Logistic regression and neural networks

Mitko Veta
M.Veta@tue.nl

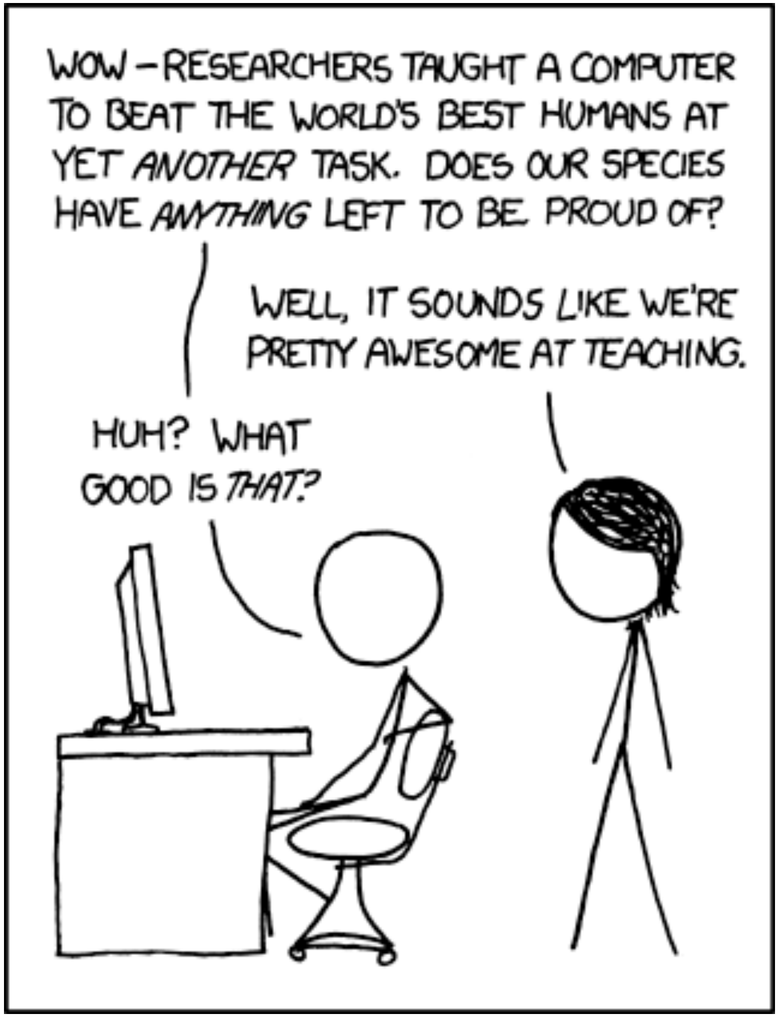

WOW – RESEARCHERS TAUGHT A COMPUTER TO BEAT THE WORLD'S BEST HUMANS AT YET *ANOTHER* TASK. DOES OUR SPECIES HAVE *ANYTHING* LEFT TO BE PROUD OF?
WELL, IT SOUNDS LIKE WE'RE PRETTY AWESOME AT TEACHING.
HUH? WHAT GOOD IS *THAT?*

Image from xkcd.com

Goals for today:

Expand the definition of linear regression to logistic regression (which is actually a classification method).

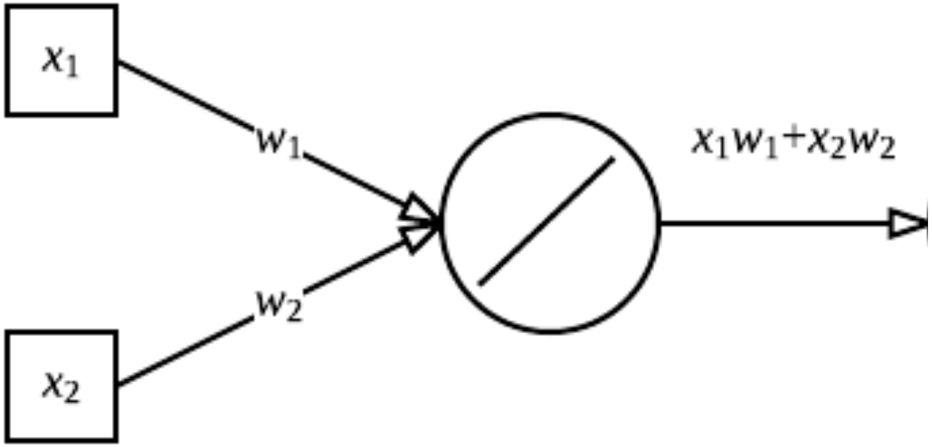
Define (deep) neural networks.

Discuss the training of neural networks (spoiler alert: it is done with the gradient descent method that was introduced during the Registration topic).

Discuss how to properly set up machine learning experiments.

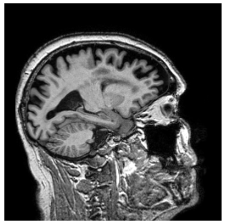
Train some neural networks in a web browser.

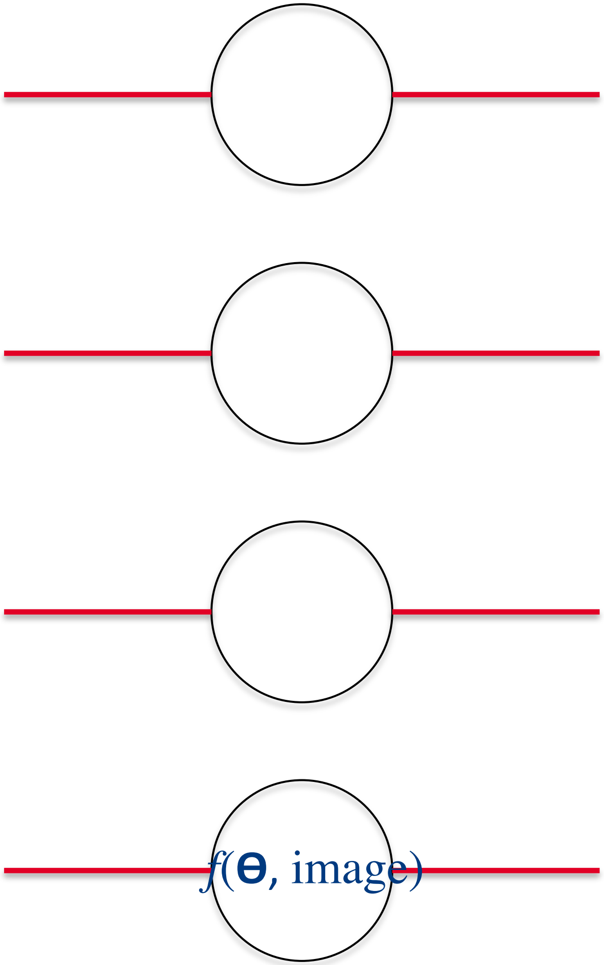
Previously:



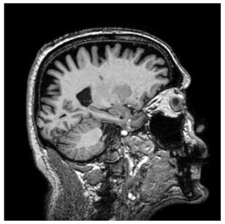
However, this model is not immediately suitable for classification as the output is a continuous value, while for classification we need to predict categories.

* Note that in the figures from now on I will be omitting the bias b for brevity.

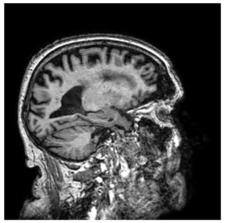




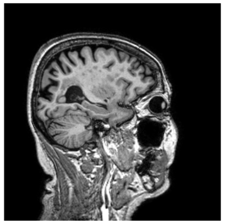
“MS”



“normal”



“MS”



$f(\boldsymbol{\theta}, \text{image})$

?

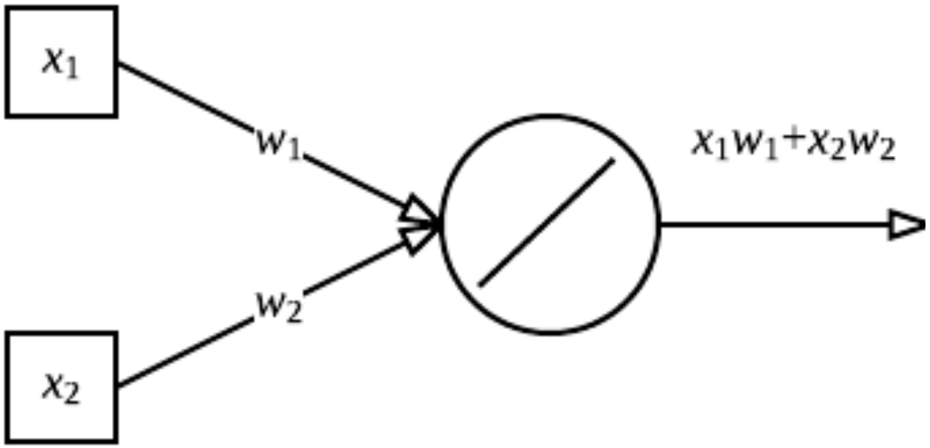
Class “MS”

$y = 1$

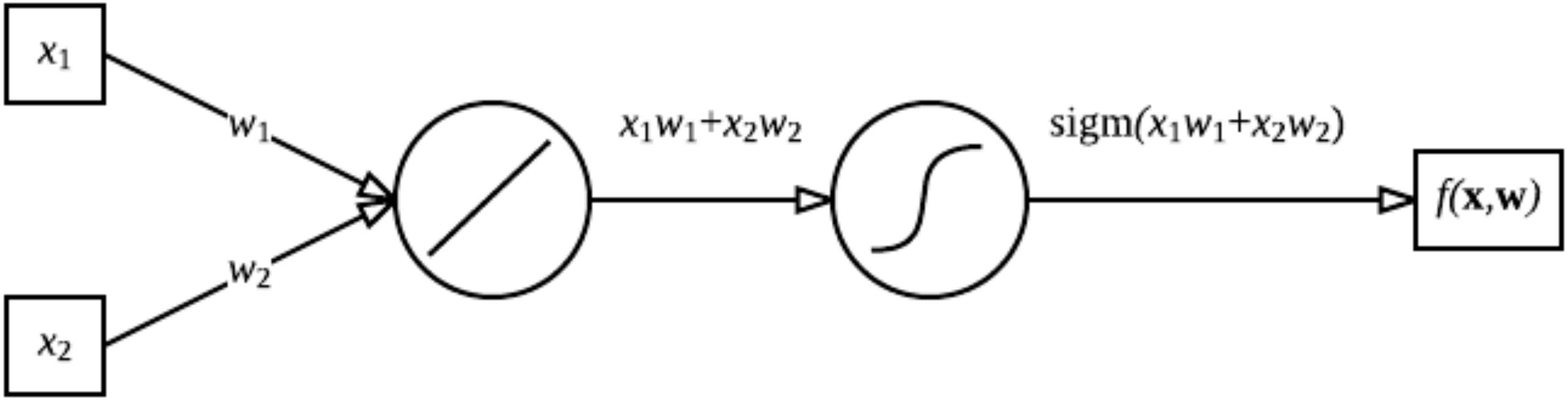
Class “normal”

$y = 0$

How to modify this model so it is suitable for classification?



Pass the output of the linear regression model through a function that will ”squash” it between 0 and 1 and thus it can be interpreted as probability.



This new model is called logistic regression (but remember, this is a model for classification).

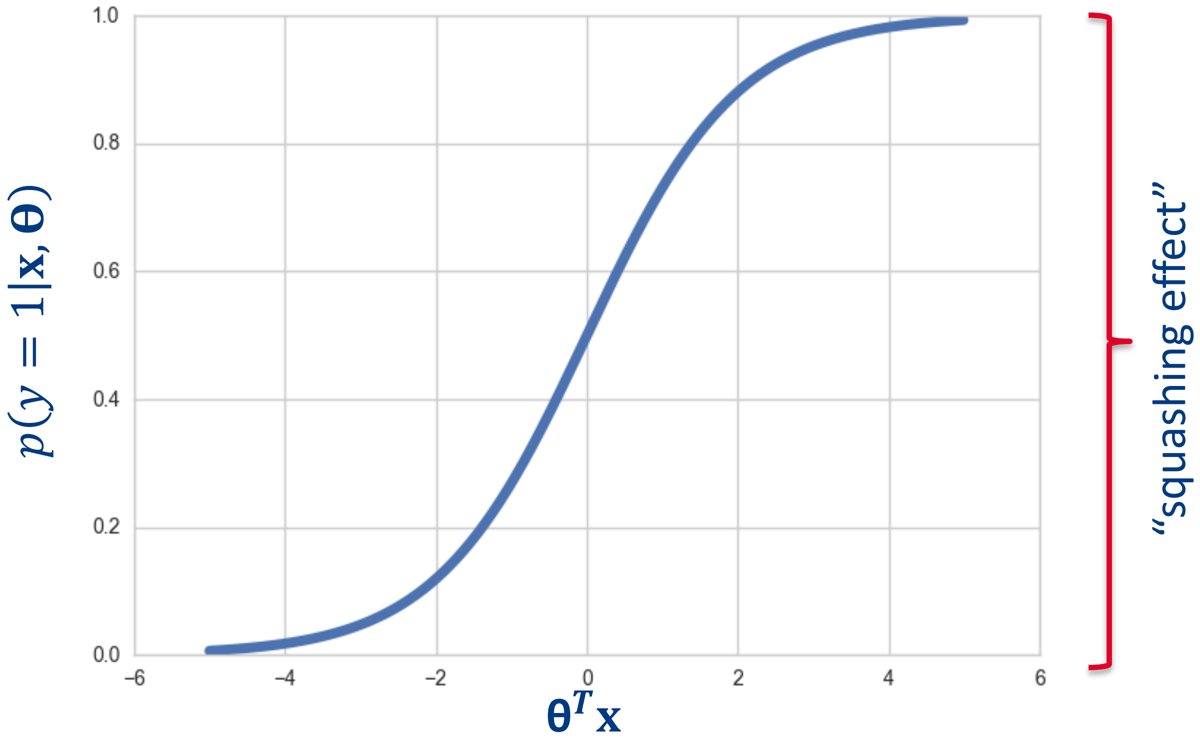
Linear model for regression:

$$\hat{y} = \boldsymbol{\theta}^{\top}\mathbf{x}$$

Linear model for classification:

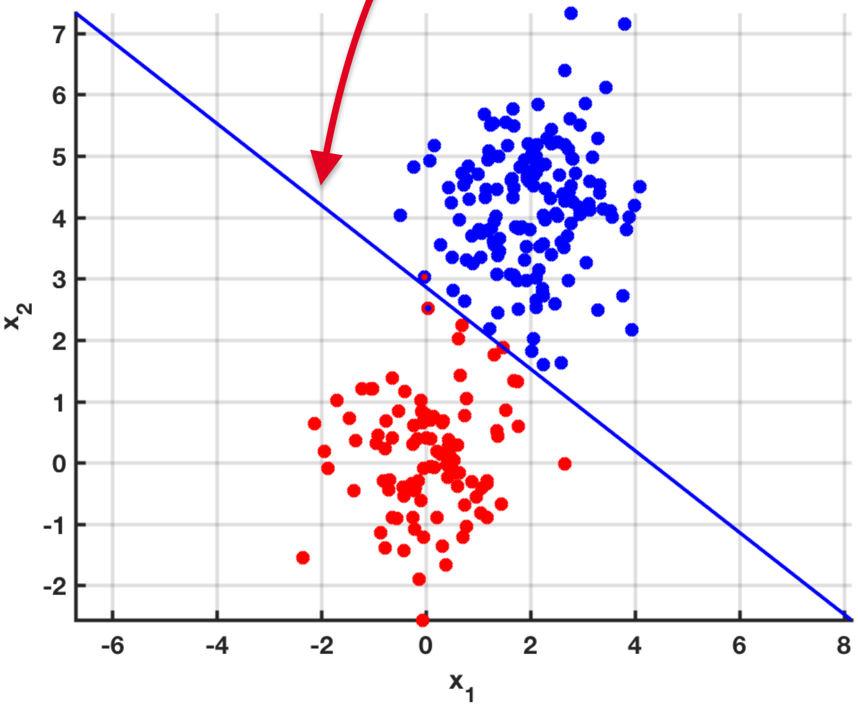
$$p(y = 1 \mid \mathbf{x}) = \sigma(\boldsymbol{\theta}^{\top}\mathbf{x})$$

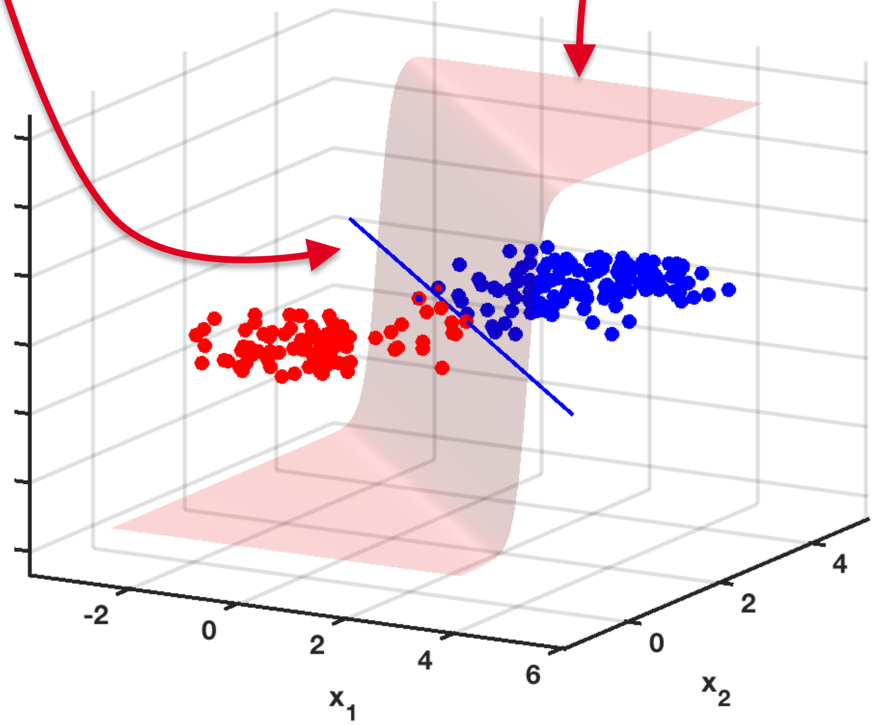
The sigmoid function:



$$p(y = 1 \mid \mathbf{x}) = p(y = 0 \mid \mathbf{x}) = \frac{1}{2}$$

$$p(y = 1 \mid \mathbf{x}) = \sigma(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x})$$





How can we define a loss function for logistic regression?

Remember that logistic regression outputs a probability:

$$f(\mathbf{x}_i, \theta) = p(y = 1 | \mathbf{x}_i, \theta)$$

We want this probability to be high when $y_i = 1$ and low when $y_i = 0$.

Consider the following expression:

$$p(y=1|\mathbf{x}_i,\theta)^{y_i}p(y=0|\mathbf{x}_i,\theta)^{1-y_i}$$

$$p(y=1|\mathbf{x}_i,\theta)^{y_i}\left\{1-p(y=1|\mathbf{x}_i,\theta)\right\}^{1-y_i}$$

$$p(y = 1|\mathbf{x}_i, \theta)^{y_i} p(y = 0|\mathbf{x}_i, \theta)^{1-y_i}$$

For example, let's say that for the sample x_1 with class $y_1 = 1$ the model predicts $p(y = 1|x_1, \theta) = 0.8$.

The value of the expression above will be 0.8.

$$p(y = 1|\mathbf{x}_i, \theta)^{y_i} p(y = 0|\mathbf{x}_i, \theta)^{1-y_i}$$

Another example: let's say that for the sample x_2 with class $y_2 = 0$ the model predicts $p(y = 1|x_1, \theta) = 0.3$.

The value of the expression above will be 0.7.

$$p(y=1|\mathbf{x}_i,\theta)^{y_i}p(y=0|\mathbf{x}_i,\theta)^{1-y_i}$$

This expression selects the probability for the correct class.

We can combine the probabilities for the correct class for all training samples:

$$L(\theta)=\prod_{i=1}^{N}p(y=1|\mathbf{x}_i,\theta)^{y_i}\left\{1-p(y=1|\mathbf{x}_i,\theta)\right\}^{1-y_i}$$

$$L(\theta) = \prod_{i=1}^{N} p(y=1|\mathbf{x}_i,\theta)^{y_i} \left\{1 - p(y=1|\mathbf{x}_i,\theta)\right\}^{1-y_i}$$

This function is called the likelihood of the parameters given the training data.

To find the best parameters we need to maximize it.

$$L(\theta) = \prod_{i=1}^{N} p(y = 1|\mathbf{x}_i, \theta)^{y_i} \left\{1 - p(y = 1|\mathbf{x}_i, \theta)\right\}^{1-y_i}$$

Alternatively, we can minimize the negative log-likelihood:

$$J(\theta) = -\sum_{I=1}^{N} y_i \log p(y = 1|\mathbf{x}_i, \theta) + (1 - y_i) \log \left\{1 - p(y = 1|\mathbf{x}_i, \theta)\right\}$$

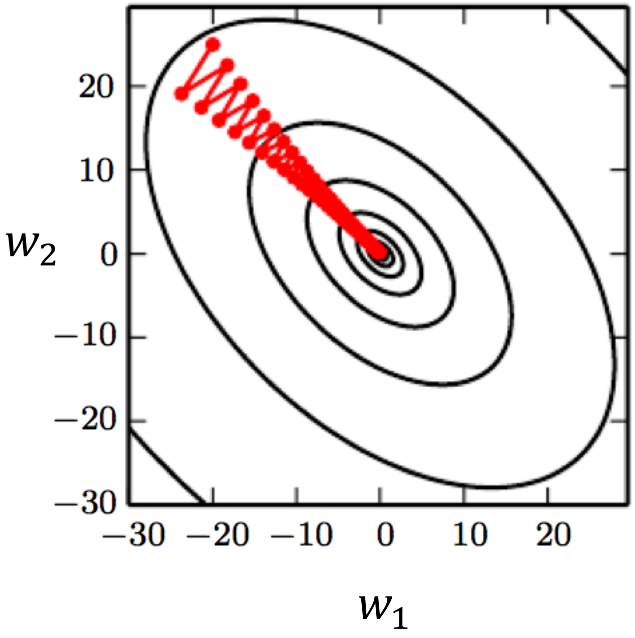
Compared to linear regression, there is no closed-form solution for the parameters of logistic regression.

The optimal parameters are found with gradient descent.

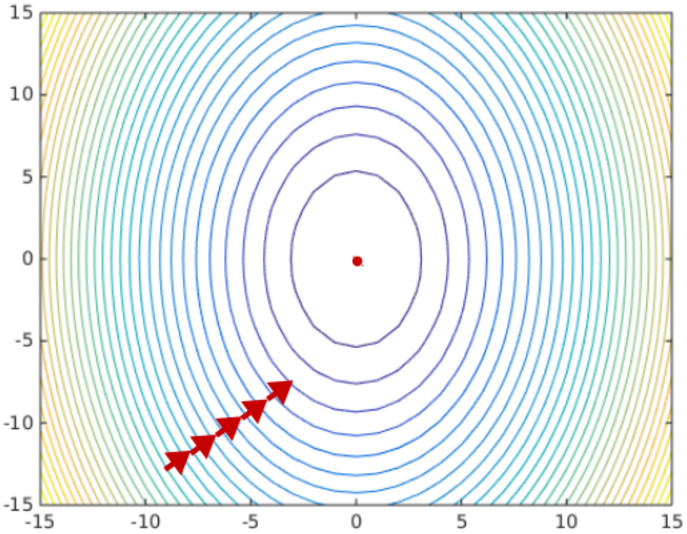
Gradient descent algorithm:

1. Initialize the parameters $\boldsymbol{\theta}$
2. While some stopping criterion is not met, repeat:
3. Compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
4. Update the parameters:

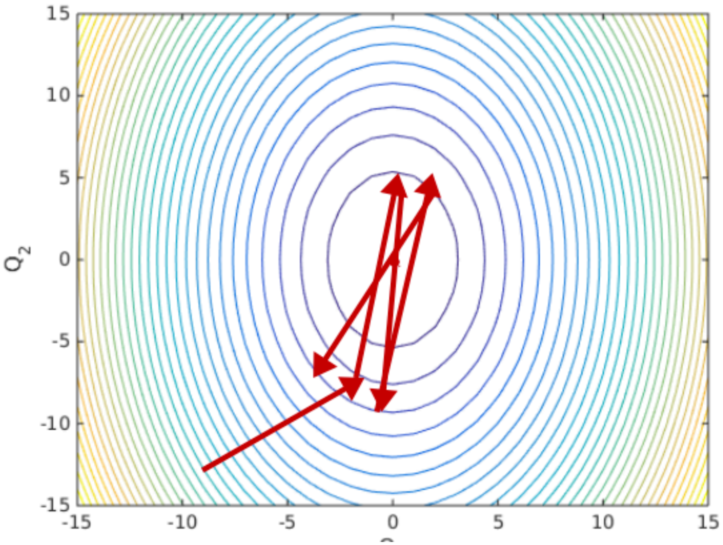
   $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mu \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$



μ is the learning rate (”step size”).



Too low.



Too high.

When the number of samples *m* is very large, $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is very expensive to compute (since the expression for $J(\boldsymbol{\theta})$ includes a sum over all training samples).

Solution: estimate $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ with a batch of samples $N' << N$ that are randomly selected.

This method is called stochastic gradient descent.

Two-class problems – sigmoid:

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

Multi-class problems – softmax:

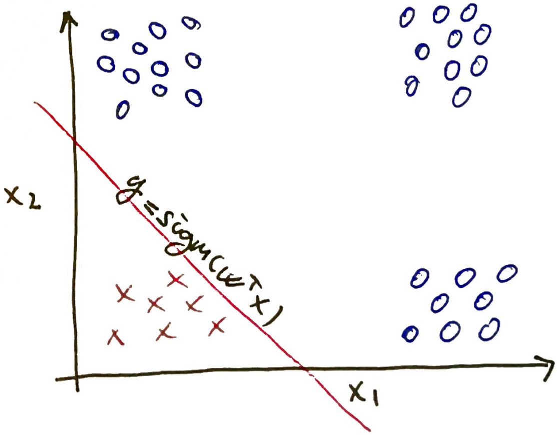
$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

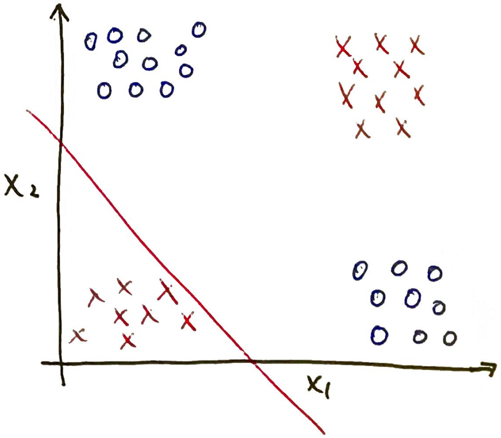
Examples:

https://goo.gl/PuoEBz

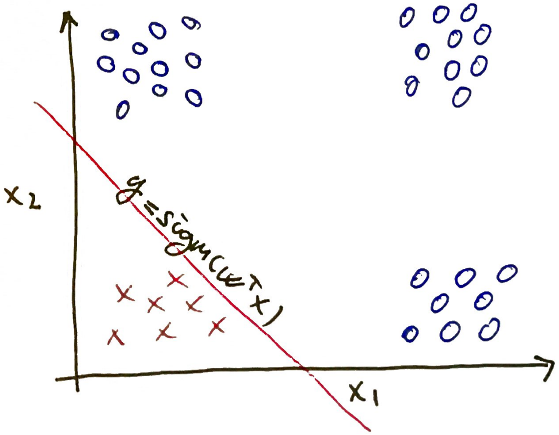
- Train a logistic regression model for an “easy” dataset
- Repeat for more difficult XOR dataset
- Experiment with nonlinear transformation of the features (similar to linear regression)

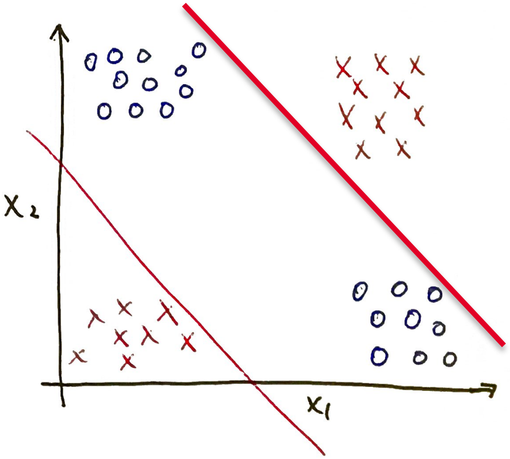
Consider these two classification problems:



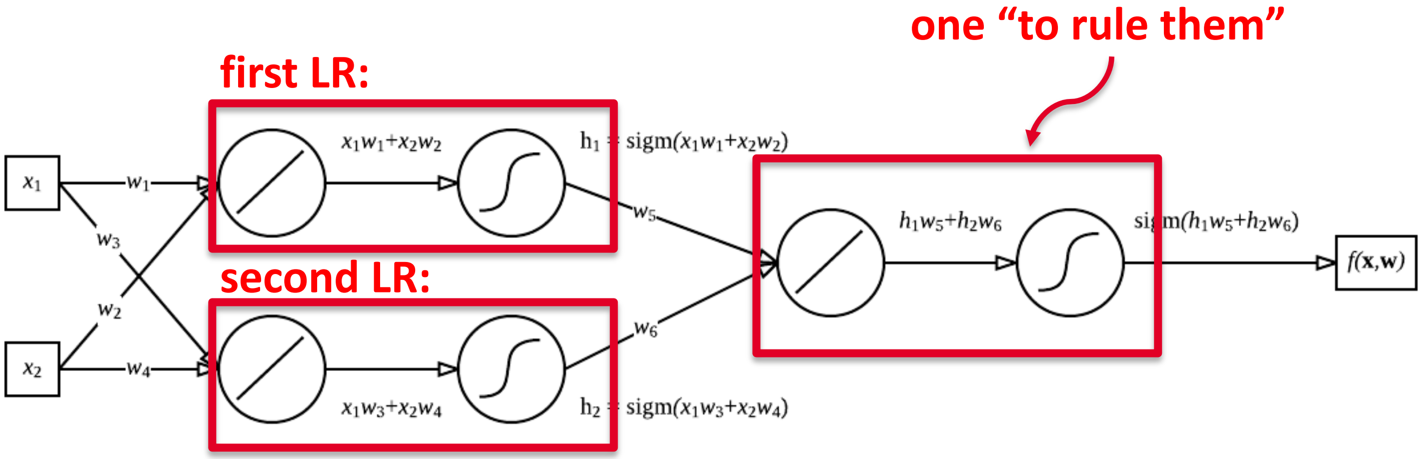


The problem on the right can be solved by combining two linear classifiers:





The problem on the right can be solved by combining two linear classifiers:


one "to rule them"

first LR:

second LR:

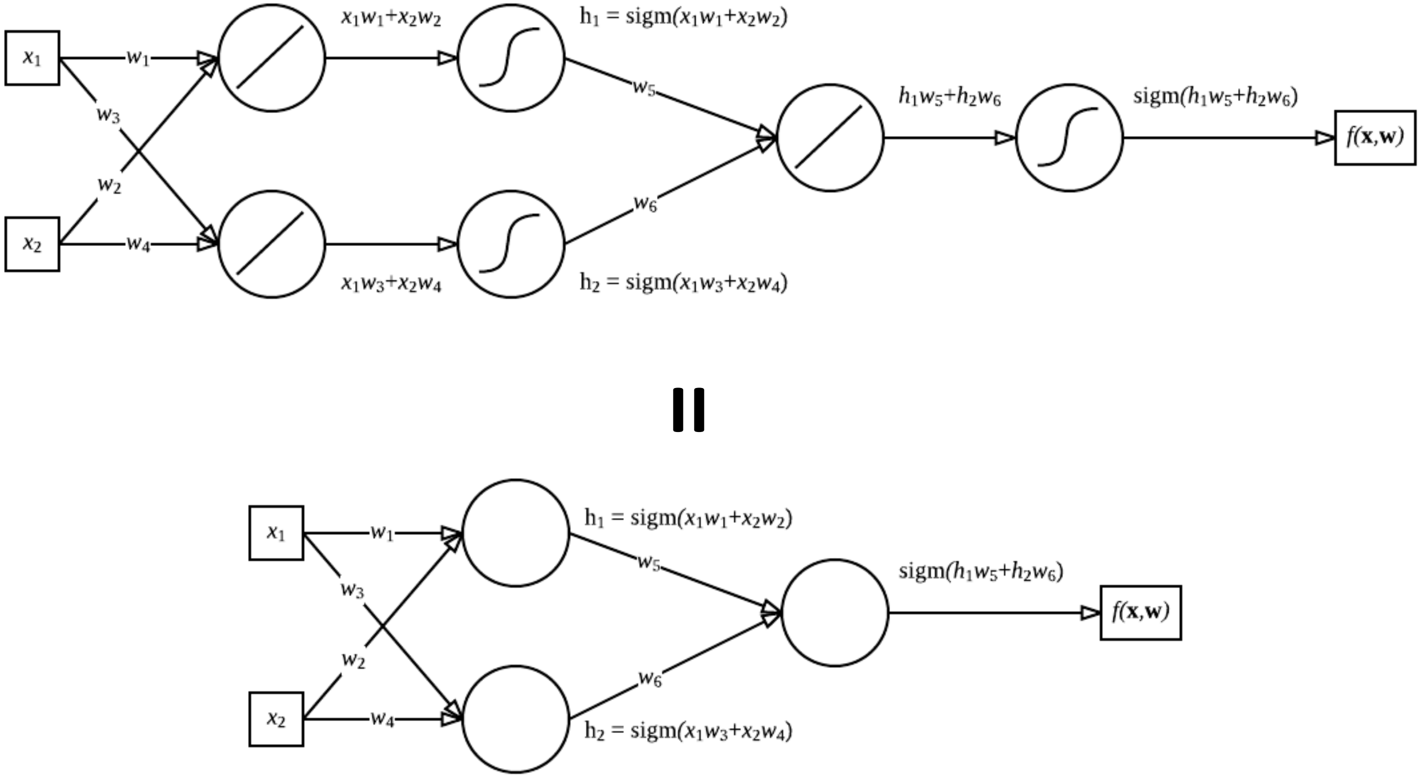
This is (a very small) feedforward neural network.

Feedforward – because the information flows in one direction (there is not feedback).

Neural – because it loosely resembles concepts in neuroscience (biological neurons).

Network – because it is a composition of many elements.

Two logistic regression classifiers = neural network


x_1
x_2
w_1
w_3
w_2
w_4
$x_1w_1+x_2w_2$
$x_1w_3+x_2w_4$
h_1 = sigm($x_1w_1+x_2w_2$)
h_2 = sigm($x_1w_3+x_2w_4$)
w_5
w_6
$h_1w_5+h_2w_6$
sigm($h_1w_5+h_2w_6$)
$f(\mathbf{x},\mathbf{w})$
=
x_1
x_2
w_1
w_3
w_2
w_4
h_1 = sigm($x_1w_1+x_2w_2$)
h_2 = sigm($x_1w_3+x_2w_4$)
w_5
w_6
sigm($h_1w_5+h_2w_6$)
$f(\mathbf{x},\mathbf{w})$

Examples:

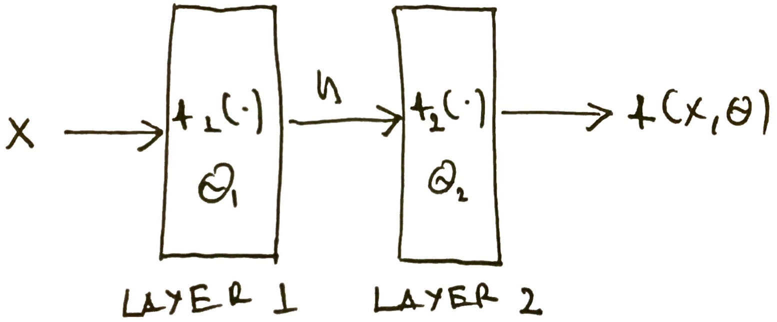
https://goo.gl/4XDAoa

- Train a small neural network (2 hidden neurons) for the XOR dataset
- Repeat for the circle dataset
- Find a solution that works well for the circle dataset

For this particular case we can think of the output function of the neural network as a composition of two layers:

$$f_1(\mathbf{x}, \boldsymbol{\theta}_1) = \mathbf{h}$$

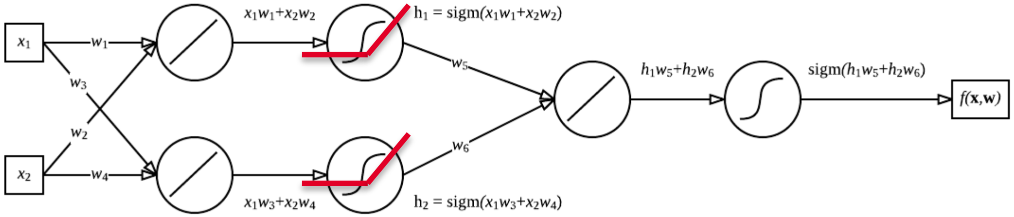
$$f(\mathbf{x}, \boldsymbol{\theta}) = f_2(\mathbf{h}, \boldsymbol{\theta}_2) = f_2(f_1(\mathbf{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2)$$



In fact, the sigmoid nonlinearity can be implemented as a separate sigmoid layer.

In “modern” neural network architectures the sigmoid nonlinearity is usually replaced with a rectified linear unit (ReLU):

$$\mathrm{ReLU}(a) = \max(0, a)$$

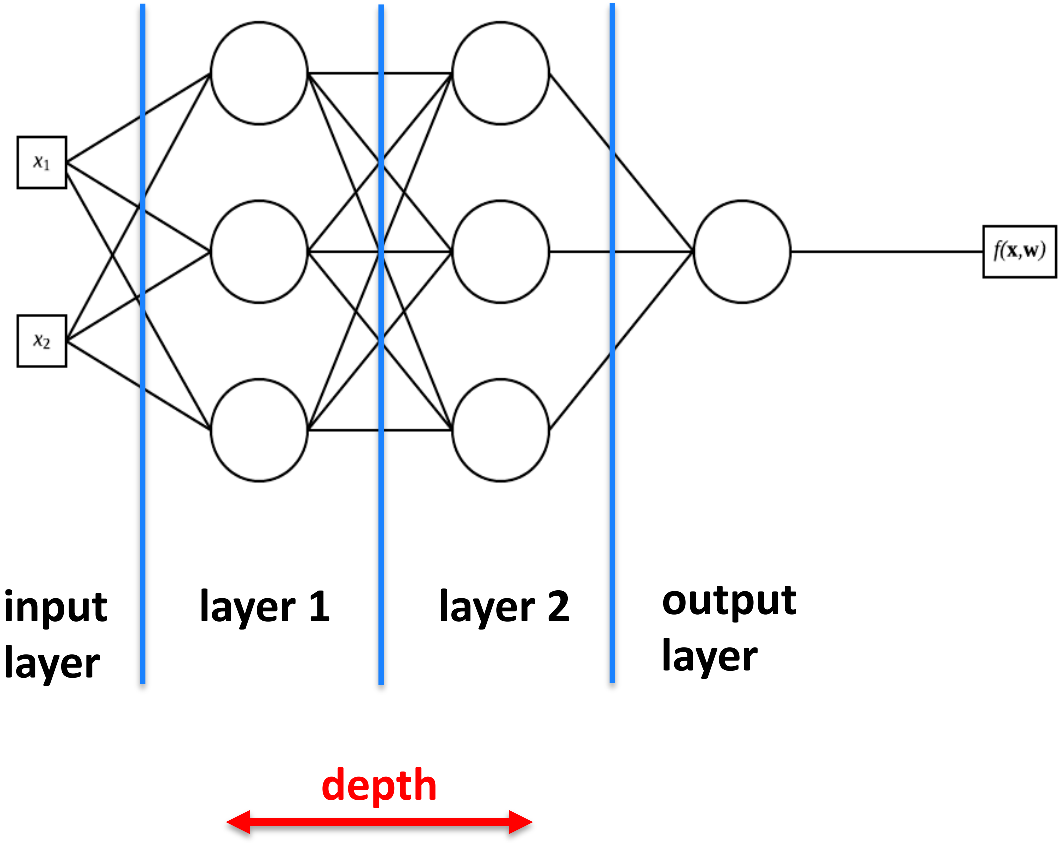

x_1
x_2
w_1
w_2
w_3
w_4
$x_1w_1+x_2w_2$
$h_1 = \mathrm{sigm}(x_1w_1+x_2w_2)$
w_5
w_6
$x_1w_3+x_2w_4$
$h_2 = \mathrm{sigm}(x_1w_3+x_2w_4)$
$h_1w_5+h_2w_6$
$\mathrm{sigm}(h_1w_5+h_2w_6)$
$f(\mathbf{x},\mathbf{w})$

In general we can have neural networks with many hidden layers:

$$f_d(\ldots f_3(f_2(f_1(\mathbf{x}))))$$

The number of layers *d* is called the depth of the network. This is were the “deep” in “deep learning” comes from.

Deep neural networks



input layer

layer 1

layer 2

output layer

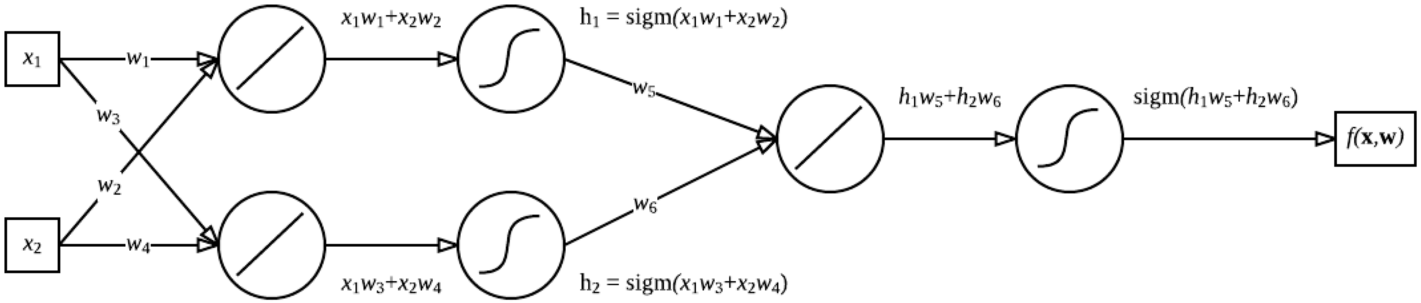
depth

Examples:

https://goo.gl/evySZ4

- Find a neural network that works well with the spiral dataset
- Compare sigmoid vs. ReLU nonlinearity

Even for a small neural network, computing the derivative w.r.t. every parameter involves evaluation of "lengthy" mathematical expressions


x_1
x_2
w_1
w_2
w_3
w_4
$x_1w_1+x_2w_2$
$x_1w_3+x_2w_4$
$\mathrm{h}_1 = \mathrm{sigm}(x_1w_1+x_2w_2)$
$\mathrm{h}_2 = \mathrm{sigm}(x_1w_3+x_2w_4)$
w_5
w_6
$h_1w_5+h_2w_6$
$\mathrm{sigm}(h_1w_5+h_2w_6)$
$f(\mathbf{x},\mathbf{w})$

For very deep networks, the straightforward approach becomes prohibitively expensive.

We need a “smarter” way of computing the gradient w.r.t. every parameter.

This “smarter” algorithm is called back-propagation. It is just the chain rule of differentiation applied to neural network layers.

The training of deep neural networks is highly dependent on the initialization.

Different initializations can result in different generalization performances (even with comparable loss on the training set).

Performing machine learning experiments

Before we had:

Training set: determine the model parameters

Testing set: independent evaluation of the performance of the selected model – this performance is reported in papers, reports etc.

Consider the following situation:

You train a huge number of classifiers (e.g. 100,000) for some classification problem.

You evaluate them all on the testing set and report the best result in your paper.

What can go wrong?

Because of the large number of classifiers you evaluate, one by accident might have a very good performance for this particular dataset.

This performance might not generalize well to another test dataset.

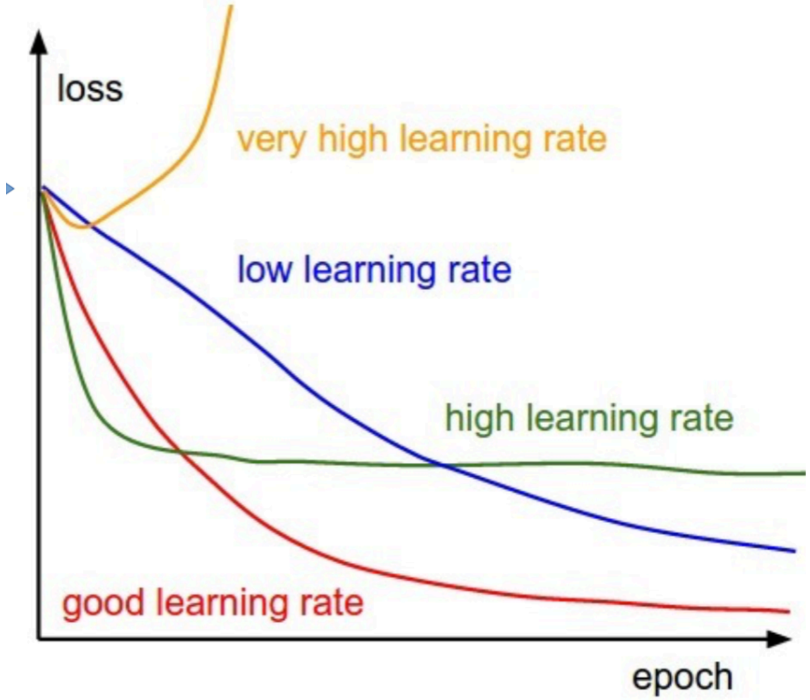
You have reported an overly optimistic and unfair estimation of the performance of your model.

Training set: determine the model parameters

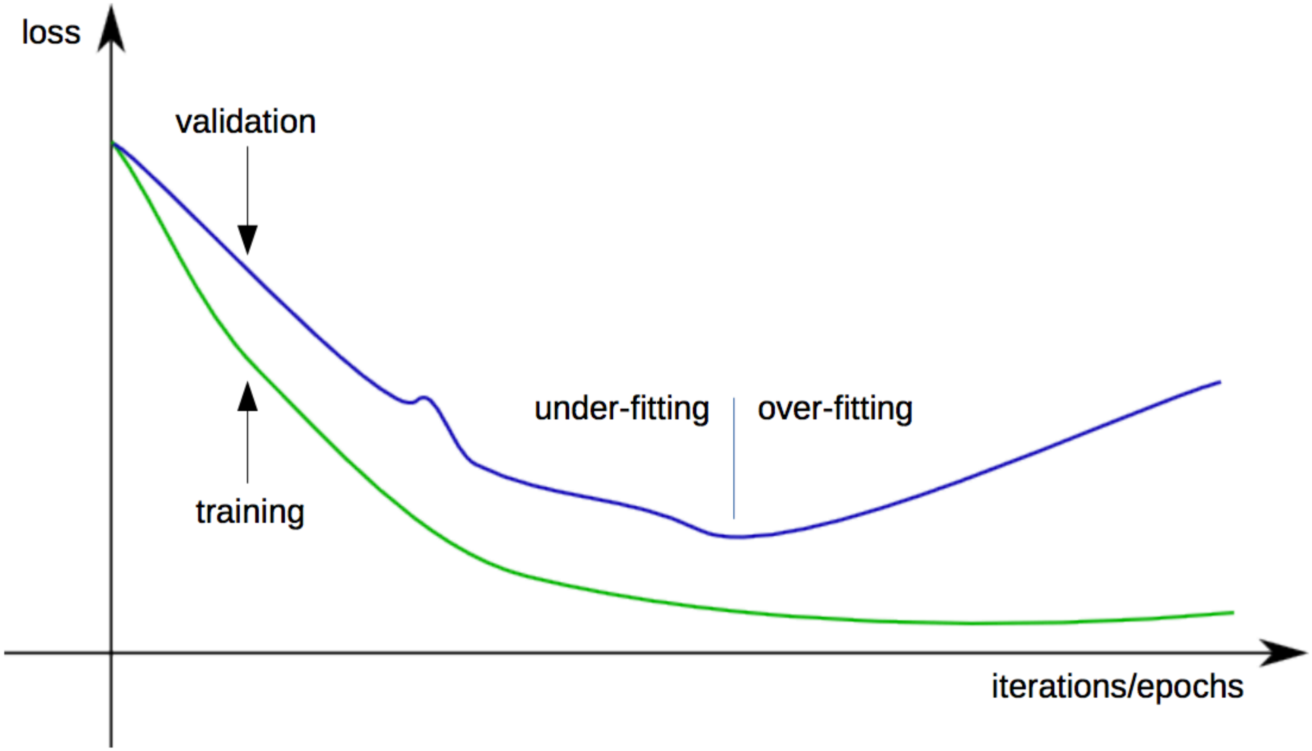
Validation set: model selection, monitor for overfitting

Testing set: independent evaluation of the performance of the selected model – this performance is reported in papers, reports etc.

The training and validation loss curves should be monitored during the training process:



Always monitor the validation loss:



Examples:

https://goo.gl/1DNwSb

- Monitor for overfitting
- Experiment with very large and very small learning rates

Goals for today (recap):

Expand the definition of linear regression to logistic regression (which is actually a classification method).

Define deep neural networks.

Discuss the training of neural networks (spoiler alert: it is done with the gradient descent method that was introduced during the Registration topic).

Discuss how to properly set up machine learning experiments.

Train some neural networks in a web browser.

Discussion point 1

You are given a dataset of 300 images originating from 100 patients (each patient has on average 3 images). What is the proper way to split the data into a training, validation and test sets?

Discussion point 2

We saw that the sigmoid nonlinearity of the neurons in the hidden layer can be replaced with ReLU. However, why do we at all need nonlinearities? Can we just remove them?

Next time:

Convolutional neural networks (a modification of neural networks that works better for images).