Classification algorithms

Logistic regression and neural networks

HAF Workshop, 22.3 – 23.3.2018, KIT

Martin Siggel

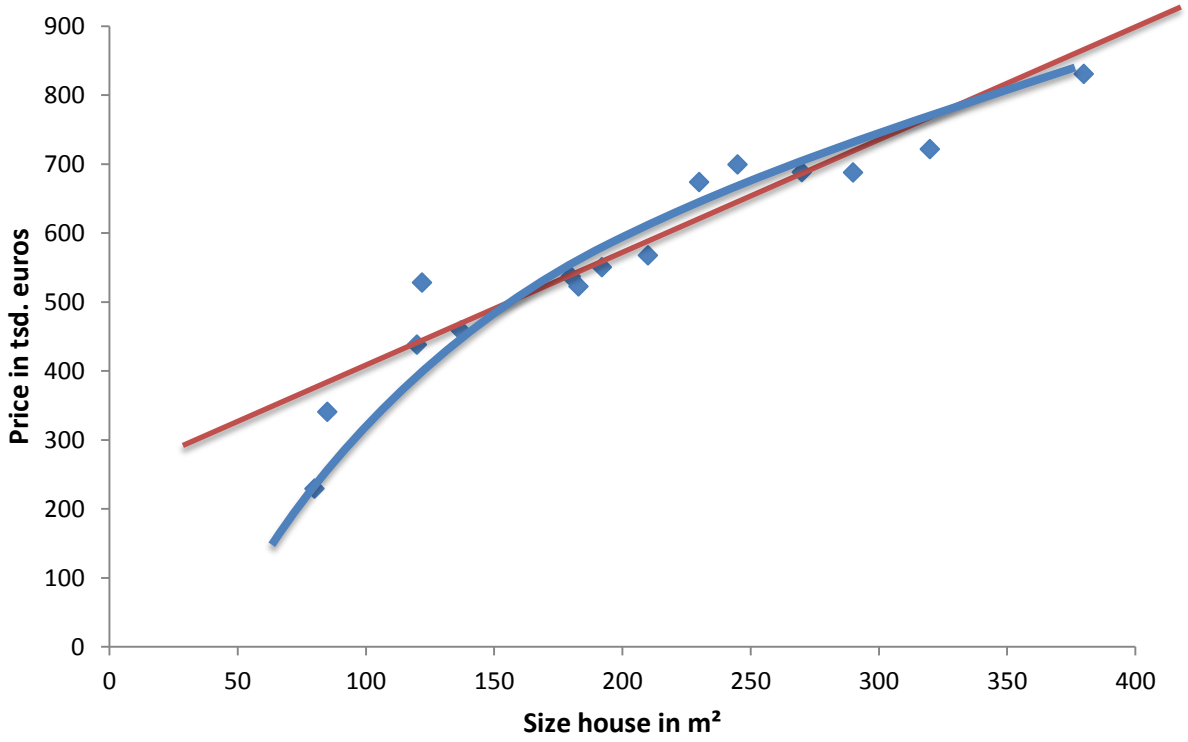
German Aerospace Center

Outline

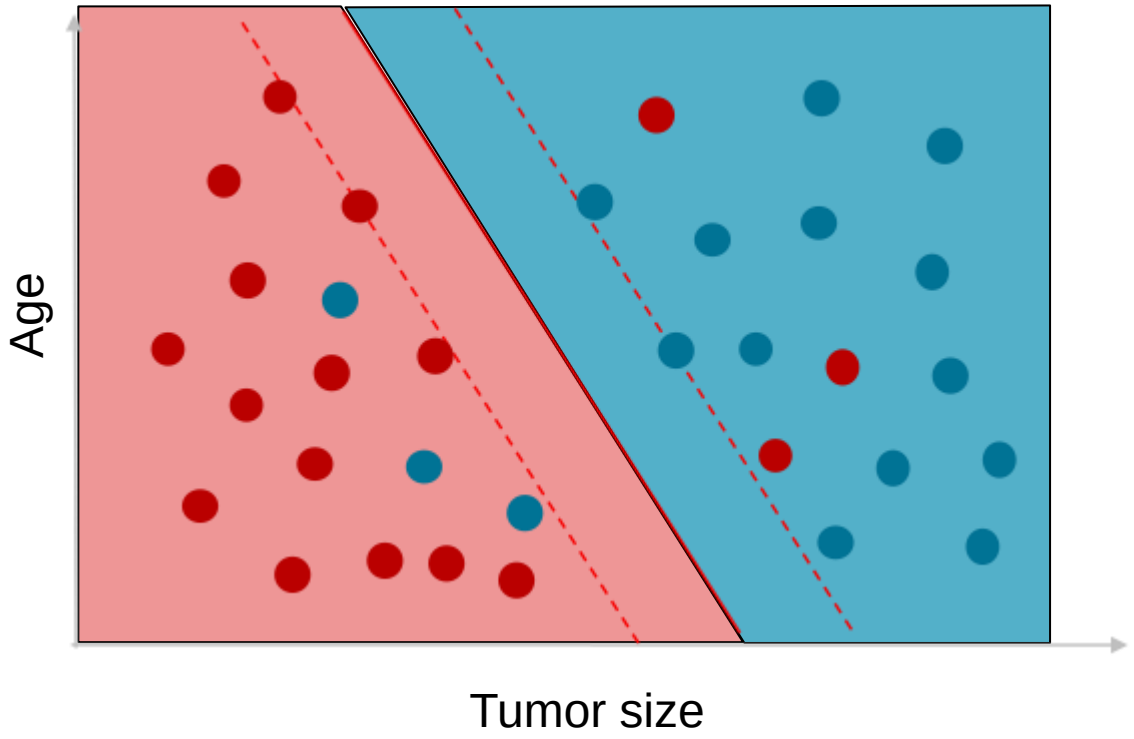
- Basics
  - Supervised learning: regression and classification
  - Math theory
  - Training
  - Optimization: gradient descent, gradient checking
  - Over- and underfitting, regularization
- Logistic Regression:
  - Linear regression vs logistic regression
  - Formalism
  - Cost vs. Loss
  - One-vs-All for multi-class classification
- Neural Networks
  - History
  - Neurons
  - Formalism
  - Loss + Back-propagation
  - Advanced topics

Basics

Big picture: supervised learning
Regression



Big picture: supervised learning
Classification



Classification = choose between discrete options

Big picture: supervised learning
Mathematical model

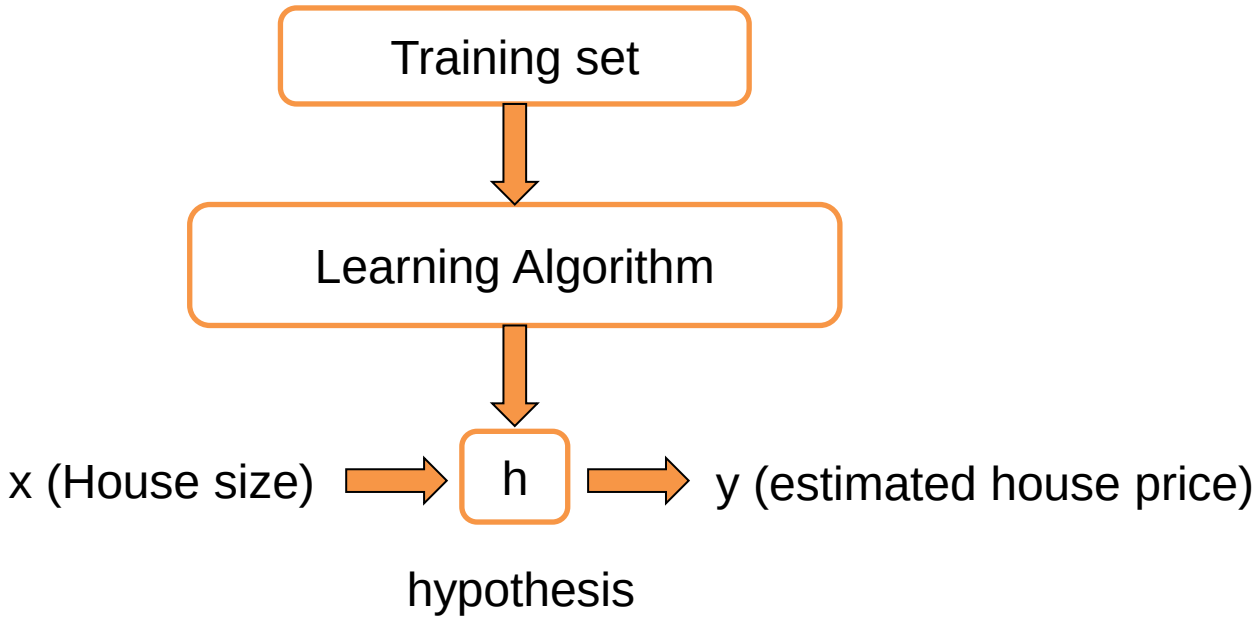
- Given a set of input data and correct answers
- Goal: find a model that fits to these data
- E.g. Given house size and number of rooms, predict house price

- Notation:
  - $x^{(i)} \in R^n$ is the i-th input for the algorithm (House size, room number)
  - $y^{(i)}$ is the ground truth answer (actual house price given features $x^{(i)}$)
  - m number of training samples
  - Training Set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

House size (x)	Price (y)
80	220
85	390
120	368
122	371
137	427
180	553
183	546
192	511

- Regression: Fit continuous data (house price)
- Classification: Predict for discrete classes (cancer, no cancer)

Big picture: supervised learning
The hypothesis



Training the hypothesis
Loss function

- The model $h(x)$ depends on some model parameters θ
- Training requires measure for quality of the model $h_\theta(x)$
- The loss function J compares predicted value $h_\theta(x^{(i)})$ with true value $y^{(i)}$, e.g.

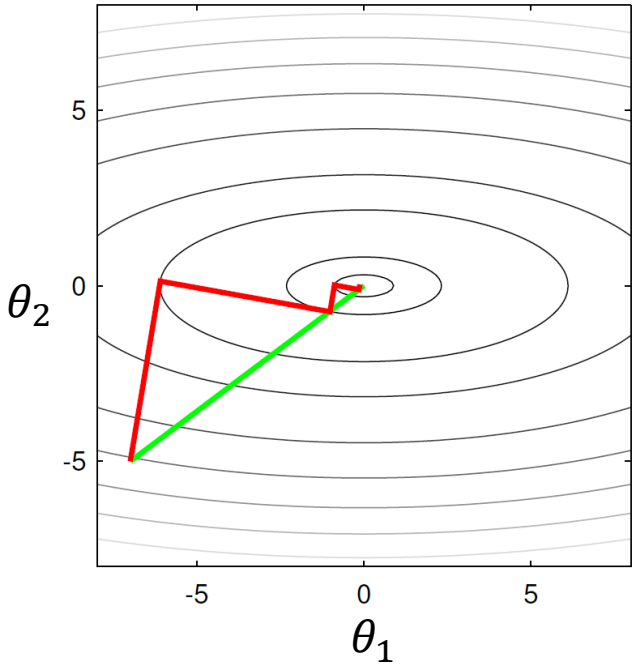
$$J = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

- Training = Find parameters θ such that loss function J is minimal

➢ Optimization problem

Training the hypothesis
Gradient descent optimization

- Mathematical formalism to minimize functions J(θ), $\theta \in R^n$
- Gradient based methods can handle millions of variables
- Iterative approach
  - Start with some initial guess θ_0
  - Iterate until convergence

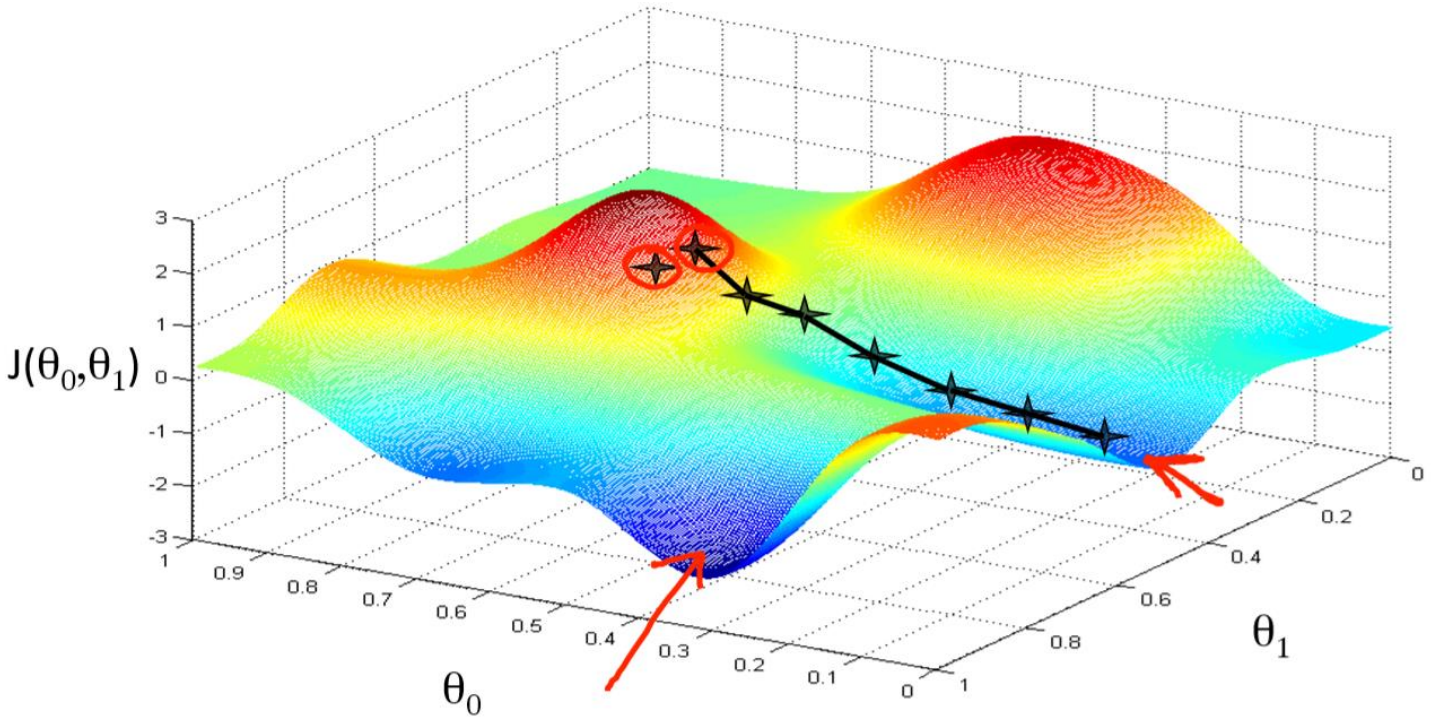


$$\theta_{i+1} = \theta_i - \alpha\vec{\nabla}J(\theta_i)$$

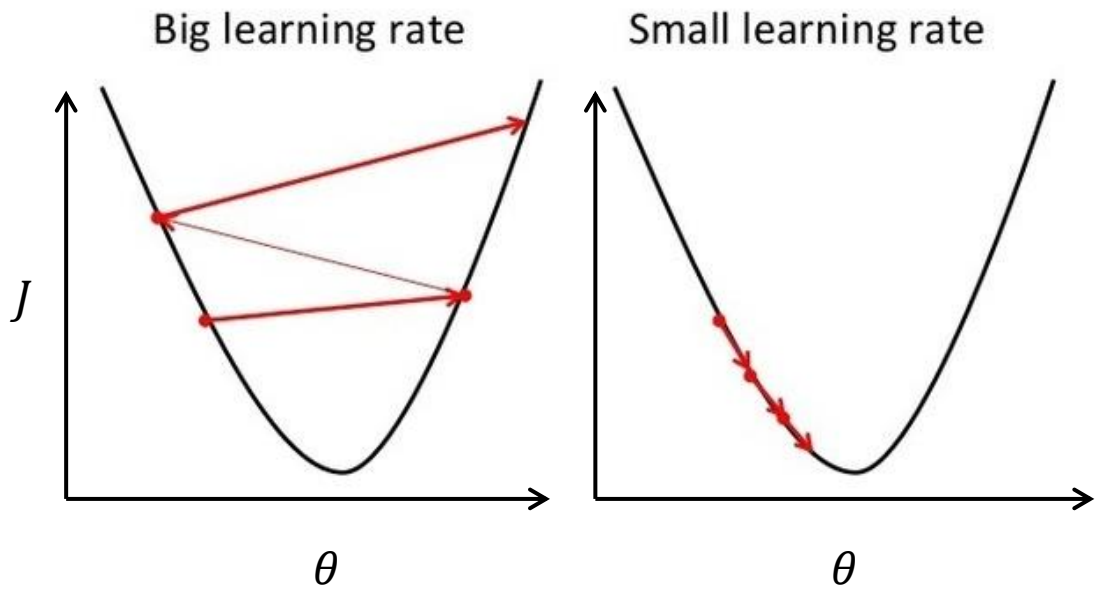
Learning Rate

Derivative of J w.r.t θ must be computed!

Training the hypothesis
Gradient descent optimization



Training the hypothesis
Gradient descent optimization



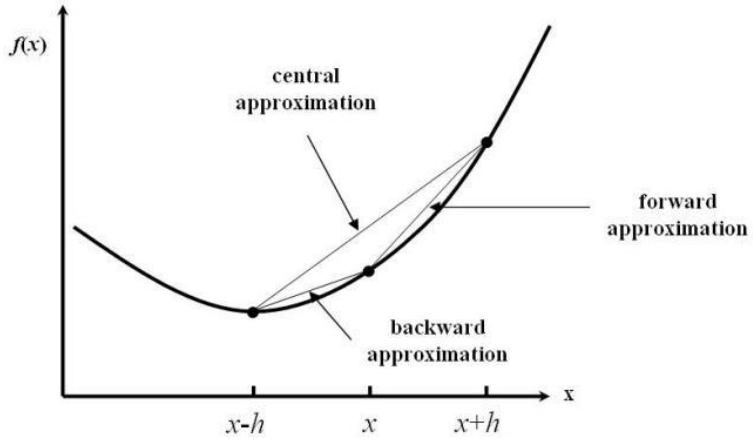
Training the hypothesis
Gradient Checking

- Implementing gradients $\frac{d}{d\theta}J(\theta)$ is difficult and error prone

- Simple (but slow) method, to check accuracy: Finite central differences

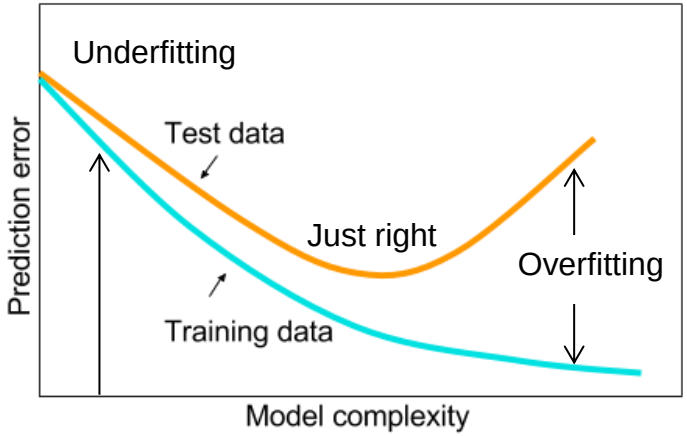
$$\frac{d\mathrm{J}}{d\theta_i}(\theta) \approx \frac{J(\theta + \vec{e}_i \cdot h) - J(\theta\ - \vec{e}_i \cdot h)}{2h}$$

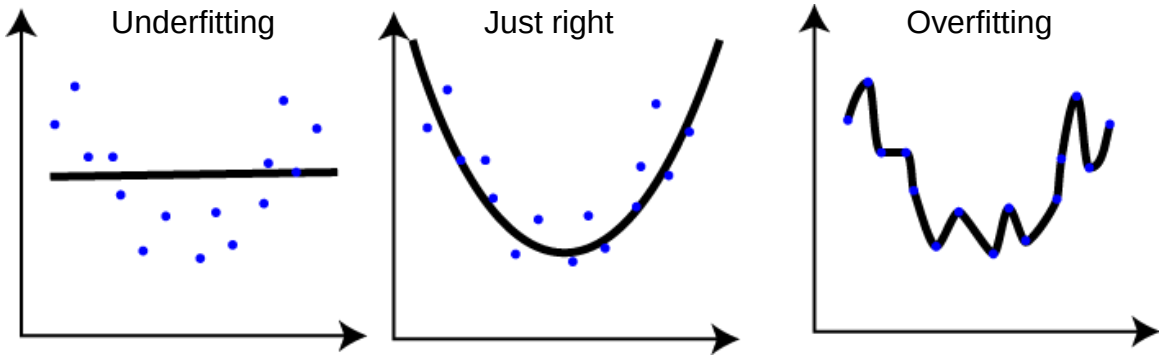
- Should always be done before training!



Training the hypothesis

Over- / underfitting





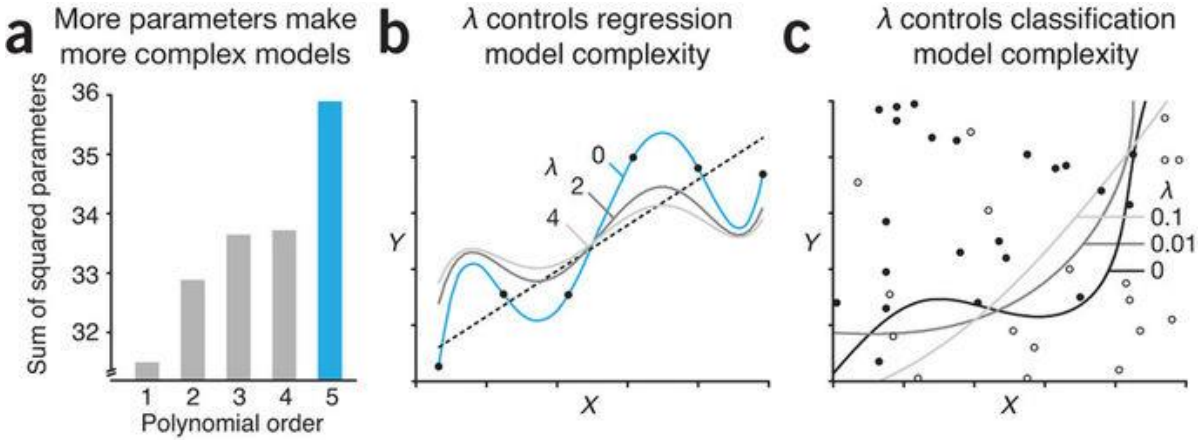
Training the hypothesis
Regularization

- Regularization tries to avoid overfitting by penalizing parameters

- Technically, add a regularization term to the loss function

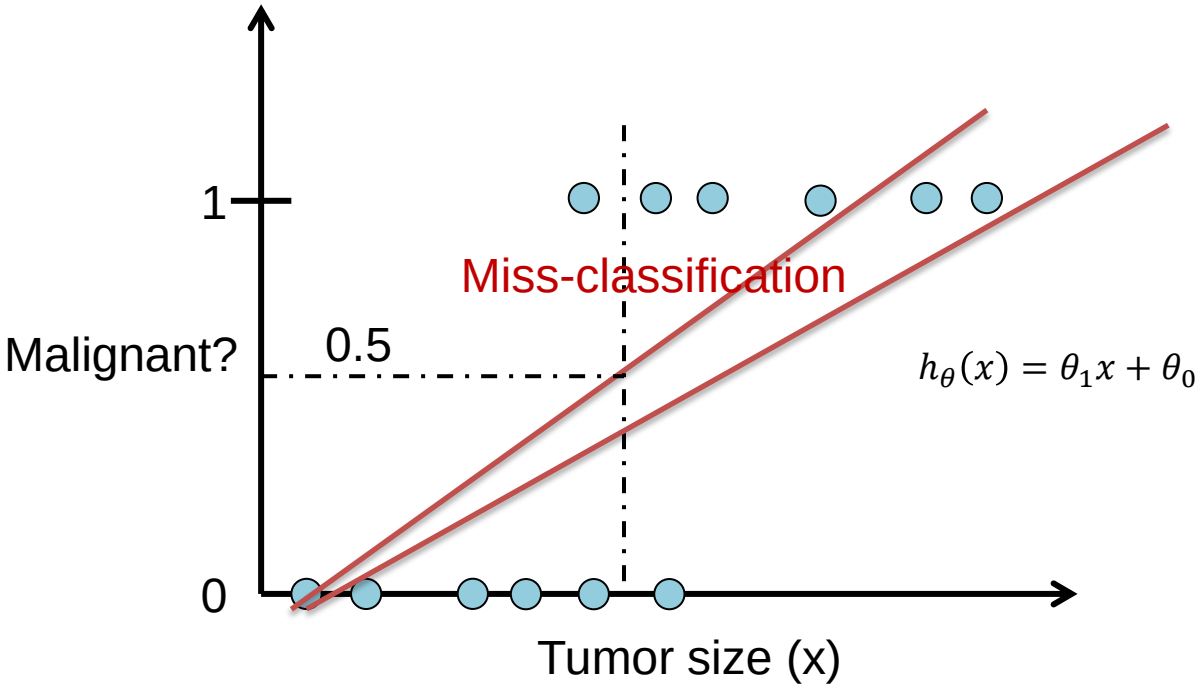
$$J = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j$$

- Note: The bias term θ_0 should not be included!



Logistic Regression

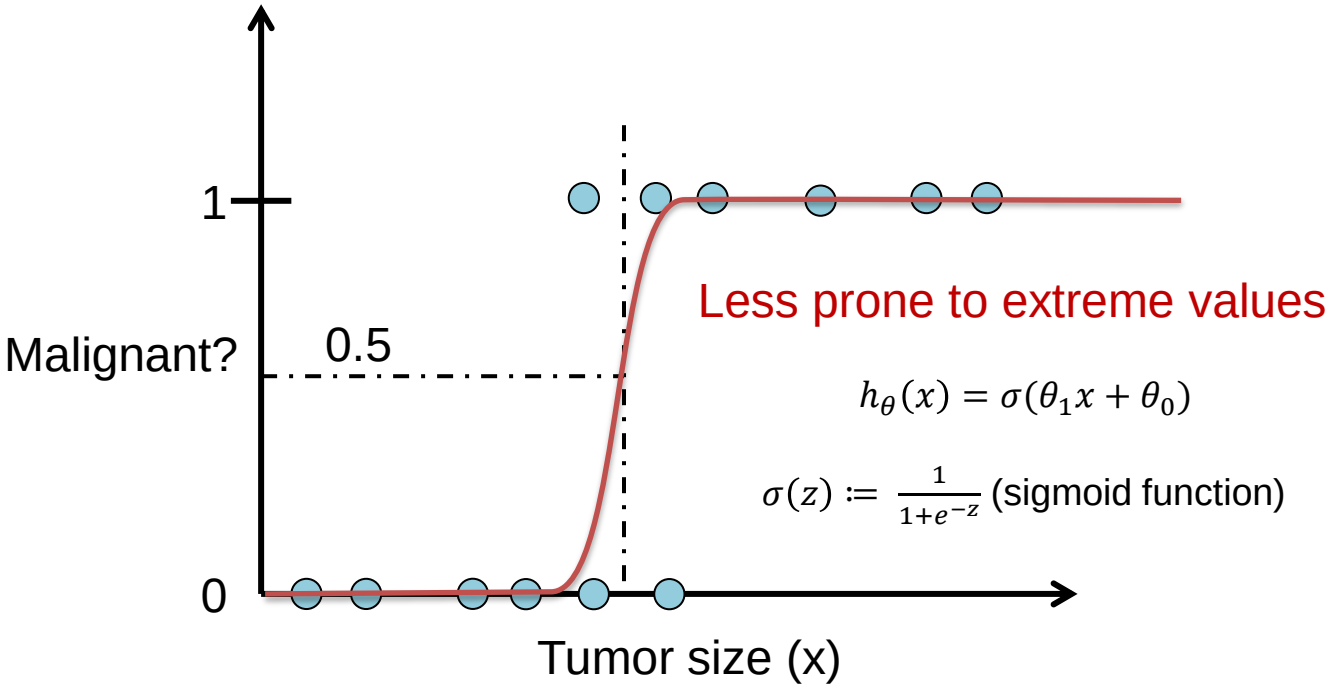
Simple linear regression



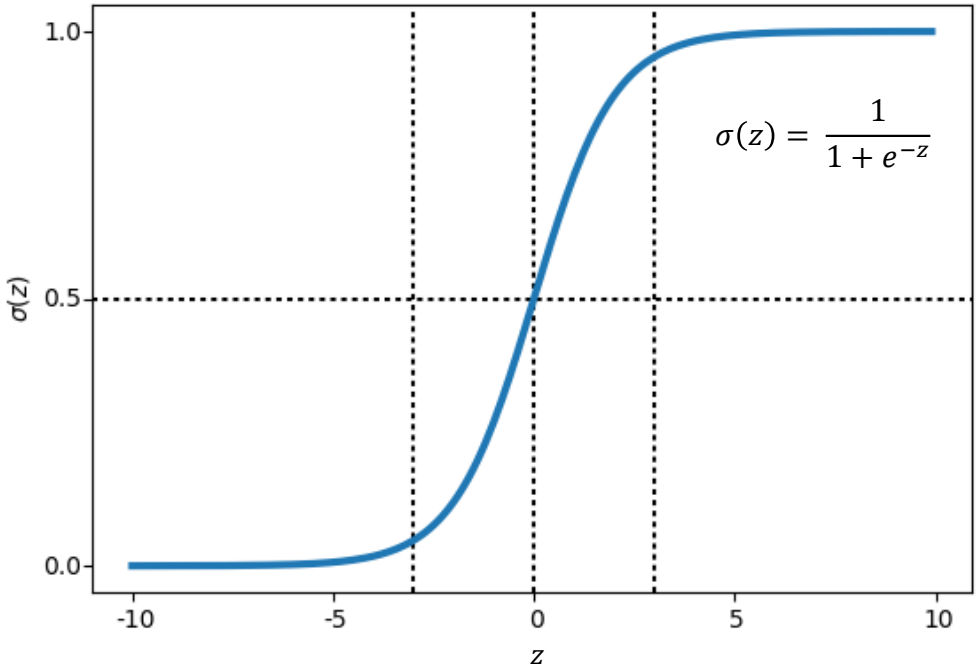
Classify:

- Malignant, if $h(x) \geq 0.5$
- Benign, if $h(x) < 0.5$

Logistic regression



Sigmoid function
(Logistic curve)



→ Measure of classification probabilty

Logistic Regression Formalism

- Let $x \in R^n$ be the n-dim input feature vector, then

$$h_\theta(x) = \sigma(\sum_{j=1}^{n} \theta_j x_j + \theta_0)$$

- Constant bias trick:
  - Define $x_0 = 1$
  - Then $h_\theta(x) = \sigma(\sum_{j=0}^{n} \theta_j x_j) = \sigma(\theta^T x)$

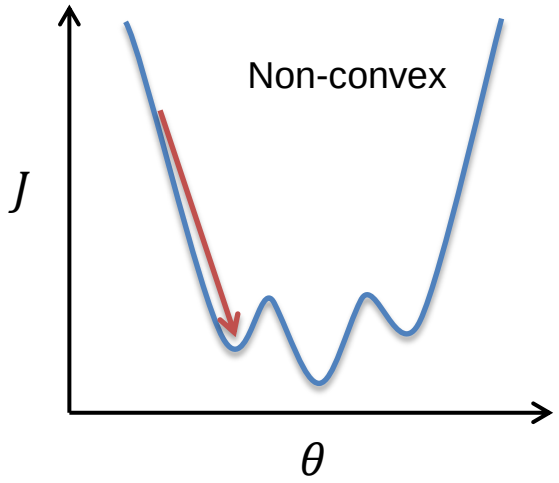
$$\boxed{h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}}$$

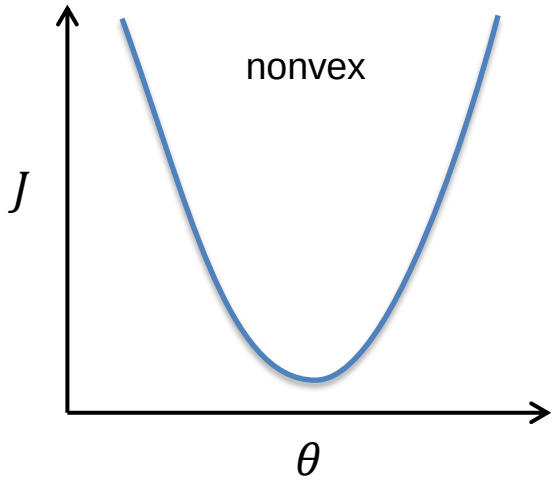
- Can be used for binary classification problems

Cost function

- Problem: standard mean squared loss is not convex in LR!

$$J = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{1}{2m}\sum_{i=1}^{m}\mathrm{Cost}(h_\theta\left(x^{(i)}\right), y^{(i)})$$



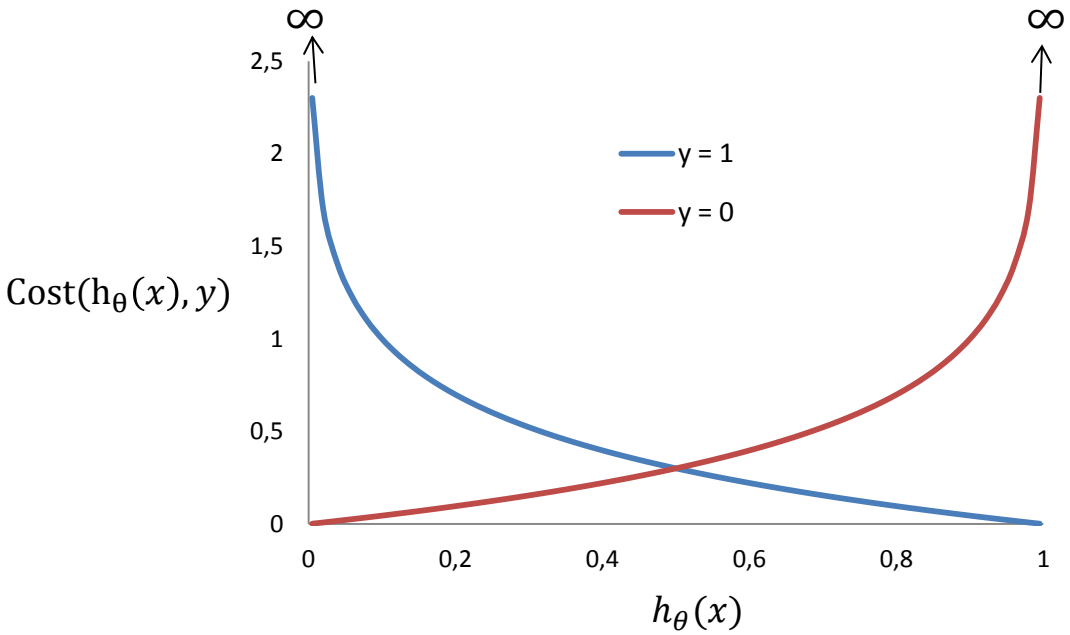


Desired

- Optimizers get stuck in local minima

Cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), \text{if } y = 1 \\ -\log(1 - h_\theta(x)), \text{if } y = 0 \end{cases}$$



Simplified Cost Function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), \text{if } y = 1 \\ -\log(1 - h_\theta(x)), \text{if } y = 0 \end{cases}$$

- Can be simpified / unified to:

$$\text{Cost}(h_\theta(x), y) = -(1 - y) \cdot \log(1 - h_\theta(x)) - y \cdot \log(h_\theta(x))$$

y=0 y=1

Loss function regularized Logistic Regression

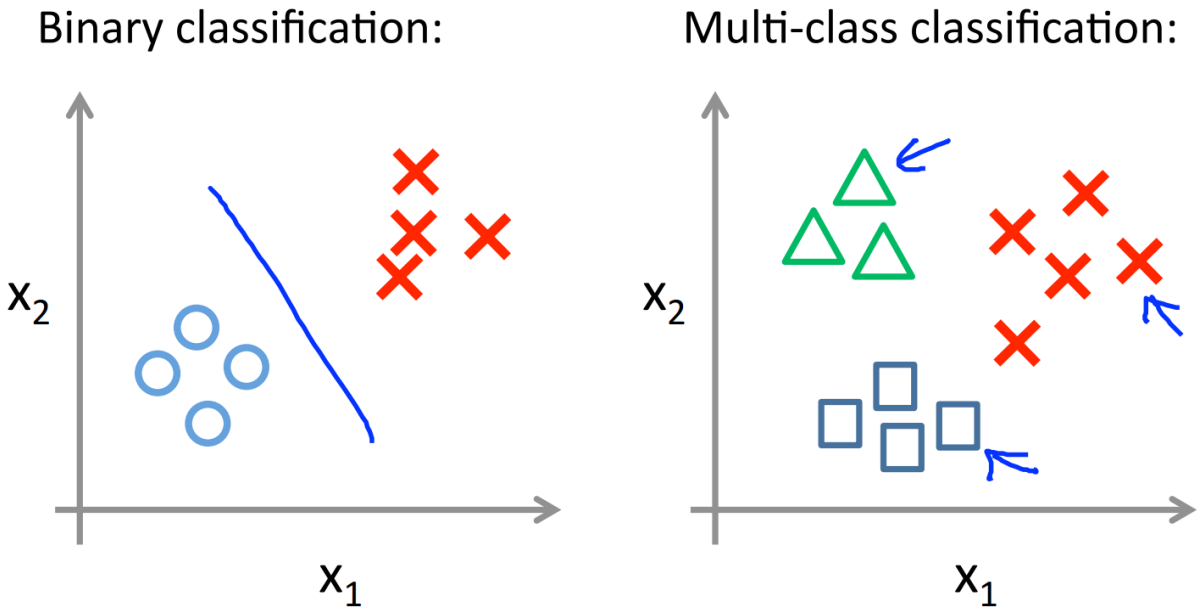
The complete loss function of logistic regression + regularization is:

$$J(\theta) \coloneqq -\frac{1}{m}\sum_{i=1}^{m} y^i \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\theta\left(x^{(i)}\right)\right) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

We can compute the derivatives as follows:

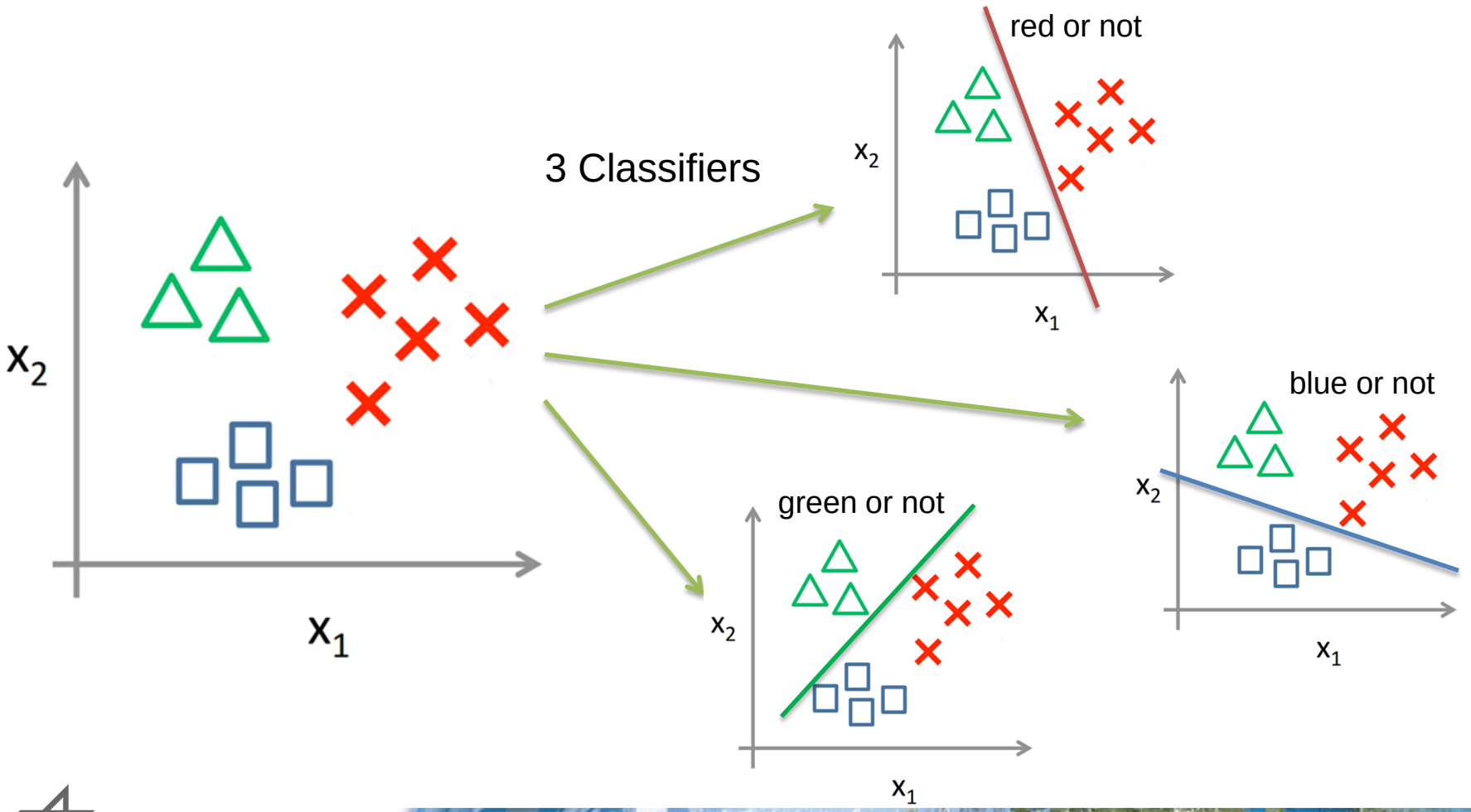
$$\frac{d}{d\theta_j}J(\theta) = \frac{1}{m}\sum_{i}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)\cdot x_j^{(i)} + \frac{\lambda}{m}\theta_j$$

Multi-class classification with Logistic Regression



Multi-class classification with Logistic Regression

One-vs-all



Multi-class classification with Logistic Regression
One-vs-all

- Train a logistic regression classifier $h_\theta^i(x)$ for each class i

- Predict class i which has the highest probability $h_\theta^i(x)$, i.e.

$$i = \text{argmax}_\text{i} \left(h_\theta^i(x) \right)$$

Excercise

- Classify hand-written digits (MNIST)

- TODO:
  1. Get a feeling for the data
  2. Implement and check the loss function of the LR method
  3. Classify images from a test data set and compute accuracy of our method.
  4. Figure out, how test set accuracy and training set accuracy depend on the number of samples.

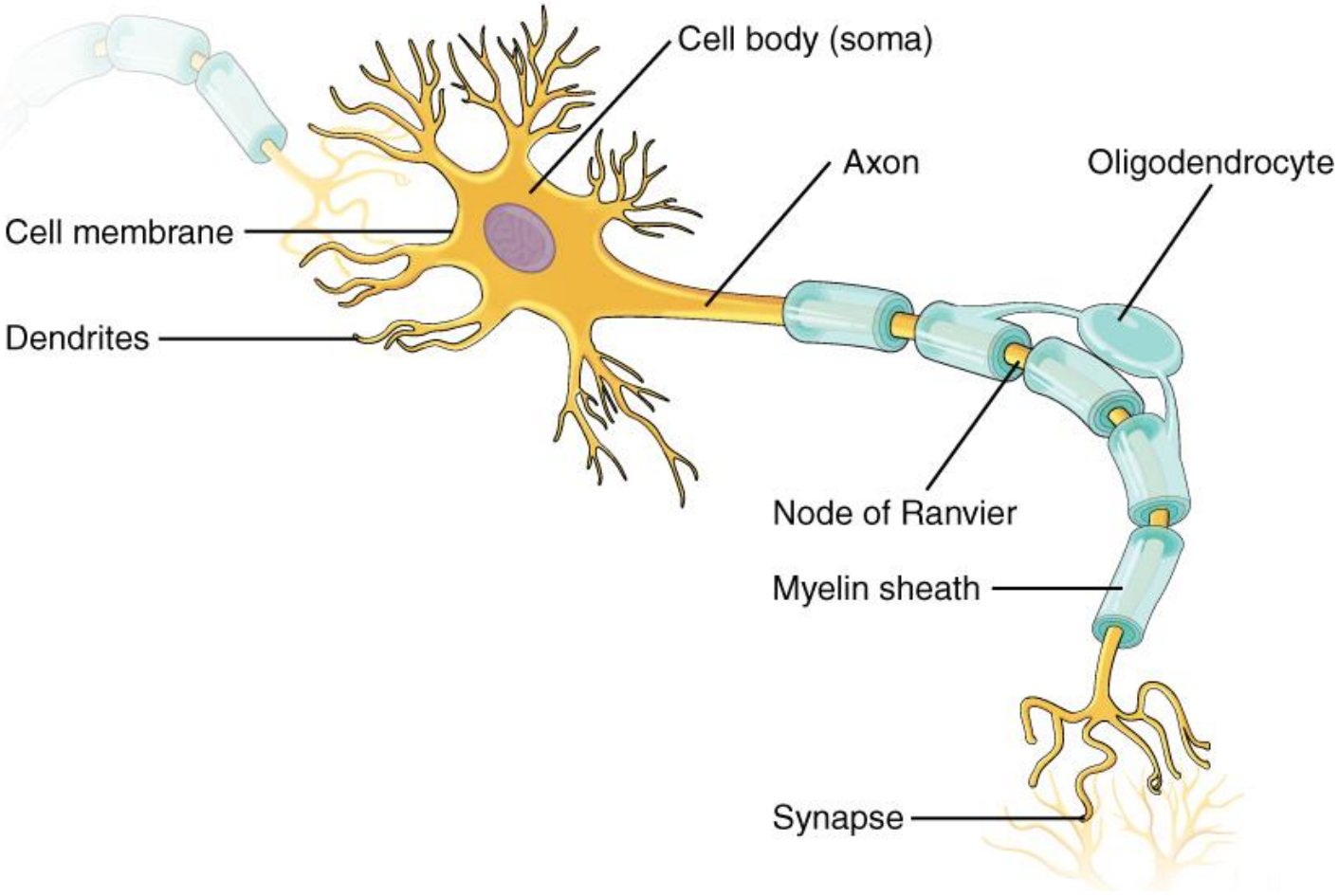
- 30 min

Neural Networks

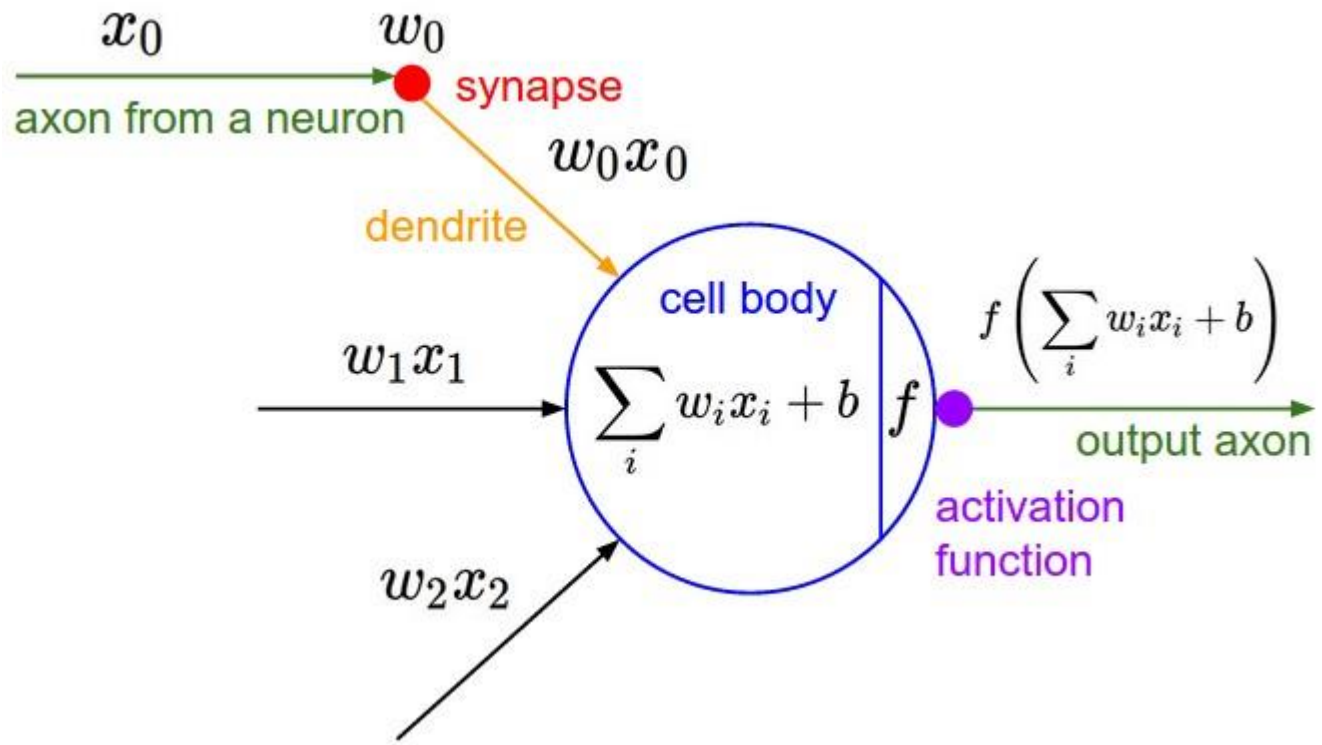
History

- Originally invented to mimic brain
- Hyped due to many successful applications in the 80s / early 90s
  - Recognition of hand-written text (Yan LeCun)
- Winter of neural networks: networks were not as good as hoped
- Lately revival of large scale deep neural networks due to better hardware, better network architectures

The Neuron

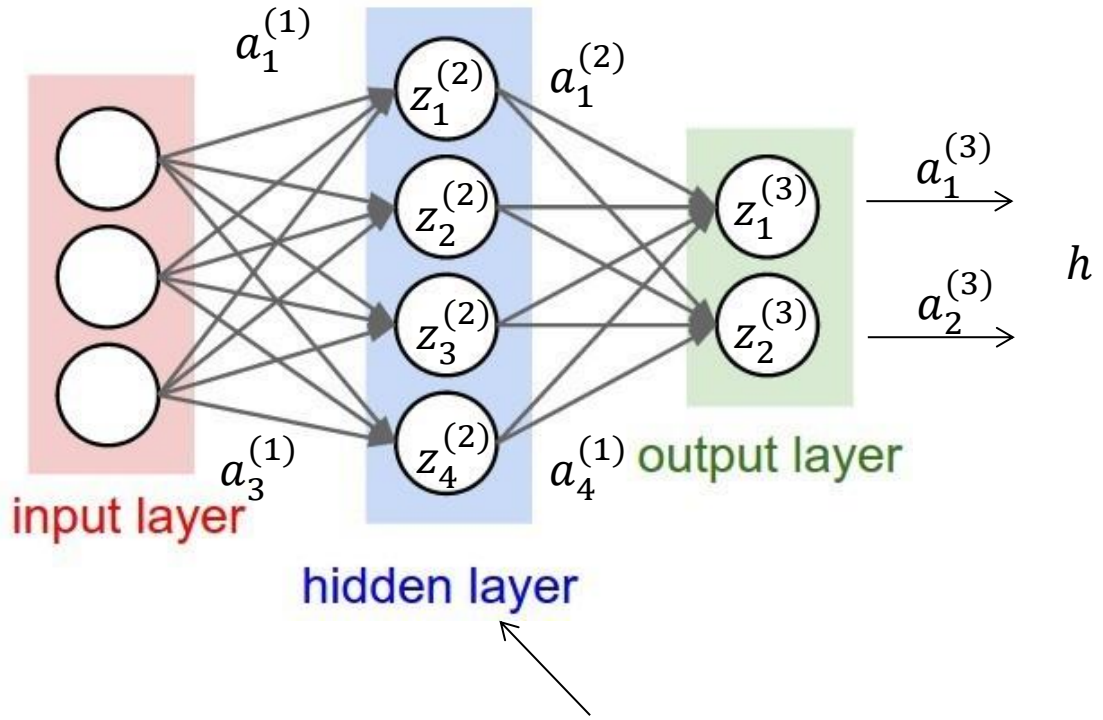

Cell body (soma)
Axon
Oligodendrocyte
Cell membrane
Dendrites
Node of Ranvier
Myelin sheath
Synapse

The Neuron
Mathematical analogon



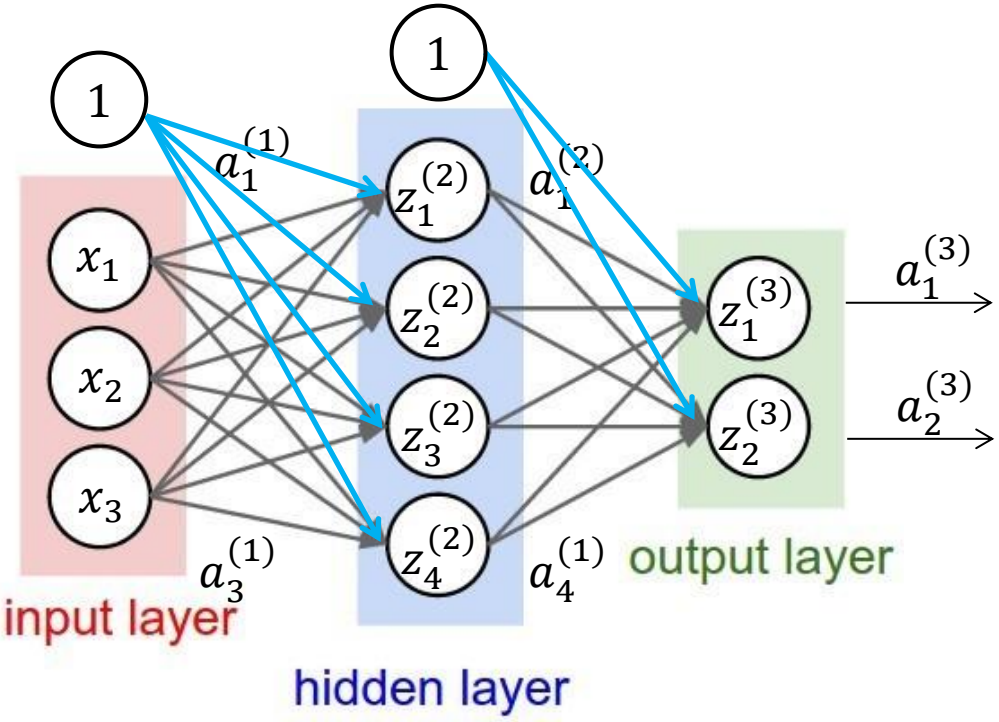
1 Neuron = Logistic Regressor

Neural Network



Learns abstract features

Neural Network



h

$$a_i^{(1)} = x_i$$

$$z_i^{(l+1)} = \sum_{j=1}^{n_l} \Theta_{ij}^{(l)} a_j^{(l)} + \theta_{i0}^{(l)}$$

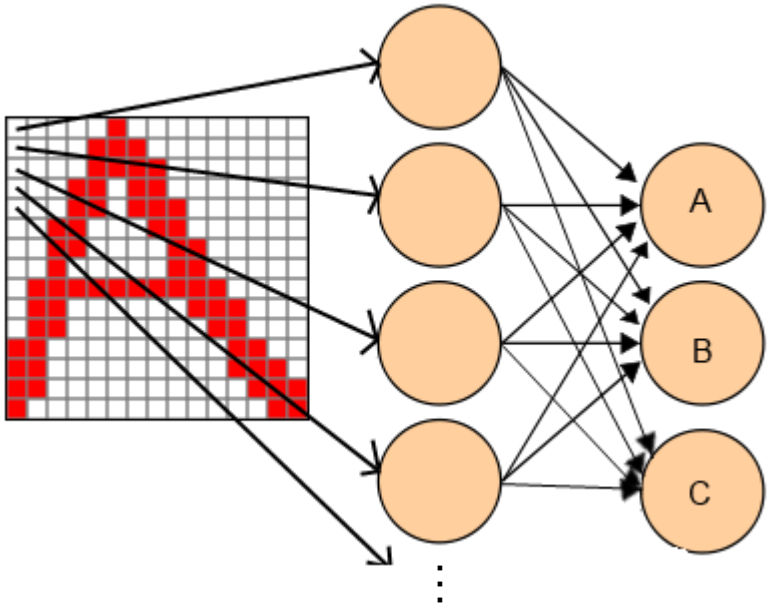
$$a_i^{l+1} = \sigma\left(z_i^{(l+1)}\right)$$

$$\Rightarrow a^{(l+1)} = \sigma\left(\Theta^{(l)} a^{(l)}\right)$$

Sigmoid function $\sigma(z) = {}^{1}/_{1+e^{-z}}$

Multi-Class classification

- Neural network has K output neurons
- Each output neuron represents a different class e.g.

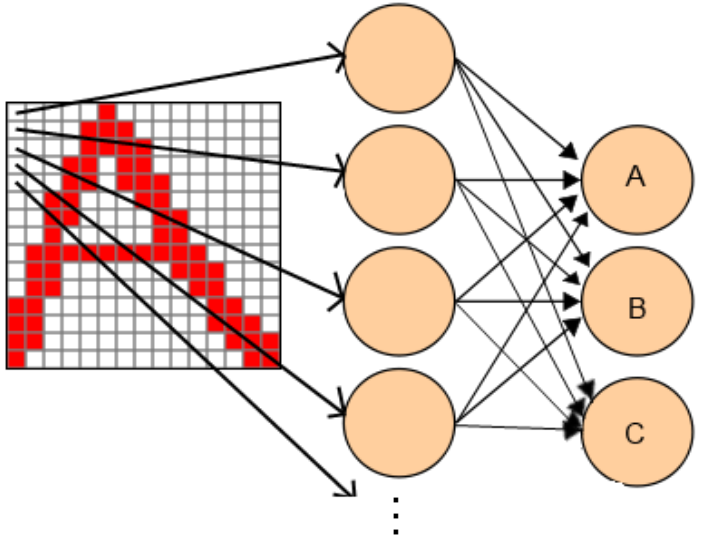


- The neuron that fires most (has highest probabilty) wins

Multi-Class classification
One-hot encoding

- The true values have to be encoded to match output topology, i.e.

$$A \triangleq \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad B \triangleq \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad C \triangleq \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$



- This encoding is called one-hot

Training
Loss function

- Loss function similar to logistic regress, but with K outputs

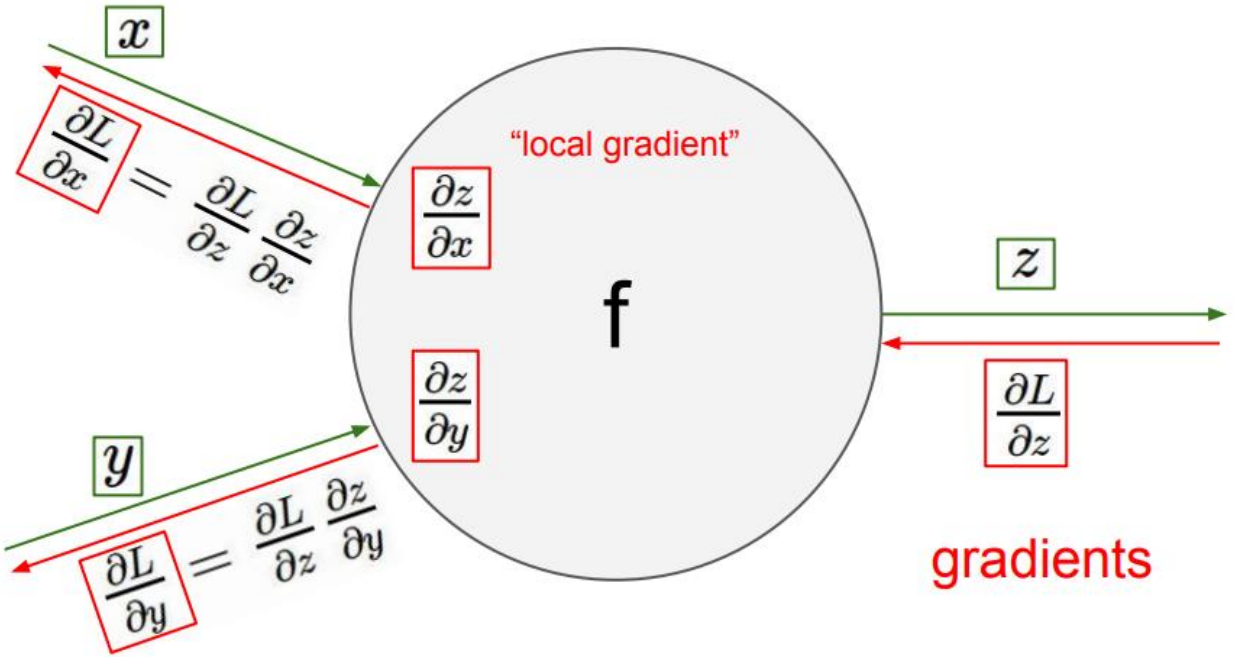
$$J(\theta) = -\frac{1}{m}\sum_{i}^{m}\sum_{k}^{K}\left[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{k=1}^{n_l}\sum_{j=1}^{n_{l+1}}\Theta_{jk}^{(l)}$$

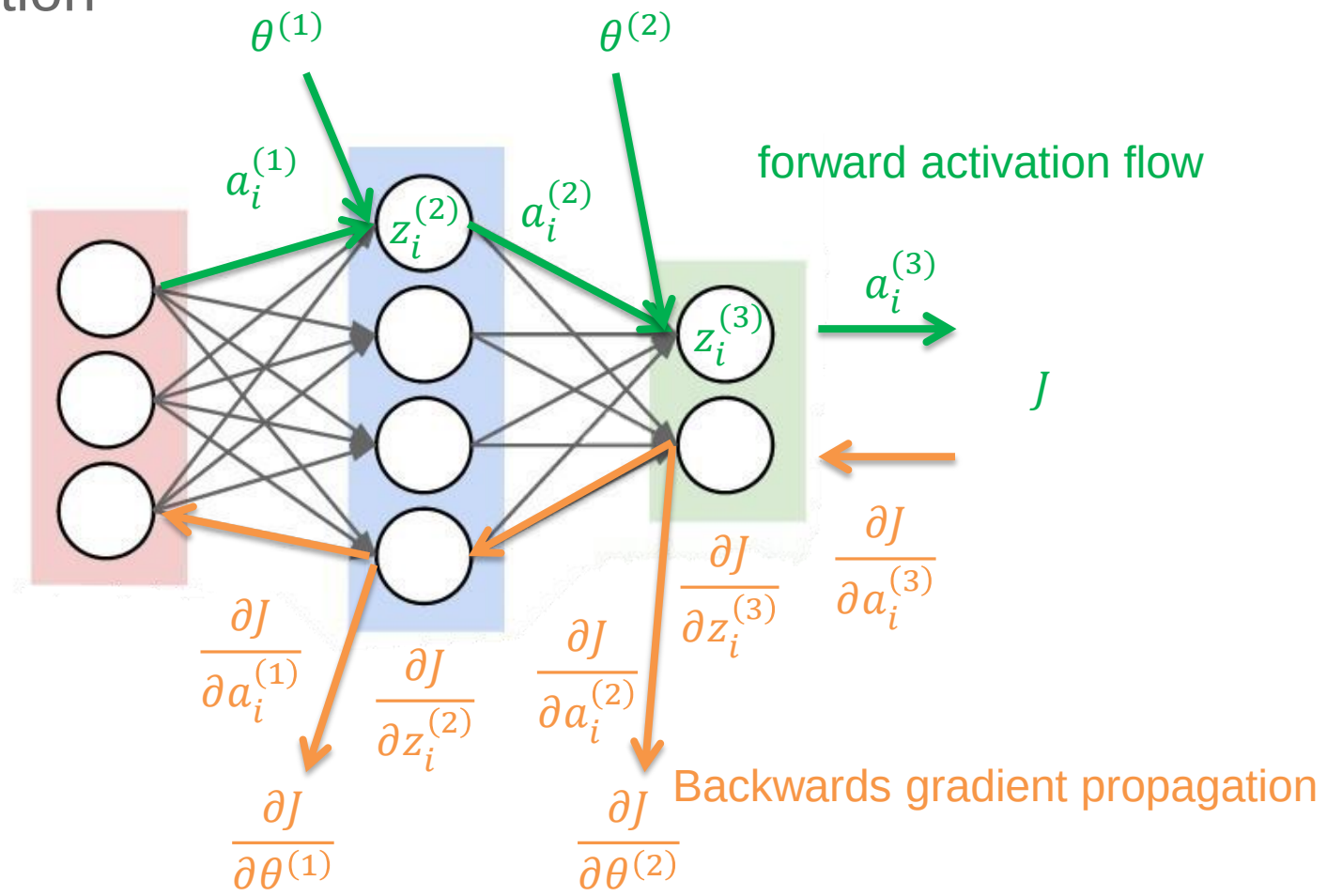
Sum over all parameters, except bias

Training
Back-propagation

Application of chain rule:

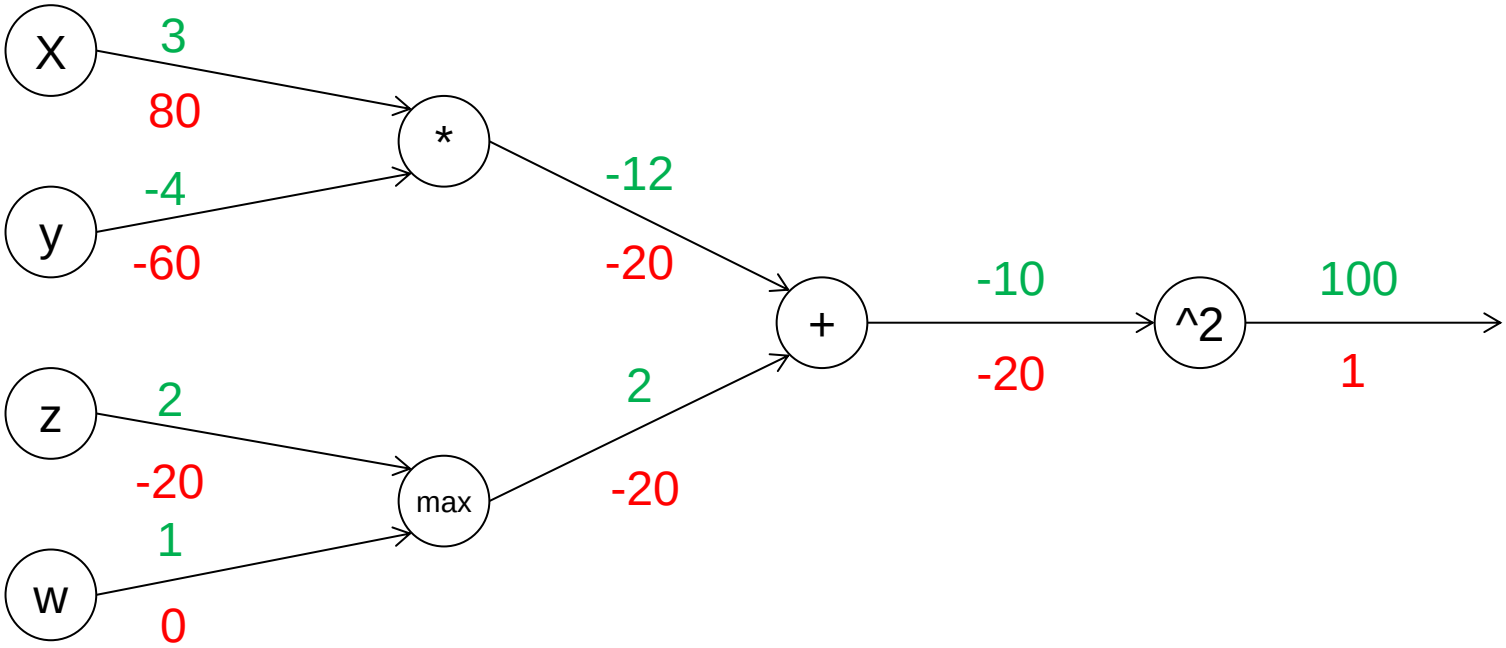


Training
Back-propagation



Training
Back-propagation, Example

- Back-propagate: $f(3,-4,2,1)$ for $f(x,y,z,w) = (xy + \max(z,w))^2$



Training
Back-propagation, Layers

- Nomenclature: $dx_\dagger := \frac{dJ}{dx}$

- $dz_\dagger^{(3)} = \frac{1}{m}(h - y)$ (Error of the output layer)
- $d\theta_\dagger^{(2)} = a^{(2)^T} dz_\dagger^{(3)}$ (Gradient of θ_2)
- $da_\dagger^{(2)} = \theta^{(2)} dz_\dagger^{(3)^T}$
- $dz_\dagger^{(2)} = da_\dagger^{(2)} \frac{d}{dz}\sigma(z^{(2)})$ (Error of the hidden layer)
- $d\theta_\dagger^{(1)} = a^{(1)^T} dz_\dagger^{(2)}$ (Gradient of θ_1)

Just believe me ;)

Add regularization part

- $d\theta_\dagger^{(1)} = d\theta_\dagger^{(1)} + \frac{\lambda}{m}\theta^{(1)}$
- $d\theta_\dagger^{(2)} = d\theta_\dagger^{(2)} + \frac{\lambda}{m}\theta^{(2)}$

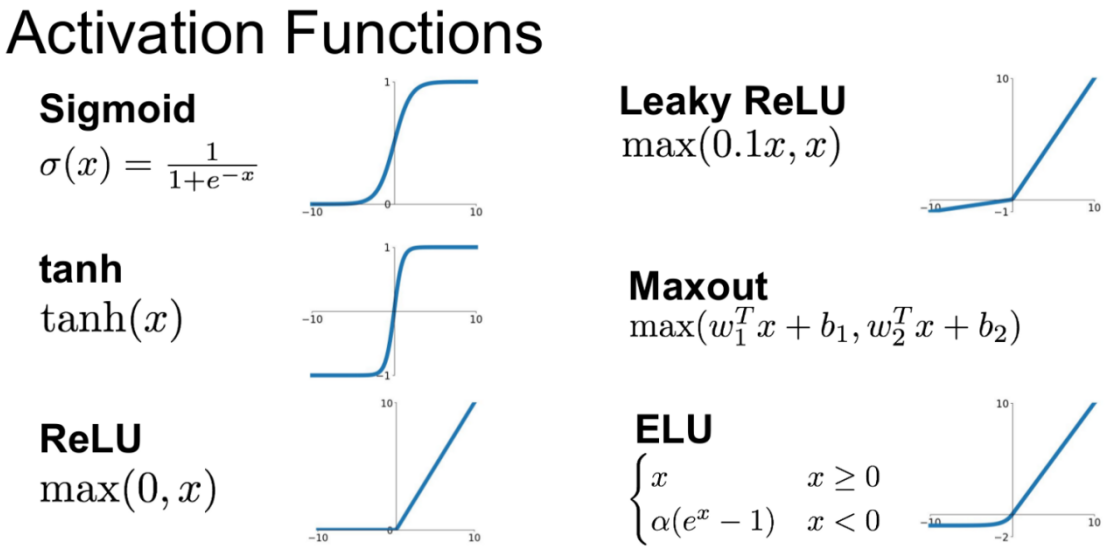
Training
Setting initial parameters θ

- Bad: all zero values
    - Lead to same gradients and same updates.
- Bad: large values
    - Activation is saturated
    - Gradients become zero
    - Optimization can‘t process
- Good: small random numbers
    - Keeps the activation in the dynamic region of the sigmoid function
    - Gradients usable
    - Breaks symmetry

Advanced Topics
Other activation functions

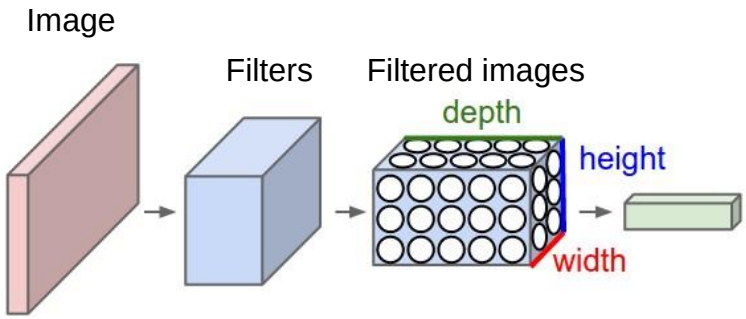
- Add non-linearity, make networks more flexible

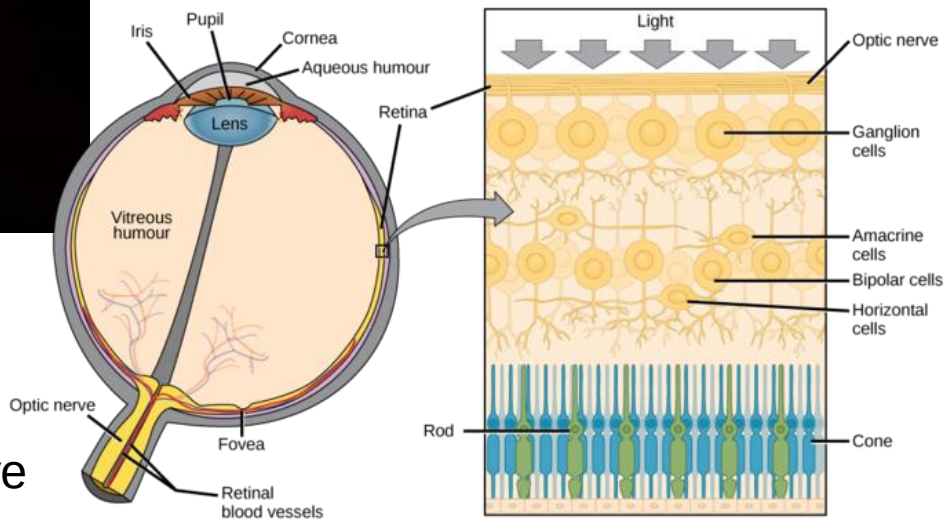
- Different in training convergence



- ReLU seems to work best for large networks!

Advanced Topics
Convolutional Networks

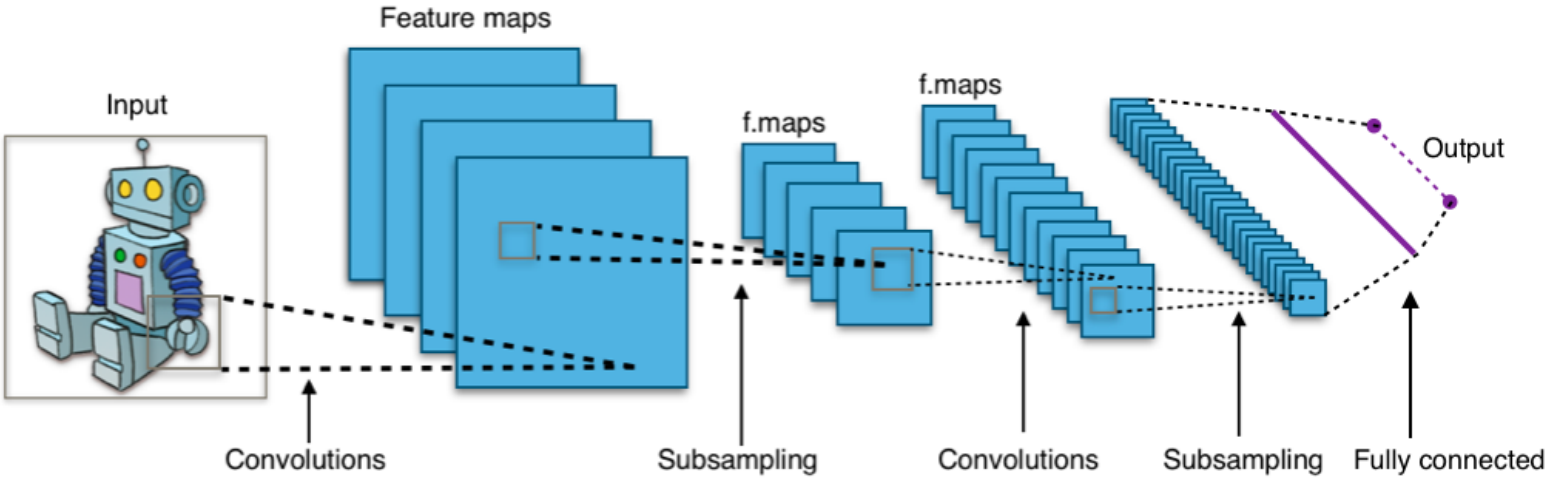




- Good for image classification
- Similar to the receptive field of the eye

Advanced Topics
Deep Networks

- Currently very successful in image classification
- Many layers, often combined with convolution

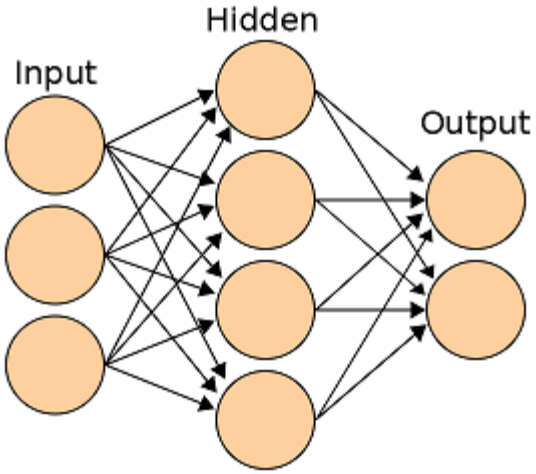


Advaned Topics
Tensor Frameworks

- Build to create large scale deep networks
- Often GPU acceleration
- Automatic gradient computation / back-propagation!
- Specialized optimization algorithms (mini-batch mode, stochastic gradients)
- Pre-configured networks …

Excercise

- Classify hand-written digits with a 3-layer network



- TODO:
  1. Implement and check the cost function of the Neural Network
  2. Classify images from a test data set and compute the accuracy.
  3. Figure out, how test set accuracy and training set accuracy depend on the number of samples.
  4. Try to improve the accuracy by changing the number of neurons in the hidden layer or by changing the regularization.

- 30 min

Further reading

- https://www.coursera.org/learn/machine-learning (Great introduction)
- http://cs231n.github.io/ (Stanford course for Conv. Networks)
- http://www.subsubroutine.com/sub-subroutine/2016/11/12/painting-like-van-gogh-with-convolutional-neural-networks (making art with neural networks)
- https://www.youtube.com/watch?v=AgkfIQ4IGaM
  (Amazing video, how neurons get specialized tasks)
- http://ruder.io/optimizing-gradient-descent/
  (Overview on optimization algorithms for neural networks)

Questions

martin.siggel@dlr.de