MASTER OF SCIENCE
IN ENGINEERING

Week 4: Overfitting, Model Selection, Curse of Dimensionality

TSM_DeLearn

Jean Hennebert
Martin Melchior

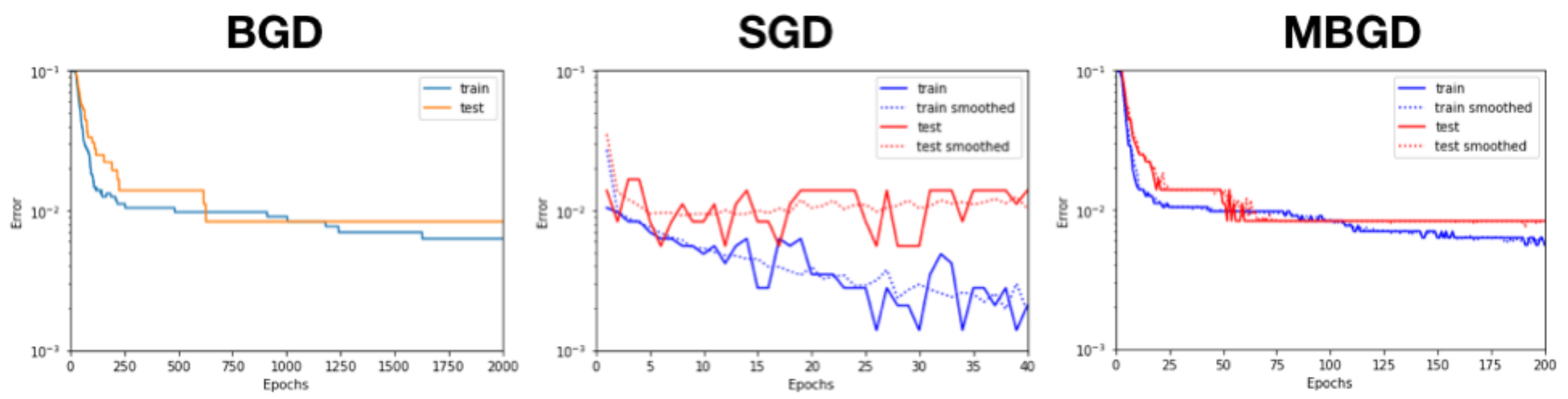
Overview

Recap from Last Week

Important Points to Resume from Last Week

SGD, MBGD

MBGD a compromise!
Tune batch size.

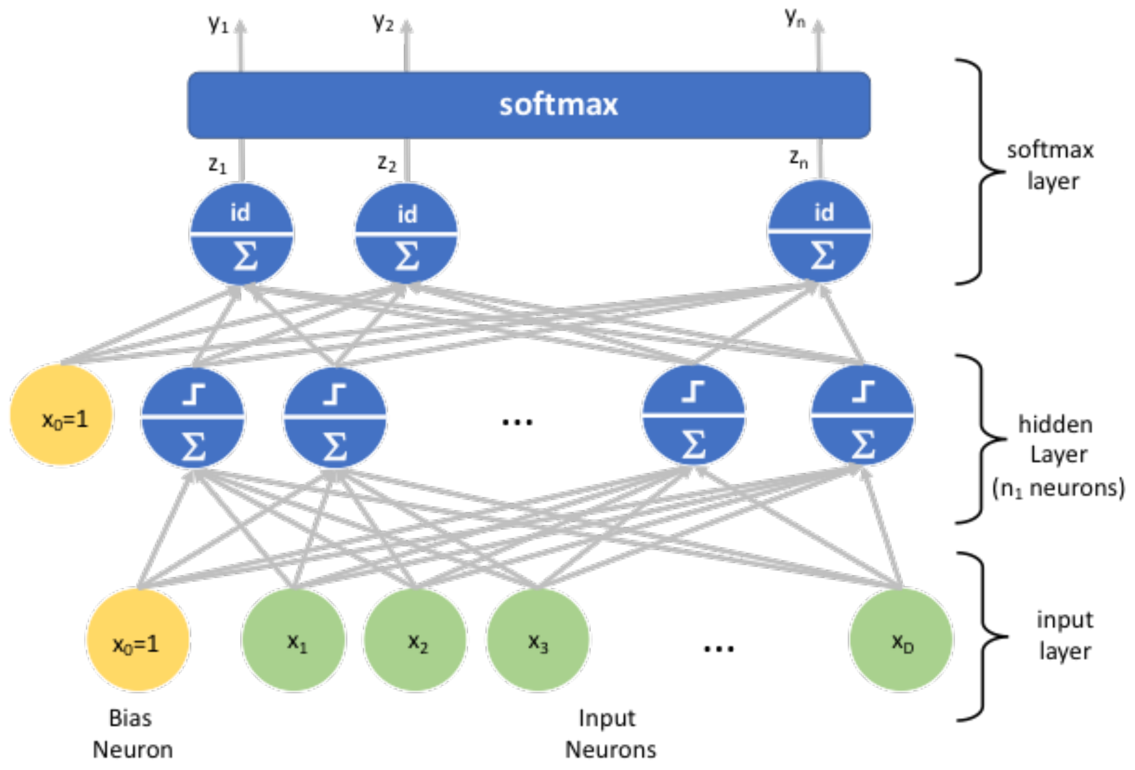


Softmax

Last layer for any n-class classification task.

MLP with Single Hidden Layer

“Shallow Networks”

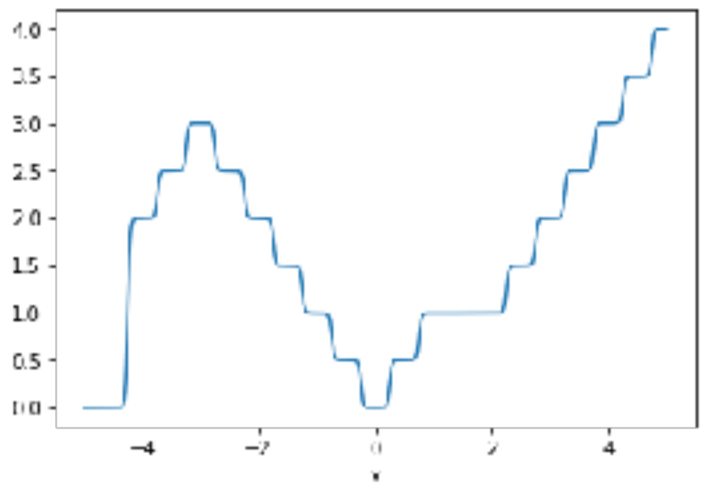


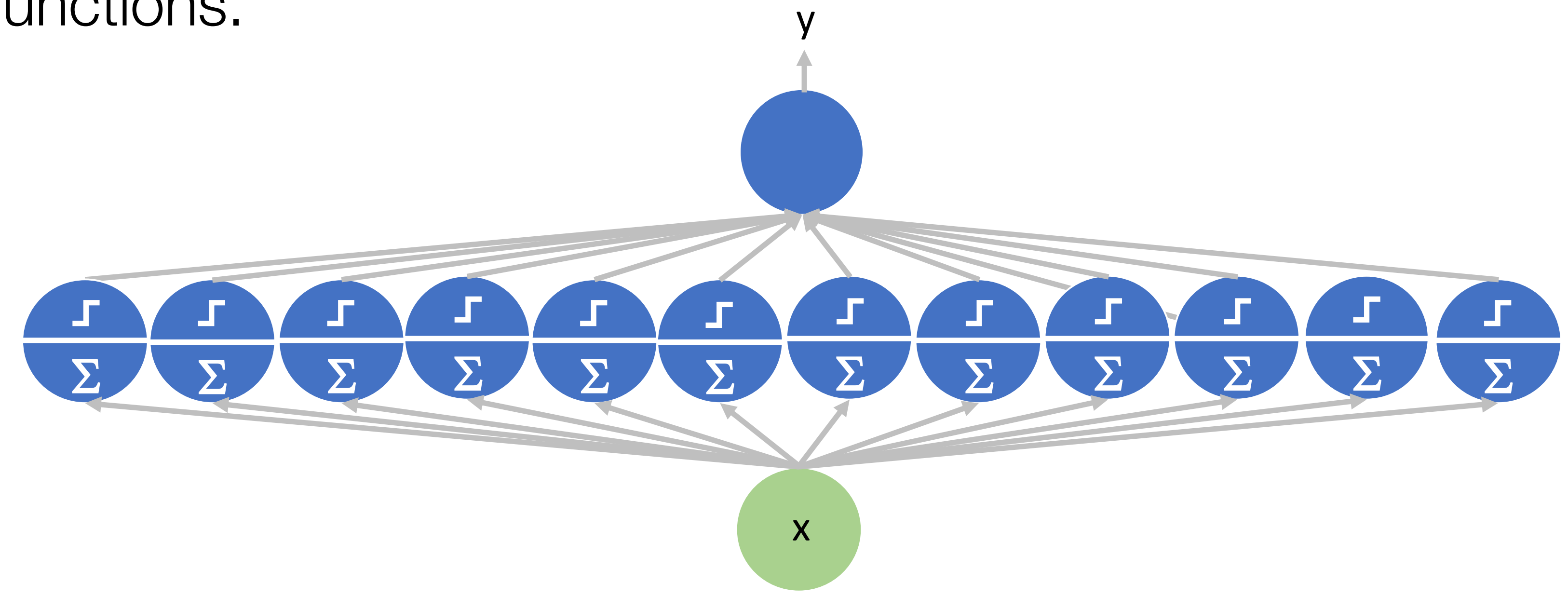
Important Points to Resume from Last Week

Universal Approximation Theorem

With shallow networks (single hidden layer) any function can be represented —
but in problems with high-dimensional input:

- an exponentially growing number of neurons is needed for obtaining better accuracy
- input data sampled on a sufficiently fine grid is needed which becomes infeasible in high-dimensional spaces such as with image or audio data ("curse of dimensionality").
- we danger to significantly overfit on the training data when using the available data and interpolate with step-functions.



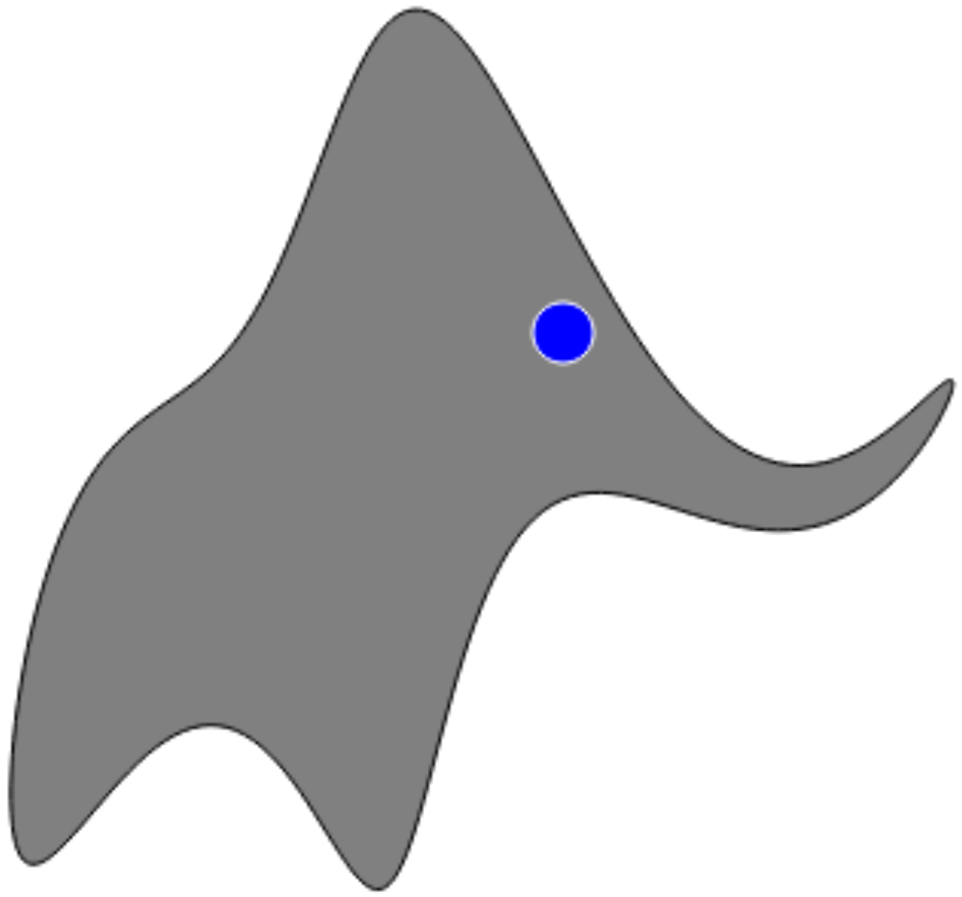


Plan for this week

- Explain overfitting and generalisation error.
- Learn the tools how to analyse and optimise them:
    - Hyper-Parameter tuning and model selection
    - Performance measures
- Get more insight into the curse of dimensionality and understand why deep architectures may help.

Overfitting and Generalisation

Typical Learning Curves for training and test data
Factors triggering overfitting
Example

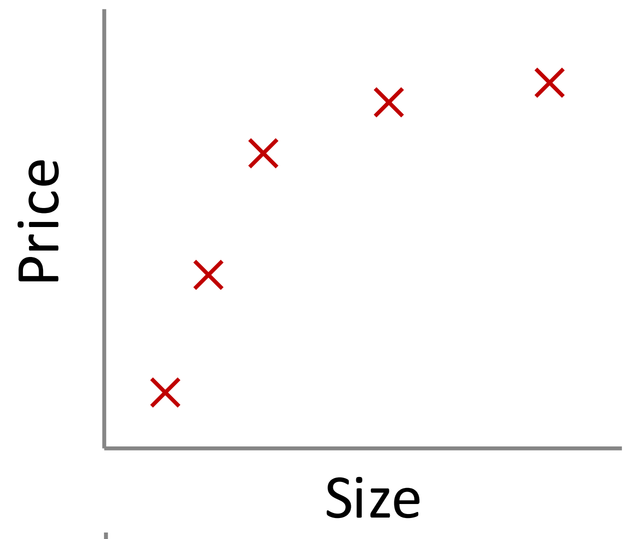


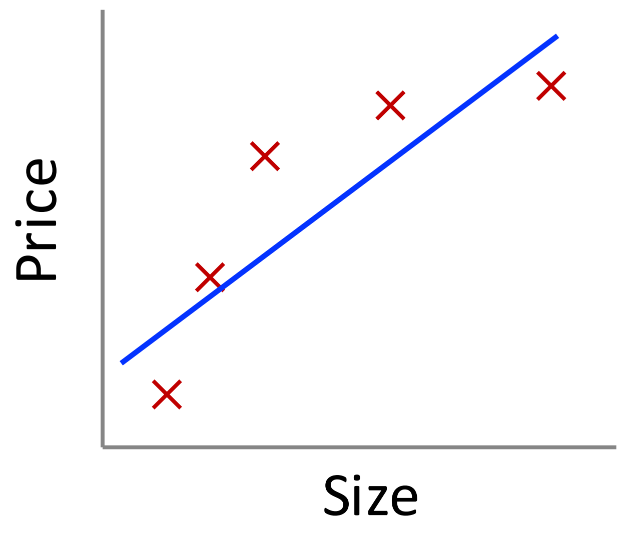
http://theoval.cmp.uea.ac.uk/~gcc/projects/elephant/

"A turning point in Freeman Dyson's life occurred during a meeting in the Spring of 1953 when Enrico Fermi criticized the complexity of Dyson's model by quoting Johnny von Neumann [1] `With four parameters I can fit an elephant, and with five I can make him wiggle his trunk'"

What is Overfitting?

Consider the data points with house prices vs size depicted on the right. What is a suitable model for prediction? The three models below (linear, quadratic, 4th order polynomial) can fit the given training data points with different accuracy (leading to different cost).

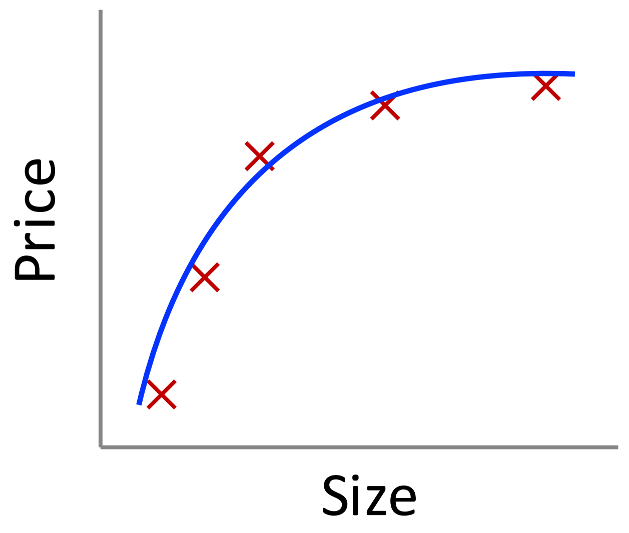




$$\theta_0 + \theta_1 x$$

Underfitting - High Bias

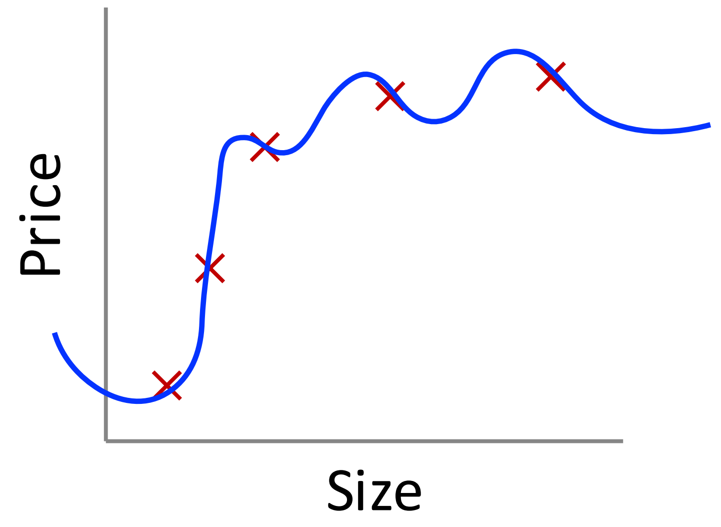
Strong bias in the way the data deviate from the linear model.



$$\theta_0 + \theta_1 x + \theta_2 x^2$$

Good Fit - “Just Right”

Model seems to capture just right the underlying structure in the data.

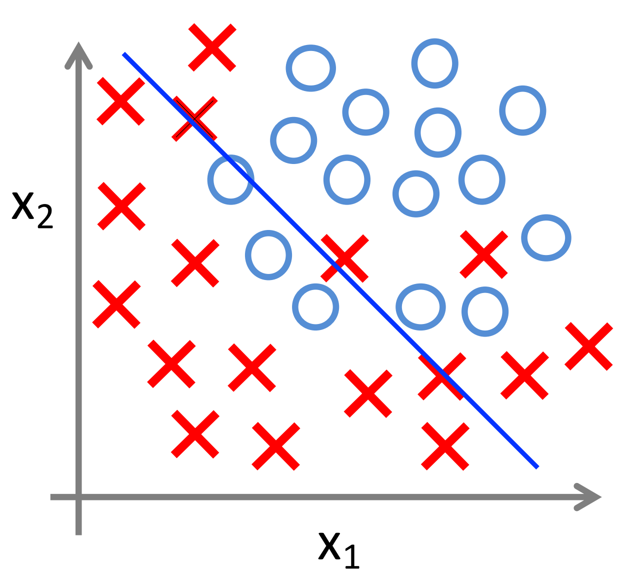


$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Overfitting - High Variance

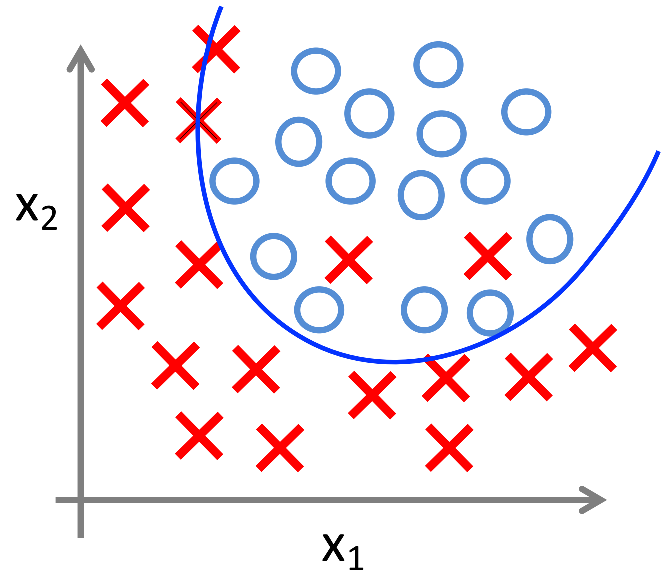
Model matches samples perfectly: too well given the number of samples; seems to capture also statistical fluctuations.

Overfitting - Binary Classification Example

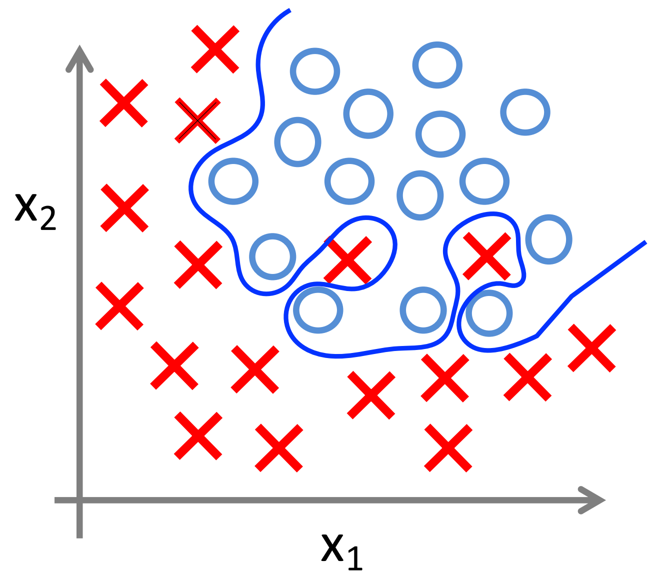


$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2$
$+\theta_3 x_1^2 + \theta_4 x_2^2$
$+\theta_5 x_1 x_2)$



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2$
$+\theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2$
$+\theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

Underfit
“high bias”

Good fit
“just right”

Overfit
“high variance”

Occam’s Razor “Among competing hypothesis the simplest is the best.”
William of Ockham (1285-1347)

Definition Overfitting

Overfitting occurs when the learned hypothesis (trained model) fits the training data set very well - but fails to generalise to new examples.

Overfitting can occur when

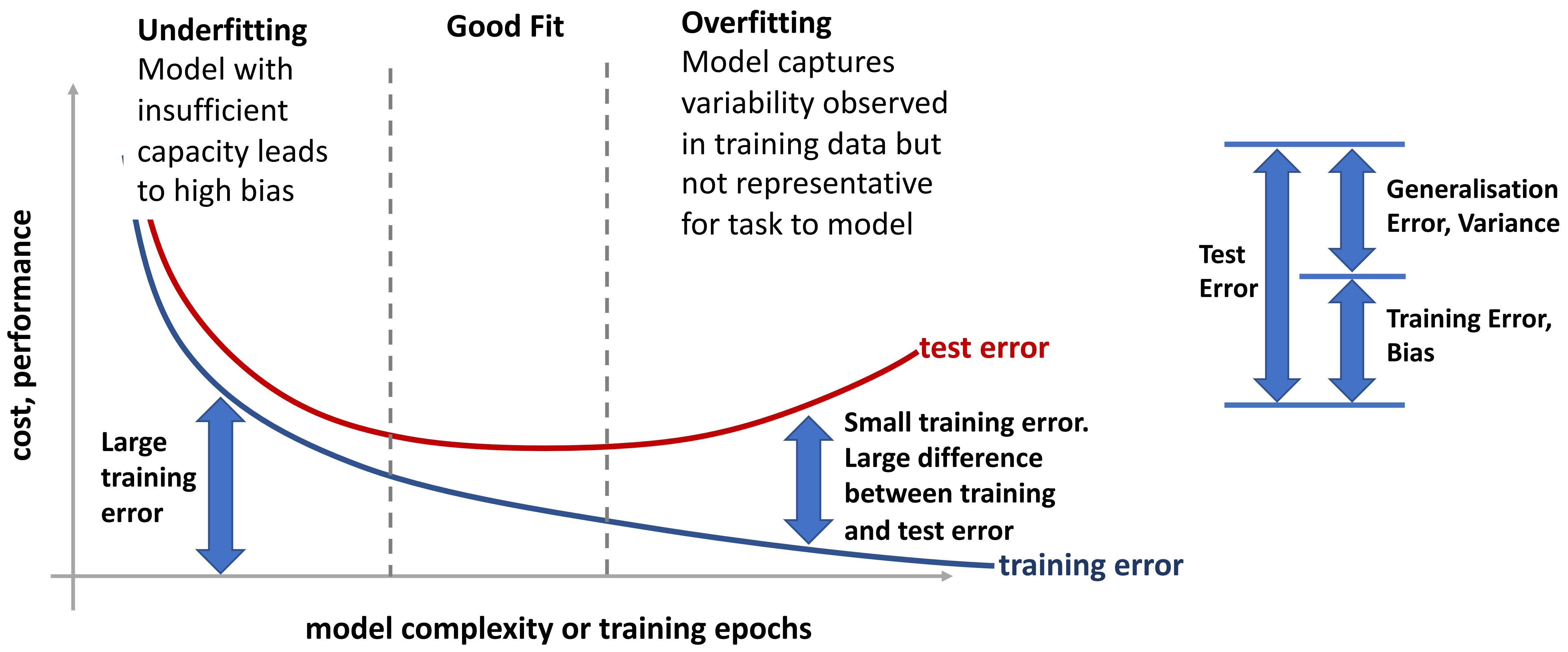
- the training set is too noisy, or
- its size is too small in comparison with the dimensionality of the input data (which introduces sampling noise), or
- the number of parameters of the model is too large, i.e. the model is too "flexible".

Then, the model is likely to detect patterns in the noise itself. These patterns will not generalise to new examples.

How to Examine Overfitting?

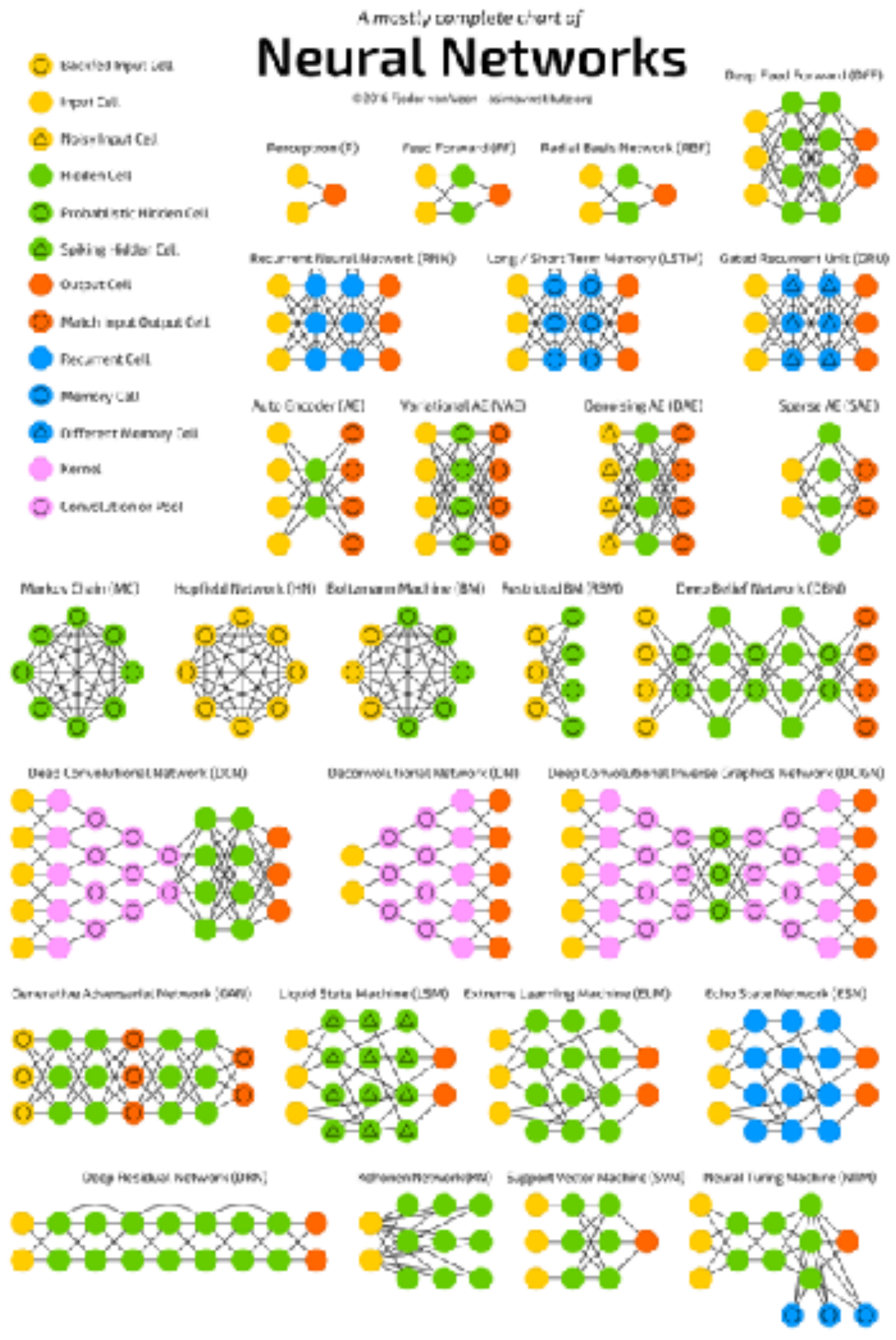
Evaluate the performance on the training and the test set

- for different models (with different model complexity) or
- after different number of training epochs, or
- by using different training set sizes.



Model Selection Process

Hyper-Parameter Tuning
Cross-Validation



Model Selection

Goal

Select from a family of models the model with the best performance.

Problem

To achieve this goal we need to evaluate the performance for different models (e.g. models with different complexities). But to avoid overfitting on the test set we must not use information from the test set to determine settings (hyper-parameters) of the model.

Solution

Further split the original training set into a two subsets:

- Training Set: Used for training
- Validation Set: Used for evaluating the the performance and model selection

Hyper-Parameters

Definition

Hyper-Parameters specify higher level properties of the mapping function (model) and/or the learning process. These parameters are optimised on the validation set.

Examples

- Learning rate
- Batch size
- Number of hidden layers in a neural network
- Number of neurons in a layer of a neural network
- …etc. (more to come)

Split Data in Training, Validation, Test Sets

Split in three disjoint datasets:

		typical split ratio	
		small datasets	large datasets
Training Set	Subset of *trustable* data used to train the model (choose the parameters that lead to a small cost). Should not be used for evaluating the model.	60%	98%
Validation Set	Subset of *trustable* data used to select models, for example by selecting the best *hyper-parameters* of the model.	20%	1%
Test Set	Subset of *trustable* data used to measure the performance of the finally selected model.	20%	1%

With *trustable* we mean

- composed of data acquired under the same conditions,
- including the same characteristics (range of values, distribution, etc),
- sufficiently large to have confidence in the parameter estimates or evaluation metrics.

Learning Curves

For a given fixed dataset, we consider different split ratios (fraction of samples used for training, the rest used for validation). Then, we expect

- the ***training cost or training error*** (=fraction of wrongly predicted samples on training set) to be increasing in the split ratio.
- the ***validation cost or validation error*** (fraction of wrongly predicted samples on validation set) to be decreasing in the split ratio; actually, the validation error becomes very wiggly for large split ratios.

$\Bigg\}$ **Explain Why**

These two curves are referred to as ***learning curves***. These can be used to do model selection and hyper-parameter tuning.

Learning Curves

For a given fixed dataset, we consider different split ratios (fraction of samples used for training, the rest used for validation). Then, we expect

- the ***training cost or training error*** (=fraction of wrongly predicted samples on training set) to be increasing in the split ratio.
- the ***validation cost or validation error*** (fraction of wrongly predicted samples on validation set) to be decreasing in the split ratio; actually, the validation error becomes very wiggly for large split ratios.

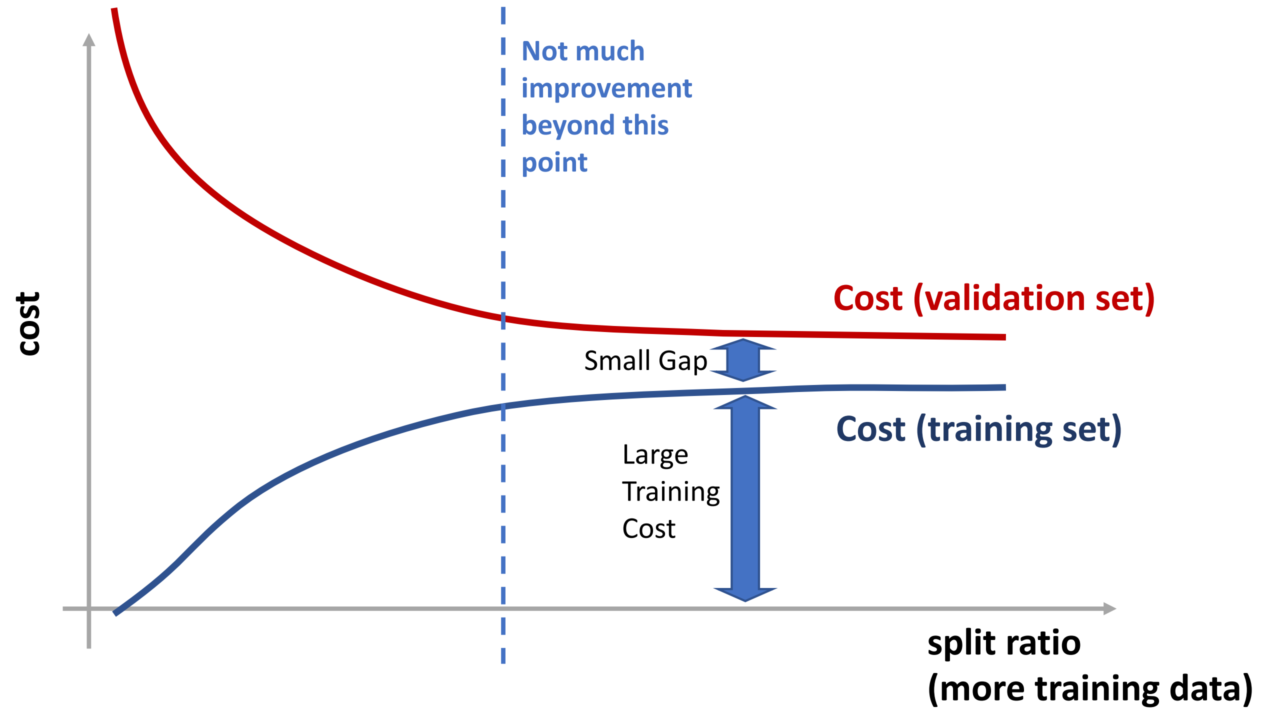
These two curves are referred to as ***learning curves***. These can be used to do model selection and hyper-parameter tuning.

with more data it gets more difficult to perfectly fit a model

the model trained with more data captures more details about the underlying problem

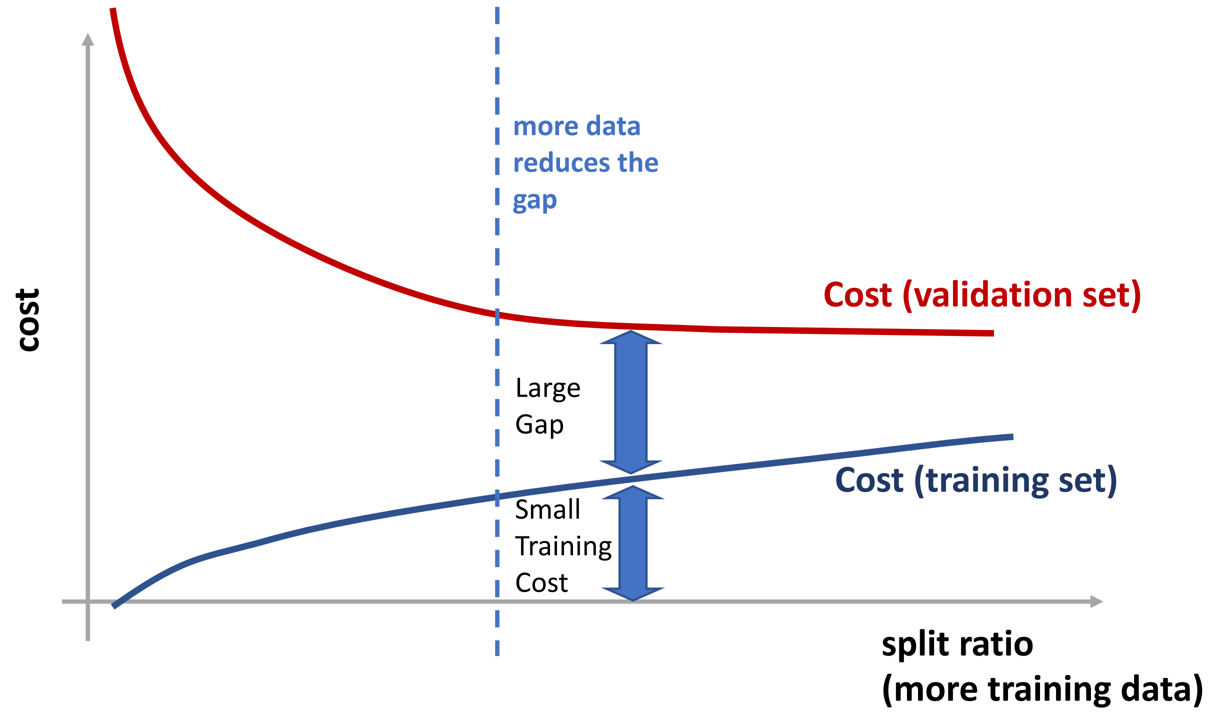
Learning Curves Analysis

High Bias, Underfitting



Example: MNIST with just softmax

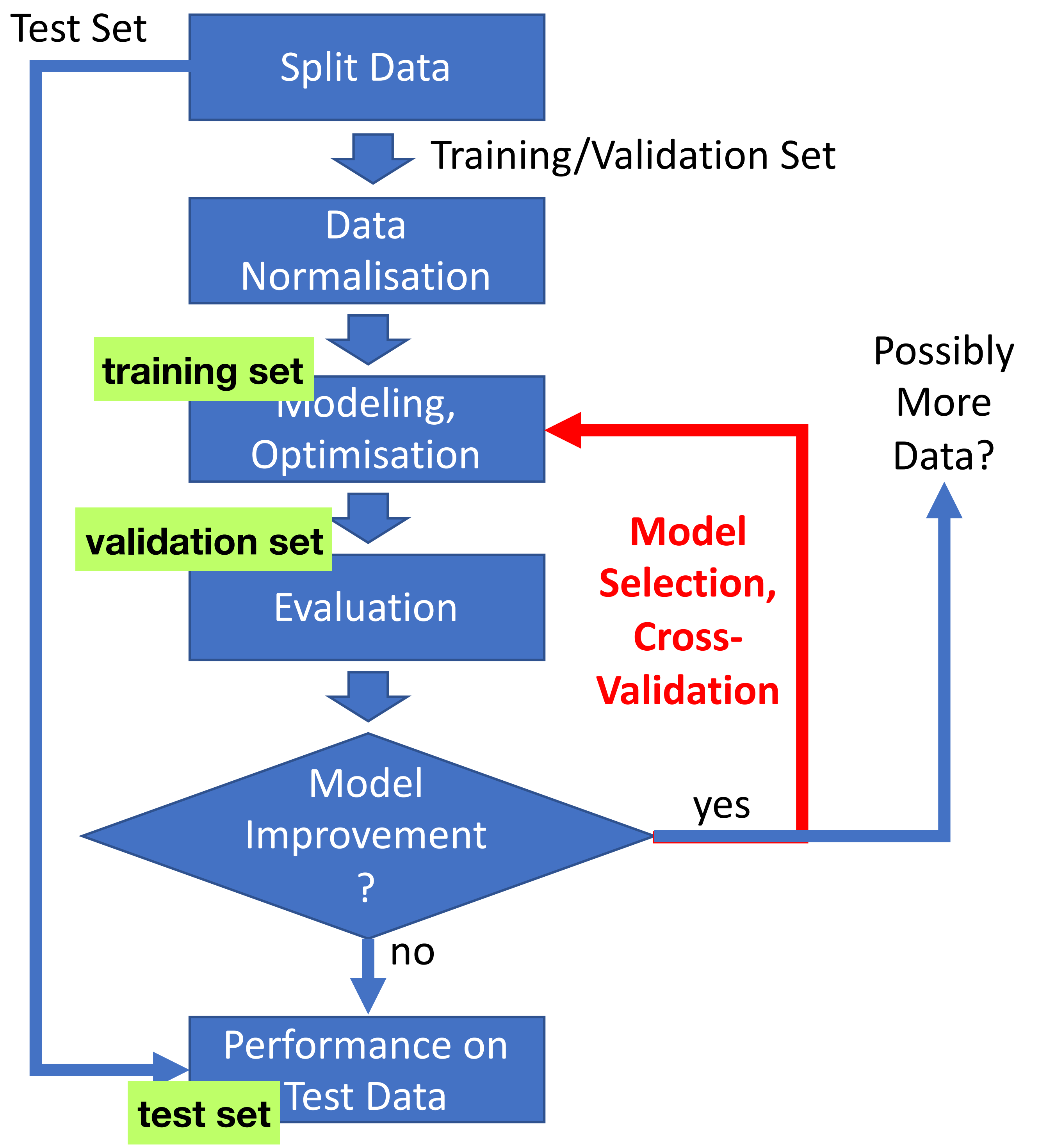
Low Bias, Overfitting



Example: Using too little training data with MNIST

See also: https://www.dataquest.io/blog/learning-curves-machine-learning/

Standard Process for Neural Networks Modelling



k-fold Cross-Validation

See ML Course

Procedure for making efficient use of the data for more robust estimation of validation set performance; it is particularly useful if data is scarce.

Algorithm

- Randomly shuffle the dataset.
- Split dataset into test set (D_{test}) and the rest (D_0)
- Split D_0 into k equally sized folds.
- For each fold:
  - Use the selected fold as validation set and the remaining folds as training set.
  - Fit the model on the remaining folds.
  - Evaluate the fitted model with the selected validation fold.
  - Retain the evaluation score (performance)
- Report the performance as average of retained evaluation scores.

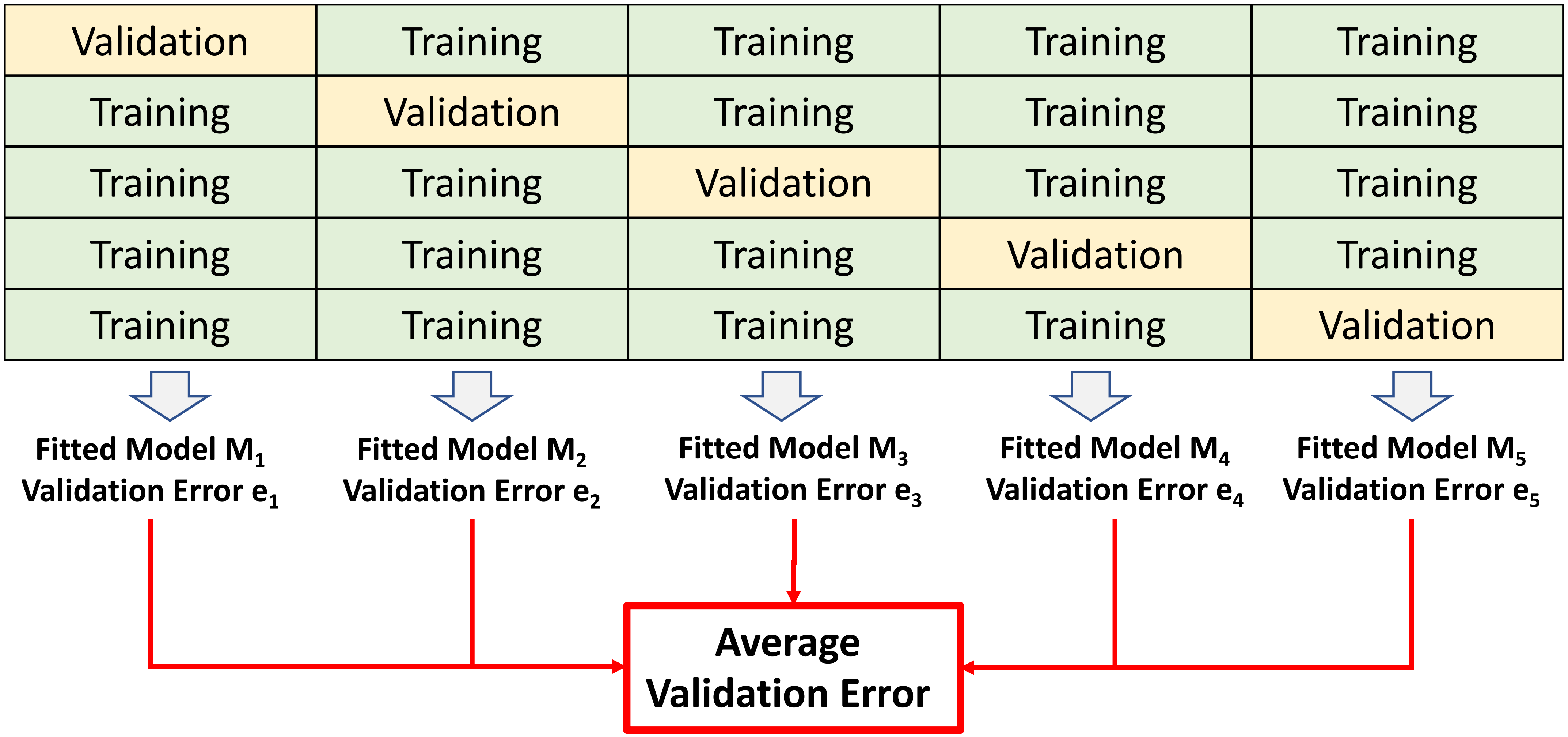
- Typical choices for k: 5 or 10
- Each sample in D_0 is used exactly once for validation.

Integrated into Model Selection Process

- Select the model, the hyper-parameters leading to the best performance
- Train it with the combined training and validation set
- Report its performance on the basis of the test set.

k-fold Cross-Validation (k=5)

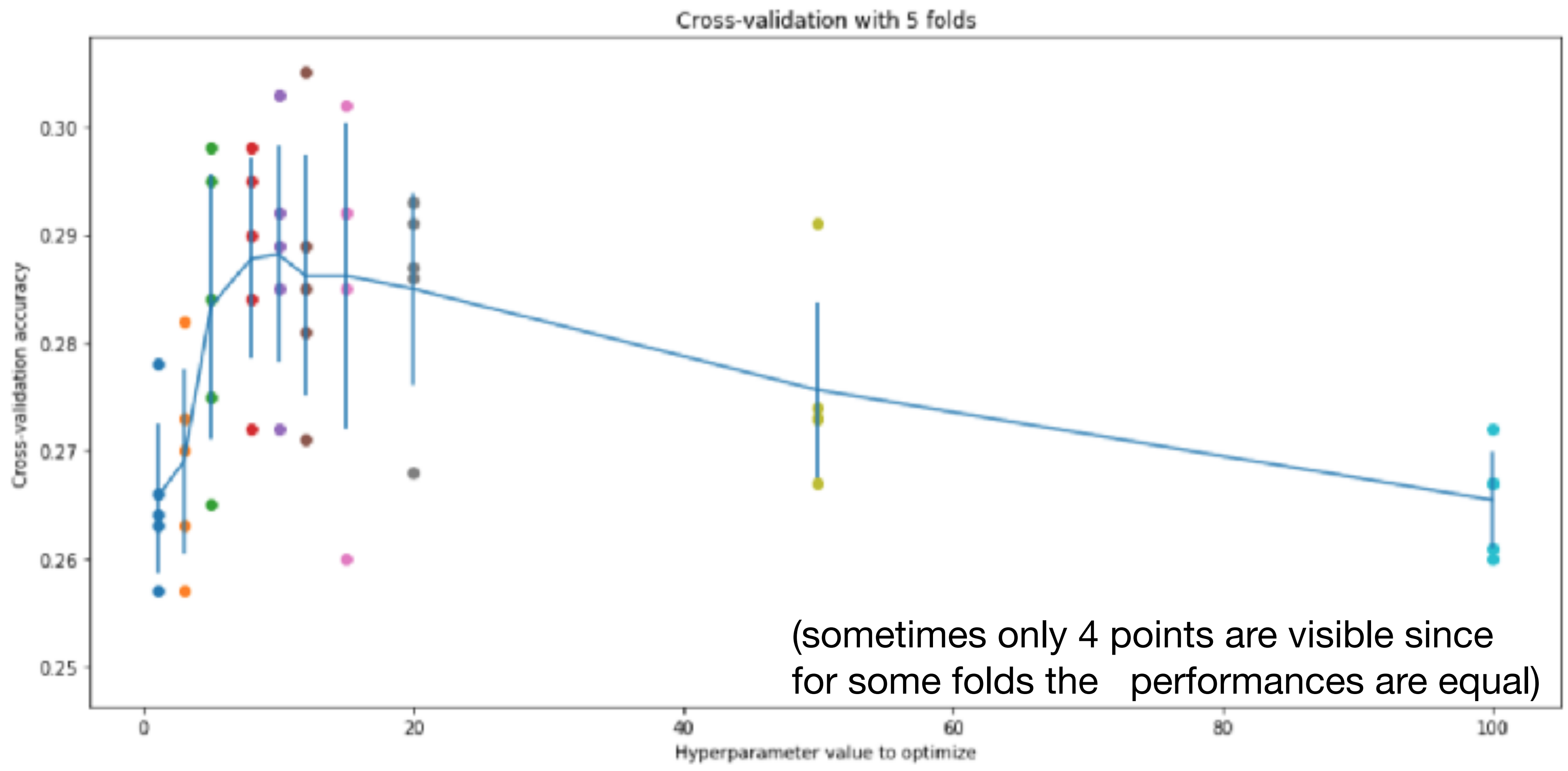
See ML Course



Validation	Training	Training	Training	Training
Training	Validation	Training	Training	Training
Training	Training	Validation	Training	Training
Training	Training	Training	Validation	Training
Training	Training	Training	Training	Validation

Fitted Model M_1 Validation Error e_1
Fitted Model M_2 Validation Error e_2
Fitted Model M_3 Validation Error e_3
Fitted Model M_4 Validation Error e_4
Fitted Model M_5 Validation Error e_5
Average Validation Error

Benefits of Cross Validation



**Example
knn on cifar**

For selected hyper-parameter values, the performance for each of the 5 folds is computed. The spread on a given hyper-parameter value shows the sensitivity of the performance metric on the split of the data and it gives a feeling about the confidence of the performance estimates.

Example: Hyper-parameter = 20 —> performance varies between 26.8% and 29.3%.

Performance Measures

Confusion Matrix
Error Measures

Confusion Matrix

A **confusion matrix** measures the performance of a classification system on a per-class basis by indicating the number of samples of actual class **'a'** predicted as class **'b'**. The rows relate to the actual class labels **'a'** and the columns to the predicted class labels **'b'**.

- Very useful to understand the type of errors the system is doing
  - Which class is confused with what?
  - Are the errors "understandable"
  - A good basis to analyse possibly bad predictions of the system.

In the example, the orange box indicates that 63 samples of total 221 of actual class **b** (8+131+63+19) are predicted as class **c**.

		predicted class			
		a	**b**	**c**	**d**
actual class	**a**	120	21	7	8
	b	8	131	63	19
	c	12	30	80	11
	d	1	11	8	40

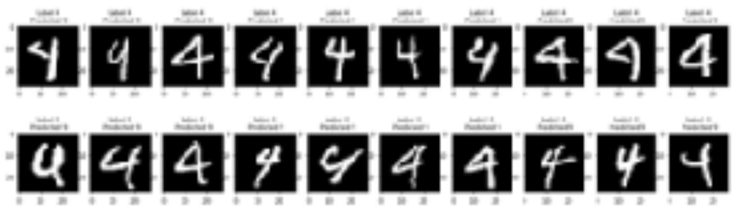
Example Original MNIST

Model used for the predictions: Simple softmax regression.
Evaluated on the test set (N_{test}=10’000).

```
from sklearn.metrics import confusion_matrix

y_pred = …
y_actual = …
confmat = confusion_matrix(y_actual, y_pred)
confmat
```

	0	1	2	3	4	5	6	7	8	9
0	902	0	3	3	3	11	5	4	3	2
1	0	1119	10	4	1	8	1	1	14	5
2	3	12	872	15	13	5	15	13	27	7
3	3	9	25	896	0	50	2	13	24	16
4	3	0	6	1	858	3	10	5	8	54
5	8	5	10	20	9	809	16	6	29	9
6	1	1	6	1	4	14	978	1	4	3
7	6	2	11	3	6	1	1	957	0	42
8	3	18	11	16	6	27	9	4	867	17
9	5	4	3	8	22	9	0	27	2	912



For example, we can extract the following information:

- The system seems to perform
  - better at predicting ‘0’, ‘1’ and ‘6’ (96.4%, 96.2%, 96.5%) — but
  - worse at predicting the digits ‘2’ (88.3%), ‘3’ (86.3%), ‘5’ (87.8%) and ‘8’ (88.7%).
- The system seems to most often misclassify
  - a ‘4’ as a ‘9’ (5.9% of all the ‘4’),
  - a ‘3’ as a ‘5’ (4.8% of all the ‘3’)
  - a ‘7’ as a ‘9’ (4.1%).

Overall Accuracy and Error Rate

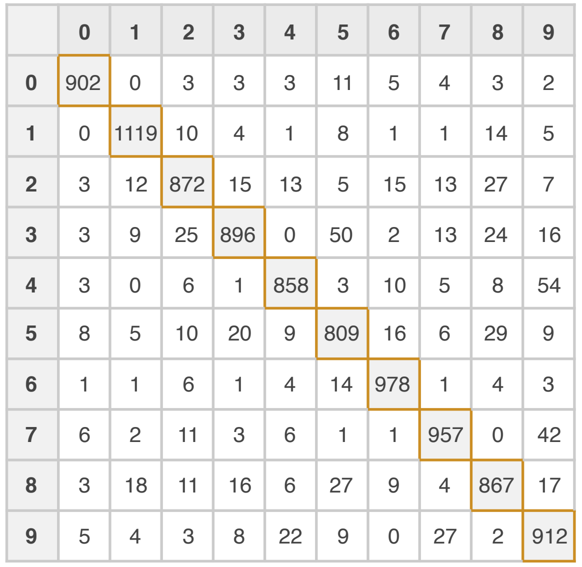
The **overall accuracy** of the system is the percentage of correct classification.
The **error rate** = 1 - accuracy

- Could give a biased view of the performance in the case of strongly un-balanced classes.
  - For example, a 95.5% accuracy system is not very good if one class represents 95% of the population.
- Can also be computed from the confusion matrix:

$$\text{accuracy} = \frac{\sum \text{diag elements}}{\#samples}$$

- Example (Original MNIST, softmax)

$$\begin{aligned} \text{accuracy} &= 9170/10000 = 91.7\% \\ \text{error rate} &= 100\% - 91.7\% = 8.3\% \end{aligned}$$



	0	1	2	3	4	5	6	7	8	9
0	902	0	3	3	3	11	5	4	3	2
1	0	1119	10	4	1	8	1	1	14	5
2	3	12	872	15	13	5	15	13	27	7
3	3	9	25	896	0	50	2	13	24	16
4	3	0	6	1	858	3	10	5	8	54
5	8	5	10	20	9	809	16	6	29	9
6	1	1	6	1	4	14	978	1	4	3
7	6	2	11	3	6	1	1	957	0	42
8	3	18	11	16	6	27	9	4	867	17
9	5	4	3	8	22	9	0	27	2	912

Confusion Table

A **confusion table** is used to measure the classification performance of a two-class system.

- It is similar to the confusion matrix but with only two classes: positive and negative
- We can use tables of confusions for n-class systems considering one class against all the other classes.

Confusion Table

		Predicted		
		Positive	Negative	*Total*
Actual	Positive	**TP**	FN	(TP+FN)
	Negative	FP	**TN**	(FP+TN)
	Total	(TP+FP)	(FN+TN)	N

Confusion Table Example

Computed from the confusion matrix for MNIST with softmax regression for class ‘5’ (again evaluated on the test set with N_{test}=10’000).

809 digits ‘5’ (out of 921) correctly recognised.
8951 out of 9079 digits correctly as not ‘5’.

		Predicted		
		P	N	Total
Actual	P	809	112	921
	N	128	8951	9079
	Total	937	9063	10000

Per Class Accuracy

The **class accuracy** of the system is the percentage of correct classification considering a given class against the others.

$$\frac{TP + TN}{\#\text{samples}}$$

See the table for our MNIST example.

E.g. for digit '5':

TP = 809, TN = 8951
—> accuracy = 97.60%

digit	TP	TN	accuracy	error
0	902	9032	99.34%	0.66%
1	1119	8786	99.05%	0.95%
2	872	8933	98.05%	1.95%
3	896	8891	97.87%	2.13%
4	858	8988	98.46%	1.54%
5	**809**	**8951**	**97.60%**	**2.40%**
6	978	8928	99.06%	0.94%
7	957	8897	98.54%	1.46%
8	867	8911	97.78%	2.22%
9	912	8853	97.65%	2.35%

Per Class Sensitivity (or Recall)

The **class sensitivity** or **recall** of the system is the percentage of correct classification for a given class.

$$\frac{TP}{TP+FN}$$

What fraction of all positives is discovered by the system?

See the table for our MNIST example.

E.g. for digit '5':

TP = 809, FN = 112
—> recall 87.84%

digit	TP	FN	recall
0	902	34	96.37%
1	1119	44	96.22%
2	872	110	88.80%
3	896	142	86.32%
4	858	90	90.51%
5	**809**	**112**	**87.84%**
6	978	35	96.54%
7	957	72	93.00%
8	867	111	88.65%
9	912	80	91.94%

Per Class Precision

The **class precision** of the system is the percentage of correct classification in the predicted outputs for a given class.

$$\frac{TP}{TP+FP}$$

What fraction of all positively classified is correctly classified?

See again on the right the table for our MNIST example.

E.g. for digit ‘5’:

TP = 809, FP = 128

—> precision = 86.34%

digit	TP	FP	F1
0	902	32	96.47%
1	1119	51	95.93%
2	872	85	89.94%
3	896	71	89.38%
4	858	64	91.76%
5	**809**	**128**	**87.08%**
6	978	59	95.41%
7	957	74	92.91%
8	867	111	88.65%
9	912	155	88.59%

F-Score

Systems can have a good precision and a bad recall or vice versa.

With a good F-Score neither of the two can be too bad. It is defined as the geometric mean of precision and recall.

F-Score $2 \cdot \frac{recall \cdot precision}{recall + precision}$

digit	TP	FP	recall	precision	F1
0	902	32	96.37%	96.57%	96.47%
1	1119	51	96.22%	95.64%	95.93%
2	872	85	88.80%	91.12%	89.94%
3	896	71	86.32%	92.66%	89.38%
4	858	64	90.51%	93.06%	91.76%
5	**809**	**128**	**87.84%**	**86.34%**	**87.08%**
6	978	59	96.54%	94.31%	95.41%
7	957	74	93.00%	92.82%	92.91%
8	867	111	88.65%	88.65%	88.65%
9	912	155	91.94%	85.47%	88.59%

See again on the right the table for our MNIST example.

E.g. for digit ‘5’:

TP = 809, FP = 128

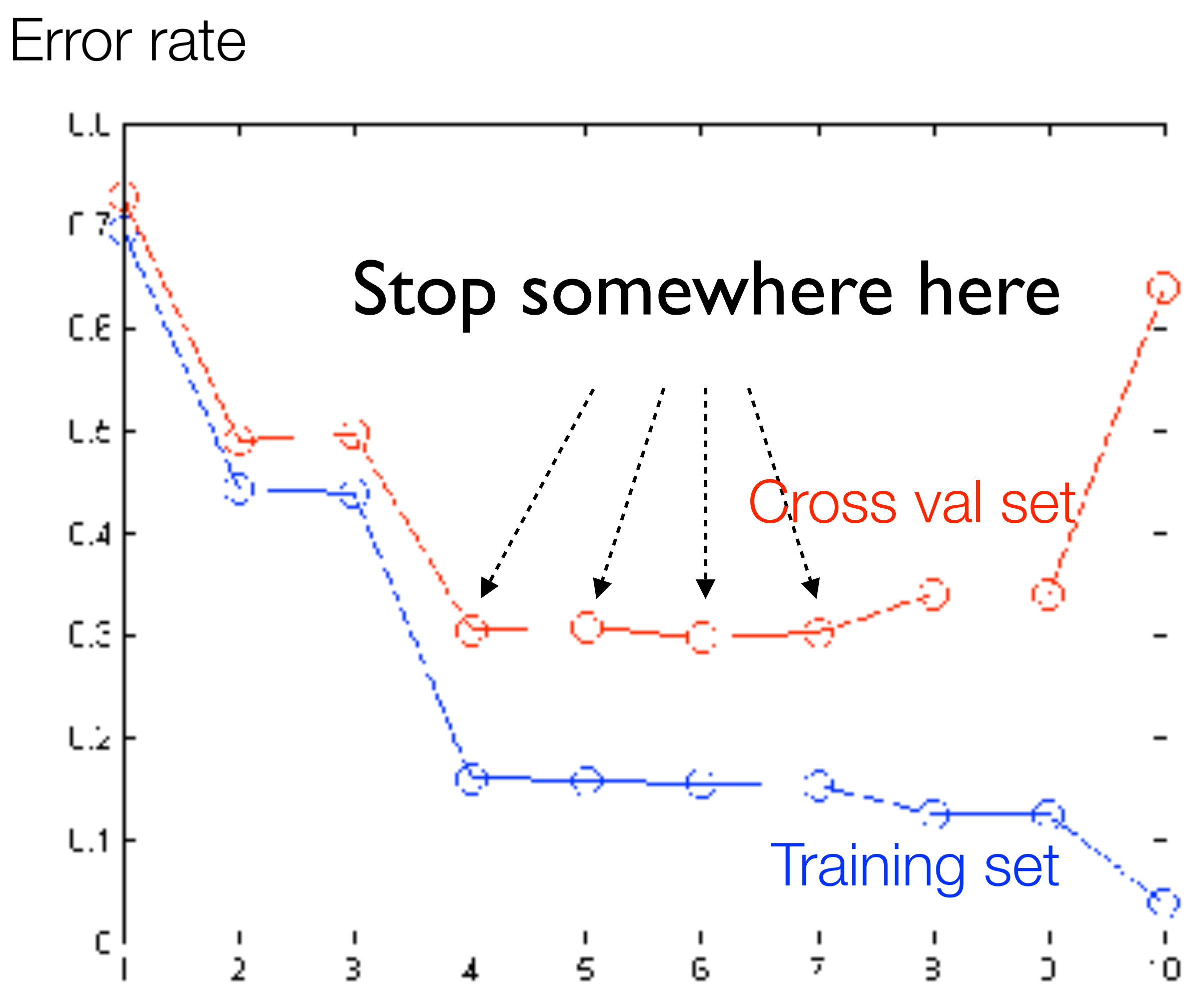
—> precision = 86.34%, F1 = 87.08%

General Rule for Model Selection

While inspecting these performance measures:

- Plot curves with these performance measures computed on the training and the validation set — vs training set size, model complexity, #epochs, etc.
- To avoid overfitting, look at the performance on the validation set.
- Possibly apply cross validation to compute them.

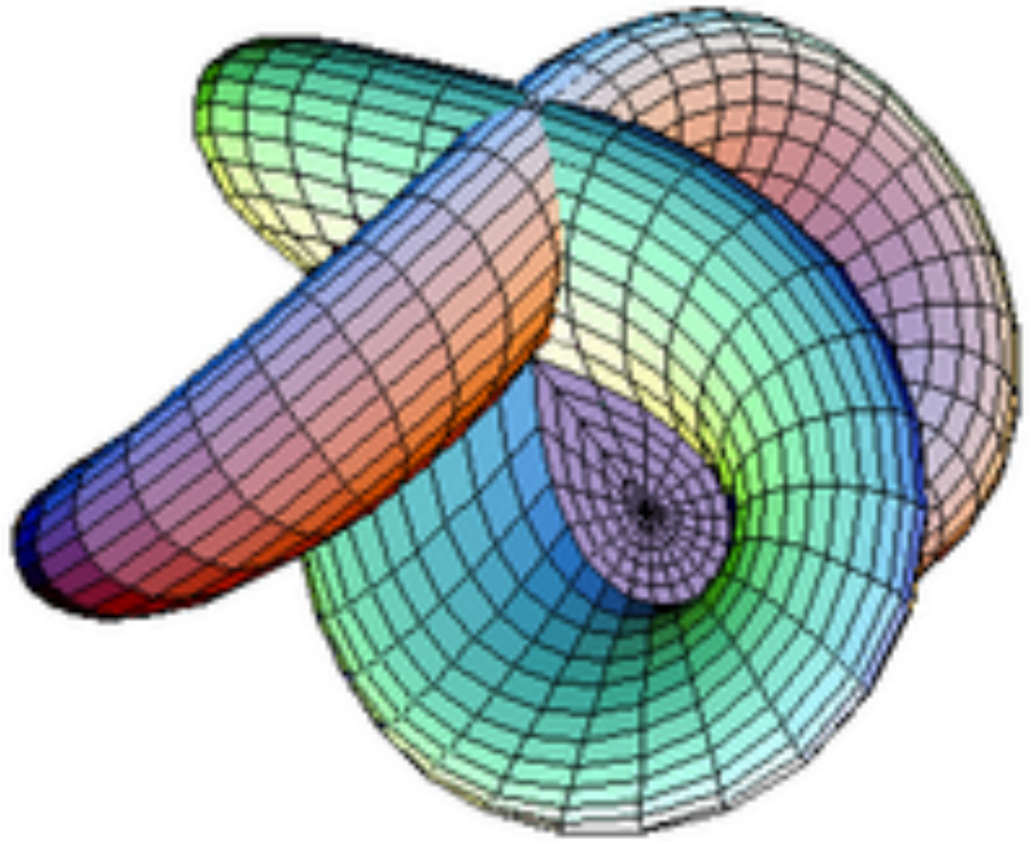
Note that there are other ways to evaluate performance (e.g. ROC, DET, AUC) — see ML course.



Curse of Dimensionality and DL

Typical Dimensionality in Deep Learning Problems

Hierarchy of Concepts



What's Curse of Dimensionality about?

Coined by Richard E. Bellmann (1961)

Wikipedia (https://en.wikipedia.org/wiki/Curse_of_dimensionality):

The curse of dimensionality refers to various phenomena that arise when analysing and organising data in high-dimensional spaces (often with hundreds or thousands of dimensions) …

… The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality.

Curse of Dimensionality in Neural Networks

- Tackling typical DL tasks with a huge amount of variability seen in the data requires models with a huge amount of parameters so that sufficient flexibility is provided to reveal the rich structure and patterns seen in the data.
- With increasing number of parameters the number of possible configurations increases exponentially.
- Accordingly, an exponentially increasing amount of data is needed to obtain a sufficient coverage and statistically significant estimates of the parameters:

$$m \propto N^d$$

m: Number of samples needed

N: Number of points along each dimension

d: Number of parameters of the model

Can become very large (10^4-10^6 or more)

Local Smoothness Assumption Not Scalable

- Classical ML methods (such as clustering, k-nearest-neighbours, SVMs, decision trees, n-grams) work with underlying local smoothness assumptions: The function to learn does not vary too much locally and stay approximately constant in a small region.

- In turn, local smoothness allows to generalise only to variations in small regions from closely located training examples. If no training examples are available in a given neighbourhood, no confident predictions can be made.

➡ Local smoothness assumption does not scale for typical deep learning applications (such as computer vision or speech recognition) with their high dimensionality.

Representation of Variational Characteristics

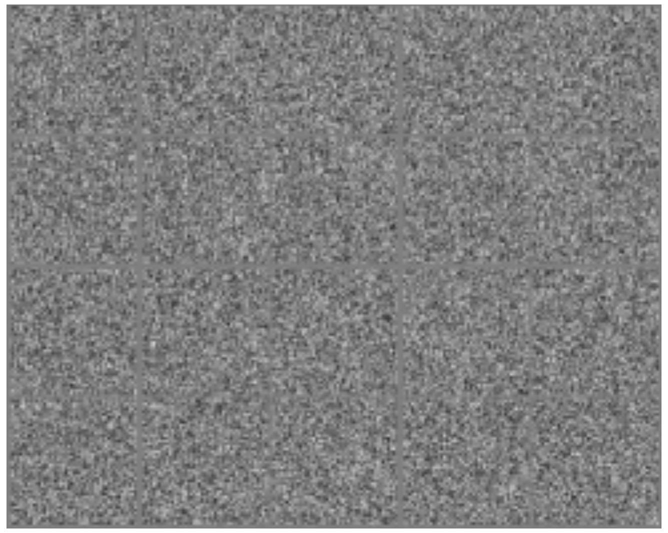
How does the number of parameters in a model suited for a given task relate to the dimensionality of the input?

(1) Goal is to find a good representation of the variational characteristics observed in the data as needed by the underlying task.

(2) The patterns found in the data typically are well represented in suitable lower dimensional manifolds.

Example

When learning the representation of images, an image with pure noise (as depicted on the right) could also be represented within a parameter space that would assign one parameter to each pixel. However, it is unlikely that we will see such an image in the training data or that we need to classify samples consisting of pure noise.



Reduced Variational Degrees of Freedom

- Images encountered in DL applications are expected to occupy a negligible portion of the volume of input space (e.g. pixel space).
- Typically, the interesting information is concentrated on some lower dimensional manifold (with much less degrees of freedom).

The variational degrees of freedom in the series of images of faces below are much smaller than to allow random noise.

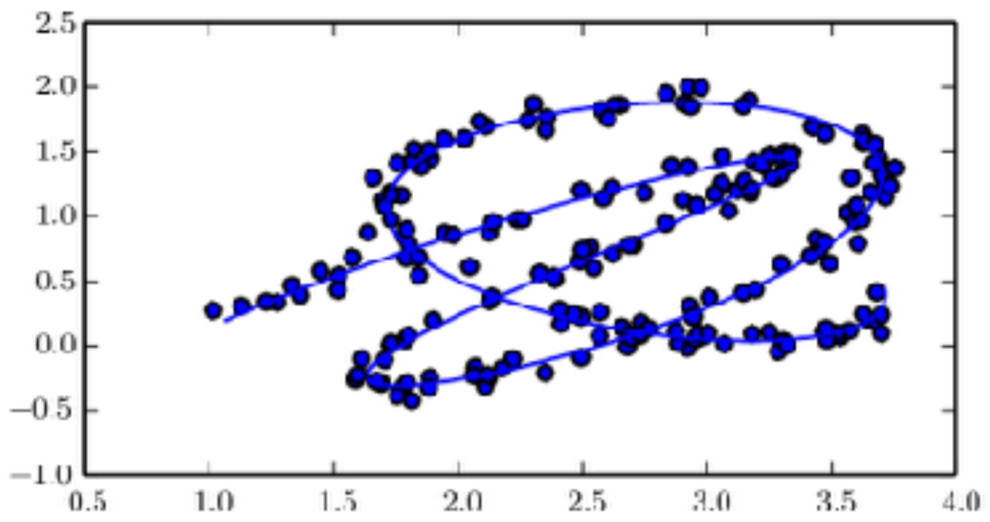


Illustration of a lower-dimensional manifold: The data points lie near a one-dimensional manifold embedded in two- dimensional space.

Composition of Features at Multiple Levels in a Hierarchy

Core Idea in DL

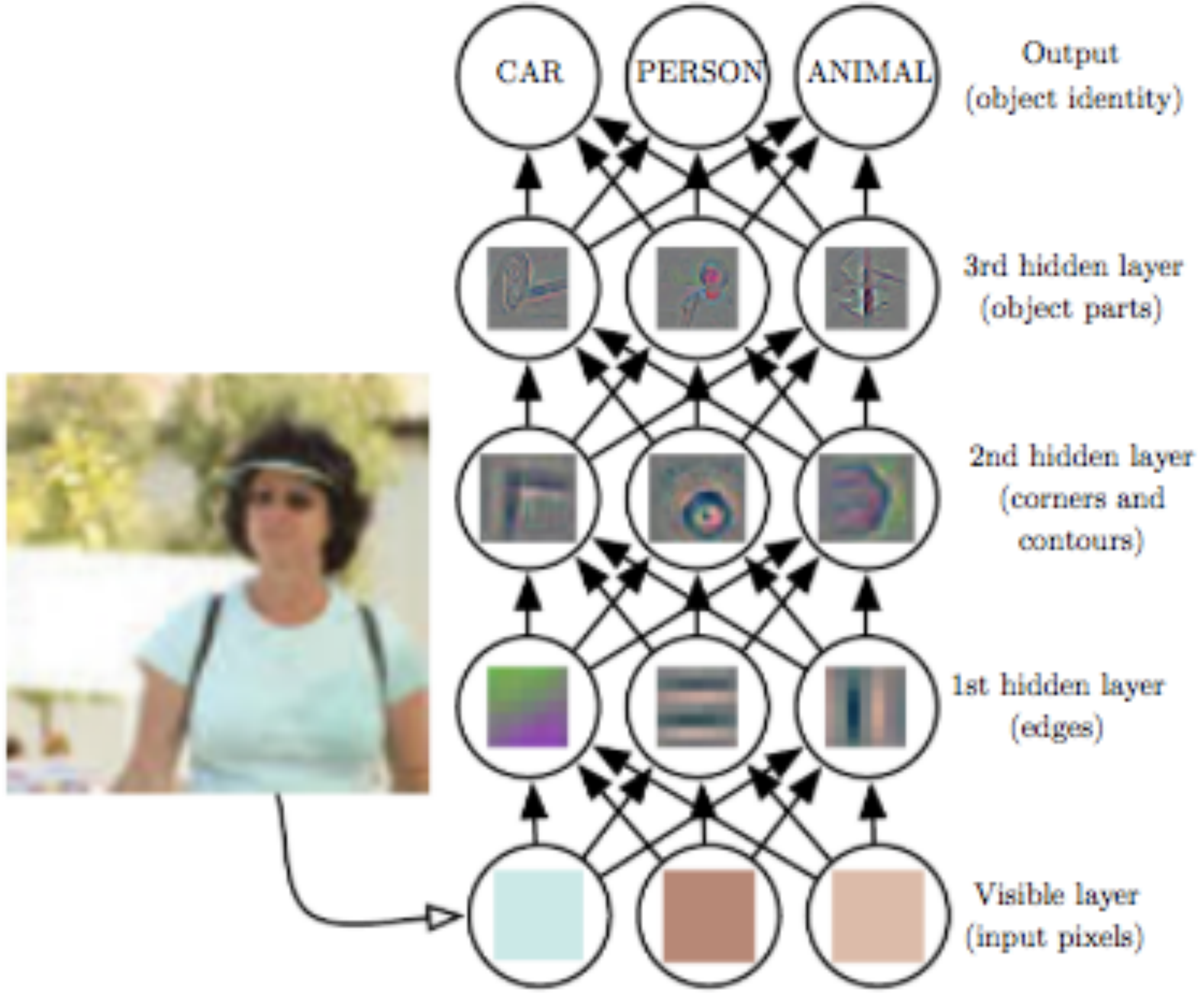
In DL problems, we assume that the data was generated by a composition of factors or features, potentially at multiple levels in a hierarchy — or, the function to be learned involves a composition of simpler functions over multiple hierarchies.
Accordingly, learning here involves discovering a set of underlying factors of variation that can be described in terms of other, simpler underlying factors.

➜ **Distributed Representations, Concept Hierarchy**

Exponential gain in the number of examples needed for a given number of regions to be distinguished.

➜ **Better generalisation properties for a wide variety of tasks!**

Example Computer Vision



more on this in the part on CNNs

Summary

You have learned

- about overfitting, the typical situations where it occurs (large models, comparatively little or noisy data) and how you can identify it from the learning curves.
- the tools to do model selection and hyper-parameter tuning with the help of the validation set and schemes such as k-fold cross-validation.
- how to compute confusion matrix that starts from the scores of a classification task and how to compute performance measures such as accuracy, precision or recall.
- some intuition about the curse of dimensionality and how deep neural networks may oppose it in problems that have hierarchies of concepts in the underlying data generating process.

Outlook

You have learned gradient descent (BGD, SGD, MBGD) and how to implement it for simple models (e.g. models consisting of just a softmax layer).

How to efficiently implement it for deep neural network?

—> Backpropagation (‘backprop’)

Backup

Illustrative Example on Peculiarities in High-Dimensional Spaces

- Consider the unit cube in *d* dimensions and decompose it in *n* smaller cubes. Then, the smaller cubes have a side length given by

$$s_{n,d} = (1/n)^{1/d}$$

Example with *n=100*:

d	s
1	0.01
10	0.631
100	0.955
1000	0.99

- In *d=100* dimensions, you need *n=100* cubes with side length of 95.5% (in each direction) to cover a unit cube.
- In higher dimensions, an increasing fraction of the volume of the unit ball is concentrated at the boundary (unit sphere).