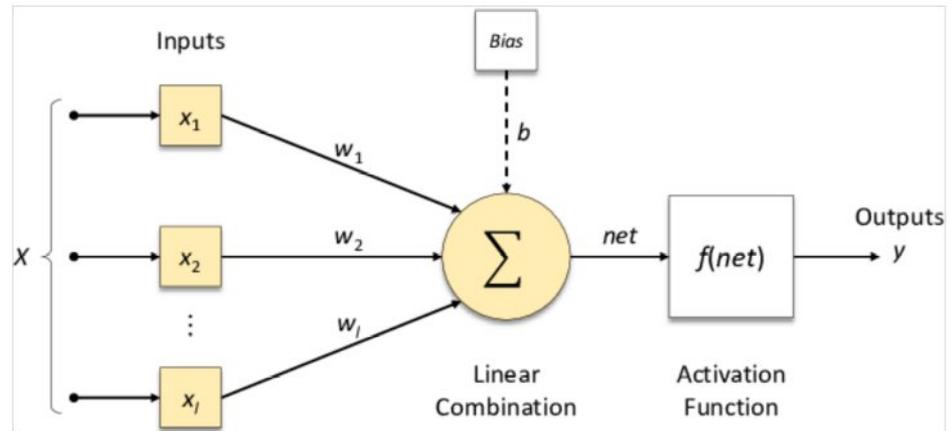


$$net = \sum_{i=1}^n x_i w_i + b$$

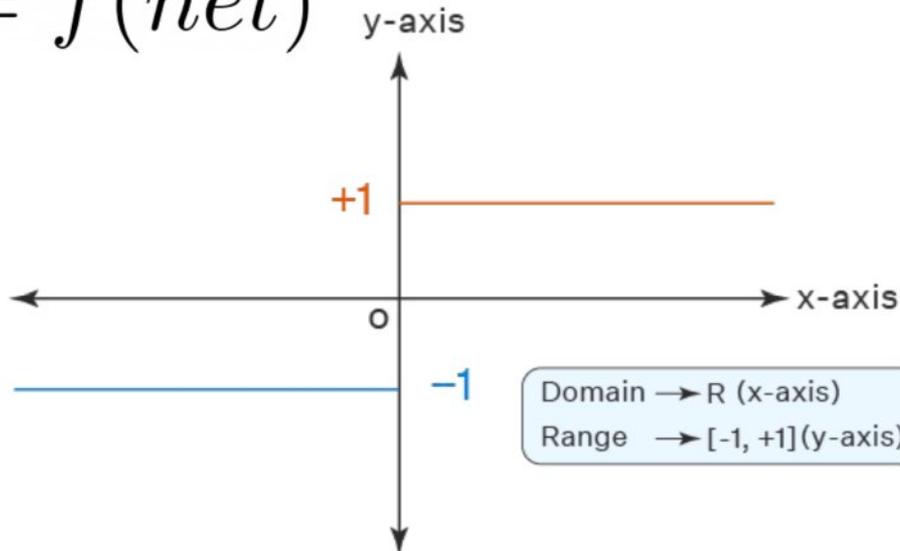
$$y = f(net)$$

Perceptron

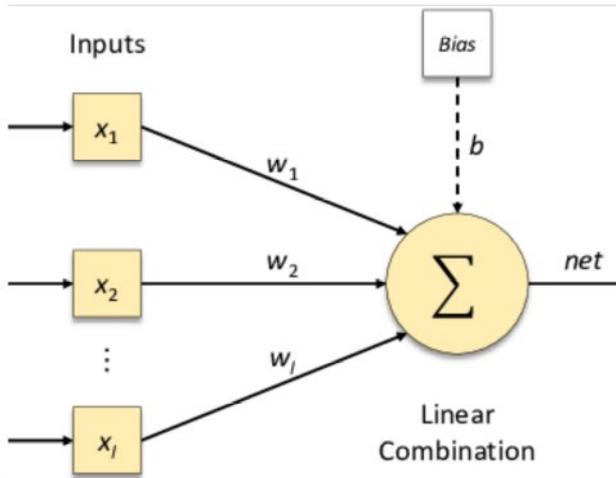
$$net = \sum_{i=1}^n x_i w_i + b$$



$$y = f(net)$$



Function of the Neuron



$$net = \sum_{i=0}^n x_i w_i$$

$$b = w_0 \text{ with}$$

$$x_0 = 1$$

The bias is usually represented by the weight with the index 0 and the input vector is extended by one digit.

Perceptron Rule

- Rosenblatt's initial perceptron rule is fairly simple and can be summarized by the following steps:
 - Initialize the weights to small random numbers.
 - For each training sample x :
 - 1. Calculate the output value

$$net = \sum_{i=0}^n x_i w_i$$

- 2. If the perceptron made an error, update the weights

$$w_{new} = w_{old} + \eta \Delta w$$

Perceptron Rule

$$w_{new} = w_{old} + \eta \Delta w$$

Let's learn w_i 's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

Gradient Descent

- Minimize the squared error on the training examples!

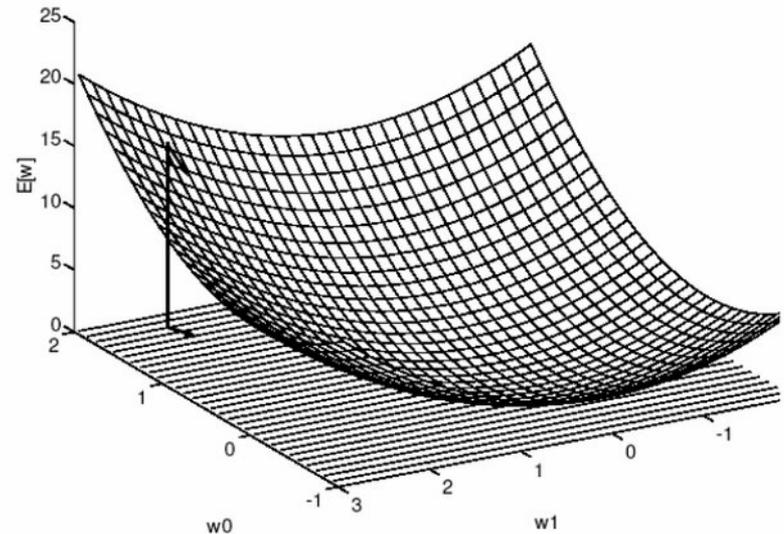
- Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- Training Rule

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$



Perceptron Rule

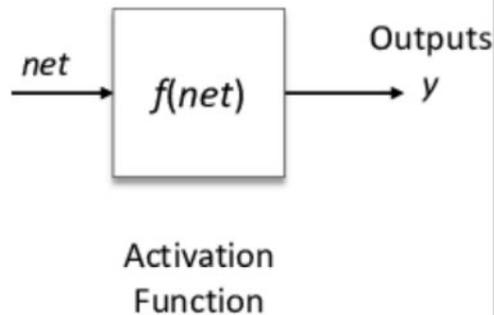
- If we made an error, update the weights

$$net = \sum_{i=0}^n x_i w_i$$

$$w_{new} = w_{old} + \eta \Delta w$$

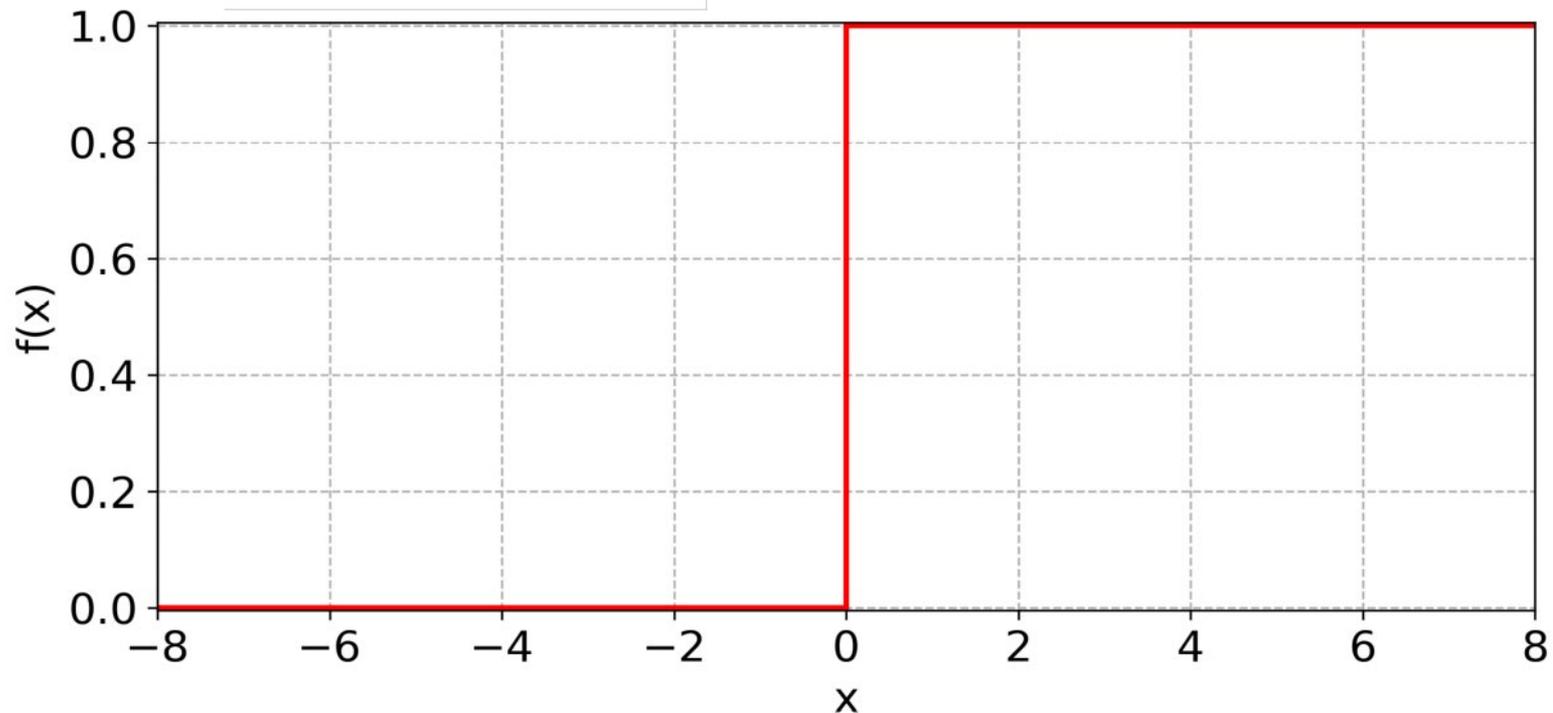
$$\Delta w_i = (target - output) x_i$$

Activation functions - Threshold function

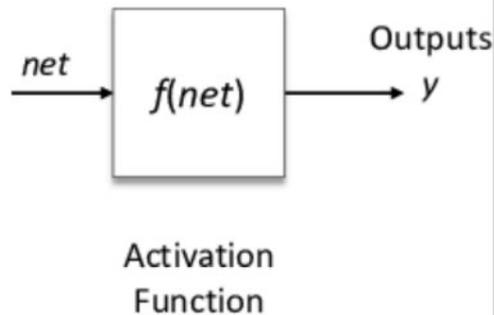


Properties

- Not differentiable

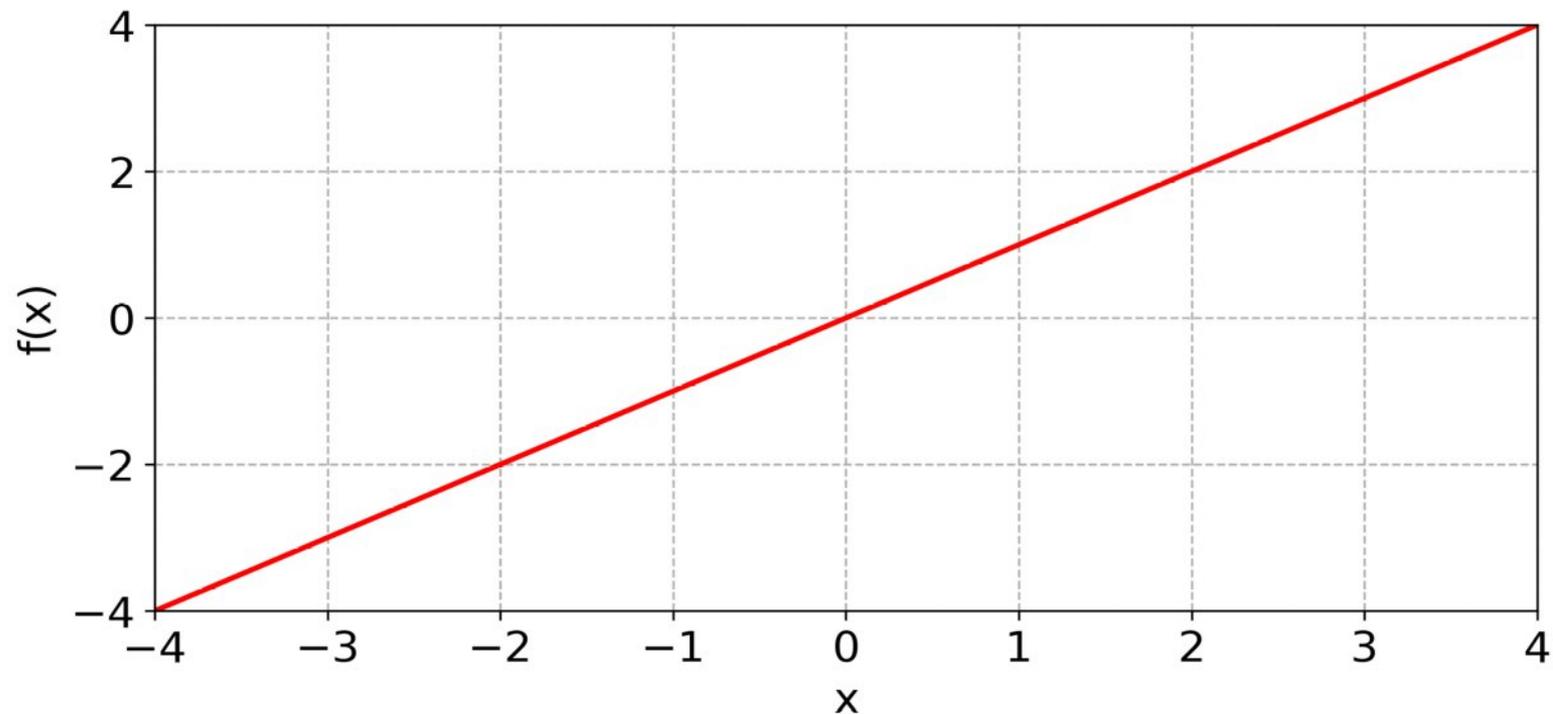


Activation functions - Linear function (Identity)

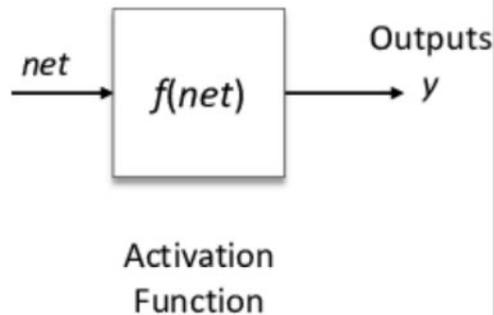


Properties

- Differentiable

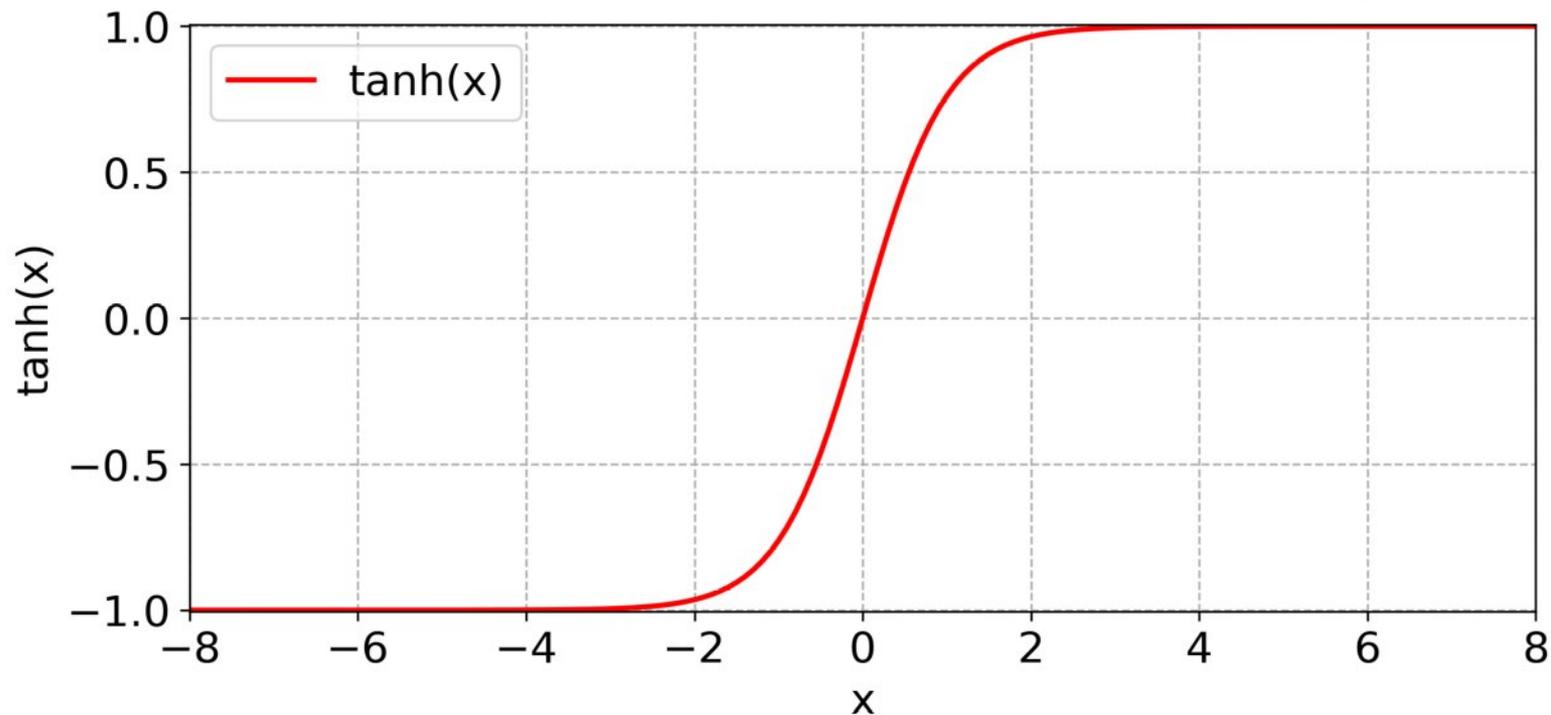


Activation functions - Tanh

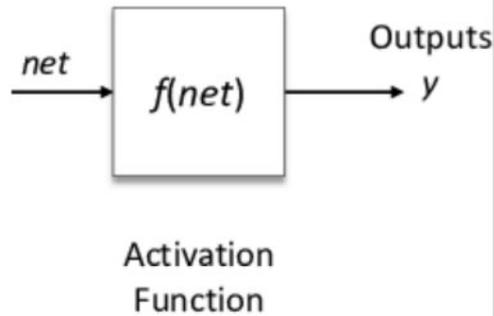


Properties

- Differentiable
- Saturation effects
- Computationally intensive

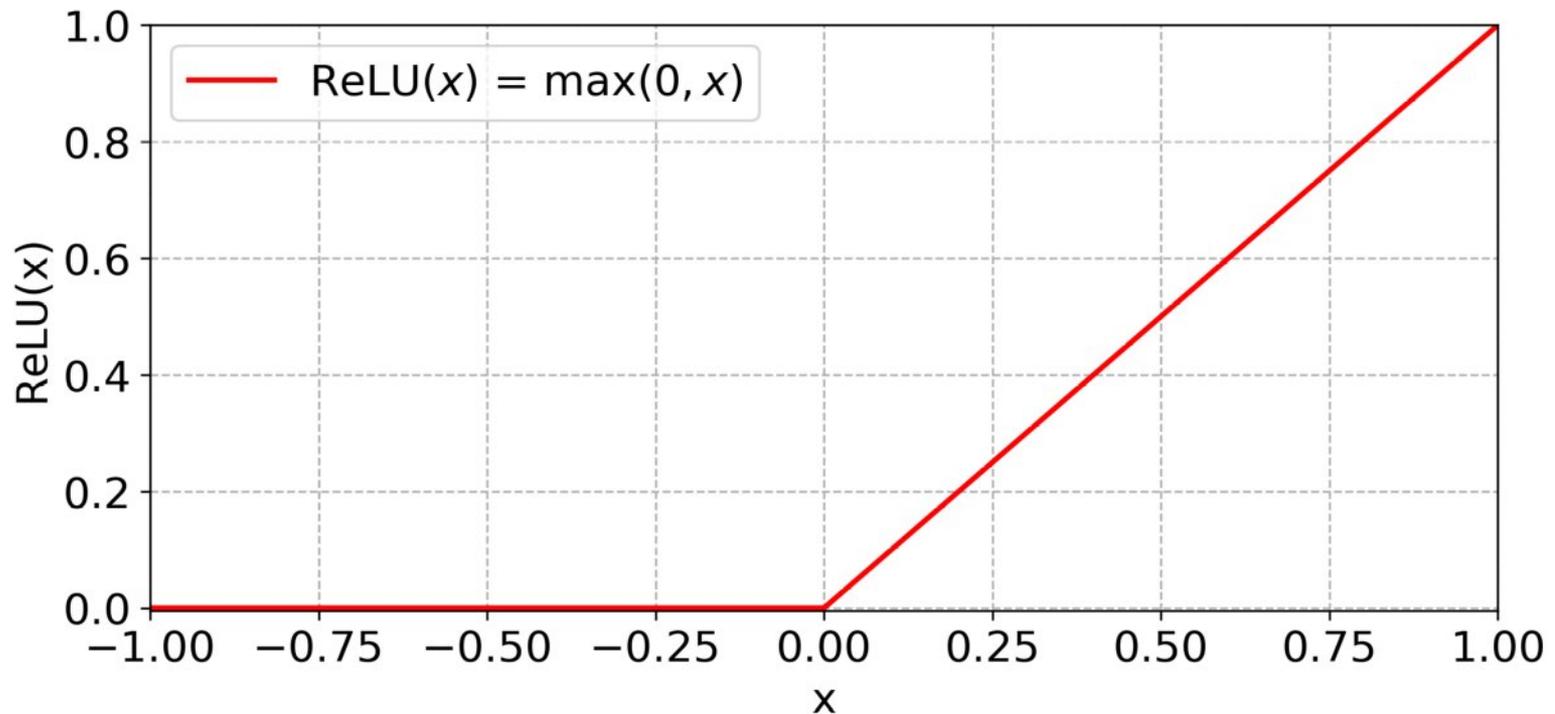


Activation functions - Rectified Linear Unit (ReLU)



Properties

- Unrestricted
- Activity only at $x > 0$





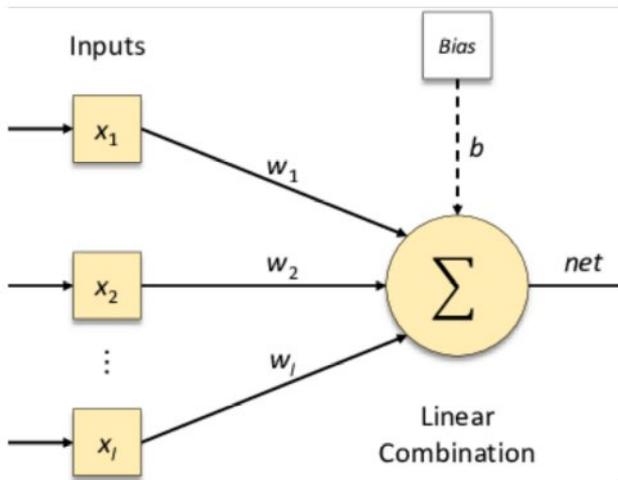
Built-in activation functions in Tensorflow (TF)

- see `tf.keras.activations`
- https://www.tensorflow.org/api_docs/python/tf/keras/activations

Build a Network

- By linking simple neurons together, a complex network is created.
- Weights determine the "total function".
- Calculation can be done in parallel

Function of the Neuron



$$net = \sum_{i=0}^n x_i w_i$$

$$b = w_0 \text{ with}$$

$$x_0 = 1$$

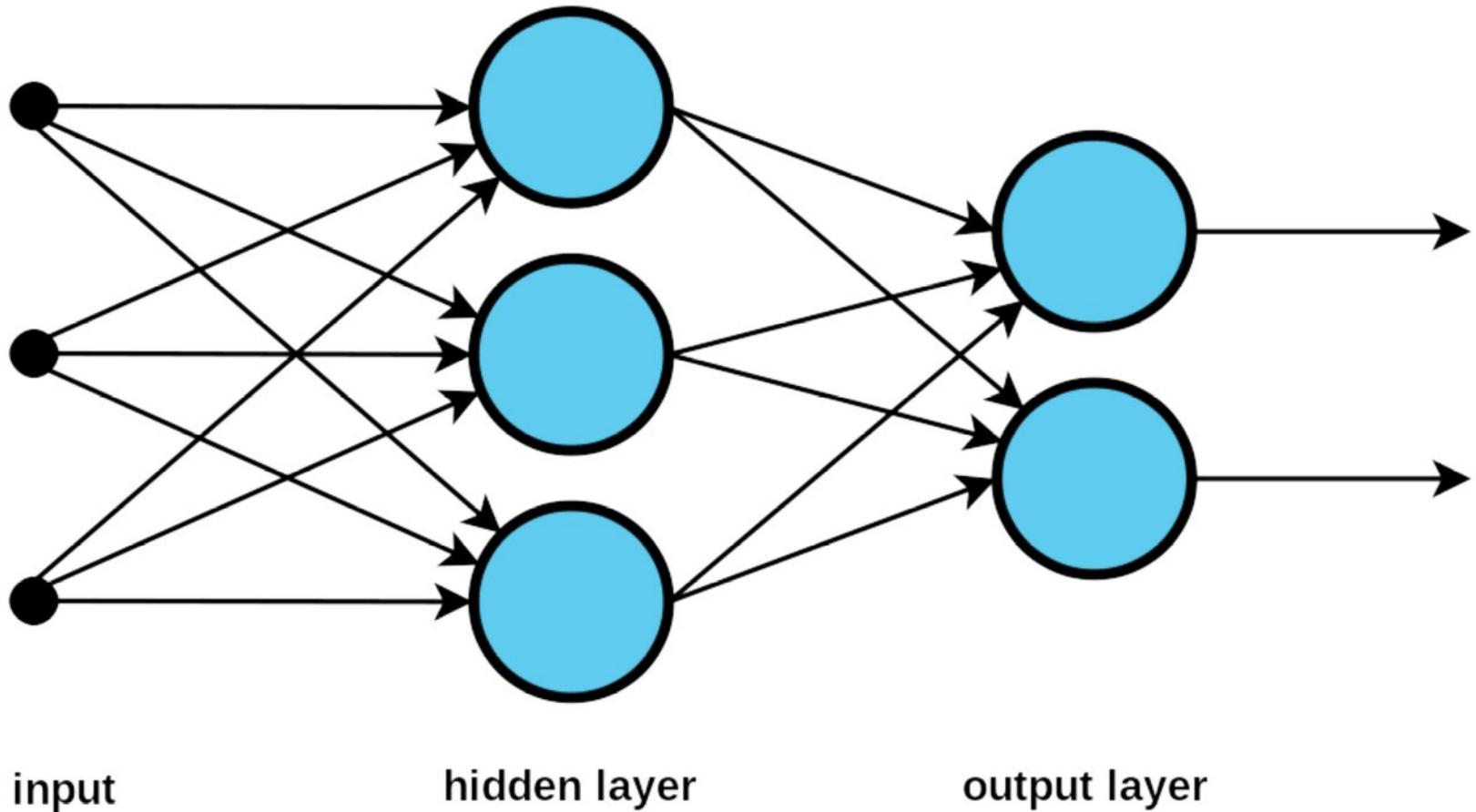
The bias is usually represented by the weight with the index 0 and the input vector is extended by one digit.

Parallel calculation on GPU

- Nvidia Tesla A100 Ampere
 - 40GB RAM
 - Tensor-Cores

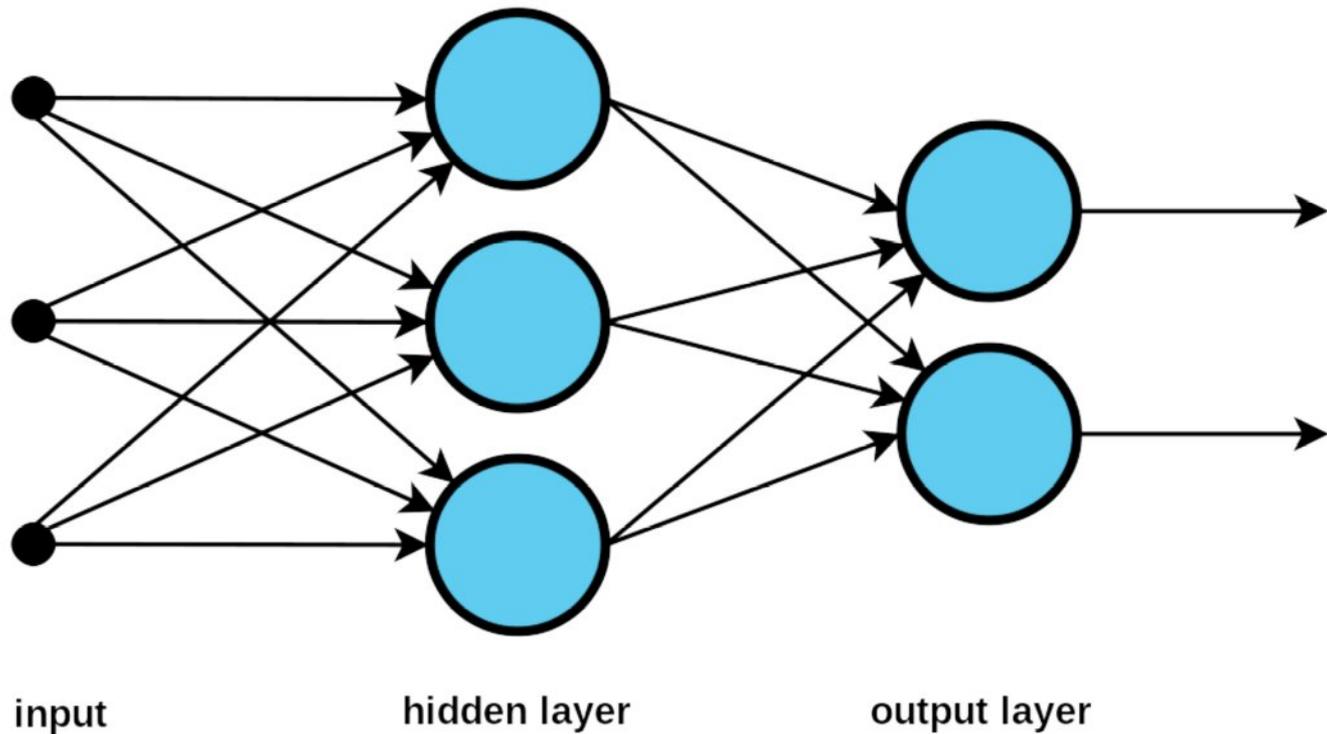


Feedforward Network



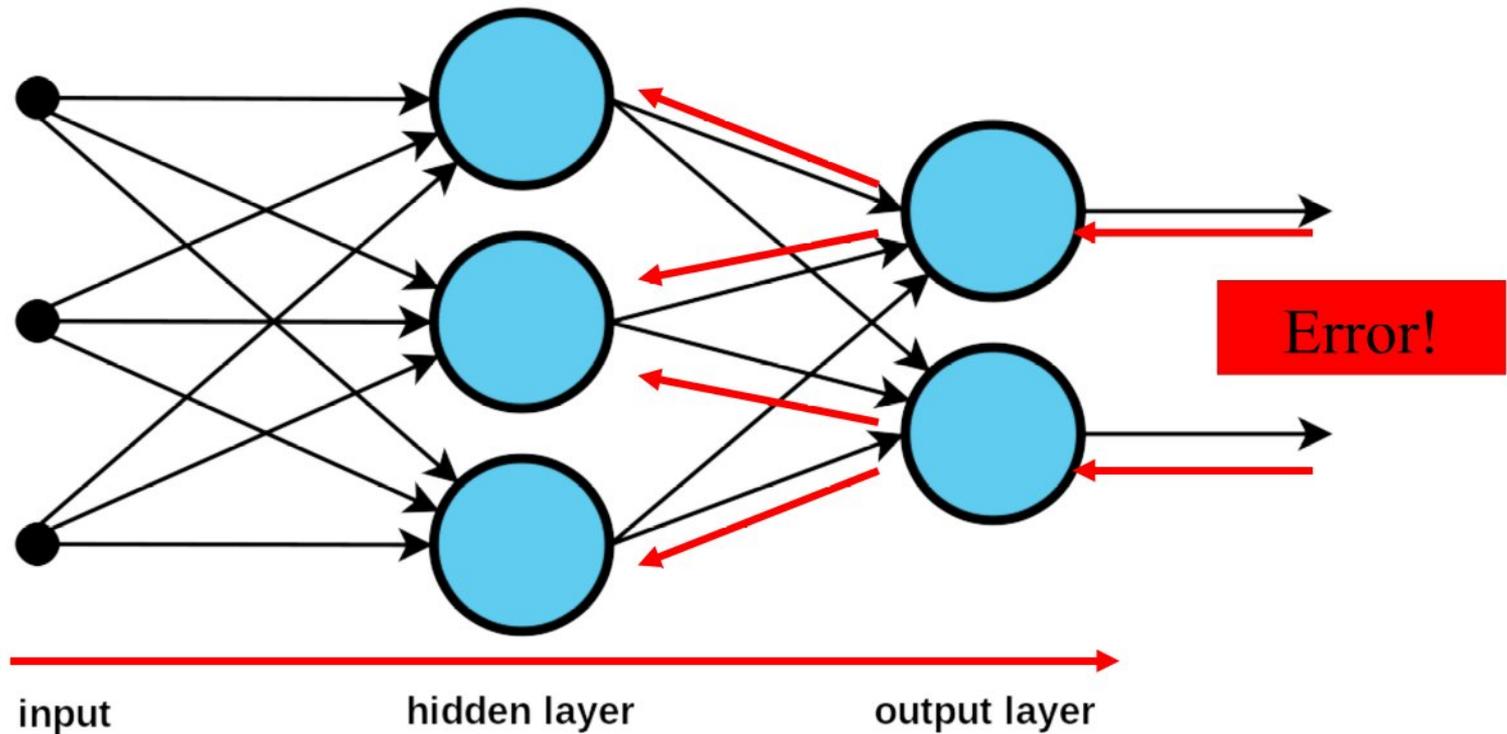
Network - Initialization

Weights are initialized with small random values at the beginning.



Network Training

- So how do we learn the weights?
 - Backpropagation of Error



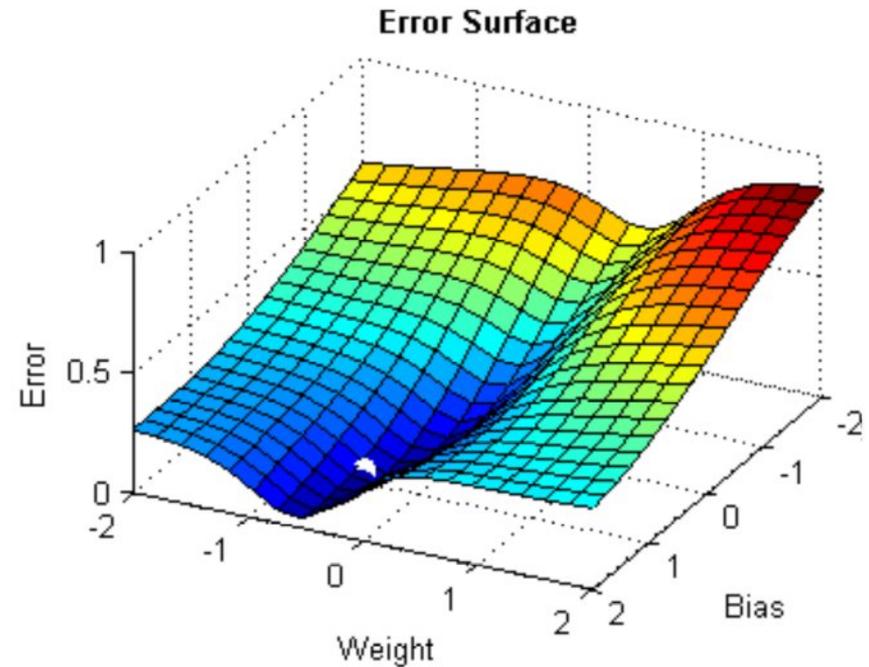
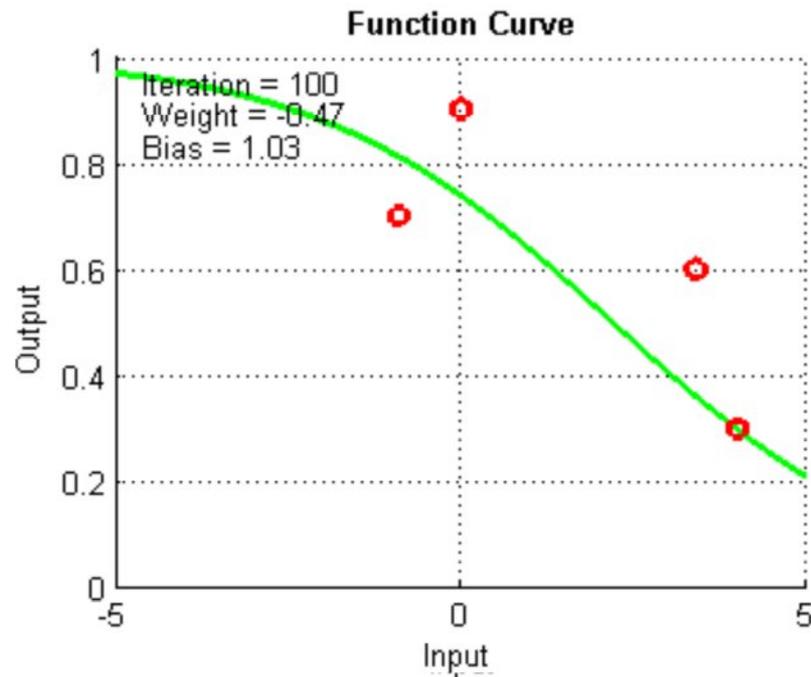


Network Training Update of the Weights

Different optimization methods are available
see [tf.keras.optimizers](https://keras.io/optimizers/)

- Gradient method
- Update with learning rate * Gradients
- Variable inertia term (Momentum)
- Procedure with variable learning rate

Example: Gradient method



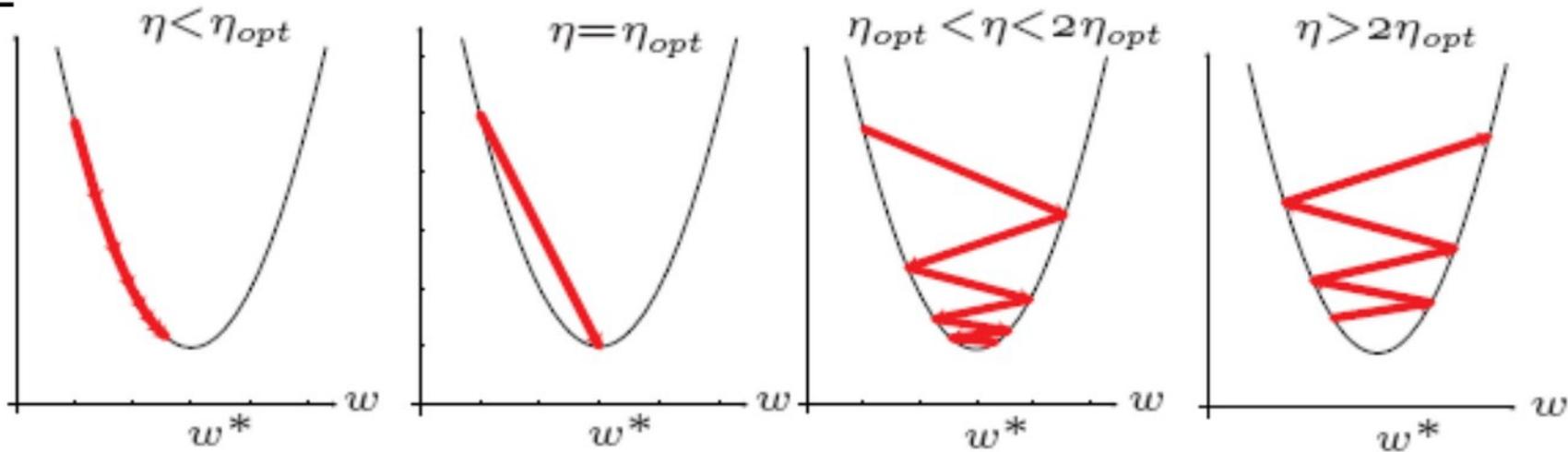
Network Training Settings

- Settings
 - learning rate
 - Regularization parameters
 - Algorithm for the optimization of weights

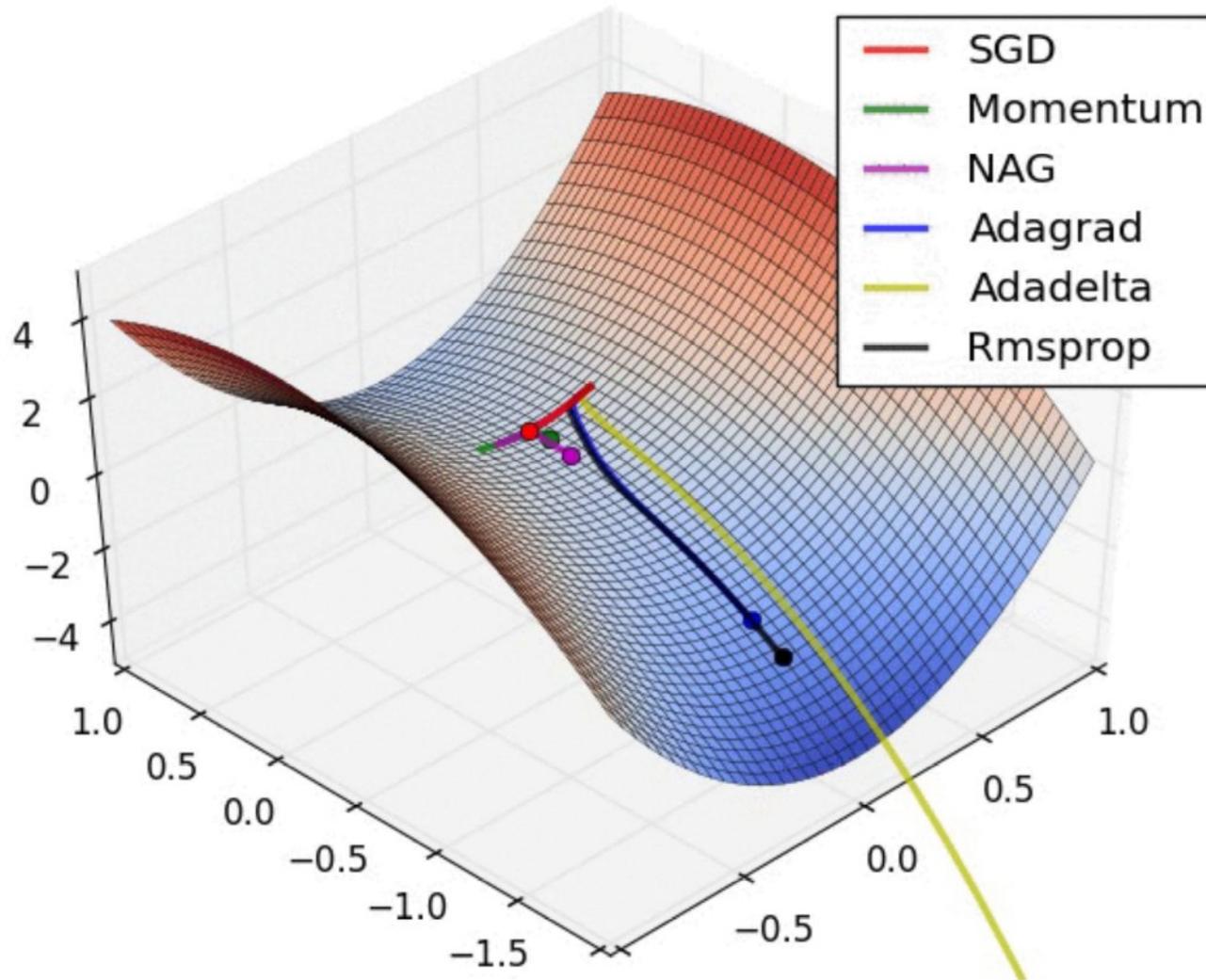
- Lots of data and big networks:
 - Training time can take several days/weeks!

Network Training Learning Rate

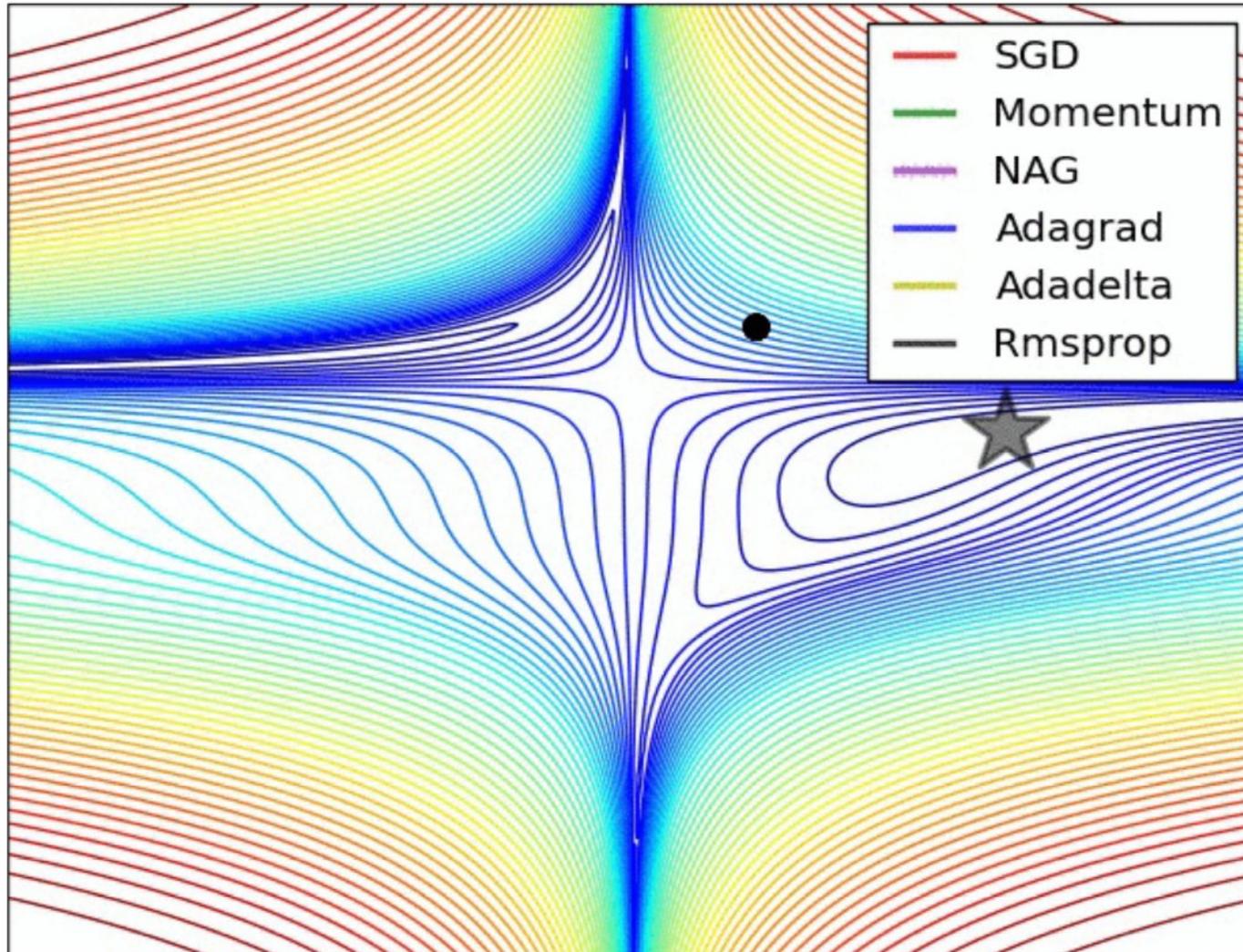
MSE



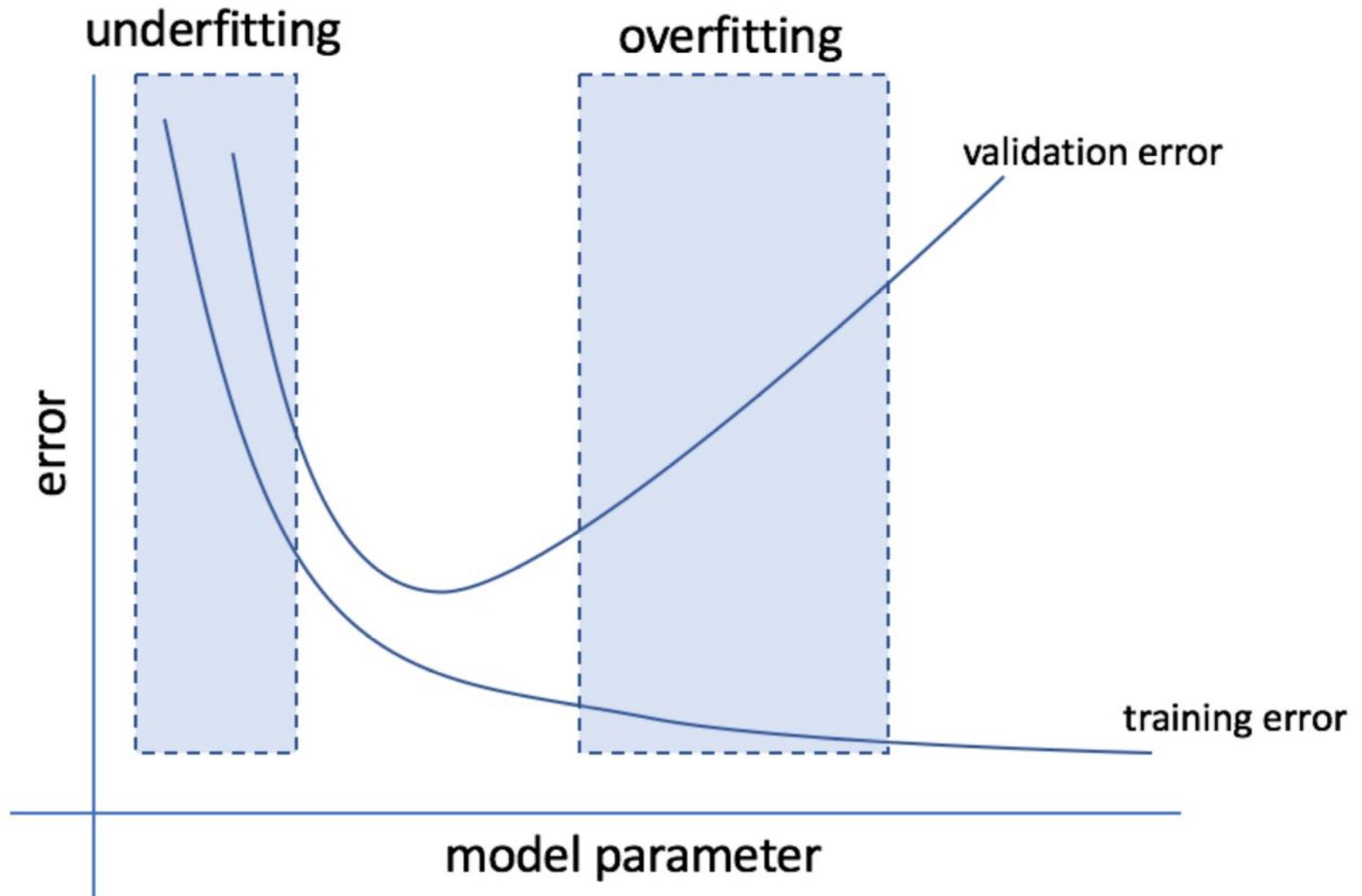
Optimization methods



Optimization methods (2)

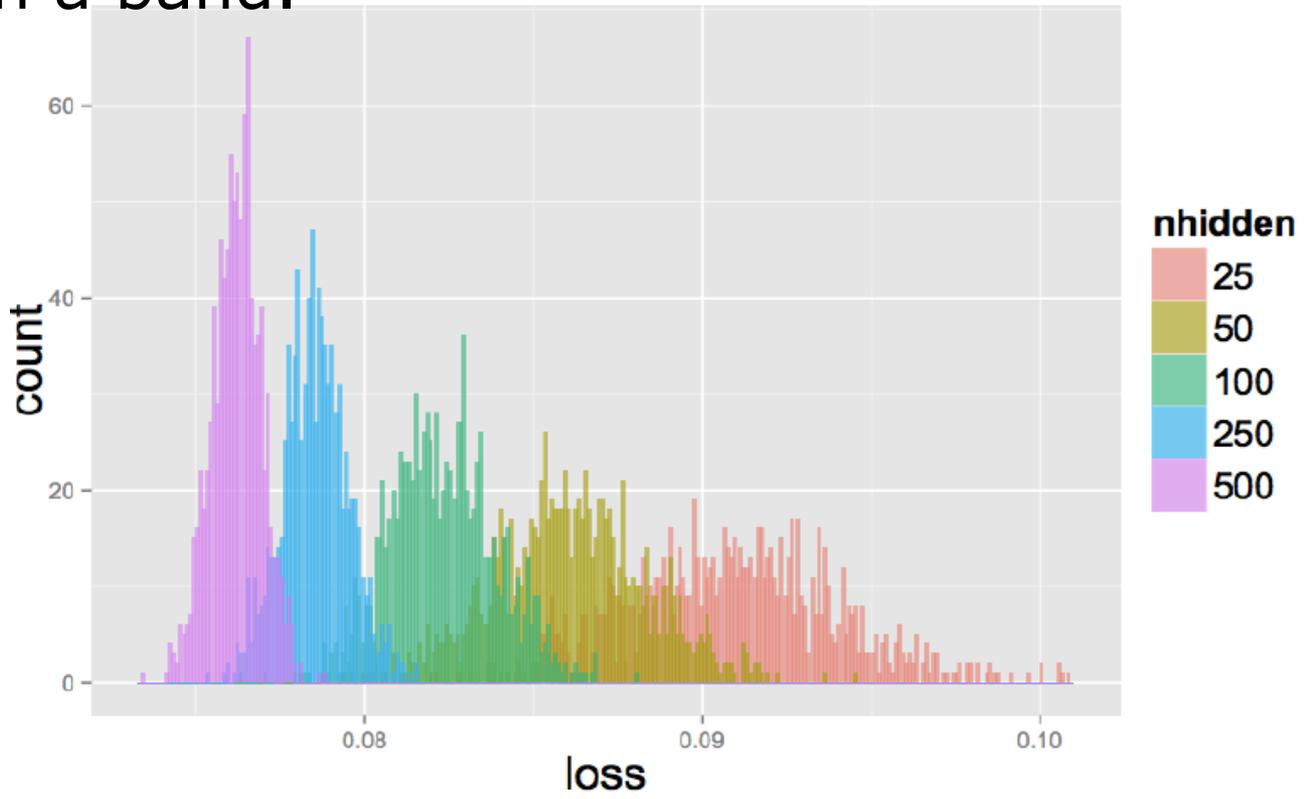


Network Training



Large Models

- Choromanska et al & LeCun 2014,
- 'The Loss Surface of Multilayer Nets'
- The low-index critical points of large models concentrate in a band.



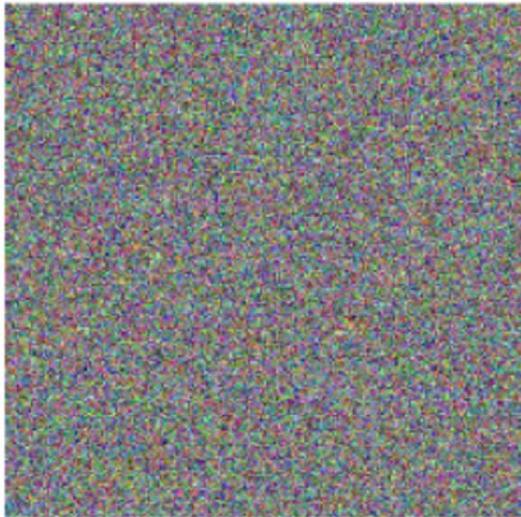
Trained NN

- Neural network after training
 - What has it learned?
 - Which structures are important for the NN?

Is an analysis & interpretation possible?

Study of learned NN

- Which input is important for an object class?
 - Noise at input
 - Output desired object class



optimize
with prior



Study of learned NN

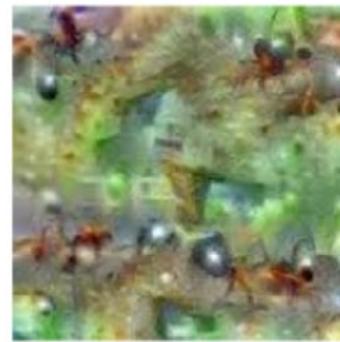
- Which input data are important for e.g. hartebeest, clownfish,...



Hartebeest



Measuring Cup



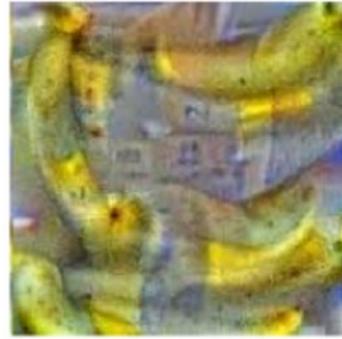
Ant



Starfish



Anemone Fish



Banana



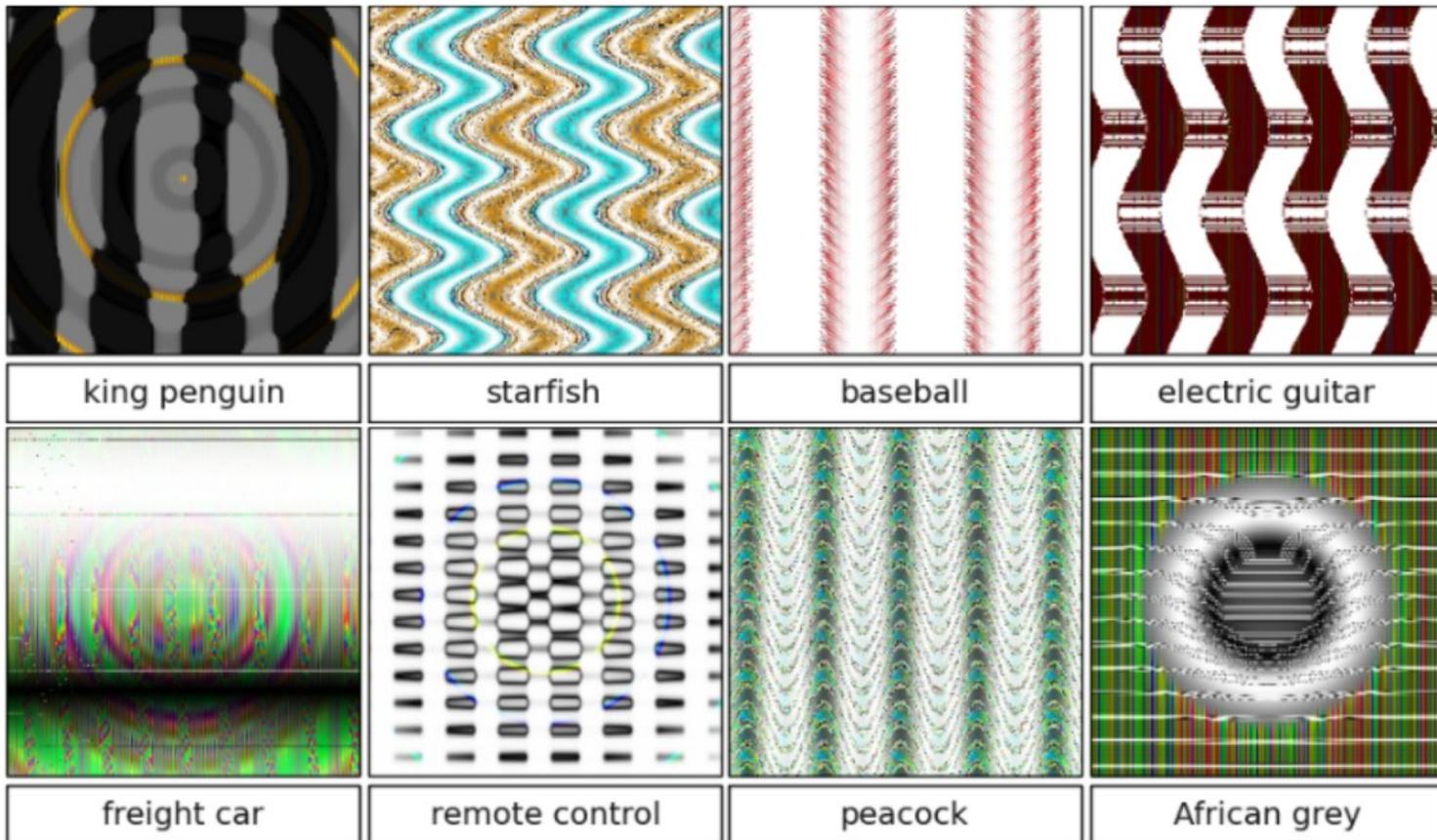
Parachute



Screw

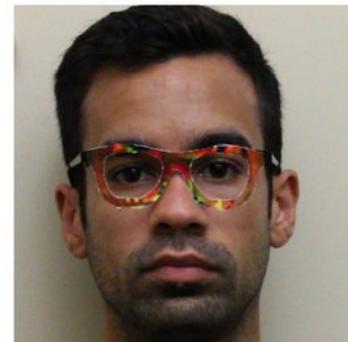
Fooling NN (1)

- Network is very certain > 99.0



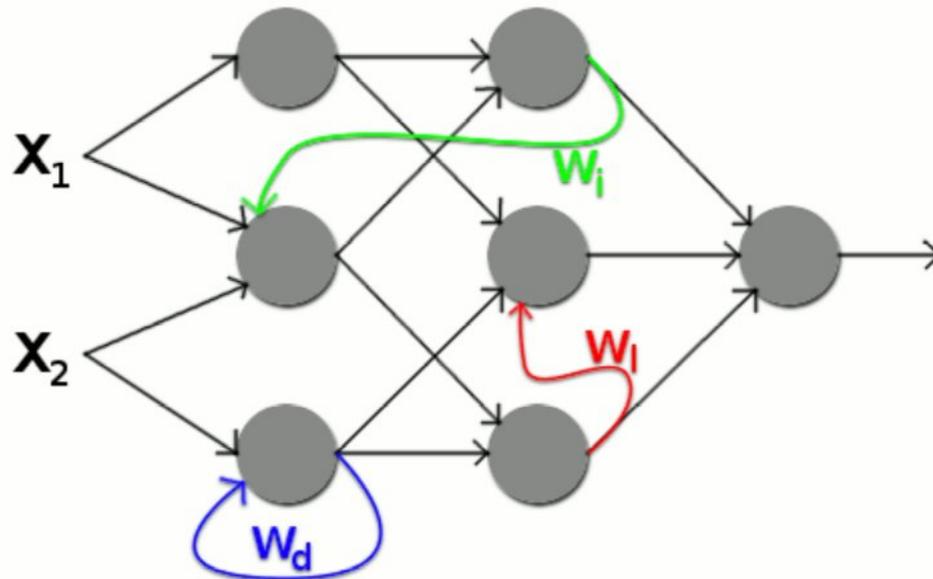
Fooling NN Access Control

- Face recognition with NN
 - Glasses → other person!



Types of Neural Networks - RNN

- Recurrent neural network (RNN)
 - direct feedback (blue)
 - indirect feedback (green)
 - lateral feedback (red)





Types of Neural Networks - RNN

- Recurrent neural network (RNN)
 - Time series processing
 - Long short-term memory (LSTM)
- Applications
 - Speech recognition (Siri, Alexa)
 - Action recognition in videos
- Tensorflow Keras RNN
 - <https://www.tensorflow.org/guide/keras/rnn>

Speech Recognition using Deep Learning

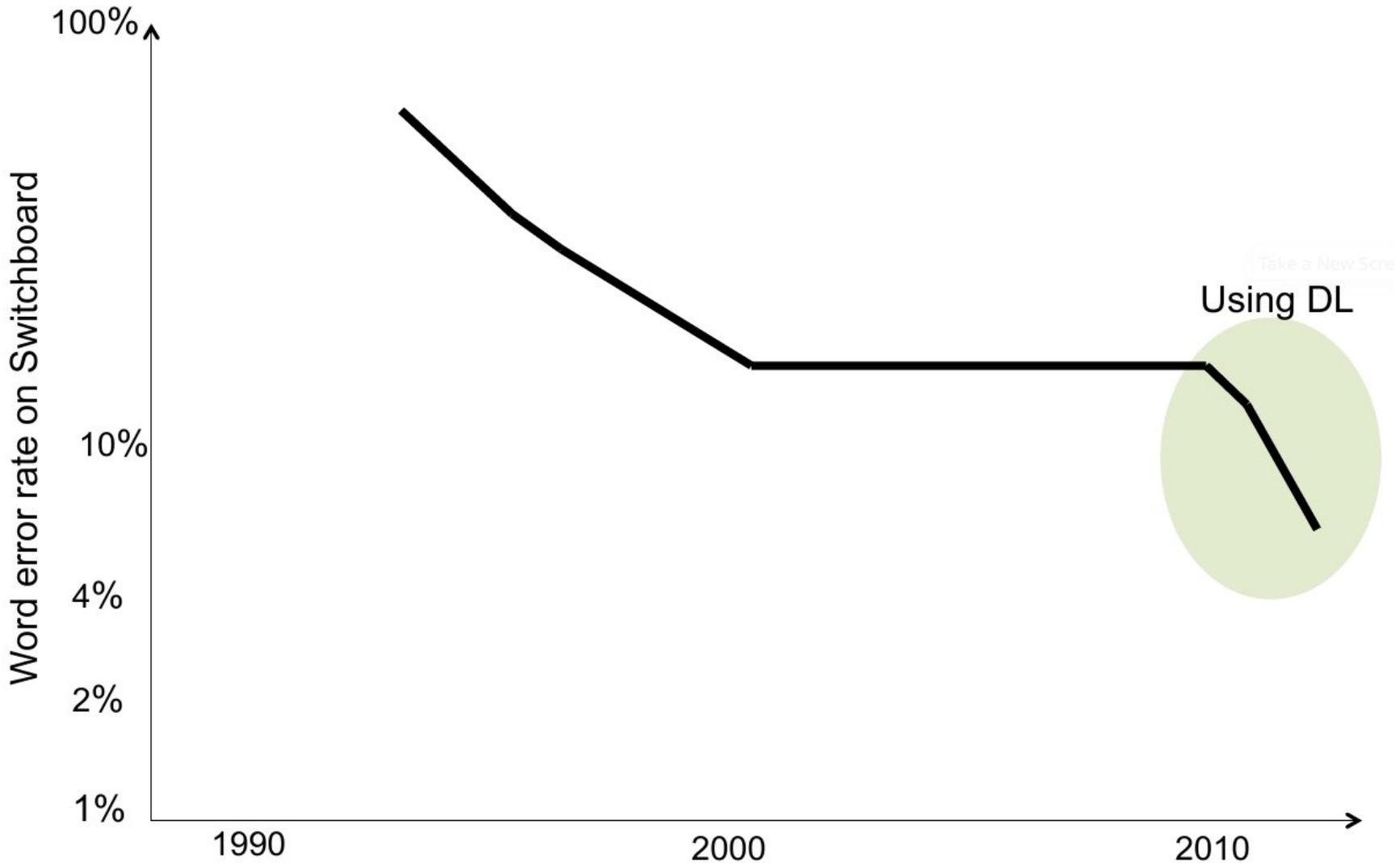


Image Classification

- Given an image \Rightarrow assign a label



This image by f8kita is licensed under CC-BY 2.0

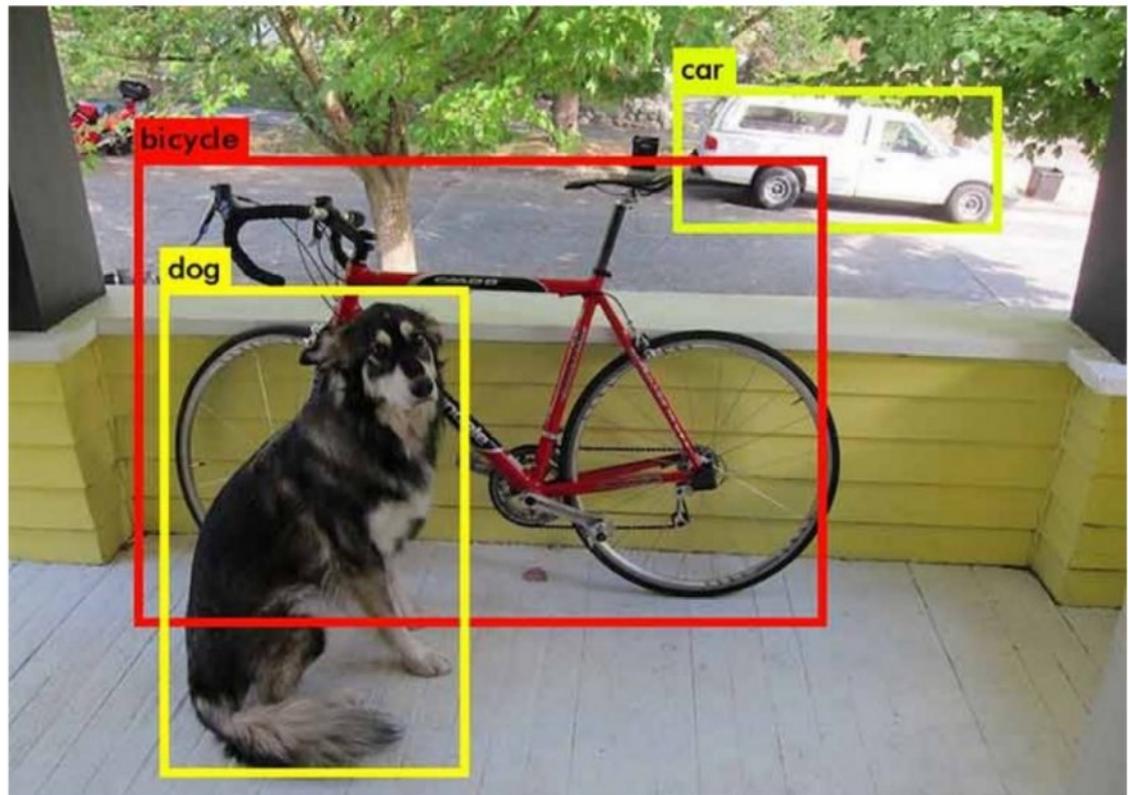
(assume given set of discrete labels)
{dog, cat, truck, plane, ...}



cat

Object Detection

- Multiple objects: The task is to classify and localize all the objects in the image.



Detection & Segmentation

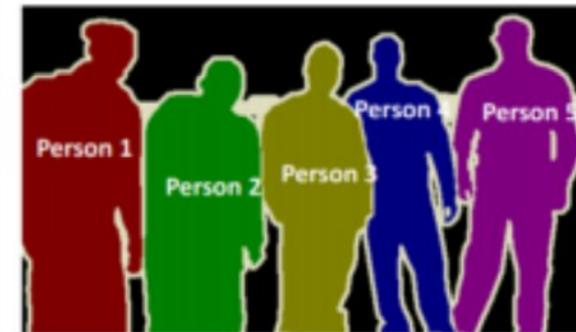
The goal of semantic image segmentation is to label each pixel of an image with a corresponding class of what is being represented.



Object Detection



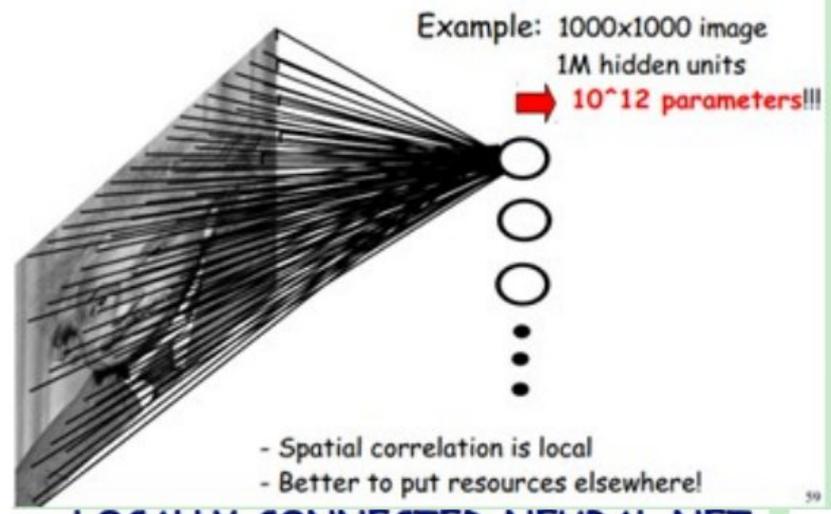
Semantic Segmentation



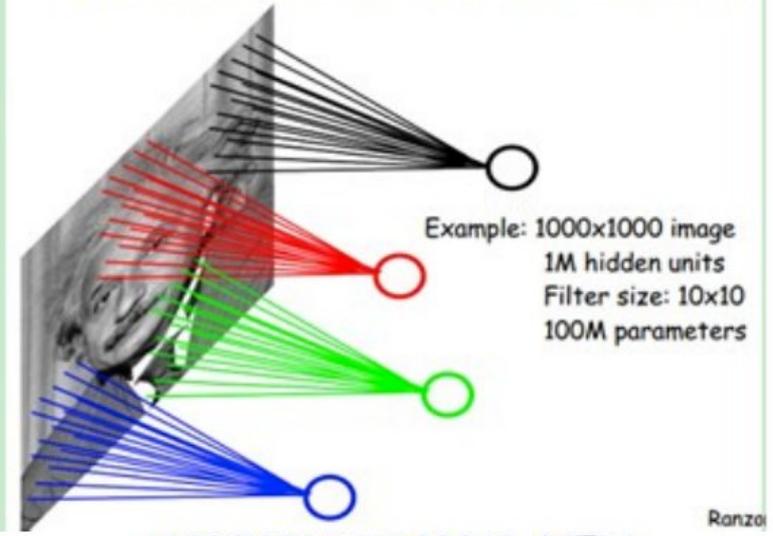
Instance Segmentation

Networks for Images

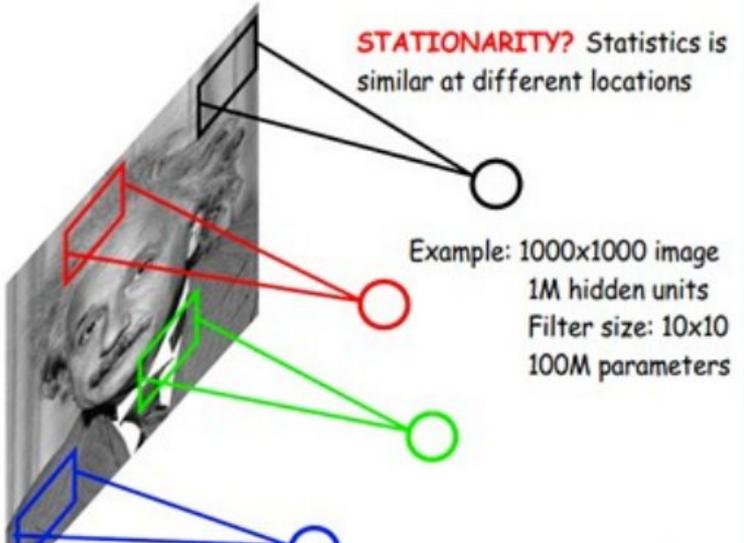
FULLY CONNECTED NEURAL NET



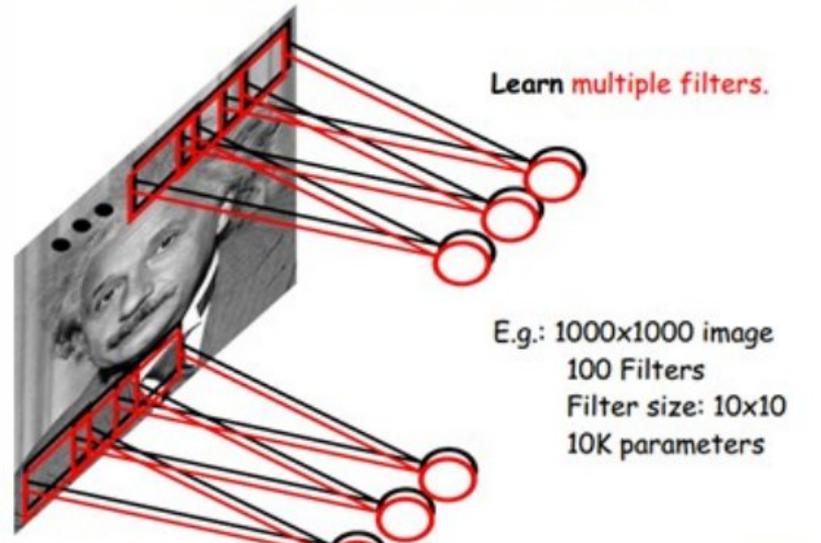
LOCALLY CONNECTED NEURAL NET



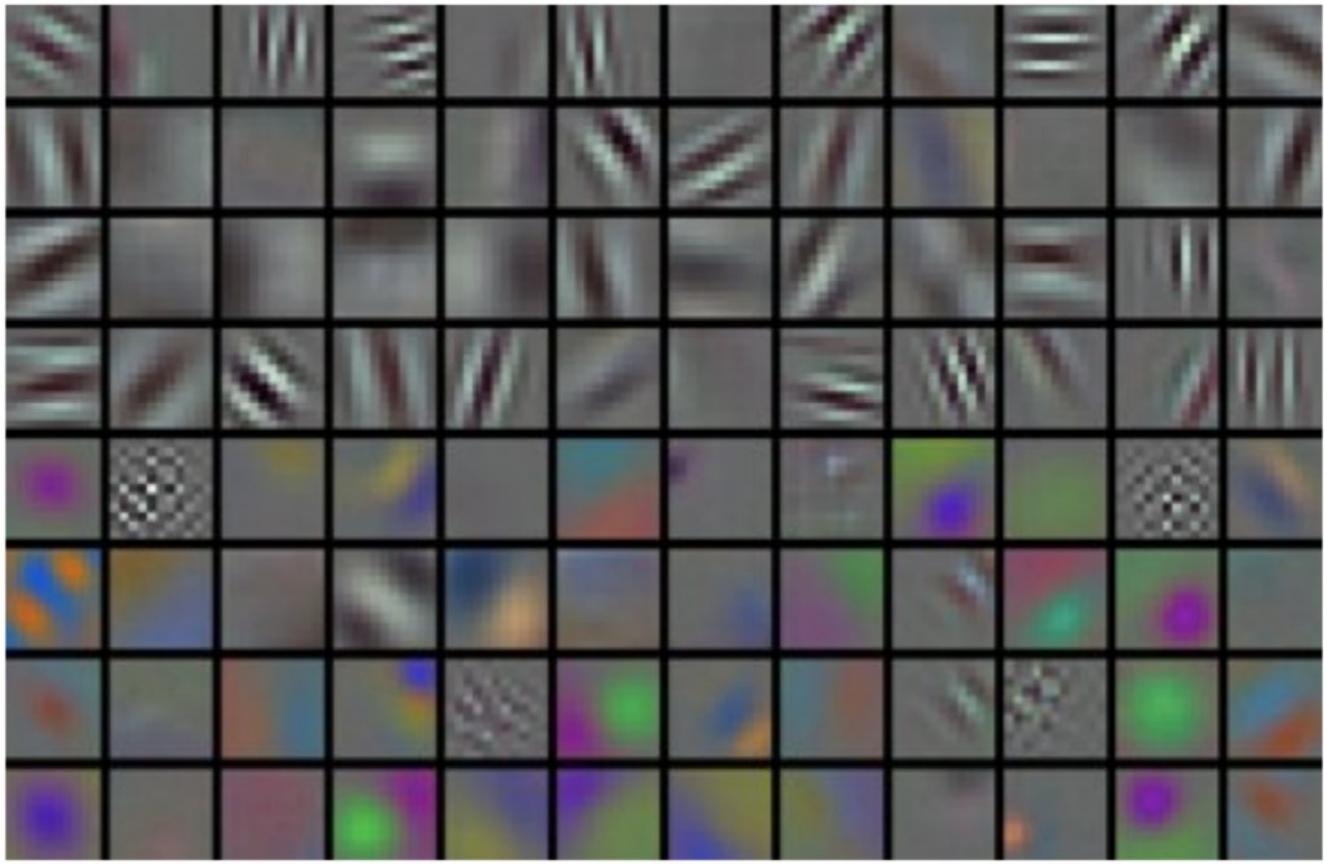
LOCALLY CONNECTED NEURAL NET



CONVOLUTIONAL NET



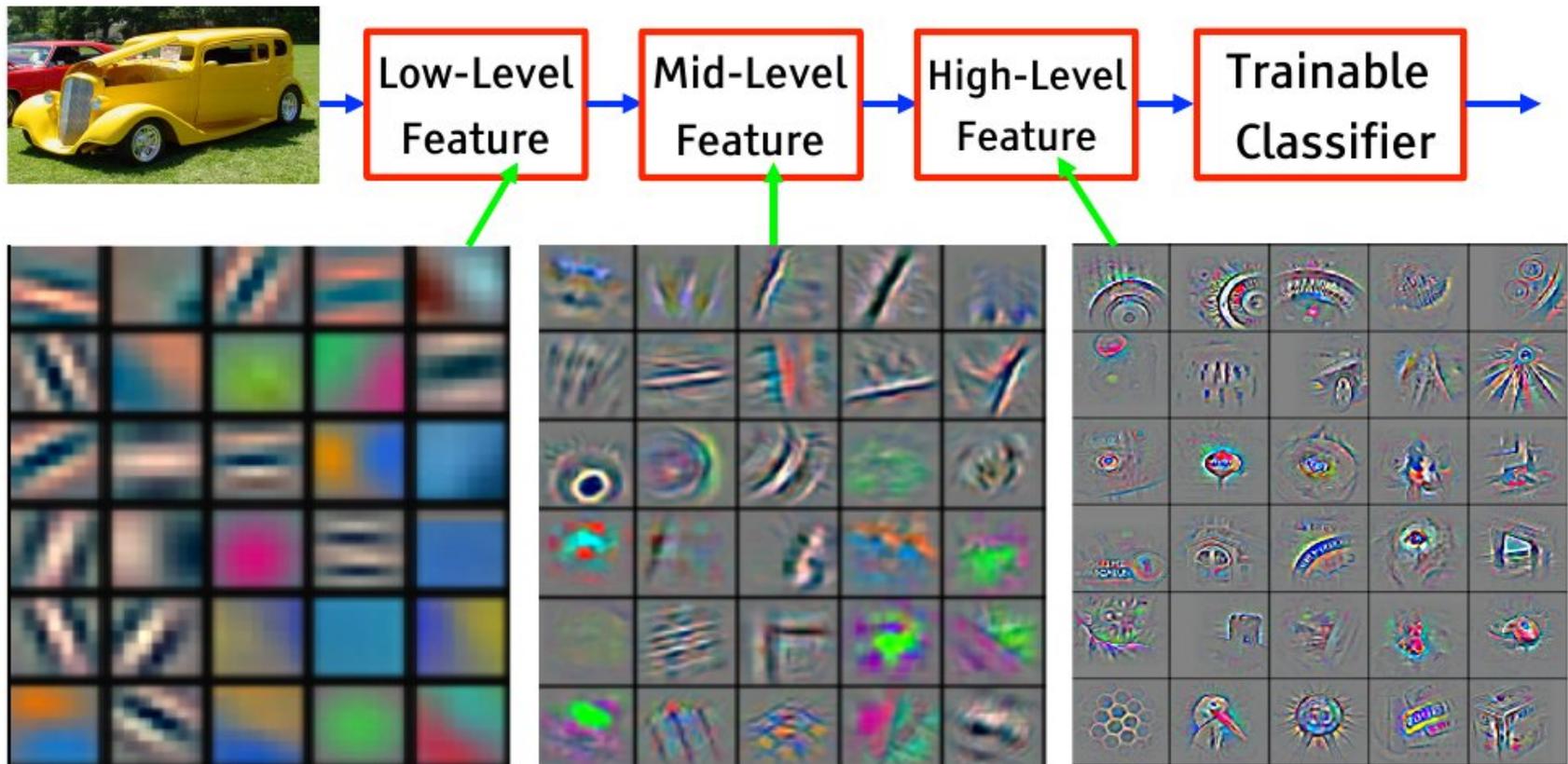
Learned Filters



Why Multiple Layers?

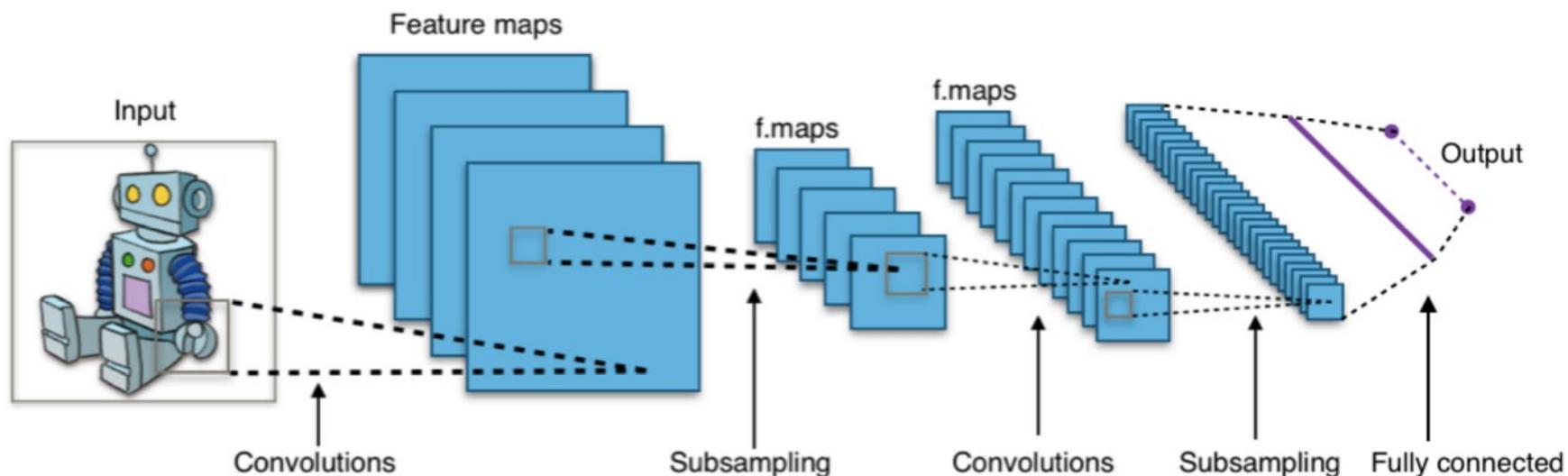
Layers represents different features!

Pixel => edge => texton => part => object



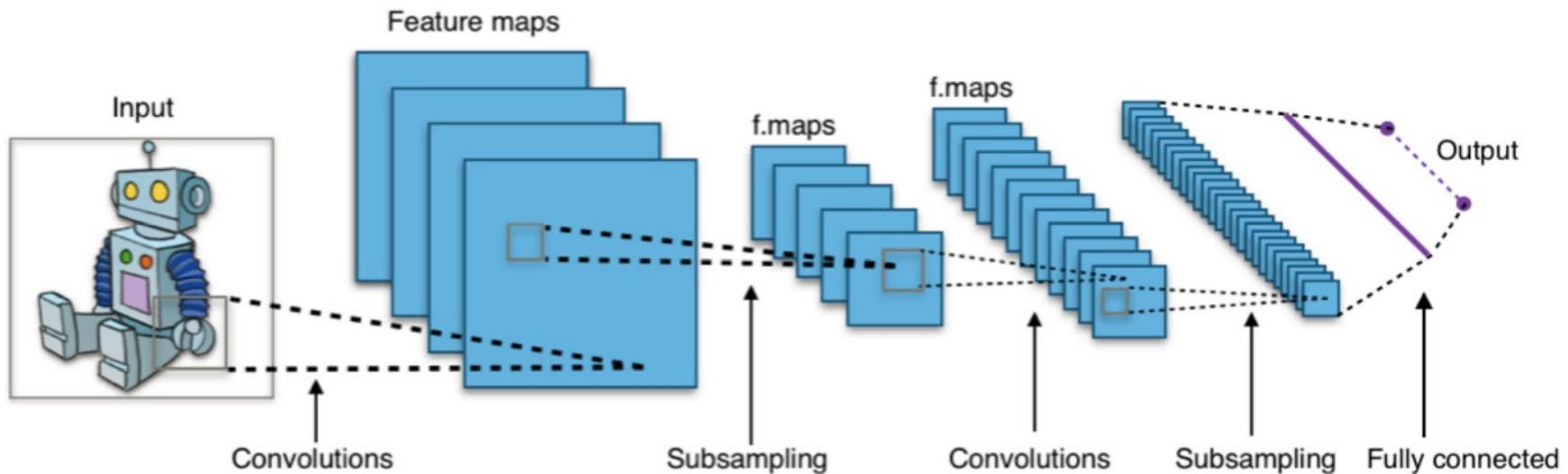
Types of Neural Networks - CNN

- Convolutional Neural Network
 - Local connectivity
 - Special arrangement



CNN Example

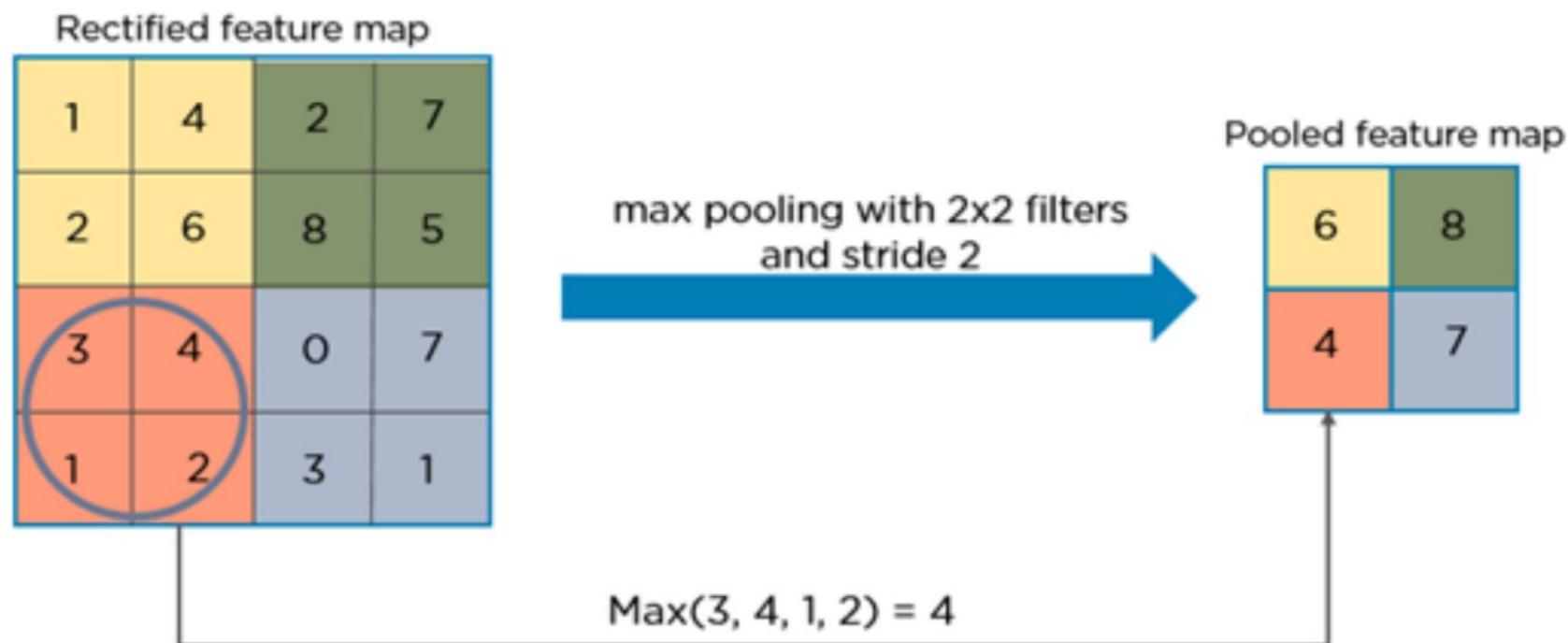
- Convolution Demo
 - <https://cs231n.github.io/convolutional-networks/>



Pooling Layer

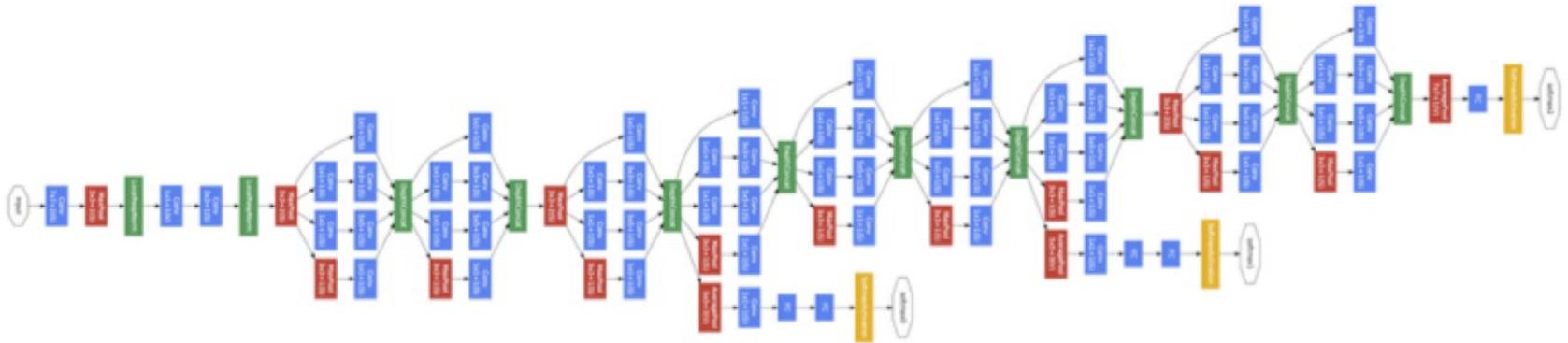
The function of pooling is to reduce the size of the feature map

=> fewer parameters in the network.



Deep learning Network GoogleNet

- Object recognition with 22 layers
- 1000 classes, error rate 6.7%



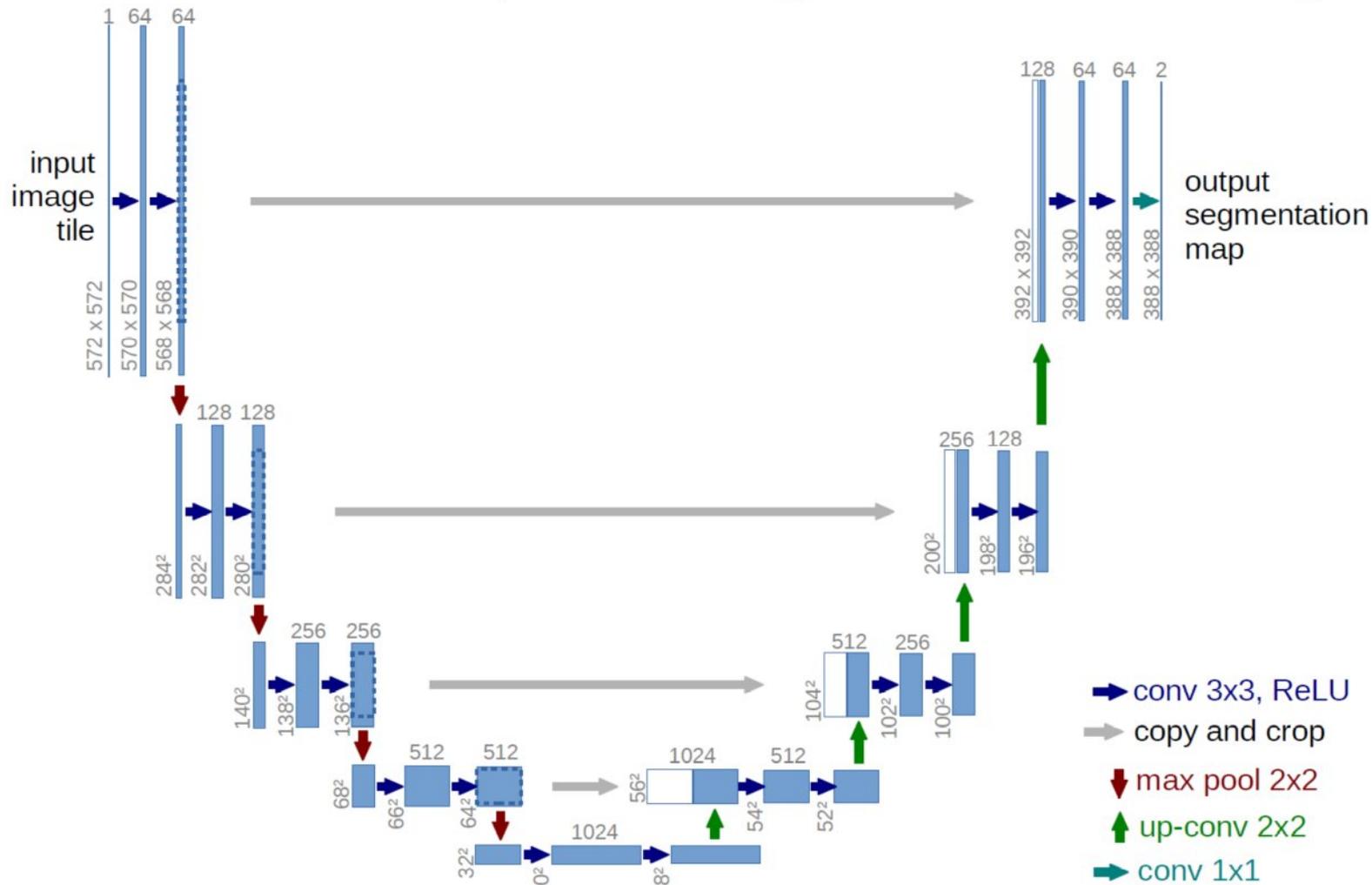
Convolution
Pooling
Softmax
Other

Deep learning Network VGGNet

- Object recognition with 16 layers
- 138 million weights
- 4 GPUs \sim 3 weeks learning time

U-Net: CNN for Biomedical Image Segmentation

U-Net for fast and precise segmentation of images!



Neural Network Toolboxes

- TensorFlow
 - Open Source
 - from Google
 - Google speech recognition
 - Google Photos
 - Python, C++, Java Interface
 - GPU and CPU support

