

Deep learning

Lecture 1

Marleen Balvert

The course - content

- Theoretical understanding of deep neural networks
- Build and use your own networks in Tensorflow
- Applications

The course - setup

- Seven lectures
- One tutorial (week 3)
- Two assignments (25% each)
 - Assignment 1: start week 3, deadline week 5
 - Assignment 2: start week 5, deadline week 7
- One final exam (50%)

Literature

- Intuitive explanation to deep learning:
<http://neuralnetworksanddeeplearning.com>
- Technical explanation to deep learning:
<https://www.deeplearningbook.org/>

Your background

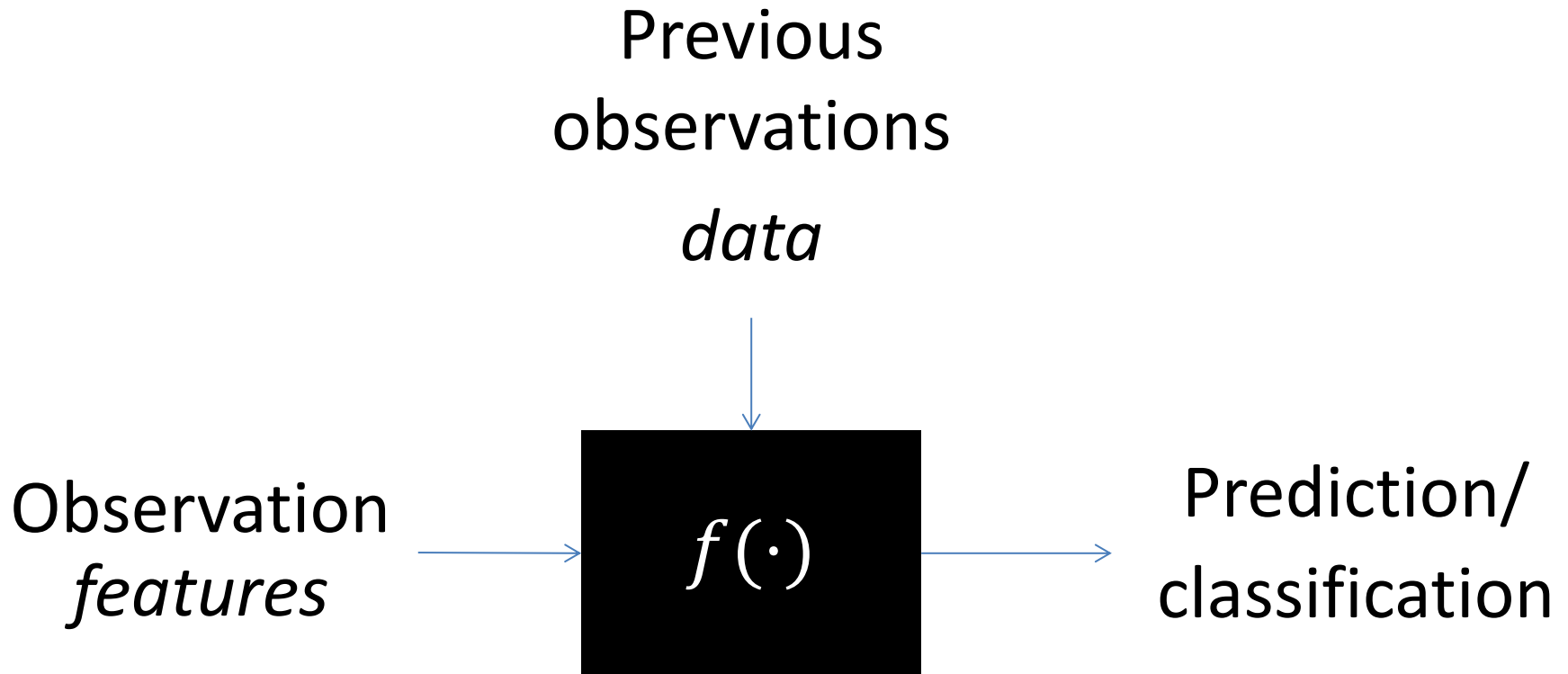
Today

- What is machine learning?
- What is a perceptron?
- What is a (deep) neural network?
- Why use deep neural networks, and when not to use them?
- What is underfitting, overfitting and capacity?
- How to train your model and optimize hyperparameters?
- How to use regularization to prevent overfitting?

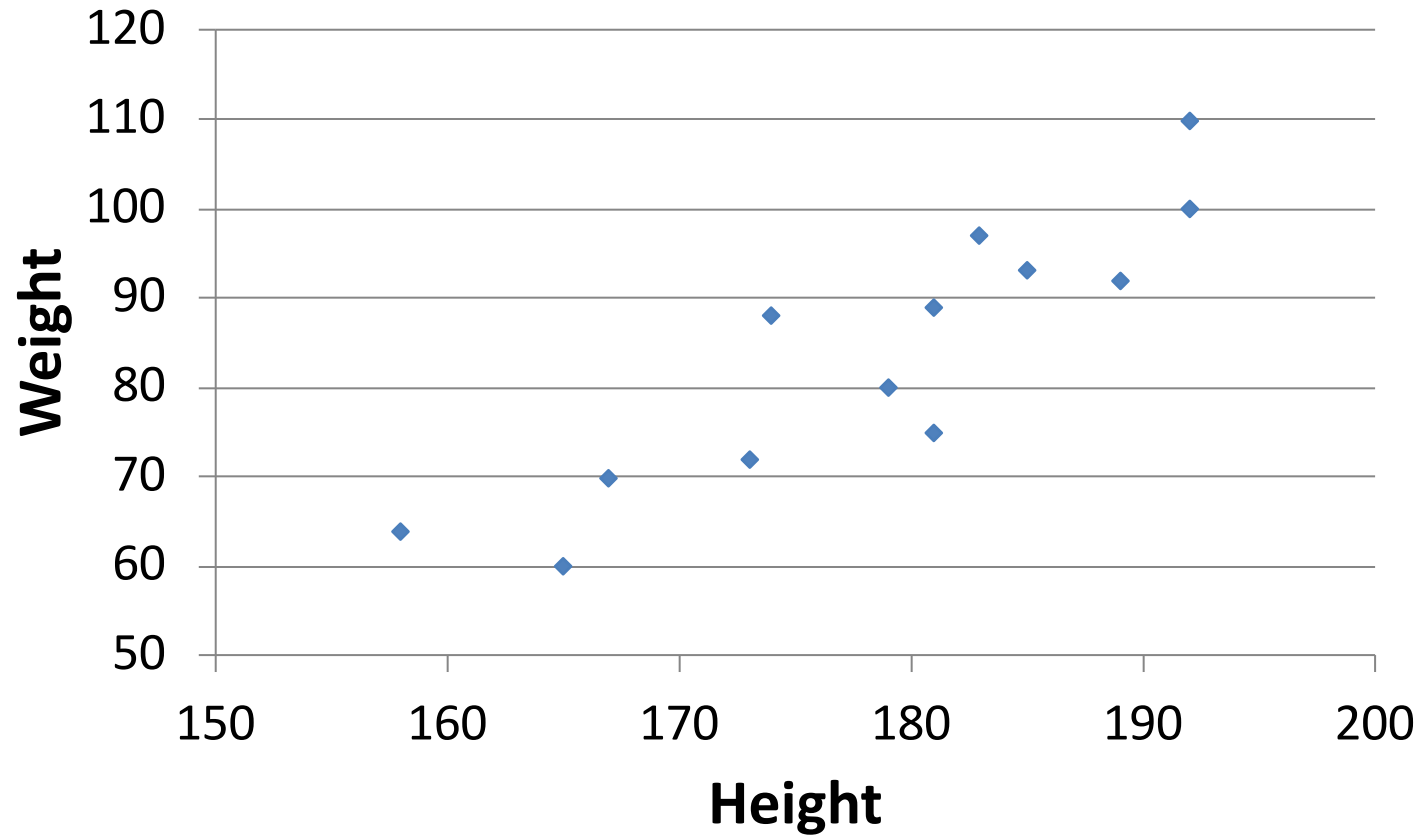
What is machine learning?

- Train a machine to learn the relationship between pre-defined *input* or *independent* variables and a chosen *output* or *dependent* variable
- Train a machine to *predict* an *output* or *dependent* variable from *input* or *independent* variables based on data

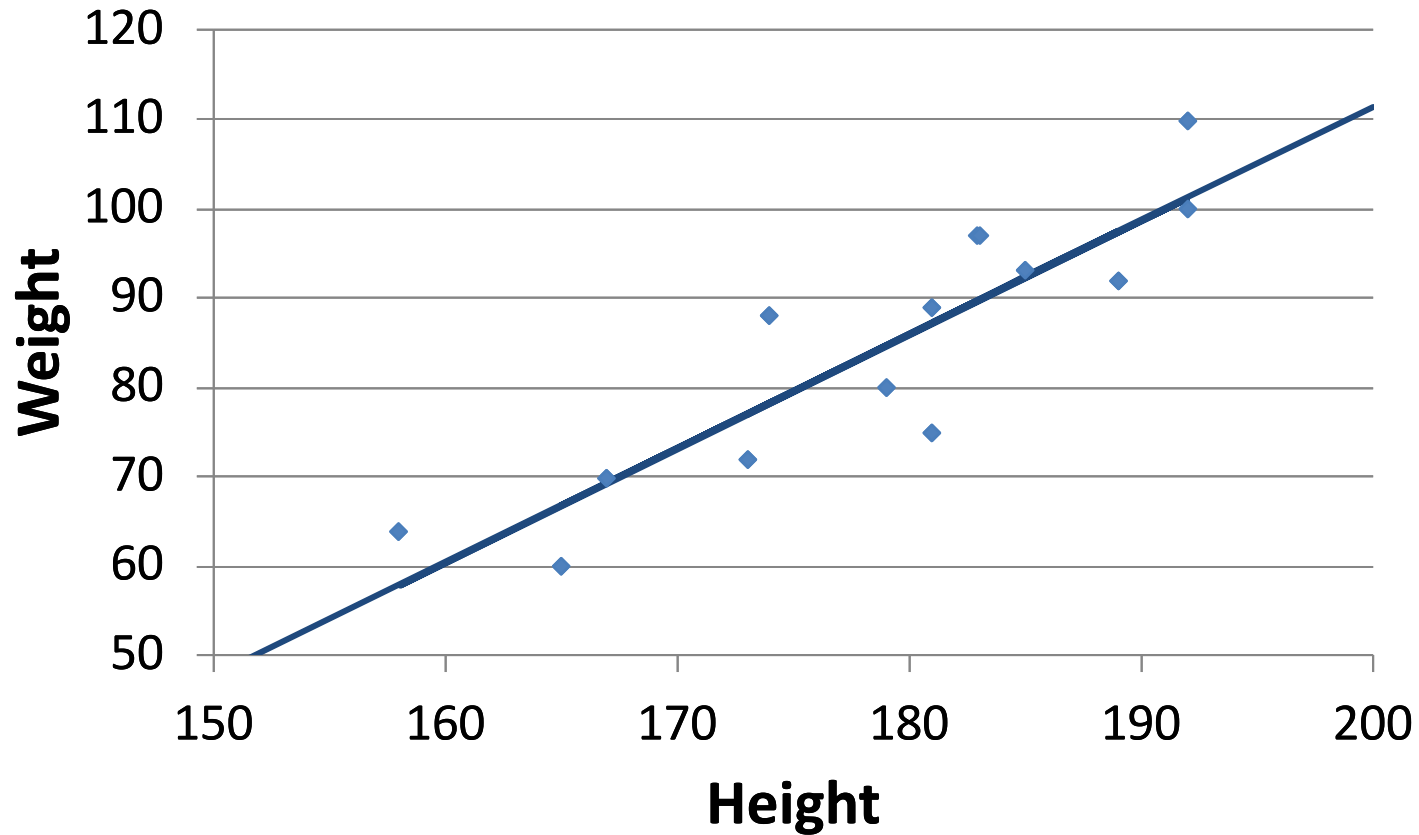
What is machine learning?



Example – regression

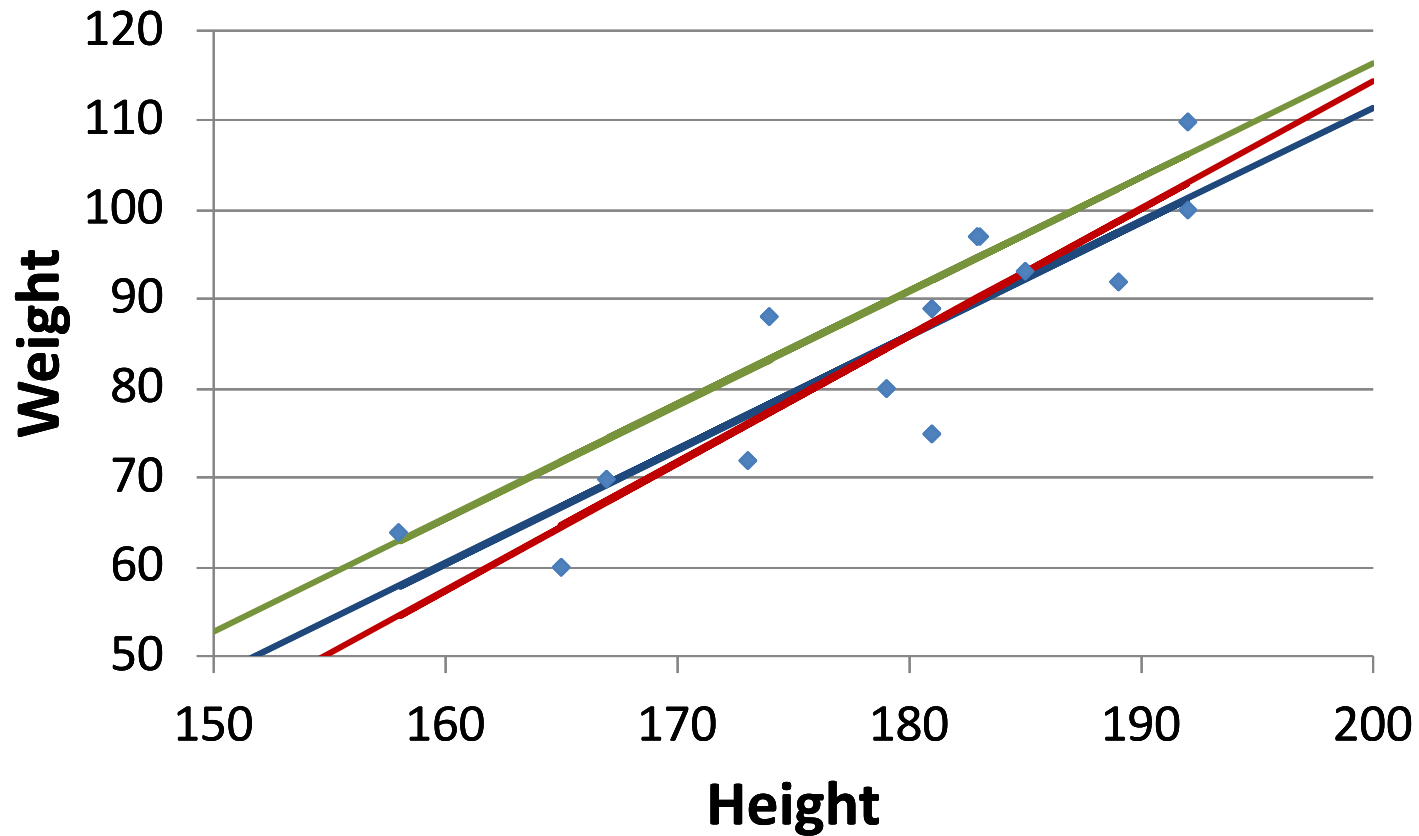


Example – regression



$$\hat{y} = w_0 + w_1 x$$

Example – regression



$$\hat{y} = w_0 + w_1 x$$

Parameter optimization

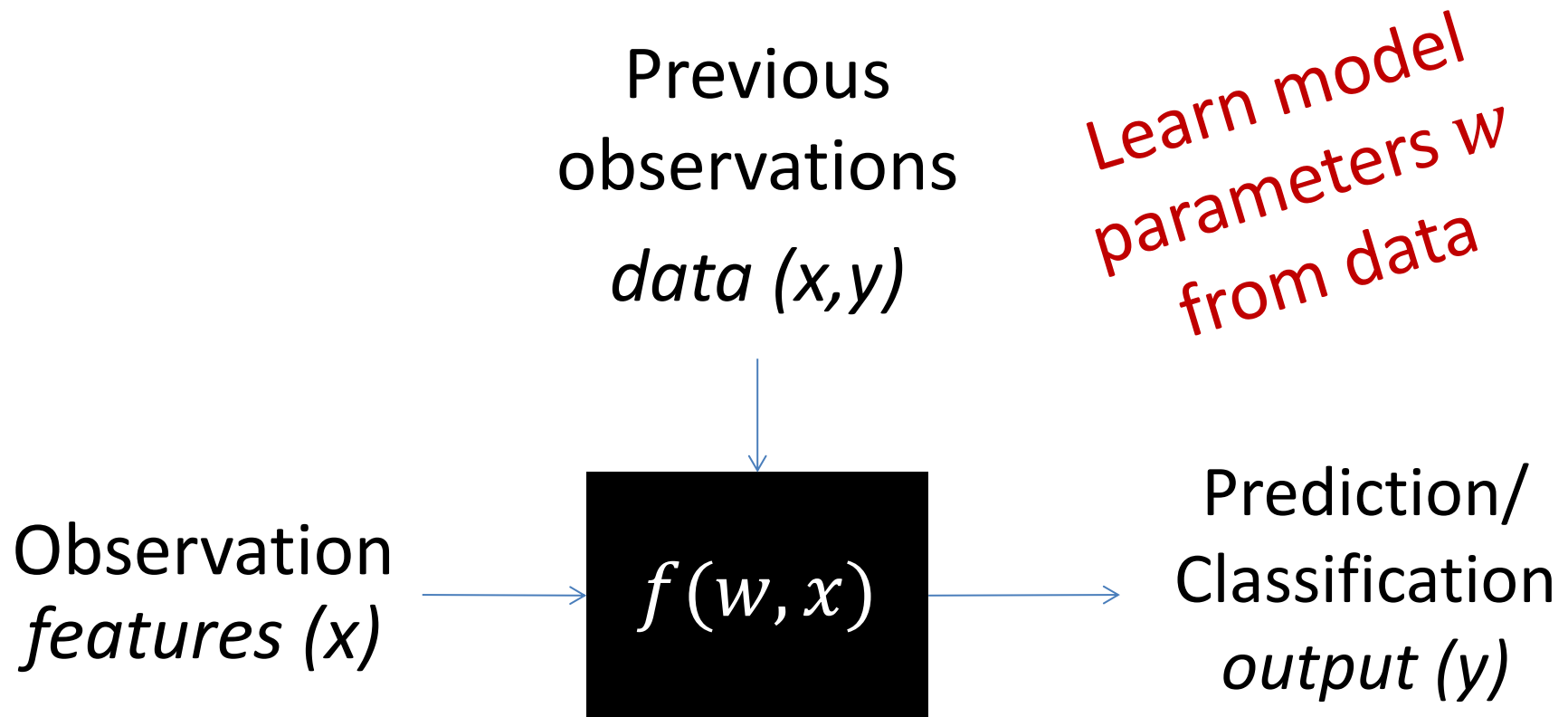
- Need a performance measure $P(w, x, y)$

$$MSE(w, x, y) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i(w, x_i) - y_i)^2$$

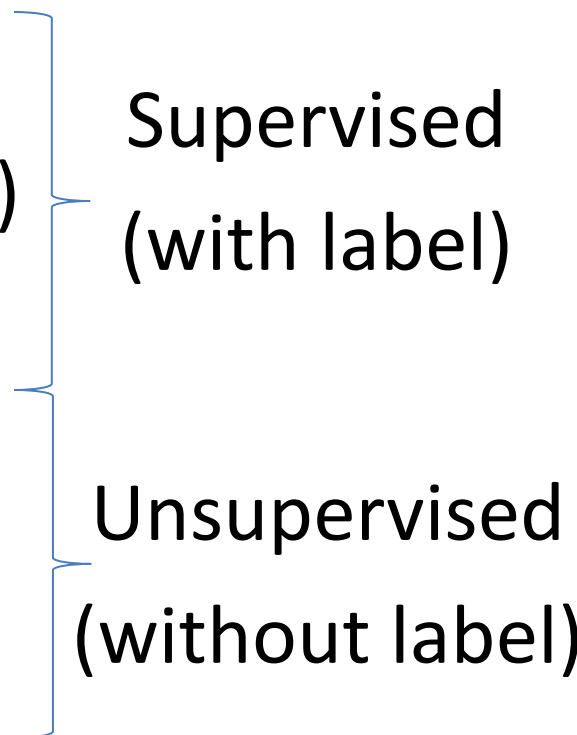
$$MAD(w, x, y) = \frac{1}{n} \sum_{i=1}^n |\hat{y}_i(w, x_i) - y_i|$$

- Solve $\min_w MSE(w, x, y)$ or $\min_w MAD(w, x, y)$

What is machine learning?



Machine learning tasks

- Regression ($f: \mathbb{R}^n \rightarrow \mathbb{R}$)
 - Classification ($f: \mathbb{R}^n \rightarrow \{0, \dots, k\}$)
 - Density estimation $p(y|x)$
 - Imputation of missing values
 - Denoising
 - Density estimation $p(x)$
 - ...
- 
- Supervised
(with label)
- Unsupervised
(without label)

Machine learning methods

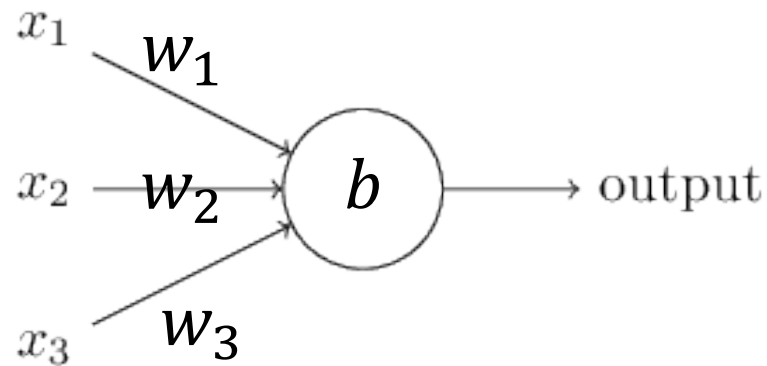
- (Logistic) regression
- Nearest neighbor classification
- Support vector machines
- Decision trees & random forest
- (Deep) neural networks
 - Feedforward neural networks
 - Convolutional neural networks
 - ...
- ...

Supervised learning steps

1. Collect data
2. Choose model
3. Optimize parameters
4. Predict for new observations

(Deep) neural networks

Perceptron



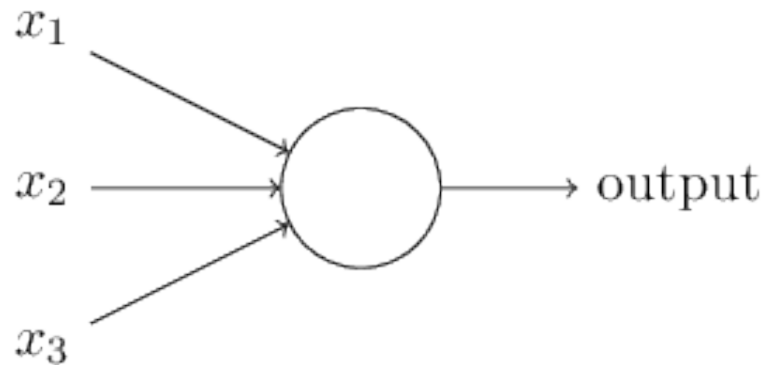
$$\text{output} = \begin{cases} 0 & \text{if } w^T x + b < 0 \\ 1 & \text{if } w^T x + b \geq 0 \end{cases}$$

weights

biases (thresholds)

Perceptron - example

Decision making: shall I go to the festival?

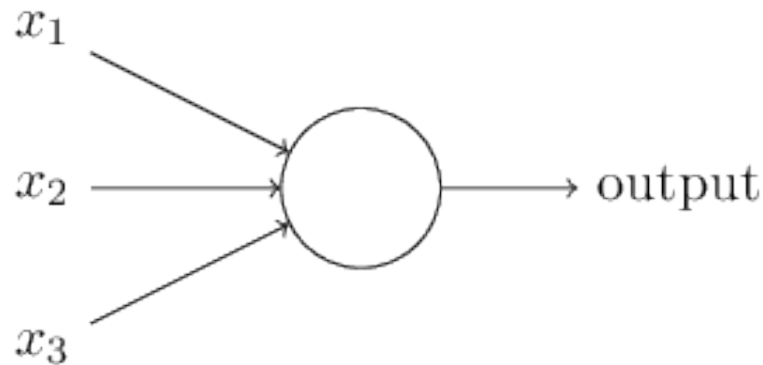


- x_1 = is it not raining?
- x_2 = are my friends going?
- x_3 = is the festival within biking distance?

I go to the festival if and only if it is not raining.

Perceptron - example

Decision making: shall I go to the festival?

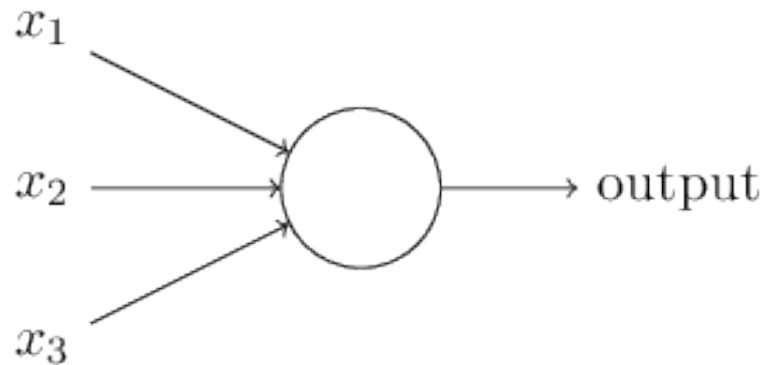


- x_1 = is it not raining?
- x_2 = are my friends going?
- x_3 = is the festival within biking distance?

I go to the festival if it is not raining, OR (if my friends are going AND it is within biking distance)

Perceptron - example

Decision making: shall I go to the festival?



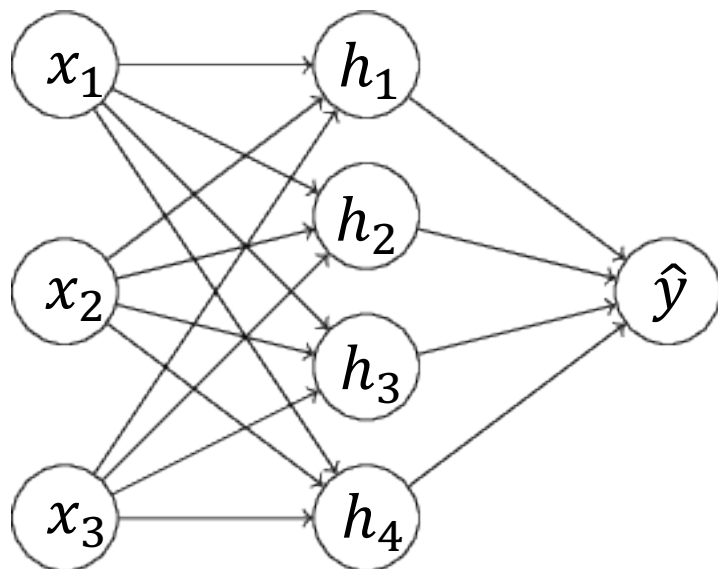
- x_1 = is it not raining?
- x_2 = are my friends going?
- x_3 = is the festival within biking distance?

If $x_1 = 1 \vee (x_2 = 1 \wedge x_3 = 1)$ then $y = 1$, 0 otherwise

Perceptrons can learn AND and OR

Neural network (NN)

Input layer Hidden layer Output layer



$$h_i = \begin{cases} 1 & \text{if } w_i^T x + b_i \geq 0 \\ 0 & \text{if } w_i^T x + b_i < 0 \end{cases}$$

$$\hat{y} = \begin{cases} 1 & \text{if } w_5^T h + b_5 \geq 0 \\ 0 & \text{if } w_5^T h + b_5 < 0 \end{cases}$$

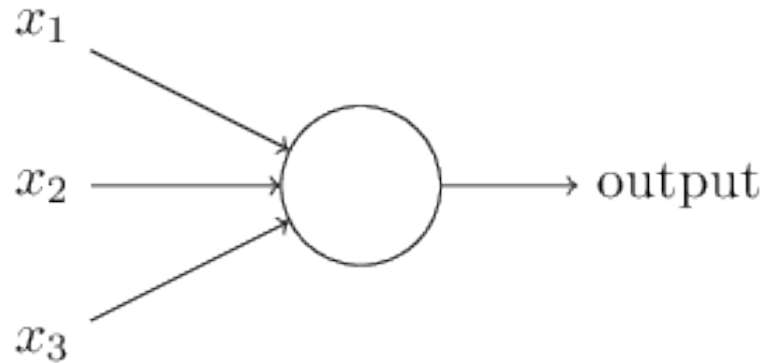
Can we learn XOR?



If $x_1 = 1$ XOR $x_2 = 1$ then $y = 1$

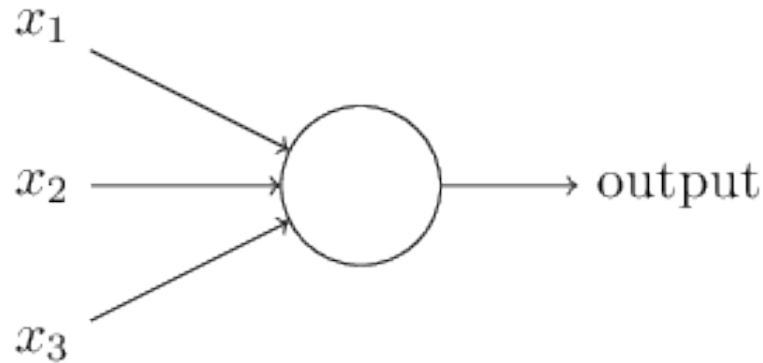
If $(x_1 = 1$ OR $x_2 = 1)$ AND NOT $(x_1 = 1$ AND $x_2 = 1)$
then $\hat{y} = 1$

Activation function: sigmoid



$$\text{output} = \begin{cases} 0 & \text{if } w^T x + b < 0 \\ 1 & \text{if } w^T x + b \geq 0 \end{cases}$$

Activation function: sigmoid

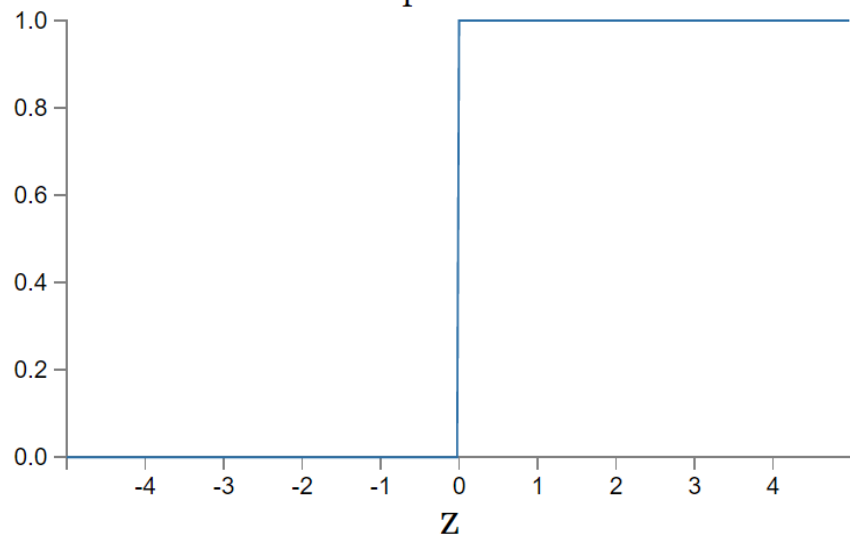


$$\text{output} = \underbrace{f(w^T x + b)}_{\text{Activation function}}$$

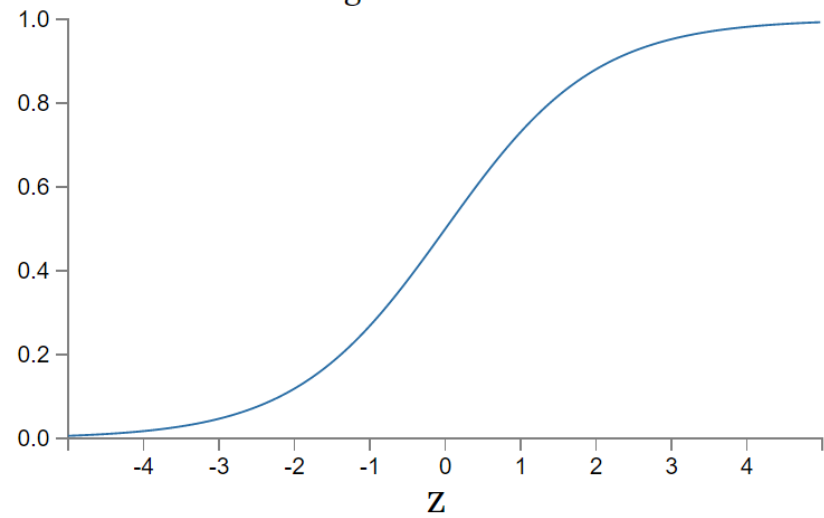
Most frequently used is the sigmoid function:

$$f(w^T x + b) = \frac{1}{1 + e^{w^T x + b}}$$

step function

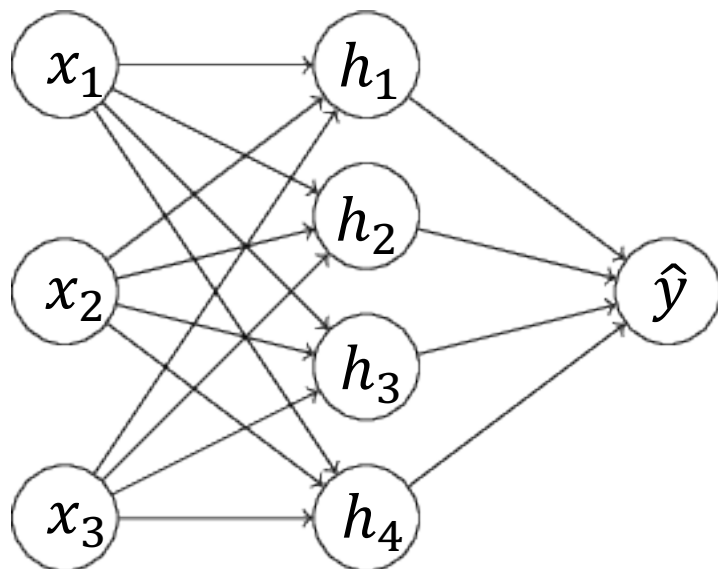


sigmoid function



Neural network (NN)

Input layer Hidden layer Output layer



$$h_i = f(w_i^T x + b_i)$$

$$\hat{y} = f(w_5^T h + b_5)$$

$$\hat{y} = f(w_1, \dots, w_5, b_1, \dots, b_5, x)$$

Goal: find W and b that minimize $P(y, \hat{y}) = P(y, x, W, b)$
where P is a performance measure
measuring the distance between y and \hat{y}

Universal approximation theorem

The theorem basically says:

A feedforward network with a single hidden layer containing a finite number of neurons can approximate any nonconstant, bounded and continuous function with arbitrary closeness, as long as there are enough hidden nodes.

(Video of intuition behind proof:

<https://www.youtube.com/watch?v=Ijqkc7OLenI>)

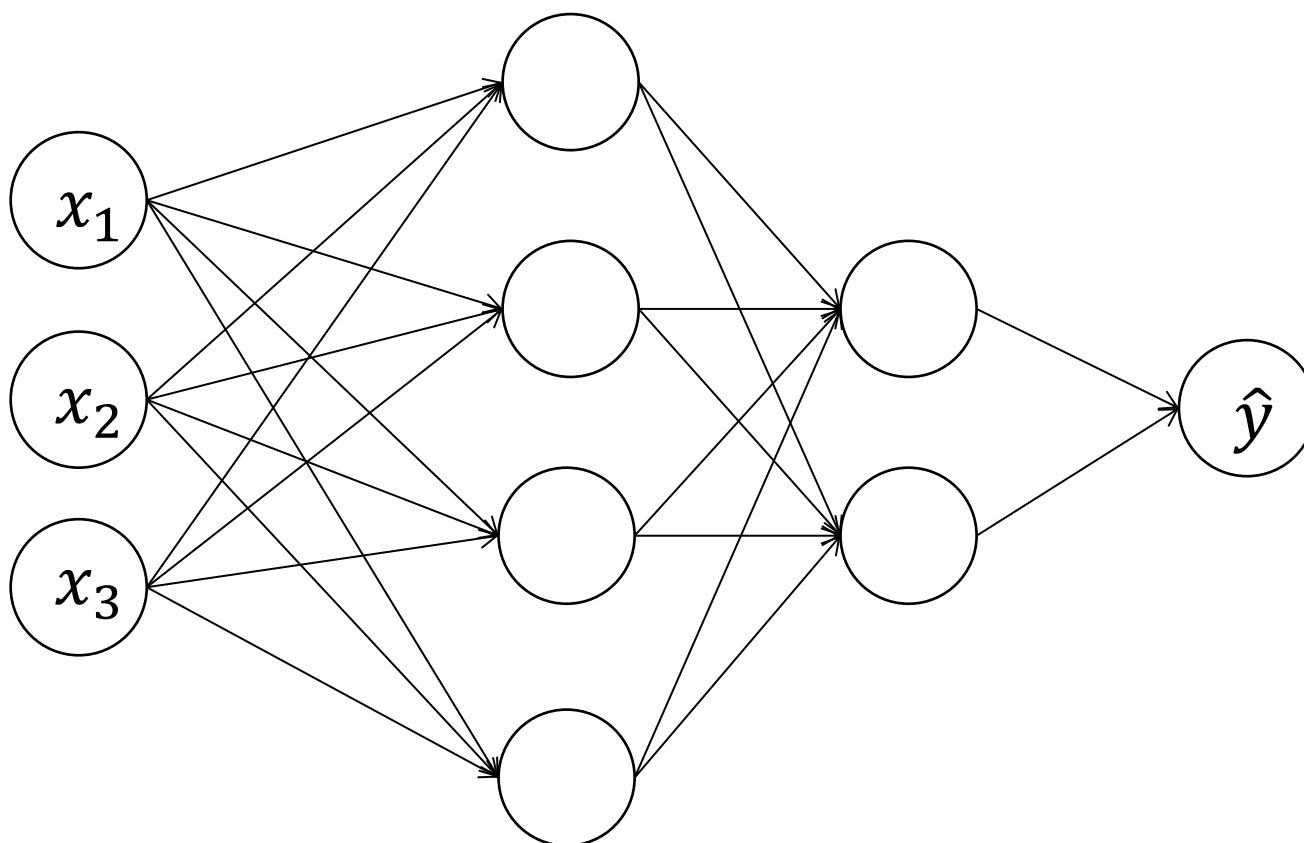
But is it learnable?

Deep neural network (DNN)

Input layer

Hidden layers

Output layer



Why DNNs?

- Allows learning of complex functions
- Can deal with the curse of dimensionality (more input variables than samples)
- Allows for parallelizing computations

Recognizing handwritten digits



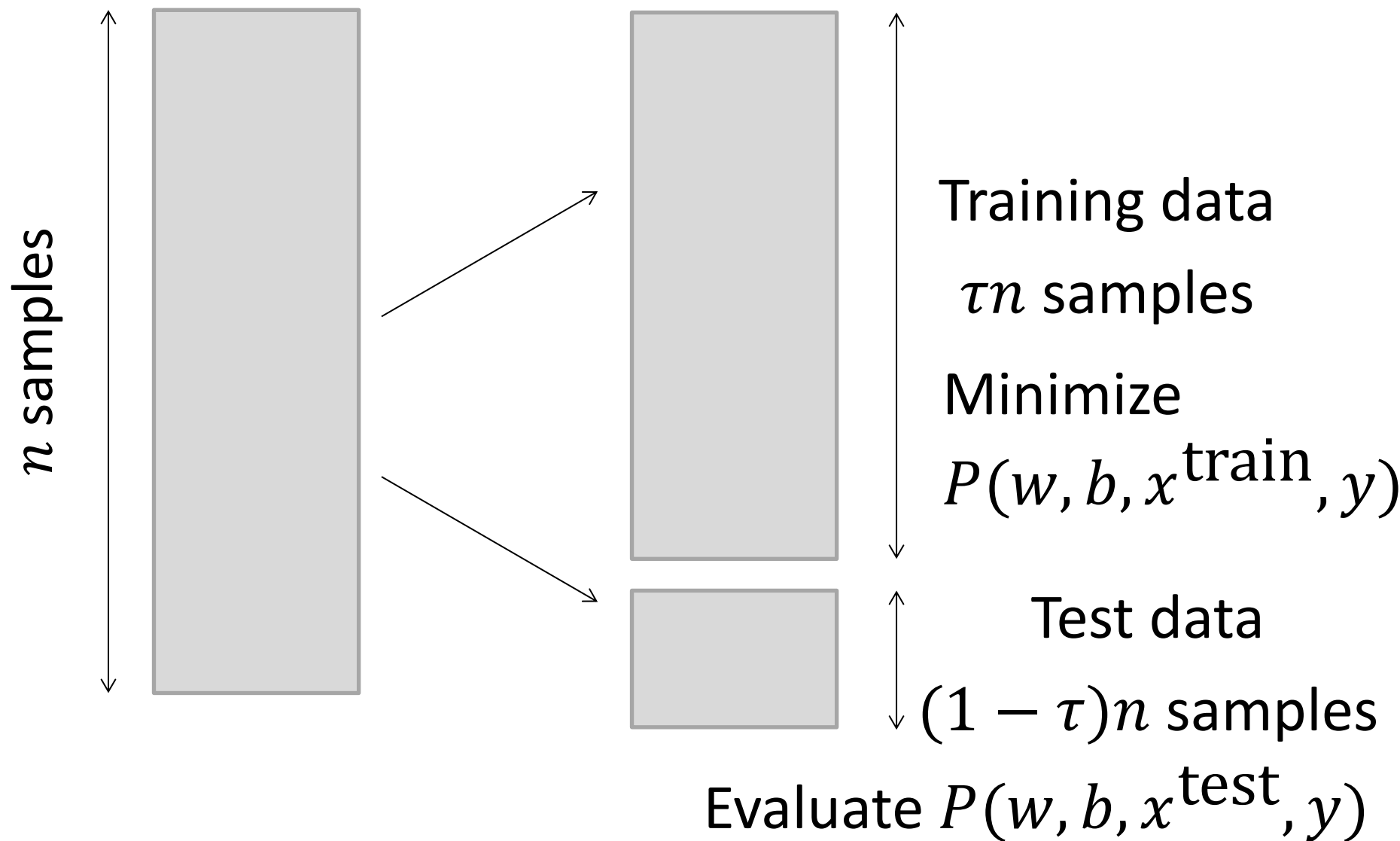
Supervised learning steps

1. Collect data
2. Choose model
3. Optimize parameters
4. Predict for new observations

Training

- Given n samples with input variables x and output y
- Find optimal w and b : minimize $P(w, b, x, y)$
- *Generalization*: the ability to perform well on previously unobserved inputs.

Train and test

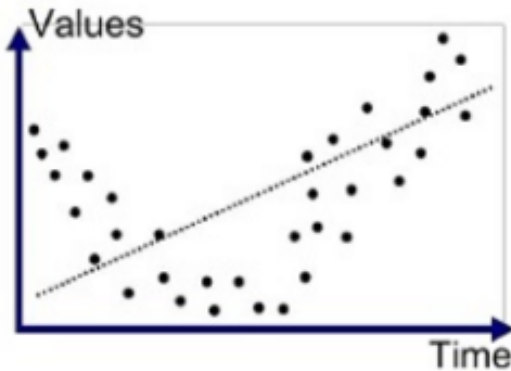


Train and test

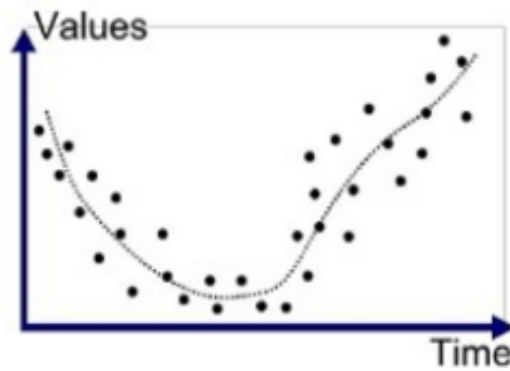
- Training: minimize train error $P(w, b, x^{\text{train}}, y)$
- Testing: evaluate test error $P(w, b, x^{\text{test}}, y)$
- In general: $P(w, b, x^{\text{test}}, y) > P(w, b, x^{\text{train}}, y)$
- Goal: $P(w, b, x^{\text{test}}, y) - P(w, b, x^{\text{train}}, y)$ small

Under- and overfitting & capacity

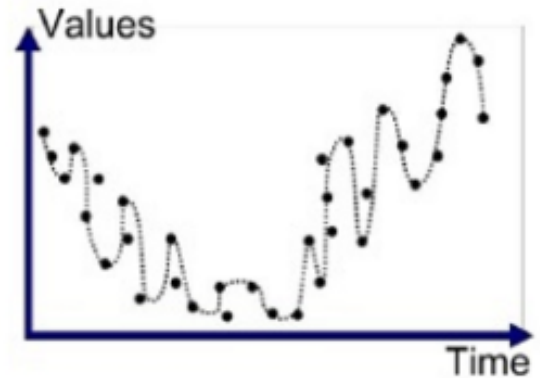
- *Underfitting*: large $P(w, b, x^{\text{train}}, y)$
- *Overfitting*: large $P(w, b, x^{\text{test}}, y) - P(w, b, x^{\text{train}}, y)$



Underfitted



Good Fit/Robust

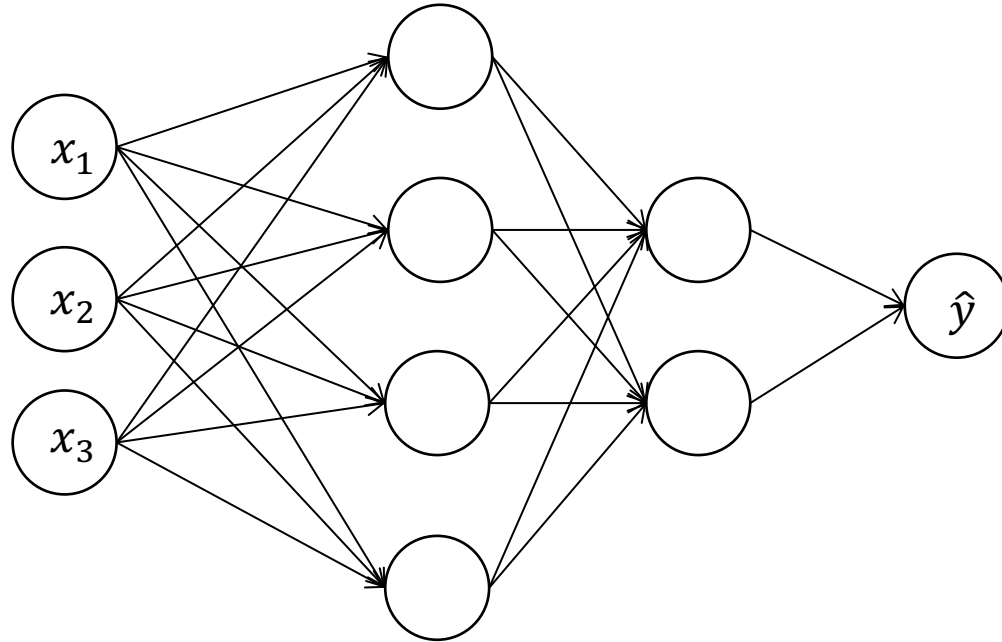


Overfitted

Under- and overfitting & capacity

- *Underfitting*: large $P(w, b, x^{\text{train}}, y)$
- *Overfitting*: large $P(w, b, x^{\text{test}}, y) - P(w, b, x^{\text{train}}, y)$
- *Capacity*: a model's ability to fit a wide range of functions

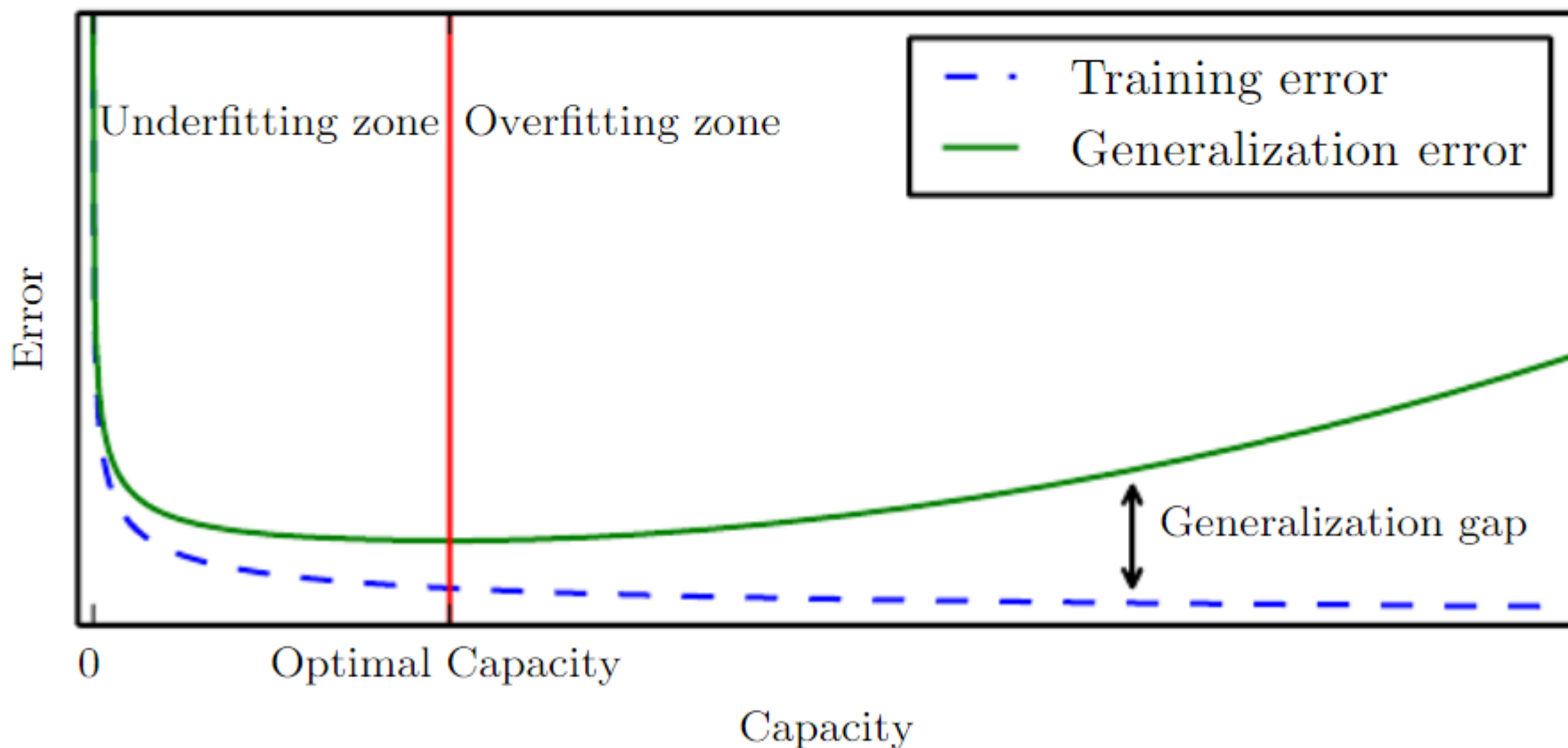
Capacity of your DNN



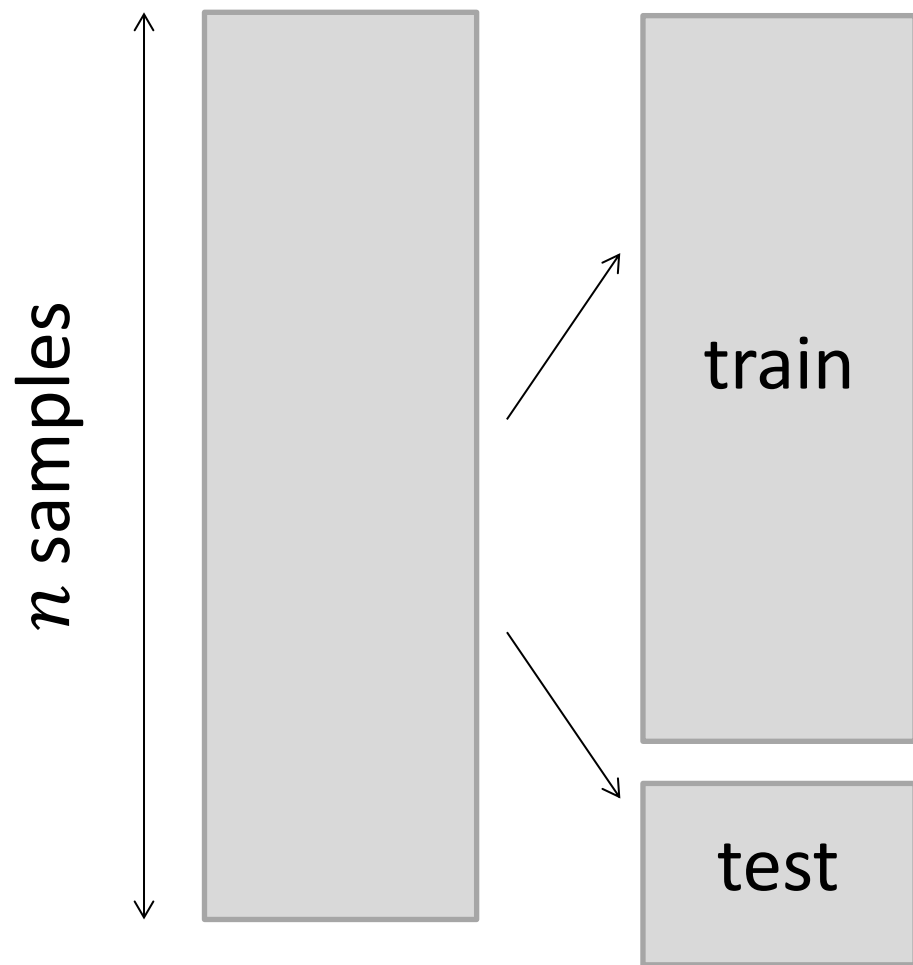
- How many layers?
- How many nodes in each layer?

→ *hyperparameter optimization*

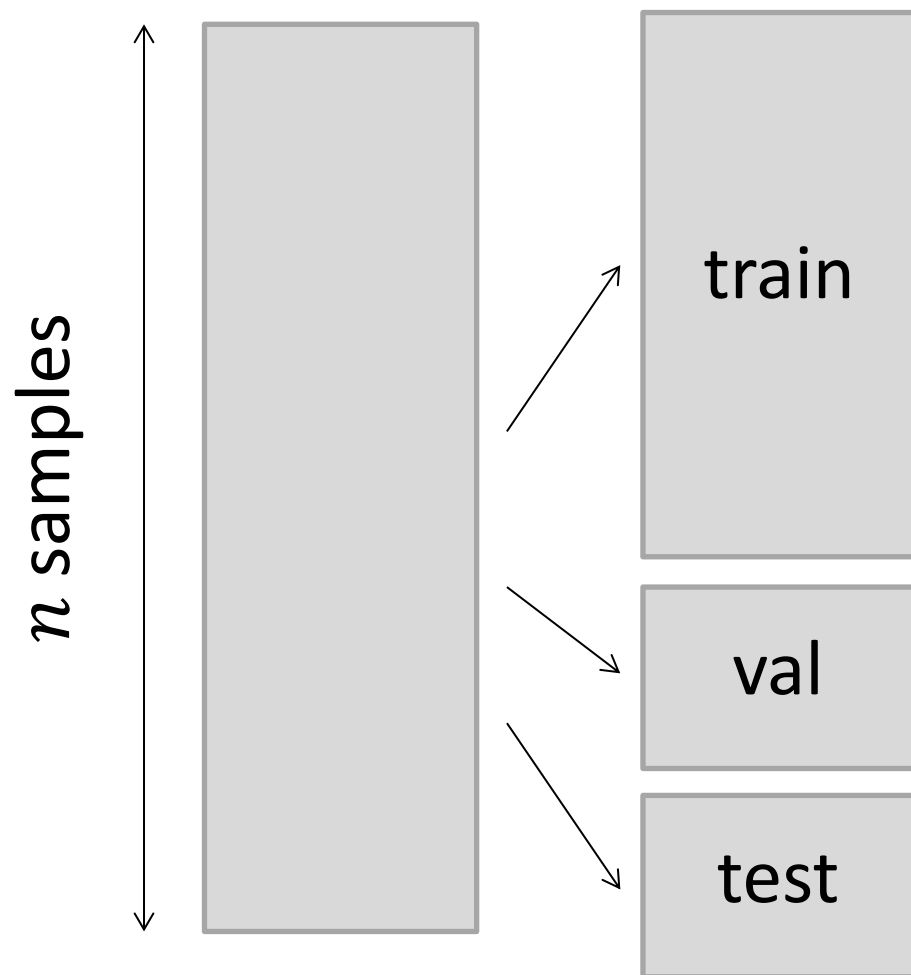
Under- and overfitting & capacity



Train and test



Train, validation and test



Hyperparameter optimization

Given a set of hyperparameters H .

Instance $h \in H$ describes no. and size of layers.

for $h \in H$ do:

train network by minimizing $P(h, w, b, x^{\text{train}}, y)$

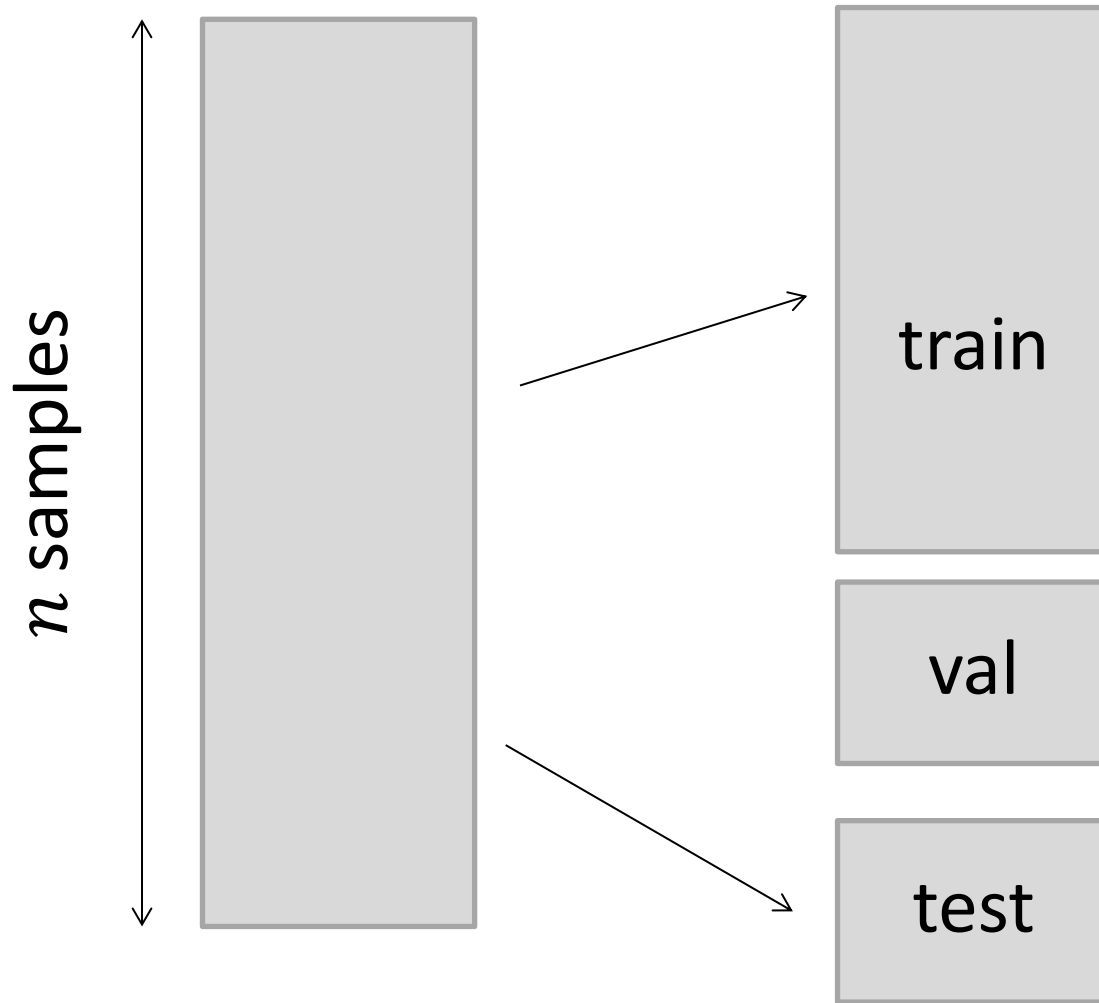
gives $P(h, w^h, b^h, x^{\text{val}}, y)$

choose $h^* = \operatorname{argmin}_h \{P(h, w^h, b^h, x^{\text{val}}, y)\}$

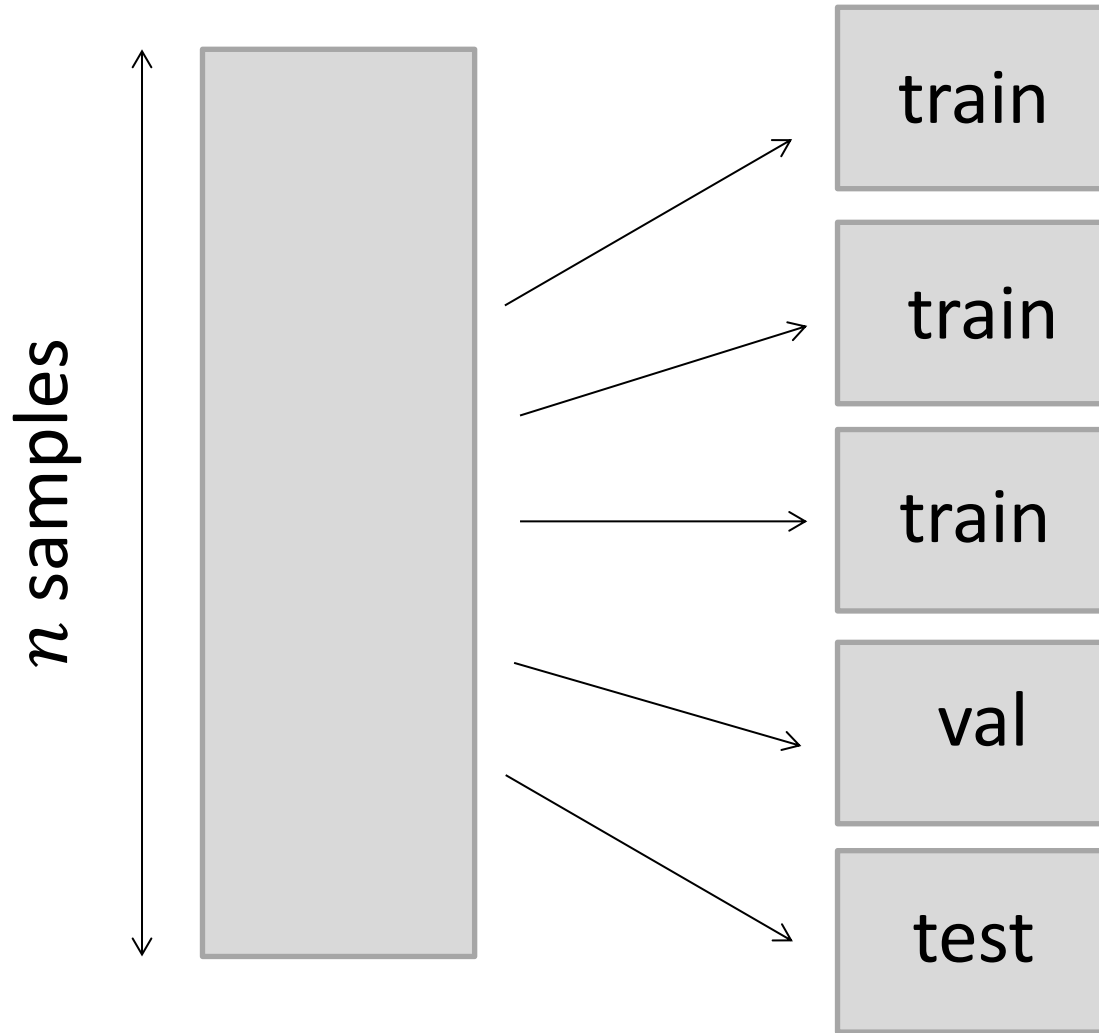
evaluate $P(h^*, w^{h^*}, b^{h^*}, x^{\text{test}}, y)$

(w^h and b^h are the optimized parameters for a network with hyperparameters h)

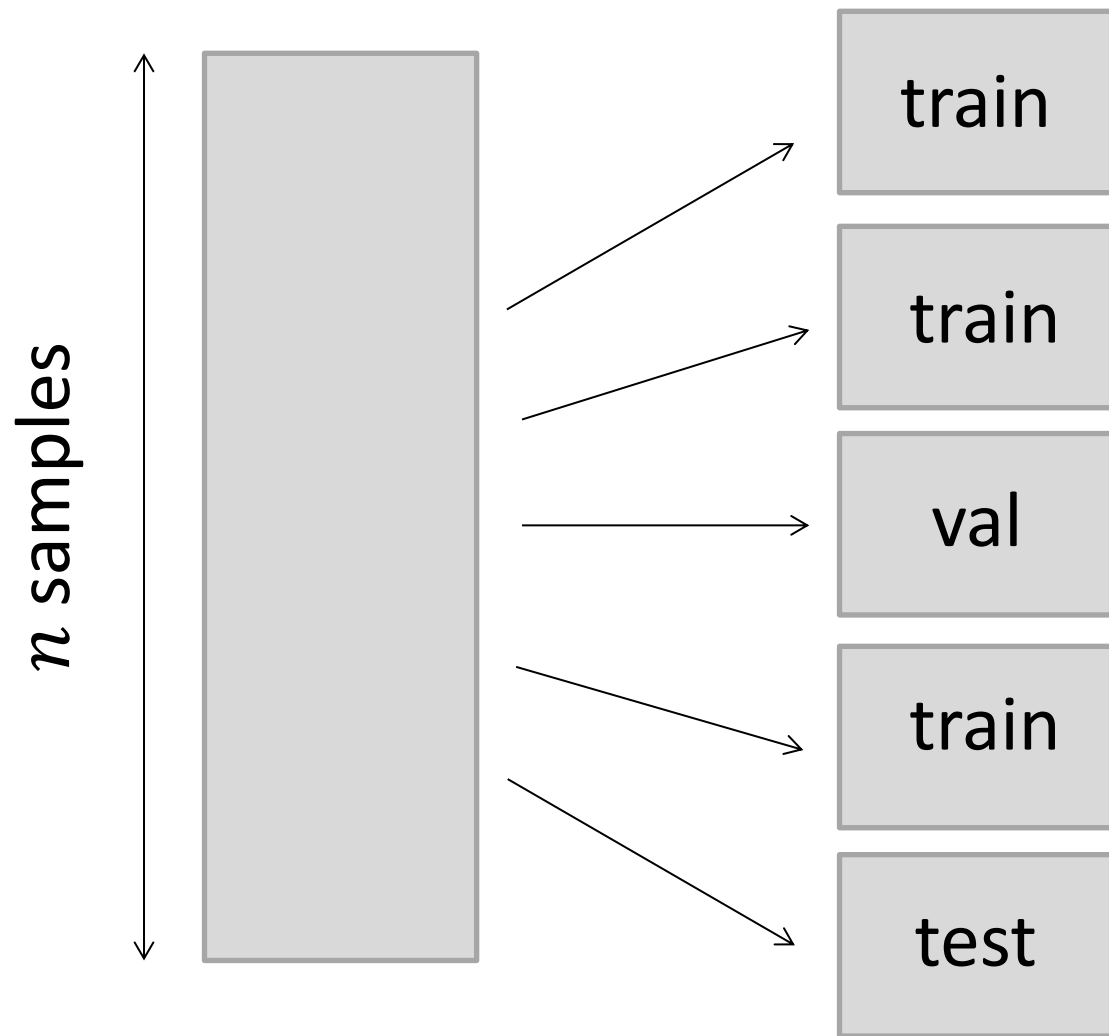
k -fold cross validation



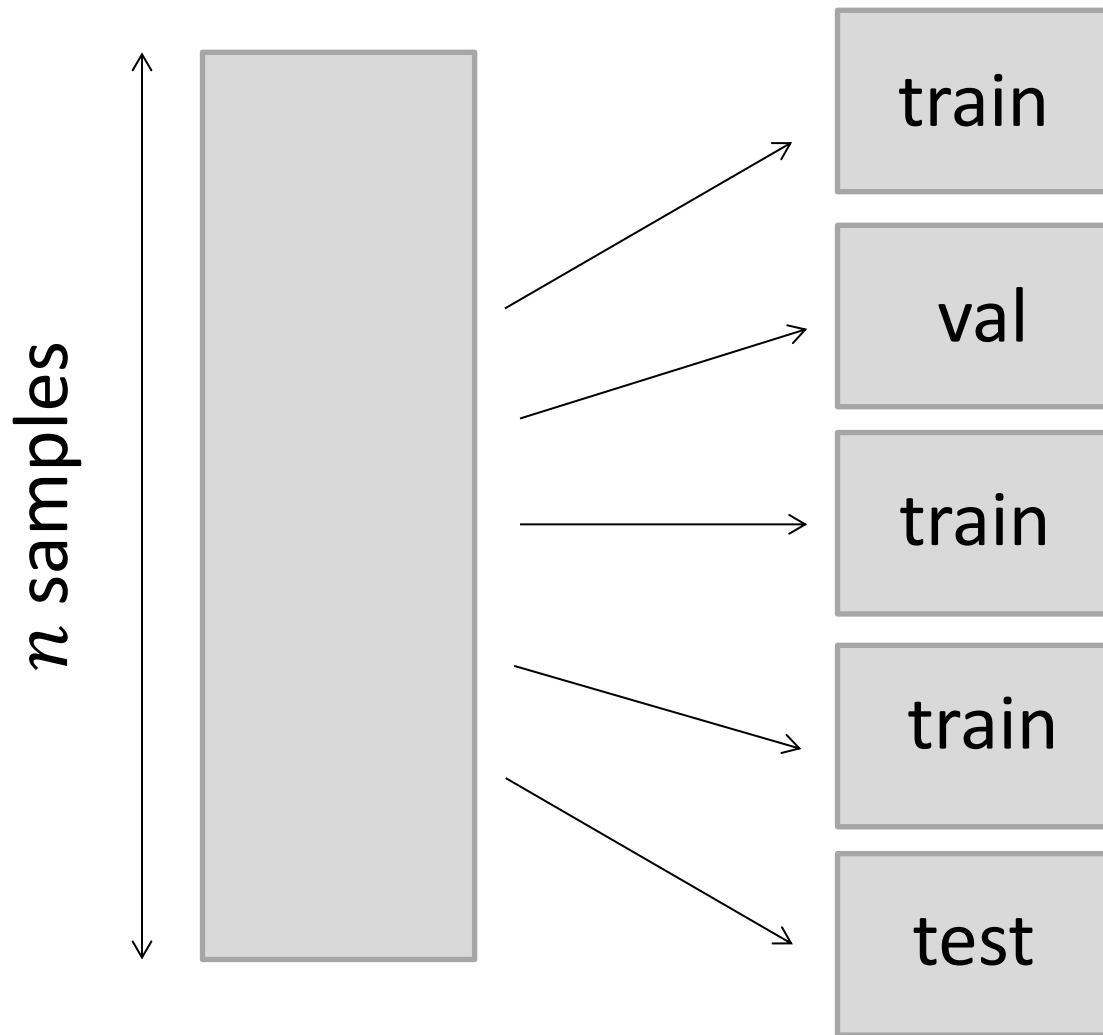
k -fold cross validation



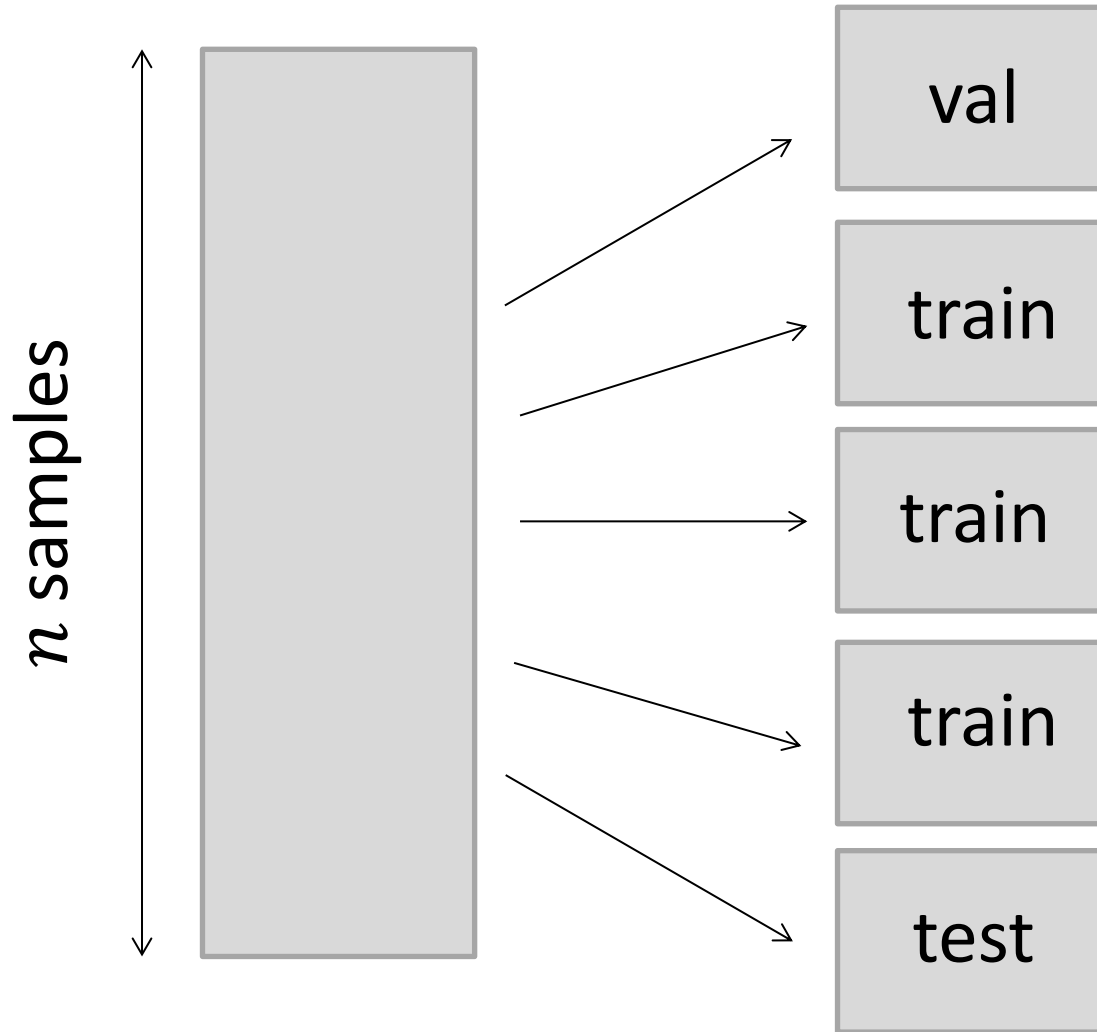
k -fold cross validation



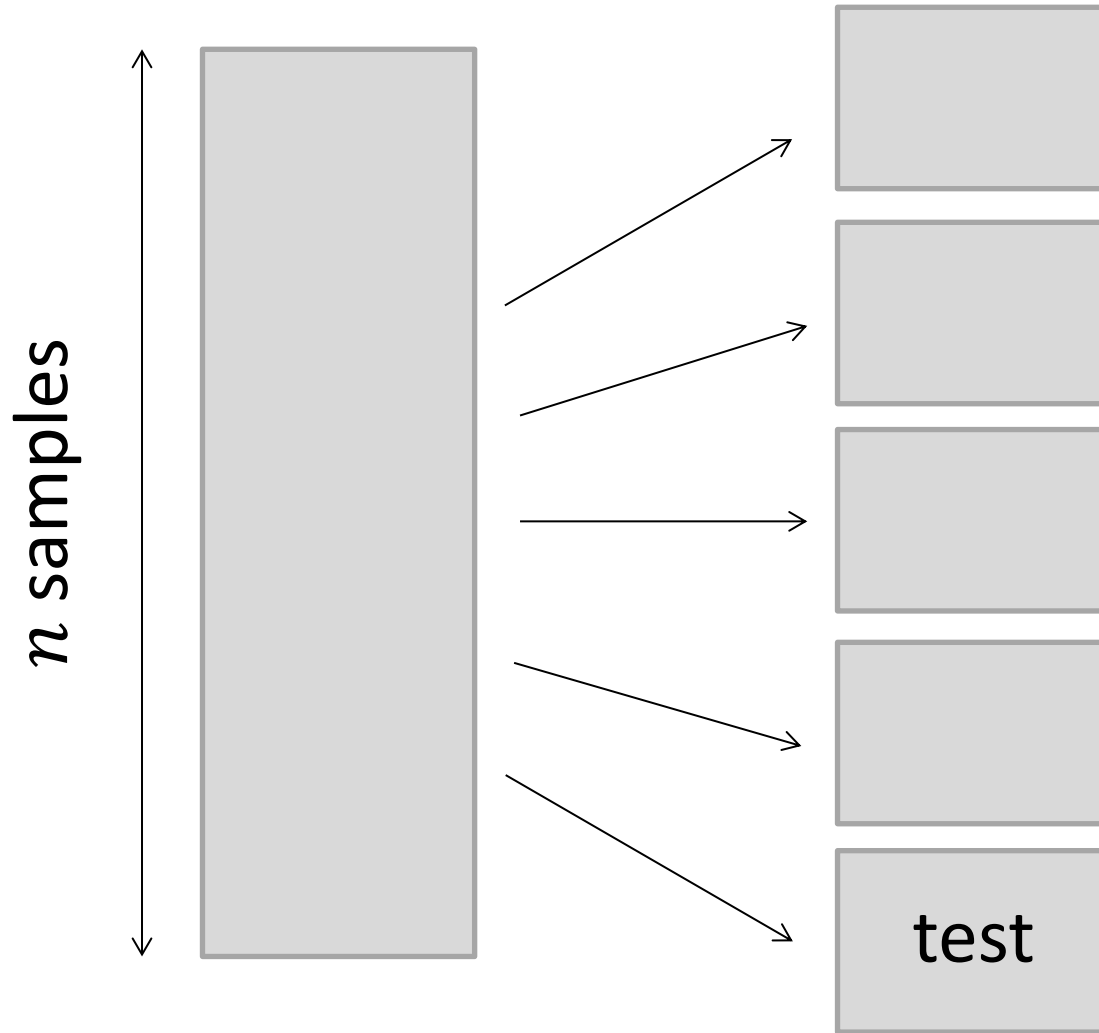
k -fold cross validation



k -fold cross validation



k -fold cross validation



- Train k times
- Different x^{val} for each fold
- Average P over the k folds

Why DNNs?

- Allows learning of complex functions
- Can deal with the curse of dimensionality
- Allows for parallelizing computations

When not to use DNNs?

- If a simpler model works at least as good
- If you know the underlying relationship (for example linear)

Dealing with overfitting

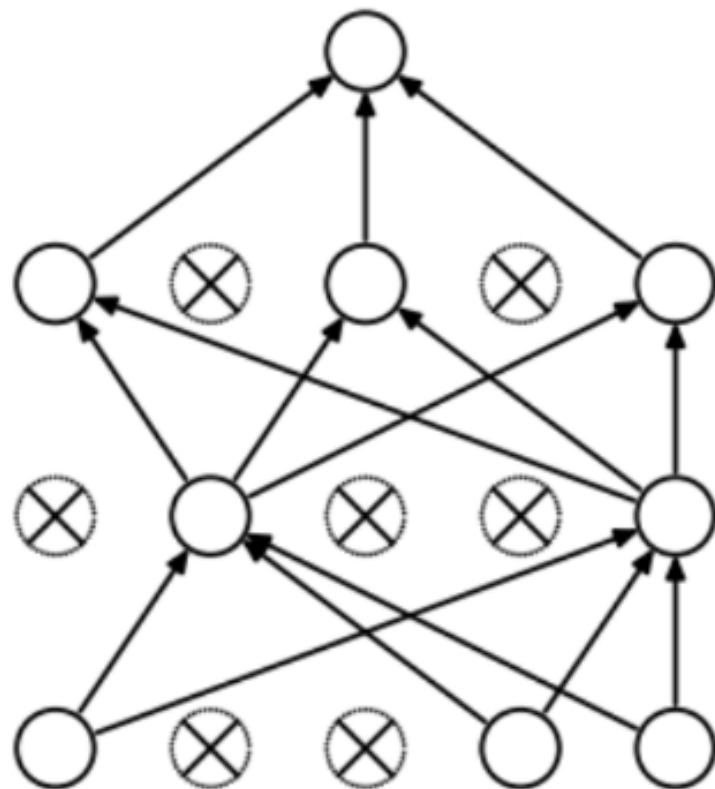
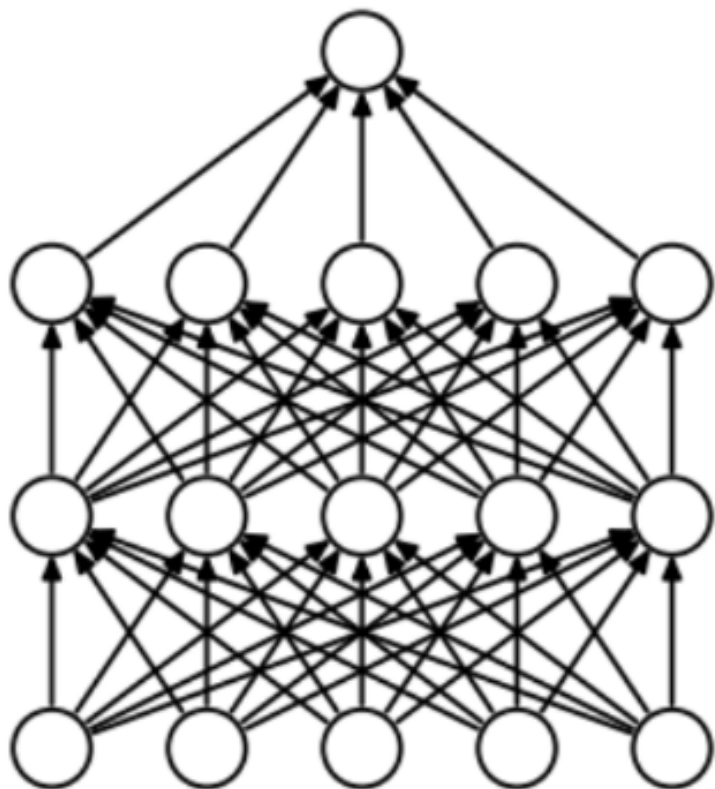
- Hyperparameter tuning
- Regularization

L1 and L2 regularization

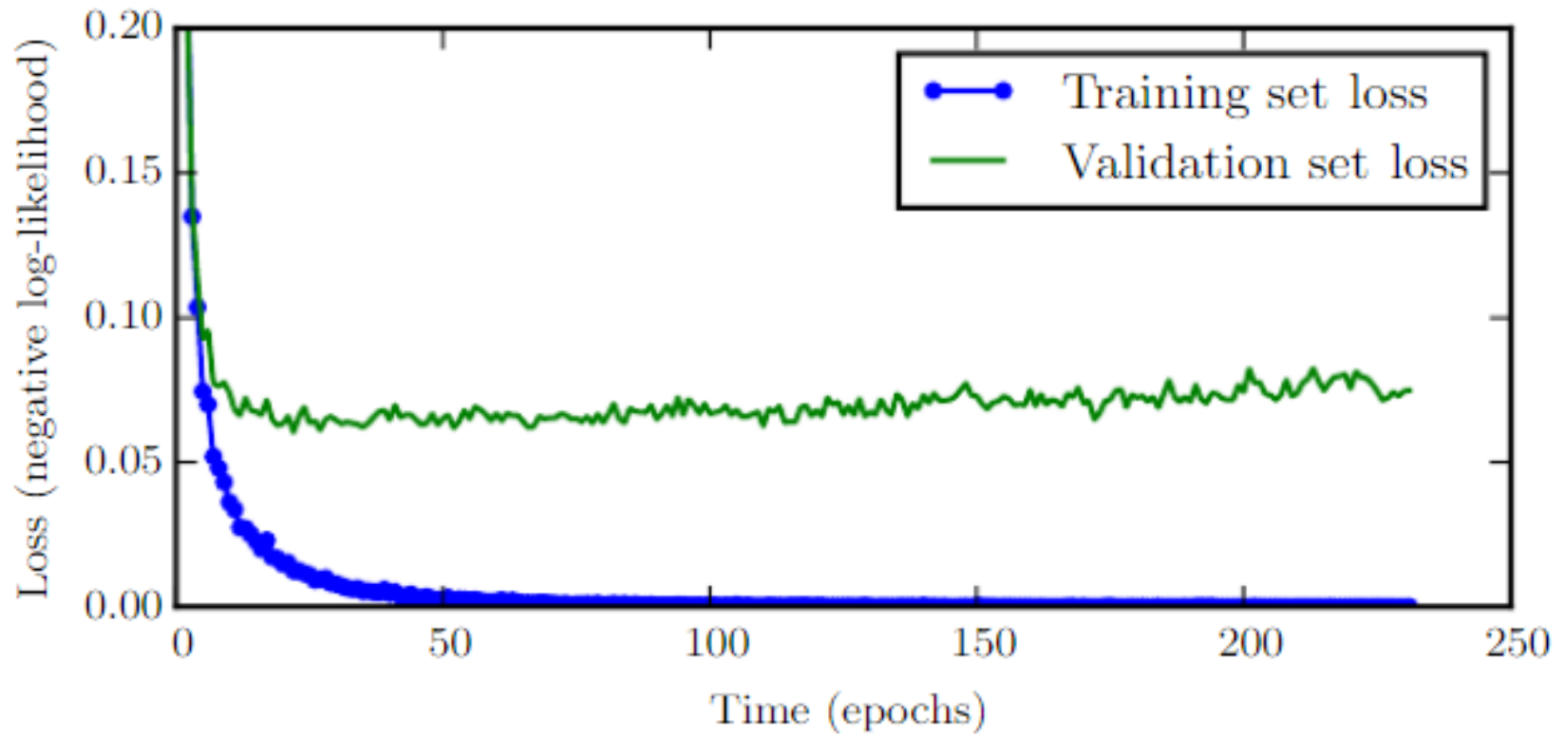
minimize $P(h, w, b, x^{\text{train}}, y) + \frac{\lambda}{m} L(w, b)$

- L1: $L(w, b) = |(w, b)|_1$
- L2: $L(w, b) = |(w, b)|_2$
- λ is a weighting factor
- m is the total number of parameters

Dropout



Early stopping



Summary

- What is machine learning?
- What is a perceptron?
- What is a (deep) neural network?
- Why use DNNs?
- What is underfitting, overfitting and capacity?
- How to train your model and optimize hyperparameters?
- How to use regularization to prevent overfitting?

Prepare for next week

- Install python
- Learn how to use python
- Read literature

Installing python (windows)

- Install miniconda (python3.6)

<https://conda.io/miniconda.html>

- Open the anaconda prompt
- Create an anaconda environment:

`conda create -n DLcourse`

- Activate the environment (not necessary after creating it, only the next time you start):

`source activate DLcourse`

- You're ready to go!

Installing python (linux)

- Install miniconda (python3.6)

<https://conda.io/miniconda.html>

- Open the command line
- Create an anaconda environment:

`conda create -n DLcourse`

- Activate the environment (not necessary after creating it, only next time you start):

`source activate DLcourse`

- You're ready to go!

Installing python (mac)

- Install miniconda (python3.6)

<https://conda.io/miniconda.html>

- Open the command line
- Create an anaconda environment:

`conda create -n DLcourse`

- Activate the environment (not necessary after creating it, only next time you start):

`source activate DLcourse`

- You're ready to go!

Learning python

- <https://www.learnpython.org/>
- <https://www.programiz.com/python-programming/tutorial>

Further reading

<http://neuralnetworksanddeeplearning.com> :

- Chapter 1 up to and including “Perceptrons”

<https://www.deeplearningbook.org/> :

- 5.1-5.3, 5.10-5.11
- 7.1, 7.8, 7.12