



Lecture 3: Basic of ML



Plan for Today

○ Numerical Computation

<https://www.deeplearningbook.org/contents/numerical.html>

○ Machine Learning Basic

<https://www.deeplearningbook.org/contents/ml.html>

- [Table of Contents](#)
- [Acknowledgements](#)
- [Notation](#)
- [1 Introduction](#)
- [Part I: Applied Math and Machine Learning Basics](#)
 - ✓ ○ [2 Linear Algebra](#)
 - ✓ ○ [3 Probability and Information Theory](#)
 - ✓ ○ [4 Numerical Computation](#)
 - ✓ ○ [5 Machine Learning Basics](#)
- [Part II: Modern Practical Deep Networks](#)
 - ✓ ○ [6 Deep Feedforward Networks](#)
 - ✓ ○ [7 Regularization for Deep Learning](#)
 - ✓ ○ [8 Optimization for Training Deep Models](#)
 - ✓ ○ [9 Convolutional Networks](#)
 - ✓ ○ [10 Sequence Modeling: Recurrent and Recursive Nets](#)
 - ✓ ○ [11 Practical Methodology](#)
 - ✓ ○ [12 Applications](#)
- [Part III: Deep Learning Research](#)
 - [13 Linear Factor Models](#)
 - ✓ ○ [14 Autoencoders](#)
 - [15 Representation Learning](#)
 - [16 Structured Probabilistic Models for Deep Learning](#)
 - [17 Monte Carlo Methods](#)
 - [18 Confronting the Partition Function](#)
 - [19 Approximate Inference](#)
 - ✓ ○ [20 Deep Generative Models](#)
- [Bibliography](#)
- [Index](#)

✓: Parts that we will cover



Machine Learning

- “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .”
- A task T : classification, regression, etc.
- A performance measure P (typically on independent test dataset):
 - Example: for classifiers, accuracy/error rate, i.e., fraction of examples correctly/incorrectly classified
- An Experience E , provided as a training dataset on which an algorithm can try to improve its task solving skill
- Supervised learning: learn to reproduce the ground-truth (e.g., flags in classification)
- Unsupervised learning: learn the pdf of the dataset and use it to make probabilistic statements about new data (e.g., anomaly detection)

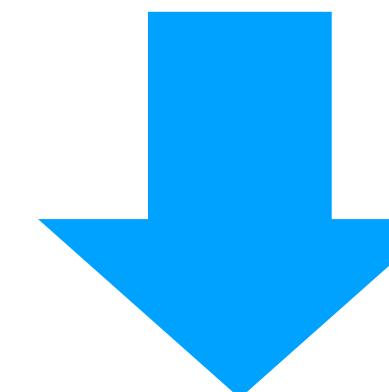
learn some quantity x_i
from the other $N-1$ x_j

$$p(\mathbf{x}) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1}).$$

Machine Learning

- Boundary between supervised and unsupervised are not so strict: the same approach can be used for supervised or unsupervised learning

Learning the pdf of the data $p(\mathbf{x})$ is an unsupervised task



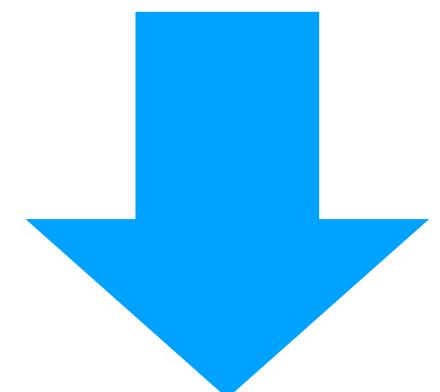
But it can be broken down in supervised tasks

$$p(\mathbf{x}) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

from the other $N-1$ x_j

learn some quantity x_i

Learning some y from some \mathbf{x} , i.e., learning $p(y | \mathbf{x})$ is a supervised task



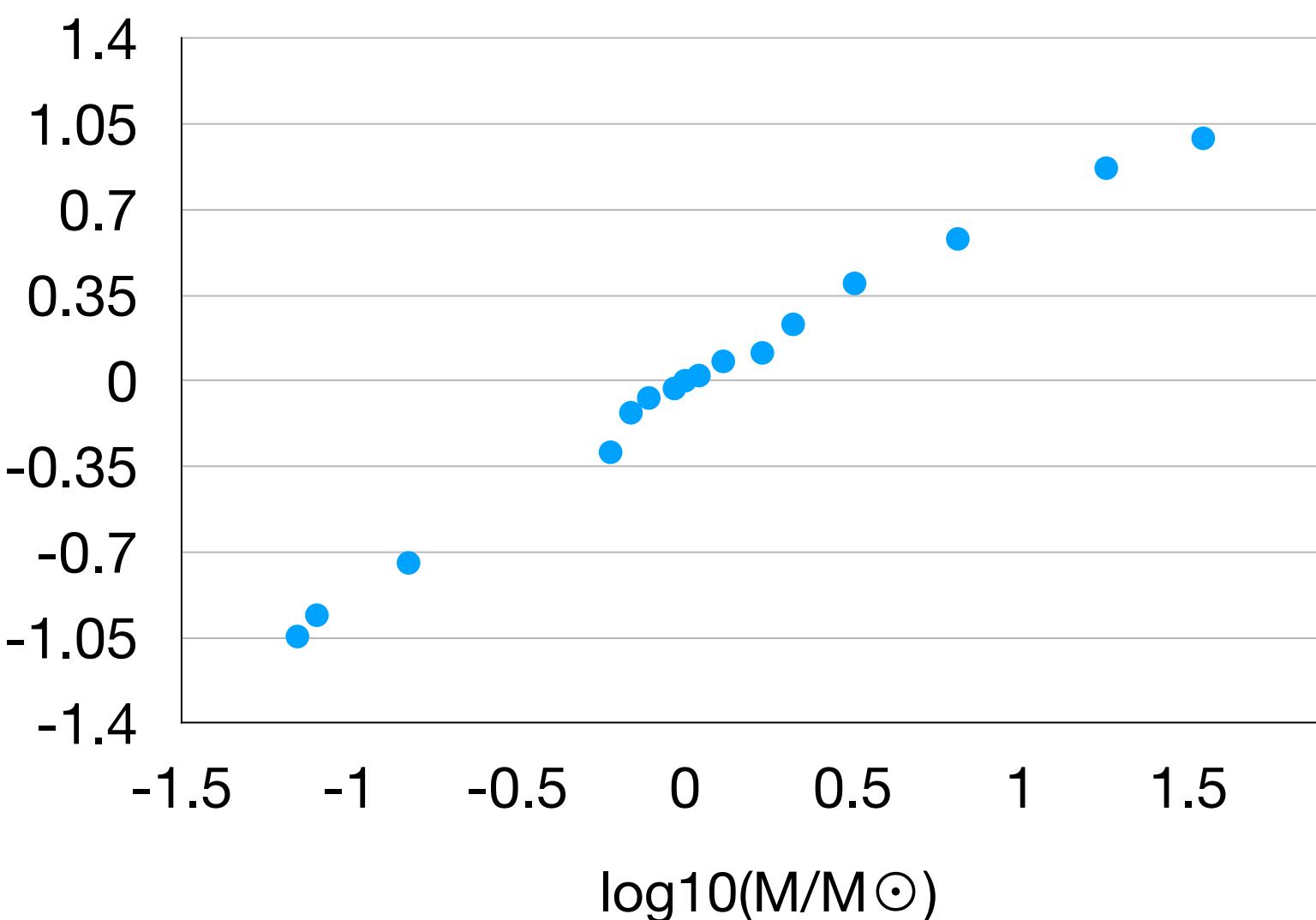
But this task can be approached with unsupervised technique, using the probability chain rule

$$p(y | \mathbf{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p(\mathbf{x}, y')}$$

learning the full pdf

learning the pdf $p(\mathbf{x})$

Machine Learning



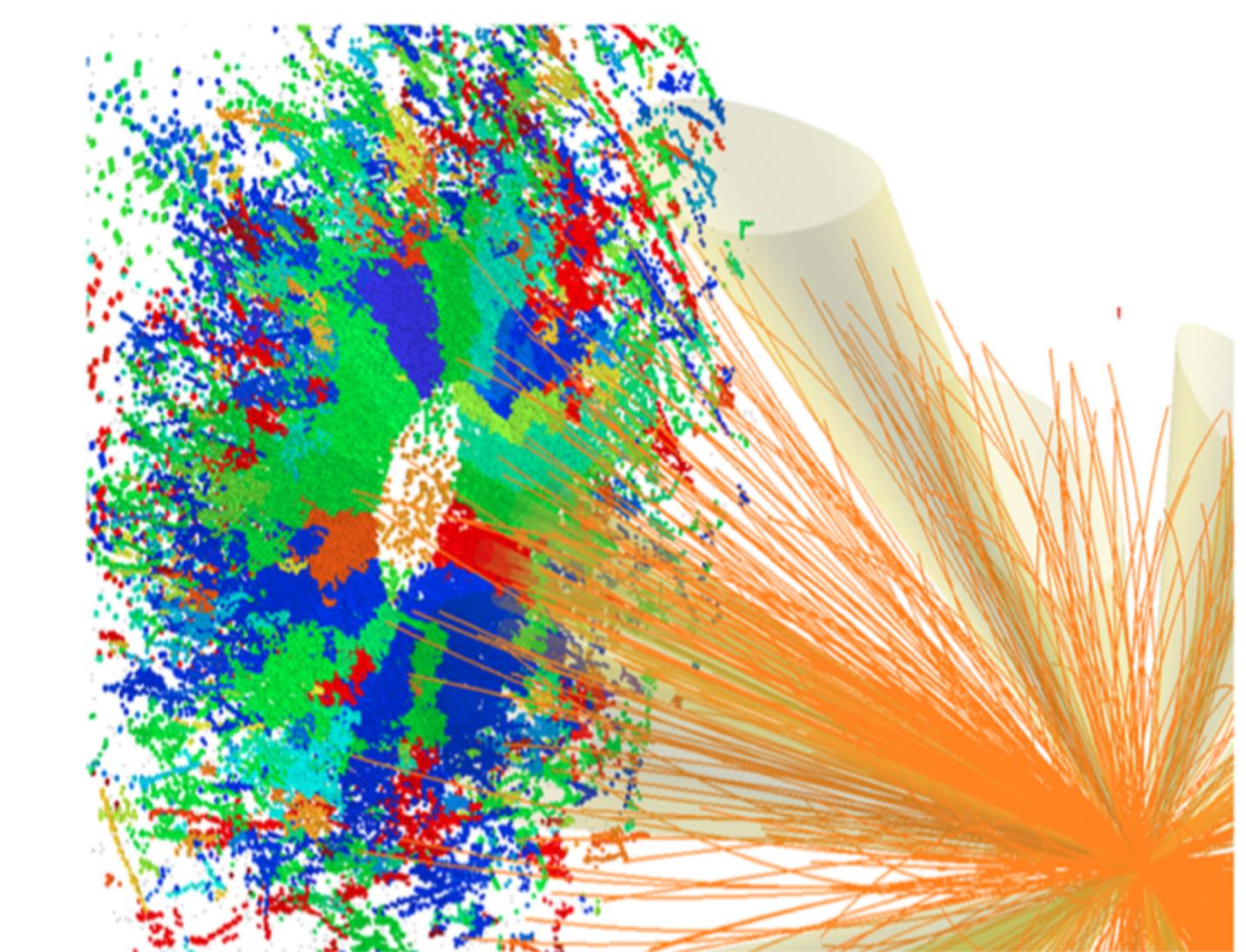
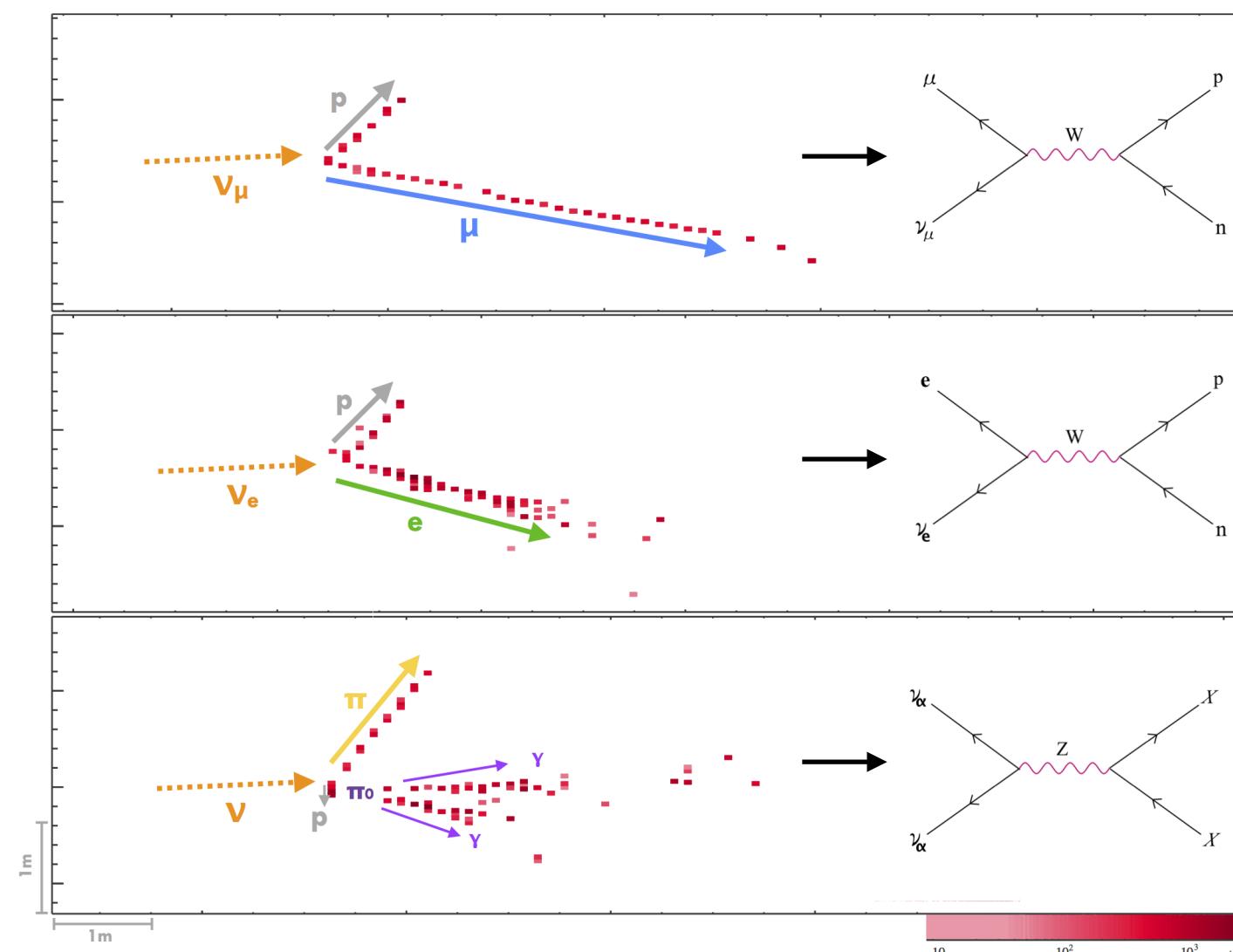
Regression

- Prediction of continuous-valued vector $y \in \mathbb{R}^K$. E.g given some input, predict particle position, particle energy etc.
- Training set: Pairs of input signal and ground-truth value

Supervised Learning

Classification

- Predict value from a finite set $\{1, \dots, C\}$, e.g label (=cat) for an image X.
- Training set: Pairs of input signal and ground truth
- Usually predict one score per class, s.t correct class has maximum score



Density modelling

- Model probability density function of data μ_X
- Training set: values x_N without quantities to predict
- Trained model should allow for evaluation of pdf or sampling from it, or both

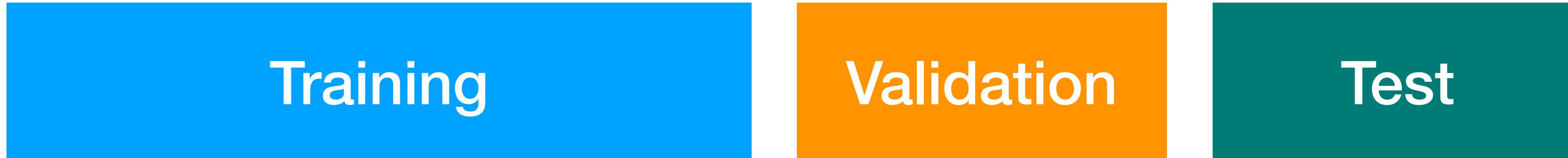
Unsupervised Learning

Machine Learning

- *We only have a finite training dataset*
 - *We cannot measure the true test error*
 - *Simple model classes underfit*
 - *Complex model classes overfit*
 - *Goal: Select the model class with lowest test error*
- Bias-variance trade-off**

Dataset Split

- *Split your sample in three:*
- *Training: the biggest chunk, where you learn from*
- *Validation: an auxiliary dataset to verify generalization and prevent overtraining*
- *Test: the dataset for the final independent check*

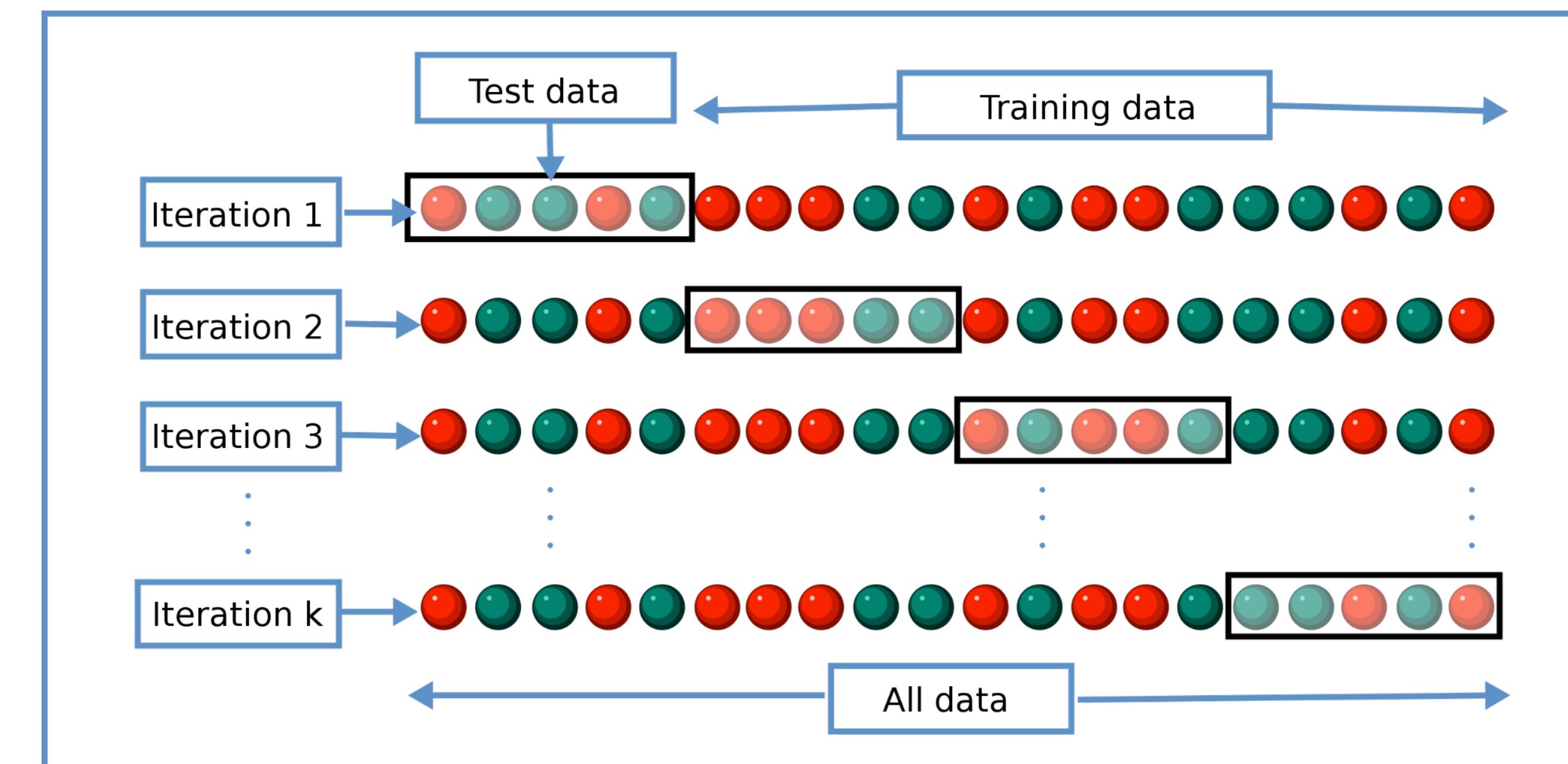


- *What separates ML from optimisation is generalization!*

k-folding

Split the original dataset into k equal parts (e.g., $k = 4$)

- Train on the $k - 1$ parts and test on the remaining one
- Repeat for every choice of the $k - 1$ parts and average the validation errors

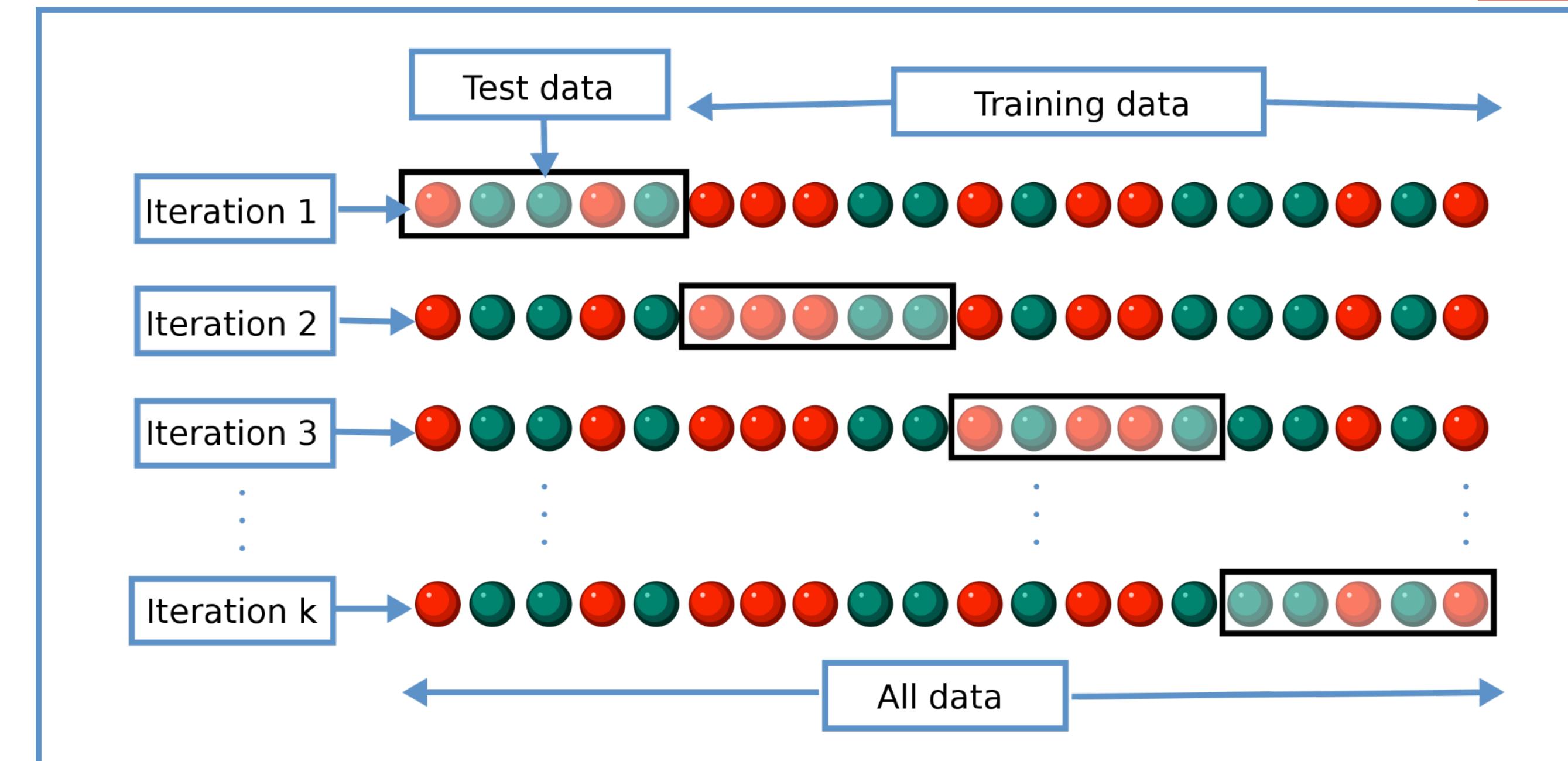
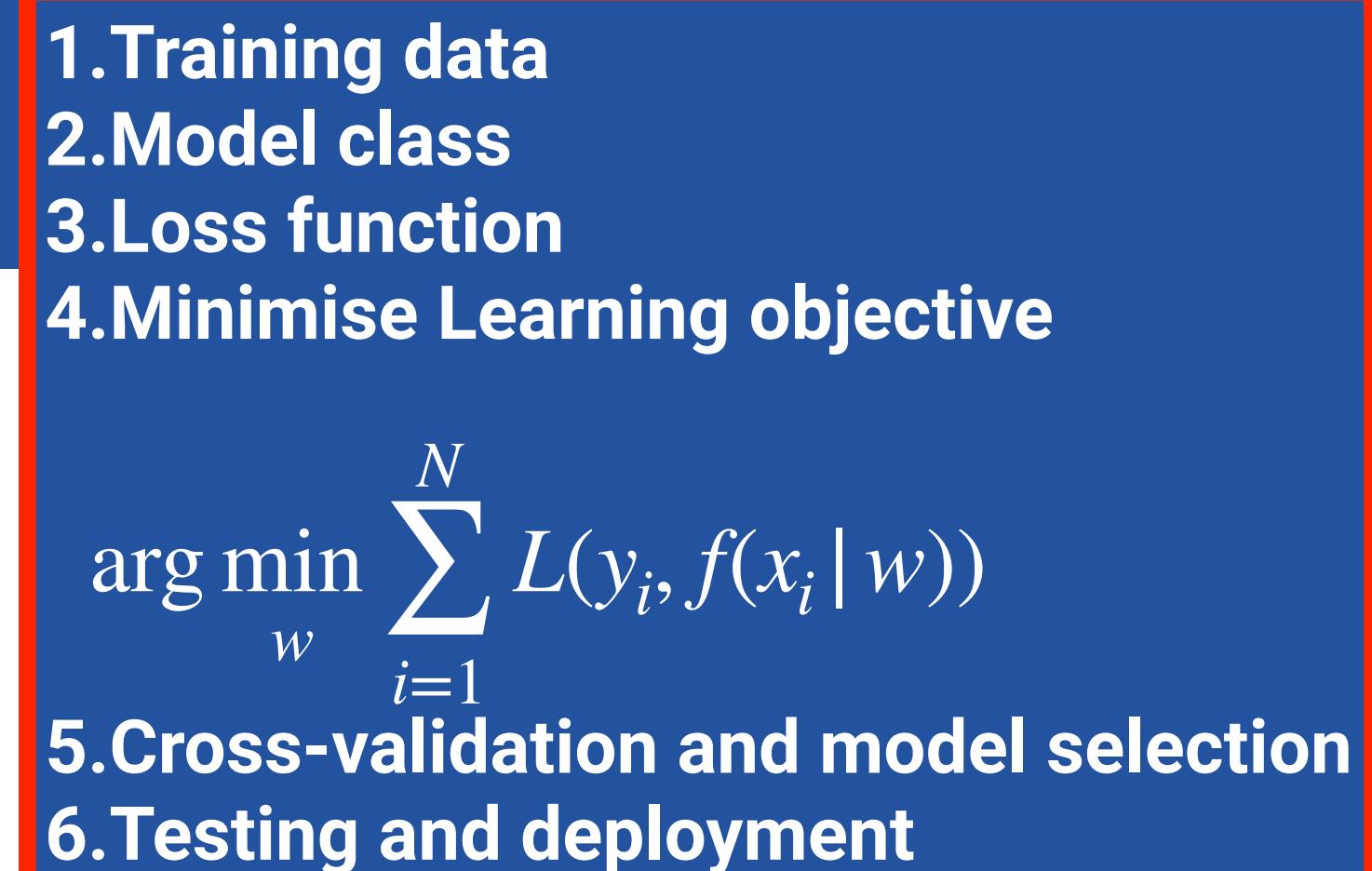


- Use all data as validation to improve the estimate of the test error (at the cost of more computation (k trainings))

k-folding

Split the original dataset into k equal parts (e.g., $k = 4$)

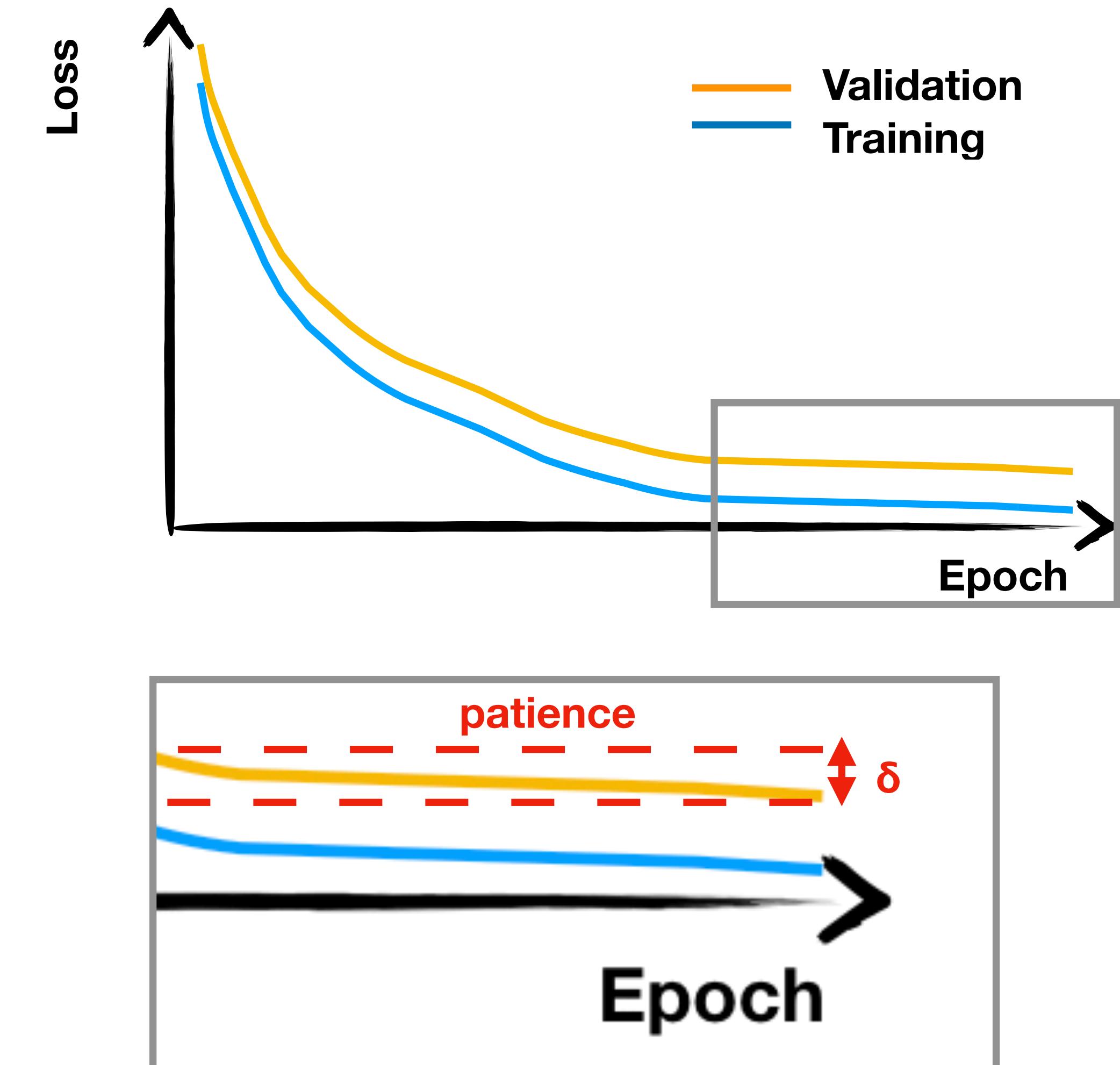
- Train on the $k - 1$ parts and test on the remaining one
- Repeat for every choice of the $k - 1$ parts and average the validation errors



- Use all data as validation to improve the estimate of the test error (at the cost of more computation (k trainings))

Learning history & Early Stopping

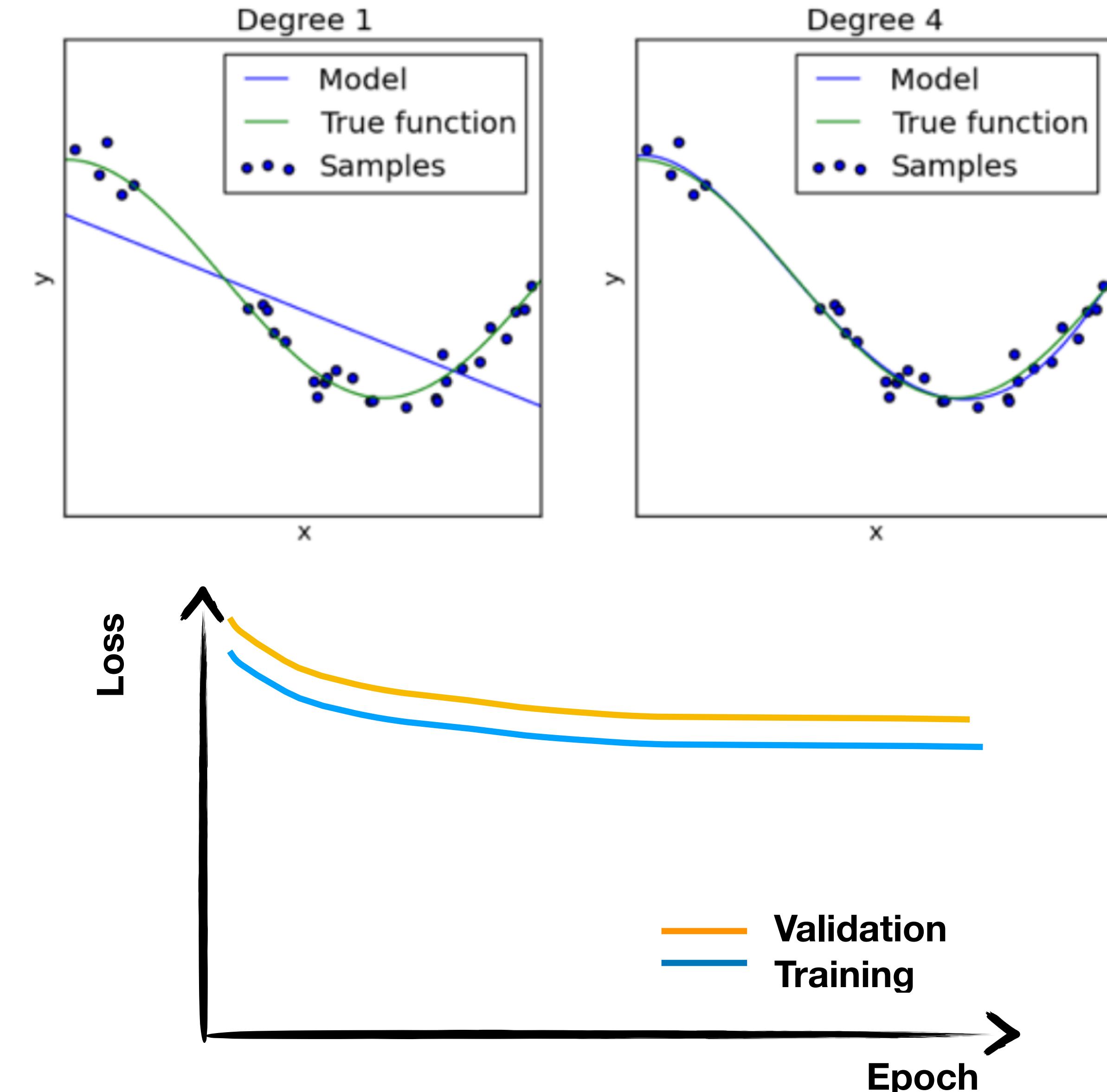
- *Train across multiple epochs*
- *1 epoch = going once through the full dataset*
- *Use small batches (64, 128, etc)*
- *Check your training history*
 - *on the training data (training loss)*
 - *and the validation ones (validation loss)*
- *Use an objective algorithm to stop (e.g., early stopping)*



EARLY STOPPING: stop the train if the validation loss didn't change more than δ in the last n epochs (patience)

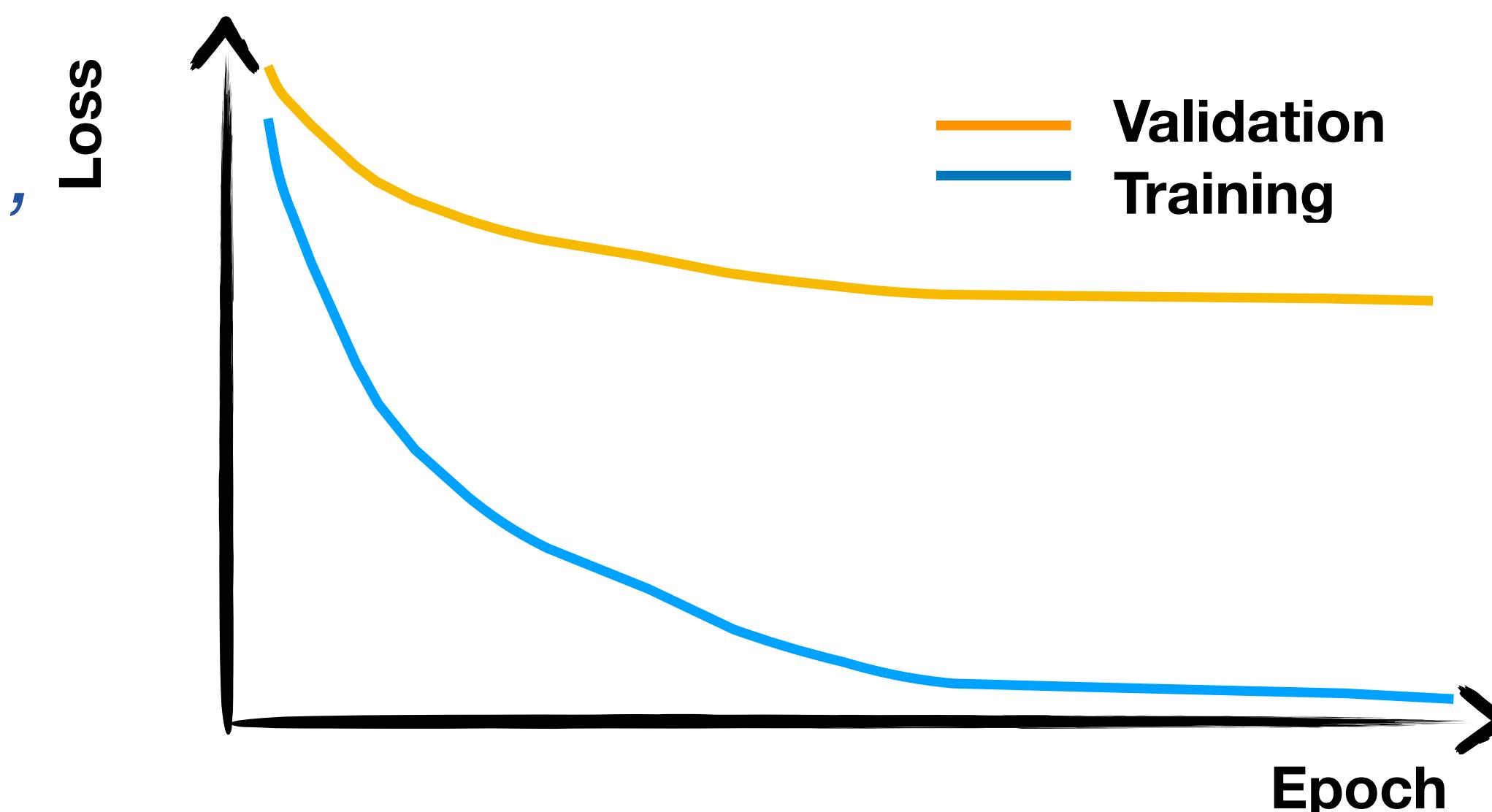
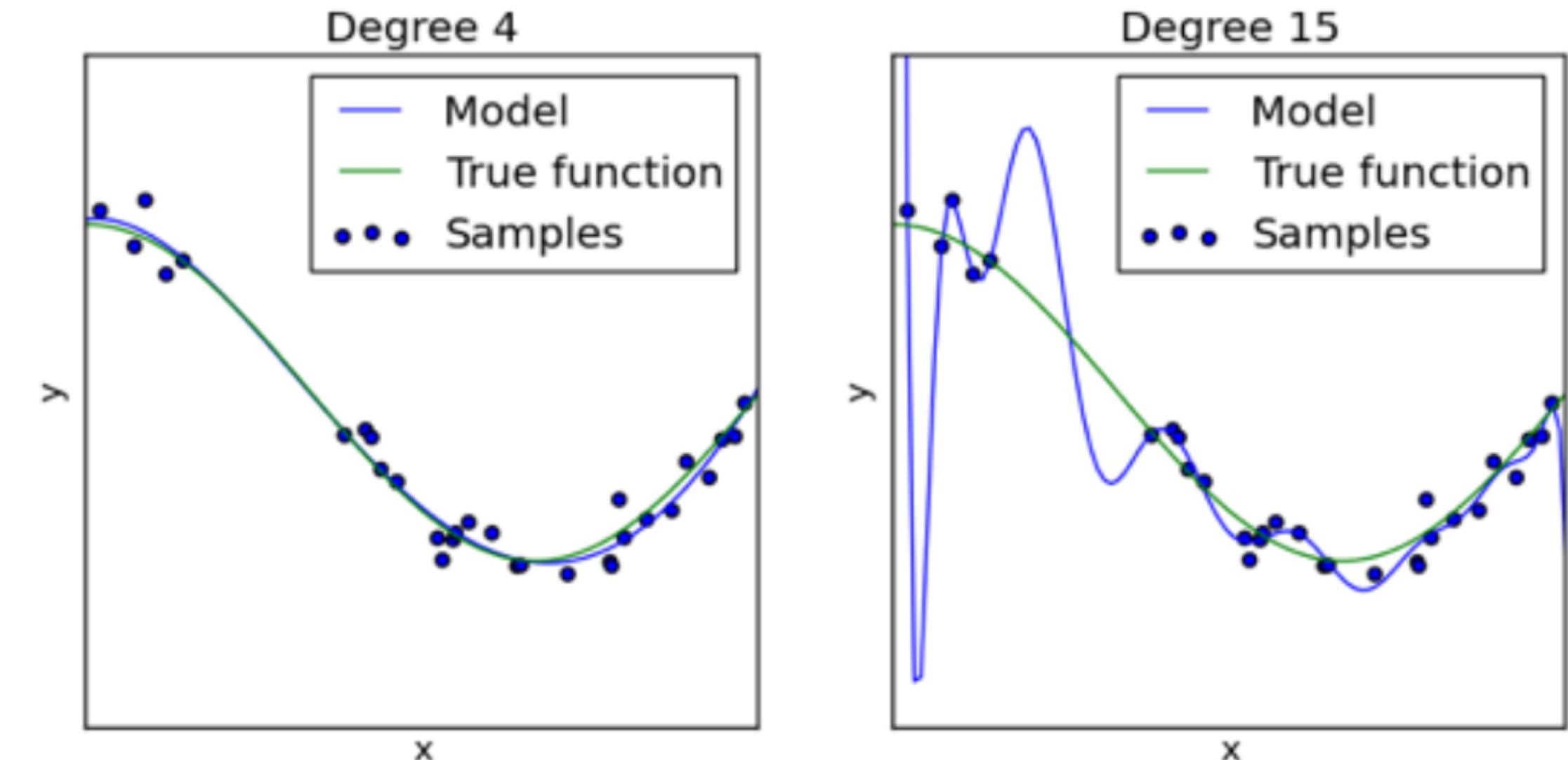
Undertraining

- *If your model has not enough flexibility, it will not be able to describe the data*
- *The training and validation loss will be close, but their value will not decrease*
- *The model is said to be underfitting, or being **biased***



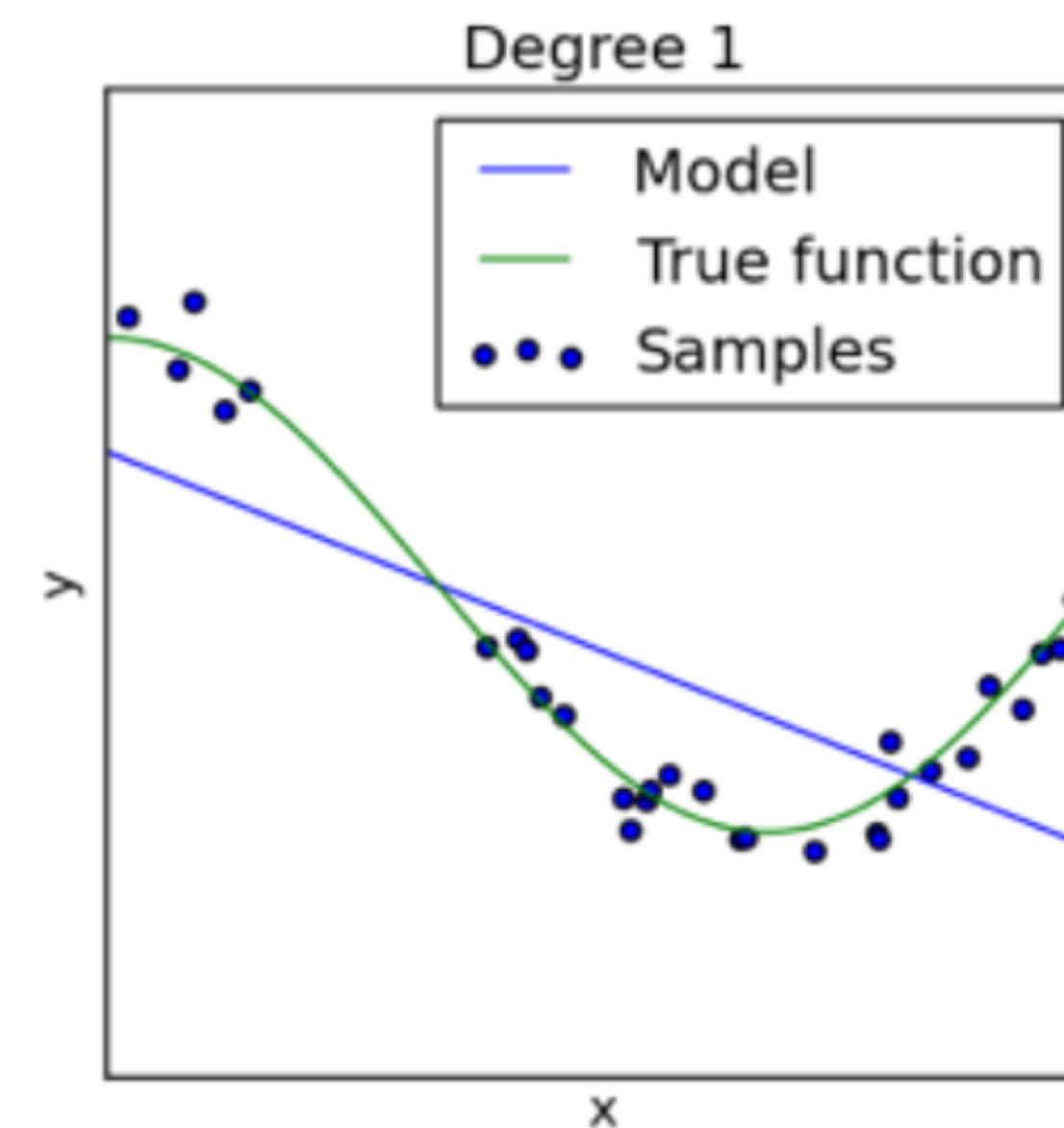
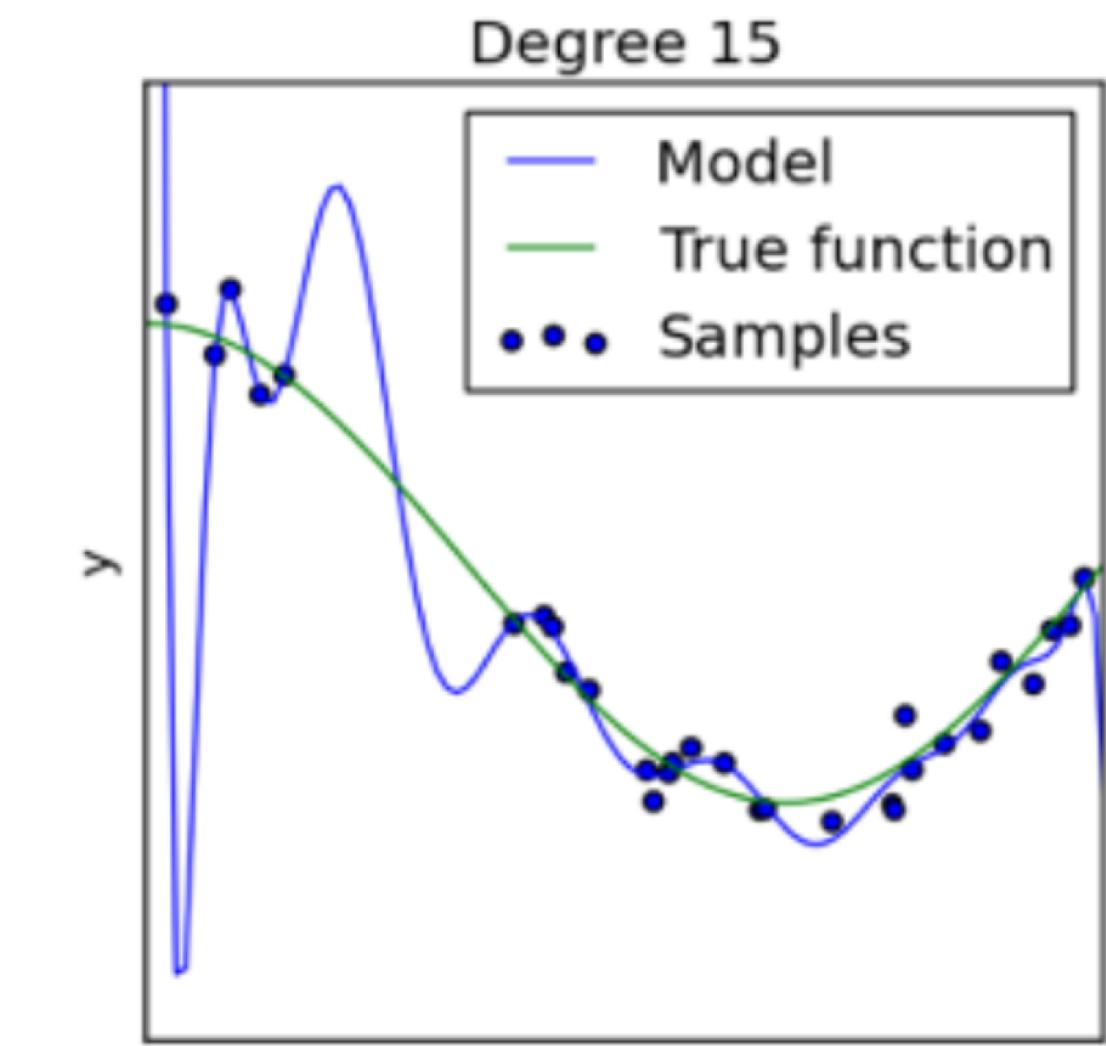
Overtraining

- Your model can learn too much of your training dataset
 - e.g., its statistical fluctuations
- Such an overfitted model would not generalise
- So, its description of the validation dataset will be bad (i.e., **the model doesn't generalise**)
- This is typically highlighted by a divergence of the training and validation loss



Bias vs Variance

- **A model would underfit if too simple:** it will not be able to model the mean value
- **A model would overfit if too complex:** it will reproduce the mean value, but it will underestimate the variance of the data
- **The generalization error** is the error made going from the training sample to another sample (e.g., the test sample)



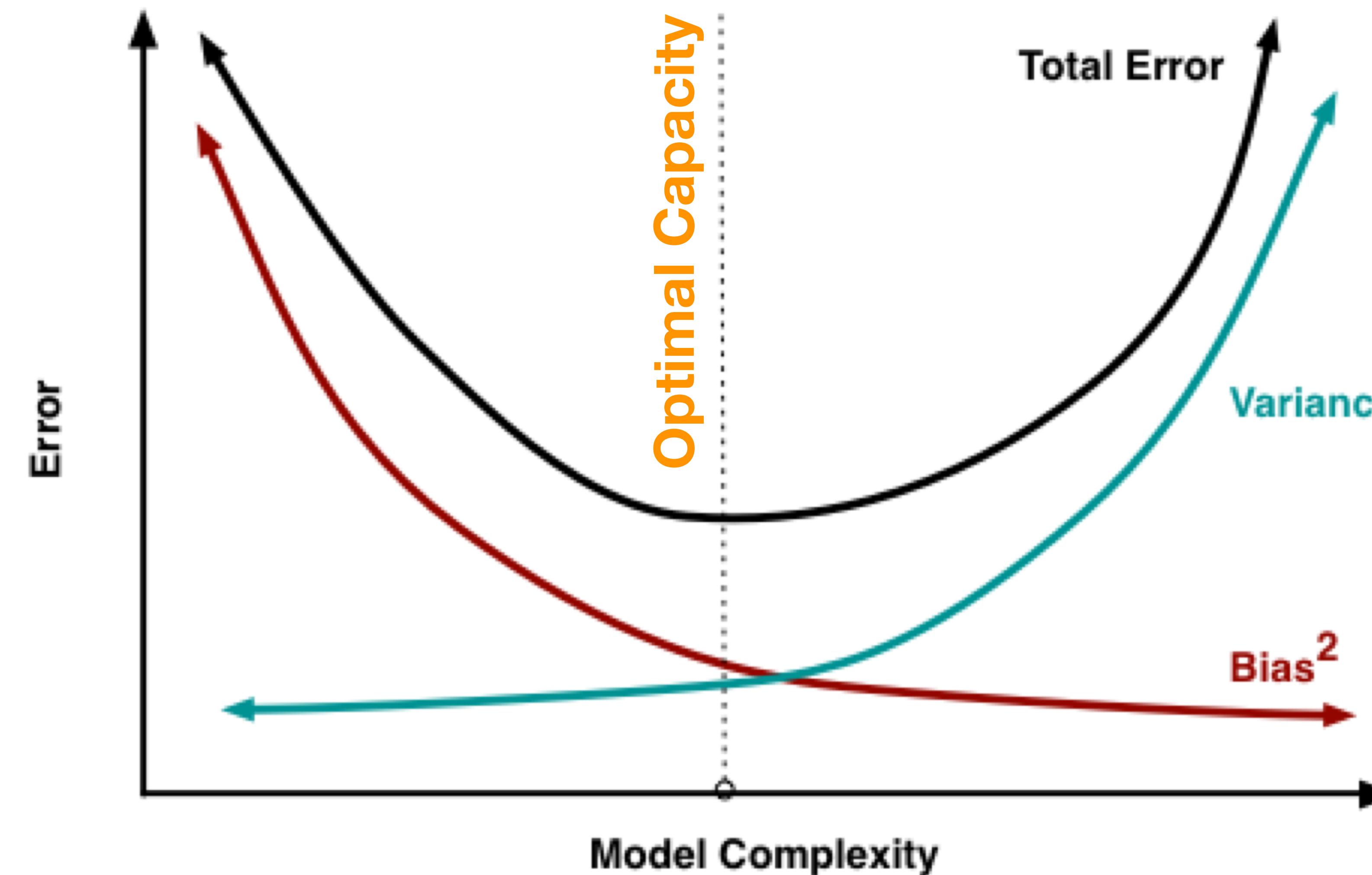
Bias vs Variance

- Generalization error can be written as the sum of three terms:
 - The *intrinsic statistical noise* in the data
 - the *bias* wrt the mean
 - the *variance* of the prediction around the mean

$$E[(y - h(x))^2] = E[(y - \bar{y})^2] + (\bar{y} - \bar{h}(x))^2 + E[(h(x) - \bar{h}(x))^2]$$

Noise **Bias Squared** **Variance**

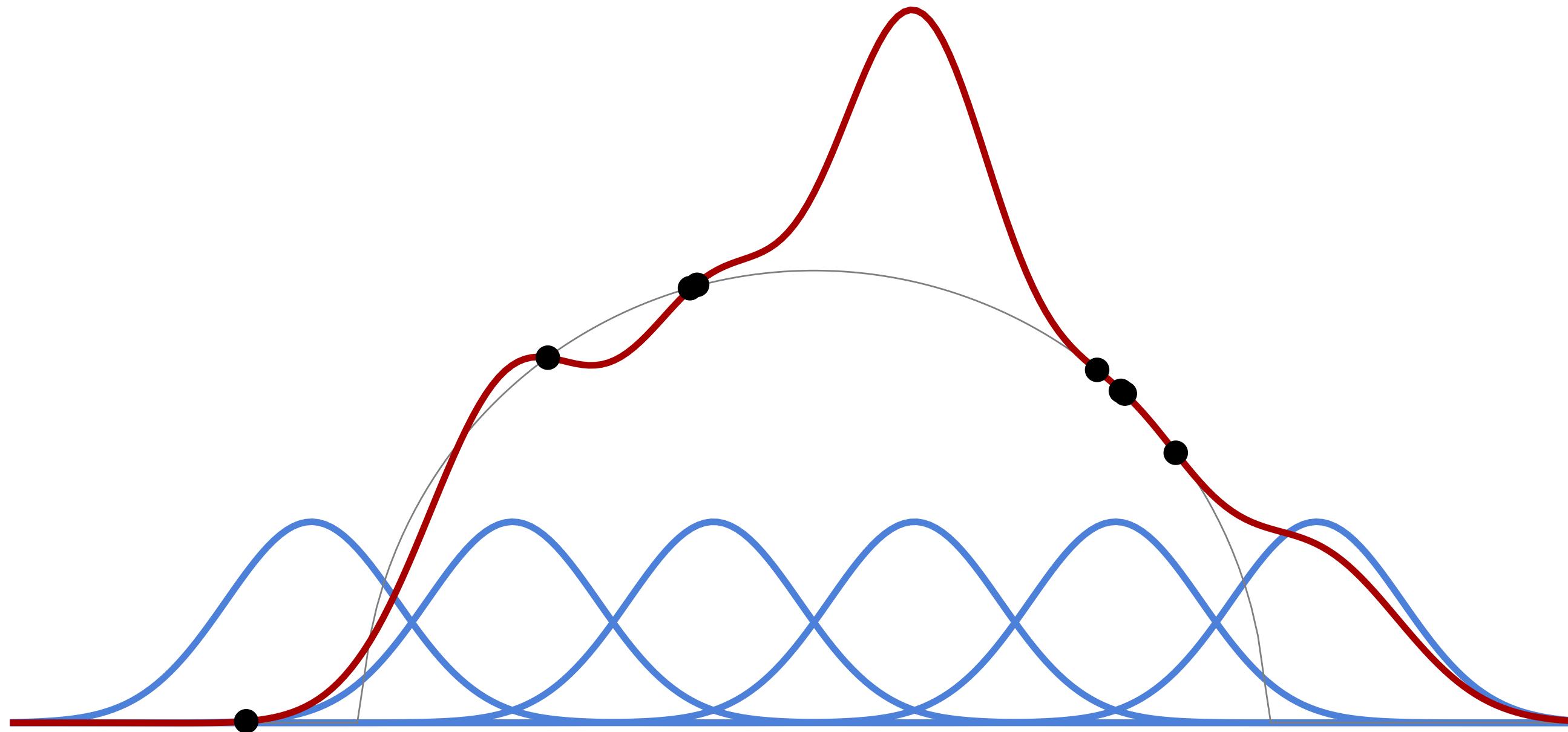
Bias vs Variance



Model capacity

Model capacity: flexibility and ability to fit diverse data

- Interplay between model capacity and amount/quality of your training data is highly important!
- Insufficient capacity → model cannot fit data, training error is high **underfitting**
- Insufficient data, training performance excellent but unrelated to underlying data structure (learn random noise in data) → **overfitting**



Amount of training data small compared to model capacity!

Model capacity

Model capacity: flexibility and ability to fit diverse data

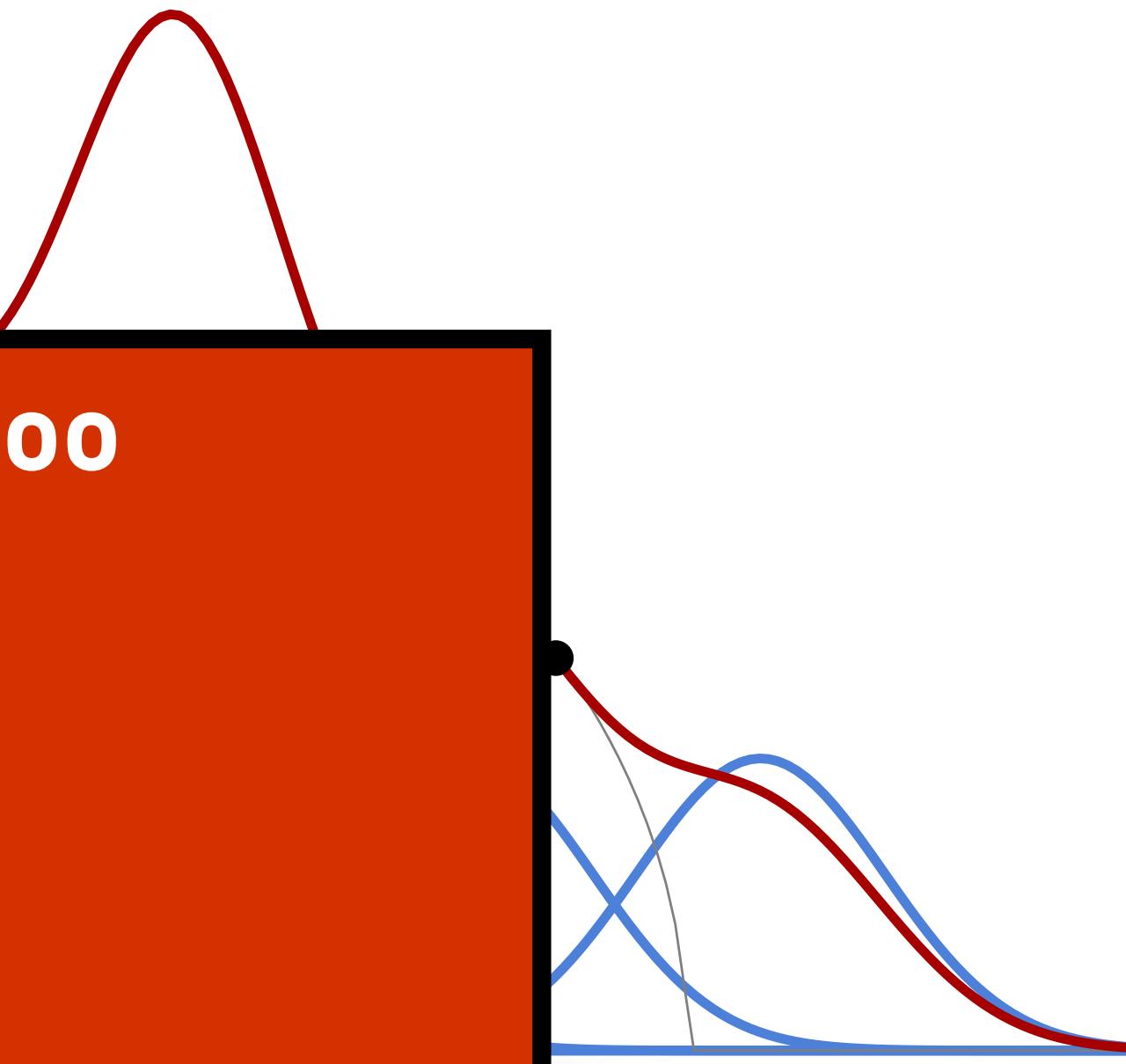
- Interplay between model capacity and

What is applied ML all about? Design models that are not too flexible, but yet able to fit the data!

- Crafting right inductive bias in a model
 - (structure corresponds to underlying structure of data)

- Insufficient data, training performance excellent but unrelated to underlying data structure (learn random noise in data) → **overfitting**

Amount of training data small compared to model capacity!





From theory to Practice

- *You can learn all Deep Learning theory, but an effective training requires skills and experience that are linked to your technical insight*
- *Many things can go wrong in training, and many of them have nothing to do with deep learning*
- *Overflows, Underflows, out-of-memory, slow code due to ineffective I/O when handling the data*
- *Even if the training works, many things can be done better*
- *Optimize I/O (data parallelism), distribute training on multiple GPUs, etc.*



Few things to keep in mind

- *PROBLEM: You are training on hardware that comes with limited numerical representation (e.g., 64 bits)*
- *This implies rounding error, that could have large effects when training or using the algorithm in practice*
- *Underflow: too small numbers are turned into 0, which kills your gradient and stop your training or give error (e.g., 1/0 -> NaN)*
- *Overflow: number is too big -> NaN. And the following mathematic is compromised*
- *Example: softmax activation function, used at the end of multi class classifier (takes a vector of scores and turns them into probabilities)*

$$\text{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}$$

You can avoid NaN evaluating the function at $\mathbf{z} = \mathbf{x} - \text{max}(\mathbf{x})$



Few things to keep in mind

- *PROBLEM: You are looking for an optimal weight configuration computing gradients. You want to avoid 0 gradient*
- *This is called conditioning, i.e., how rapidly a function changes wrt its input*
- *Example: matrix inversion. You need to find $f(x) = A^{-1}x$, given A . Given $\lambda =$ eigenvalues of A , the condition number for A is $\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$. Matrix inversion becomes numerically complicated when this number is large, i.e., when there is a big gap between the largest and smallest eigenvalue*

Regularisation in the loss

- Model complexity can be “optimized” when minimizing the loss

- A modified loss is introduced, with a penalty term attached to each model parameter

$$L_{reg} = L + \Omega(w)$$

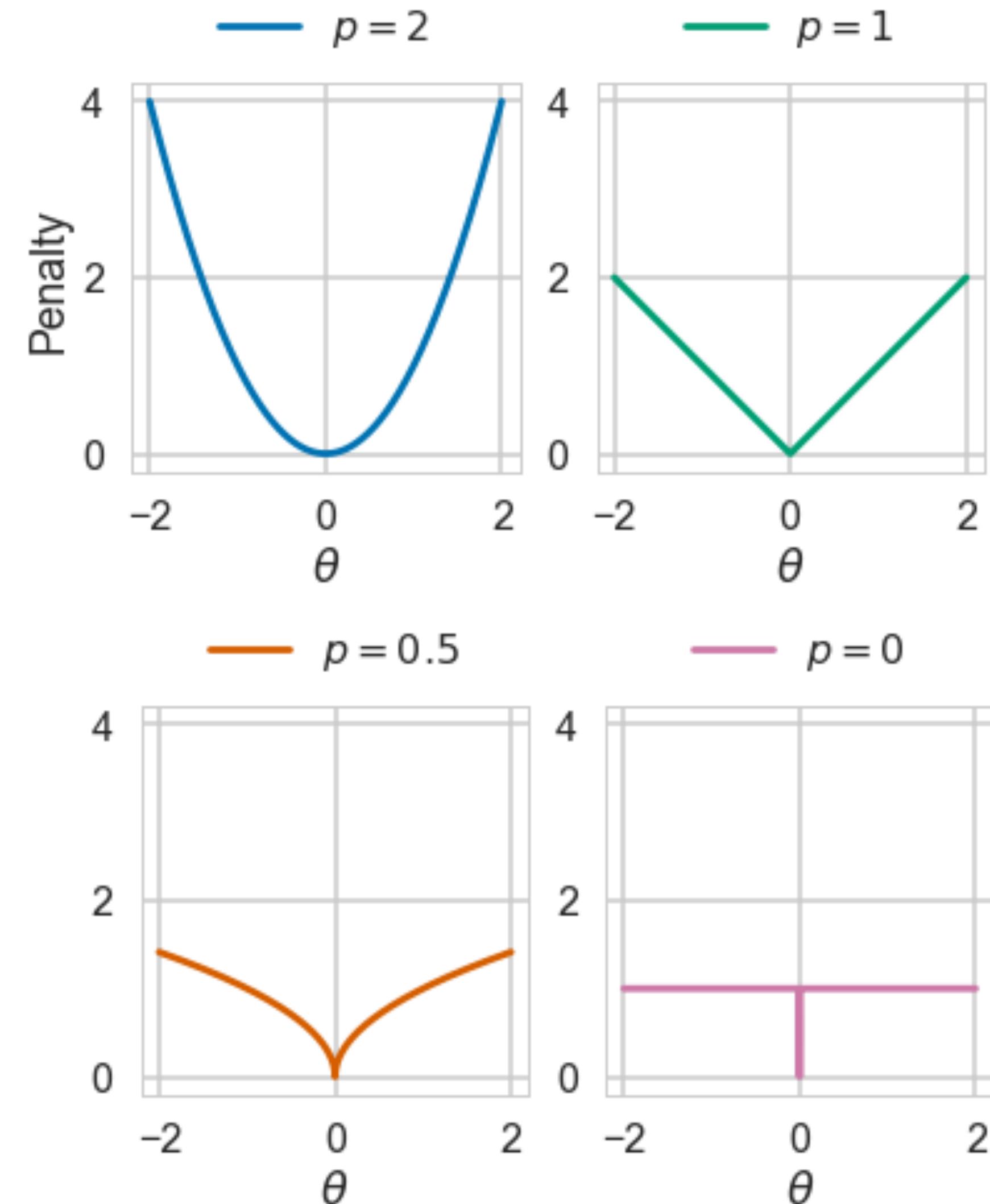
- For instance, L_p regularisation

$$L_p = \|\mathbf{w}\|^p = \sum_i |w_i|^p$$

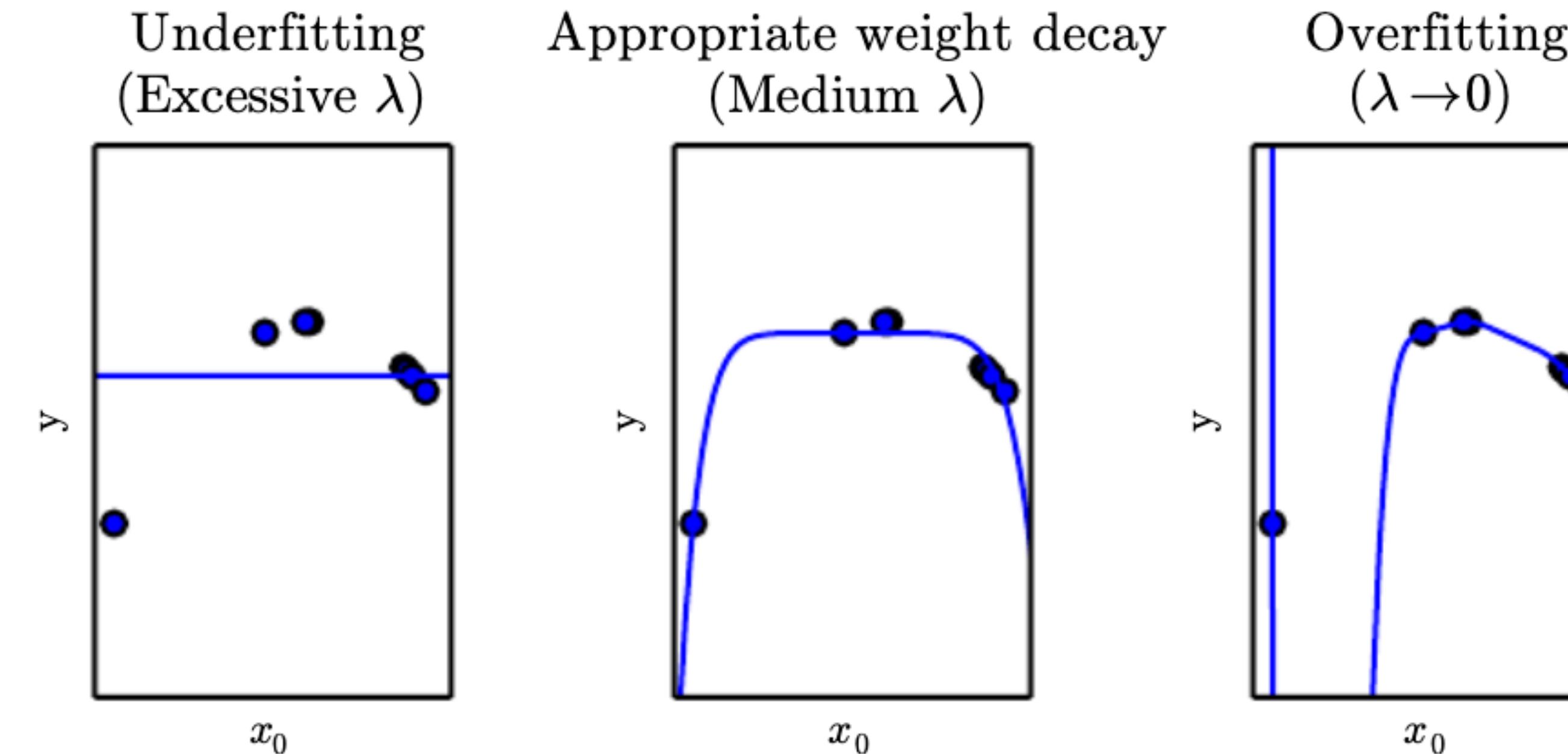
- The minimisation is a tradeoff between:

- pushing down the 1st term by taking advantage of the parameters

- pushing down the 2nd term by switching off the parameters



Regularisation in the loss

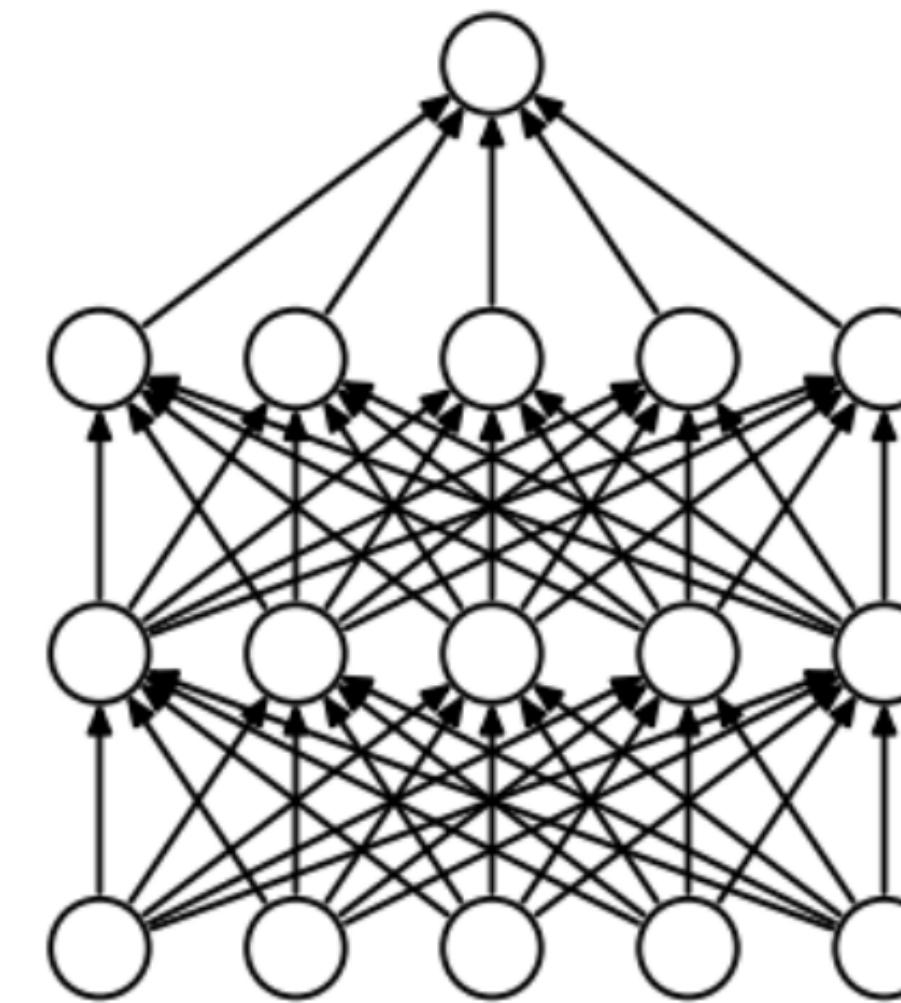


$$J(\mathbf{w}) = \text{MSE}_{\text{train}} + \lambda \mathbf{w}^\top \mathbf{w}$$

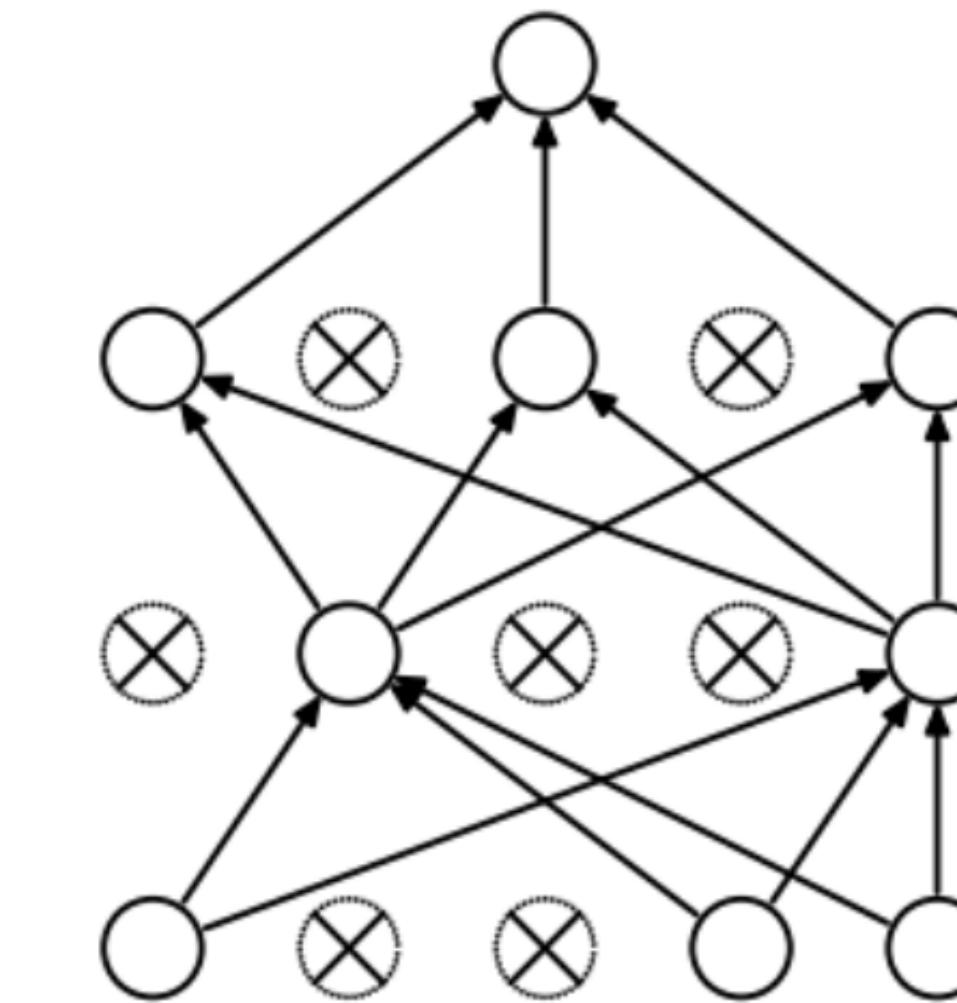
**E.g adding L2 norm to a linear regression problem encourages weights to be small
(put weight on fewer features, solutions with smaller slope)**

Regularisation with Drop out

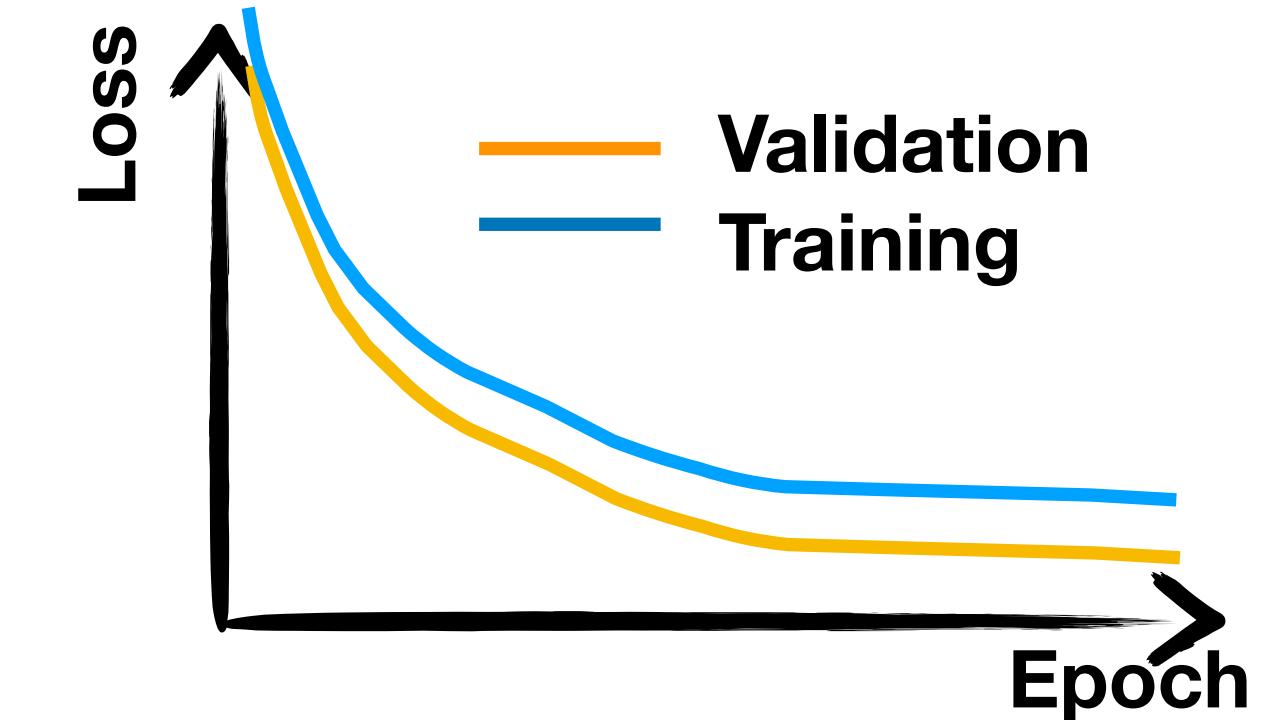
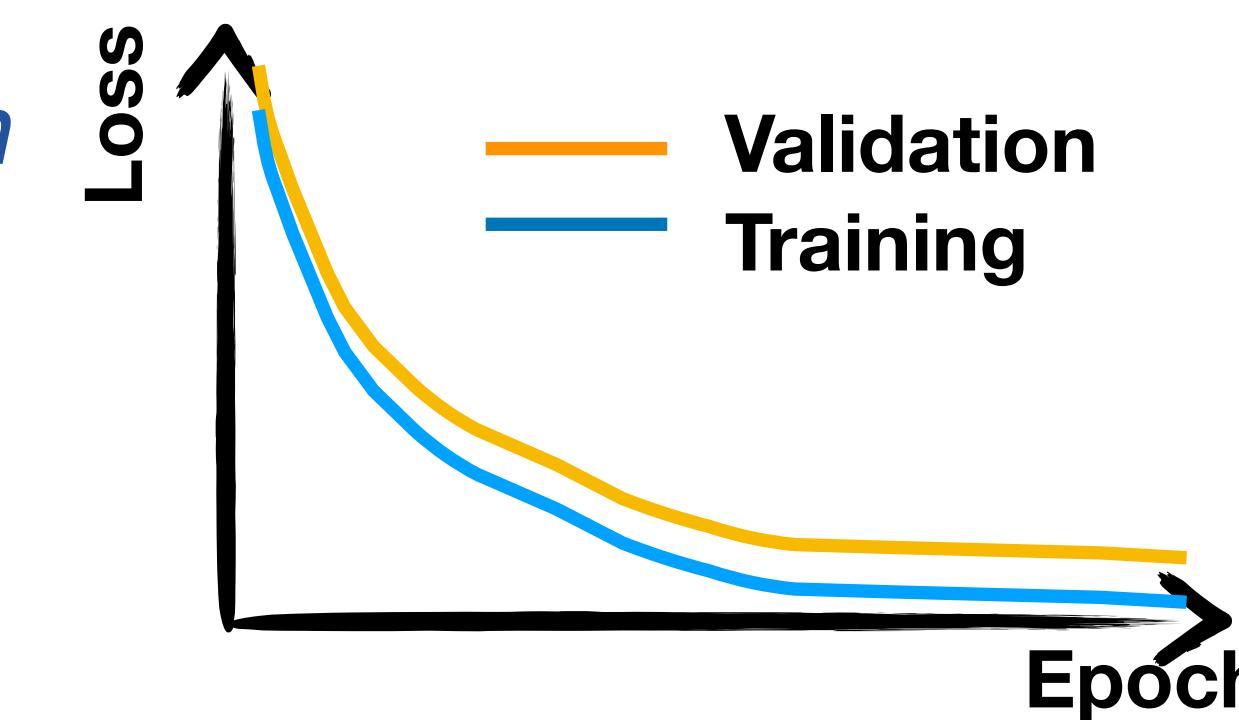
- A special kind of layer, introduced for regularisation purpose
- Randomly drop links between neurons, with probability p
- The connections are re-established during the validation and inference steps
- Typical sign of it: invert hierarchy between training and validation loss



(a) Standard Neural Net

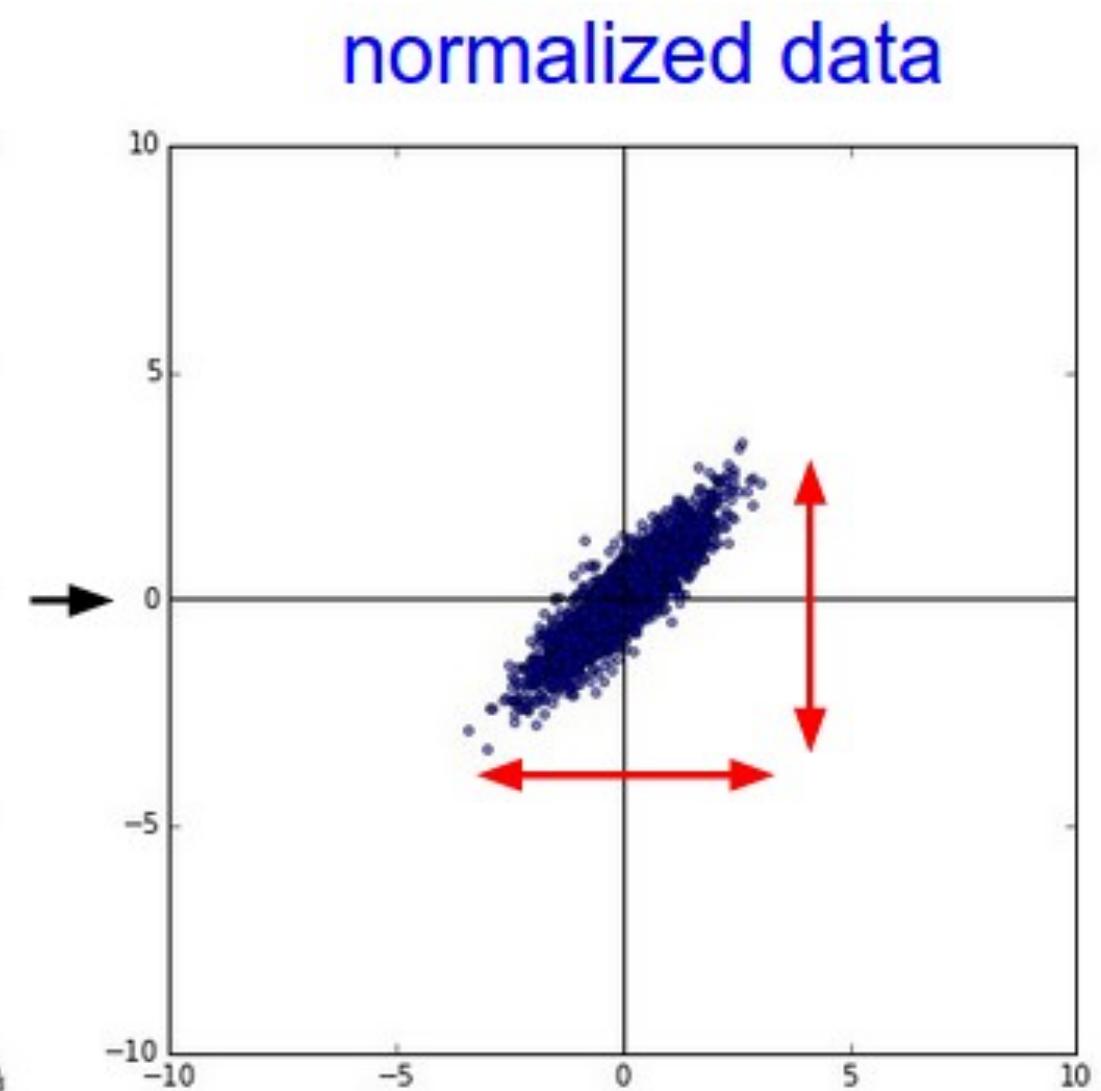
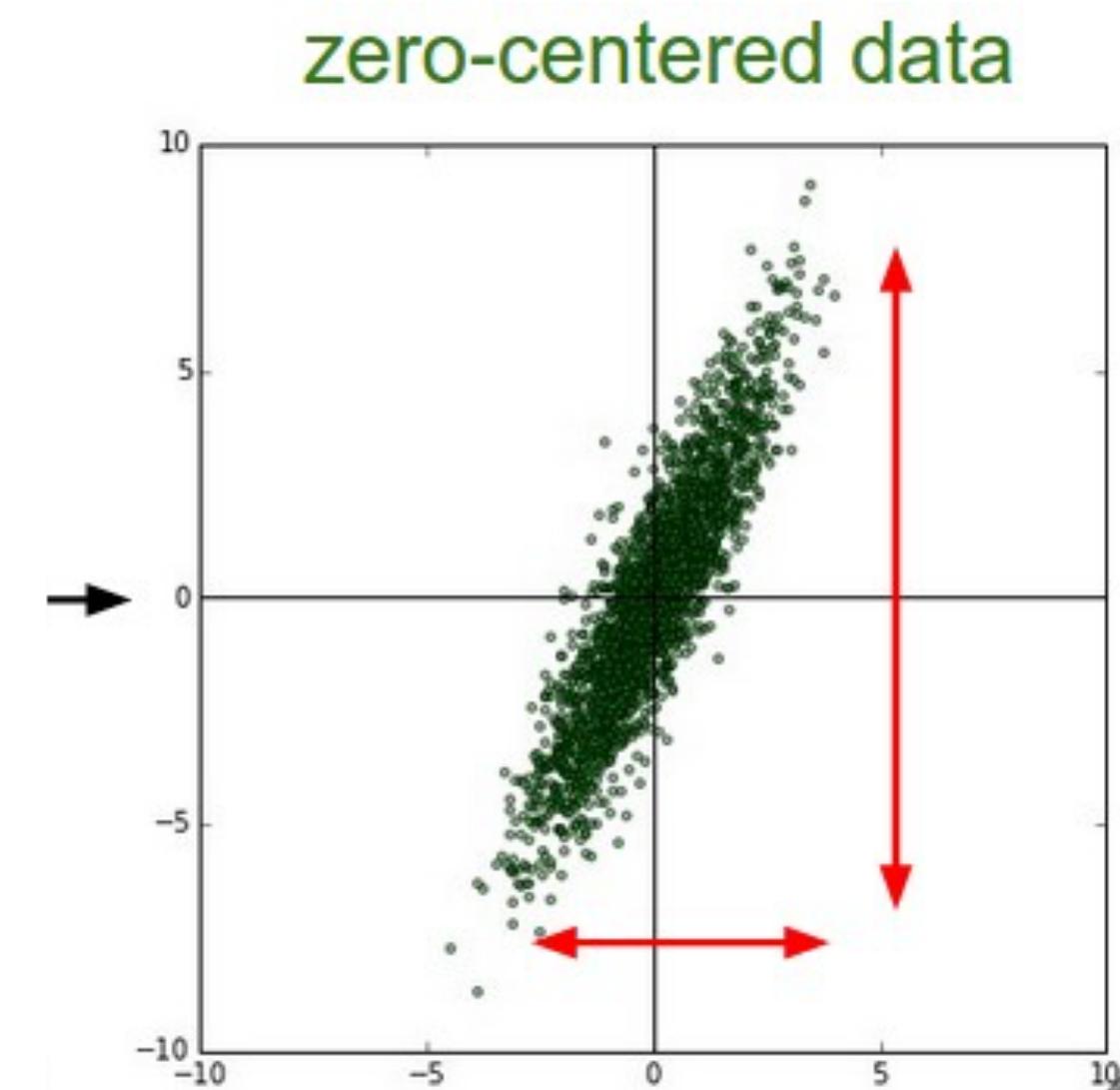
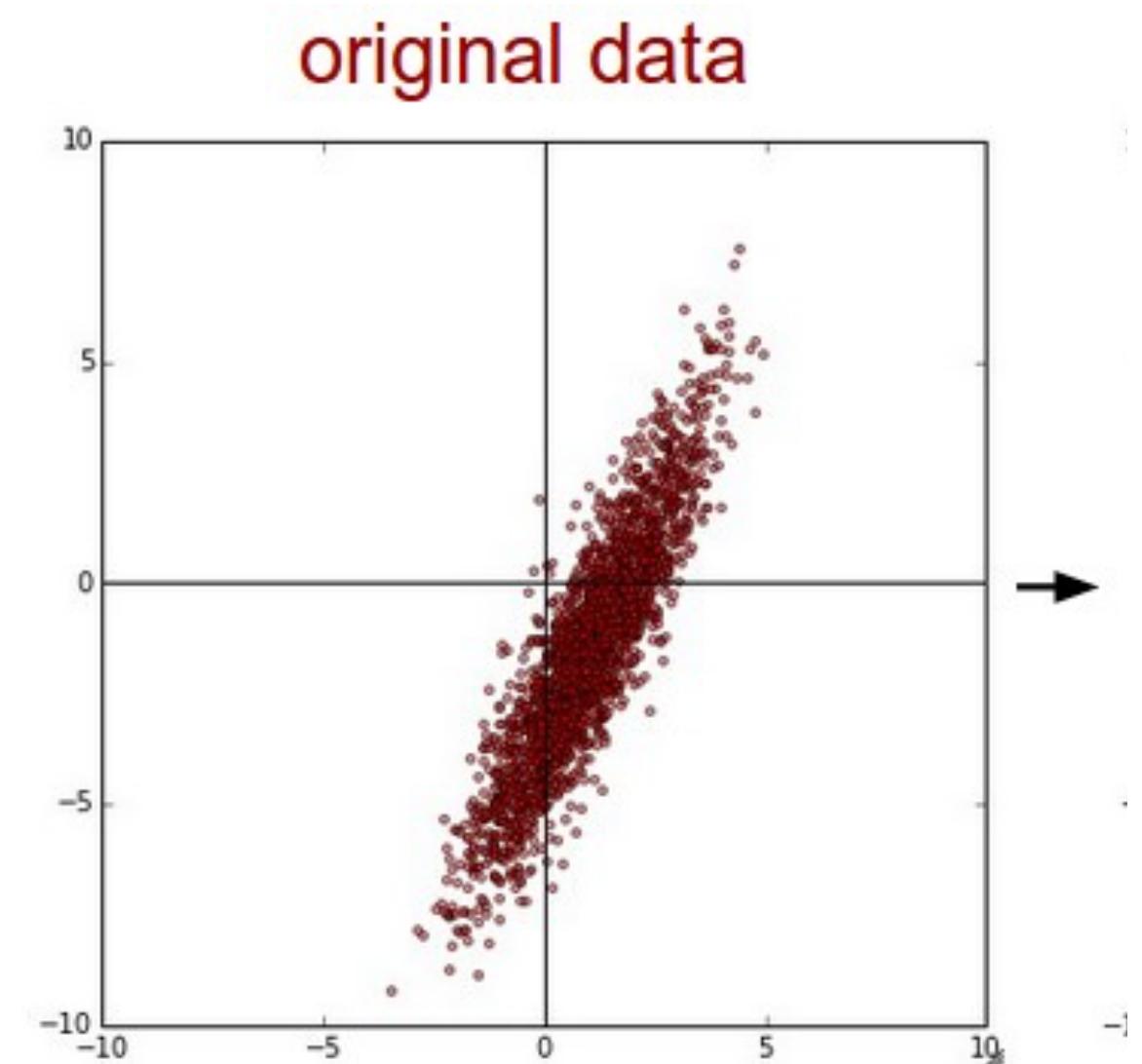


(b) After applying dropout.



Data normalization

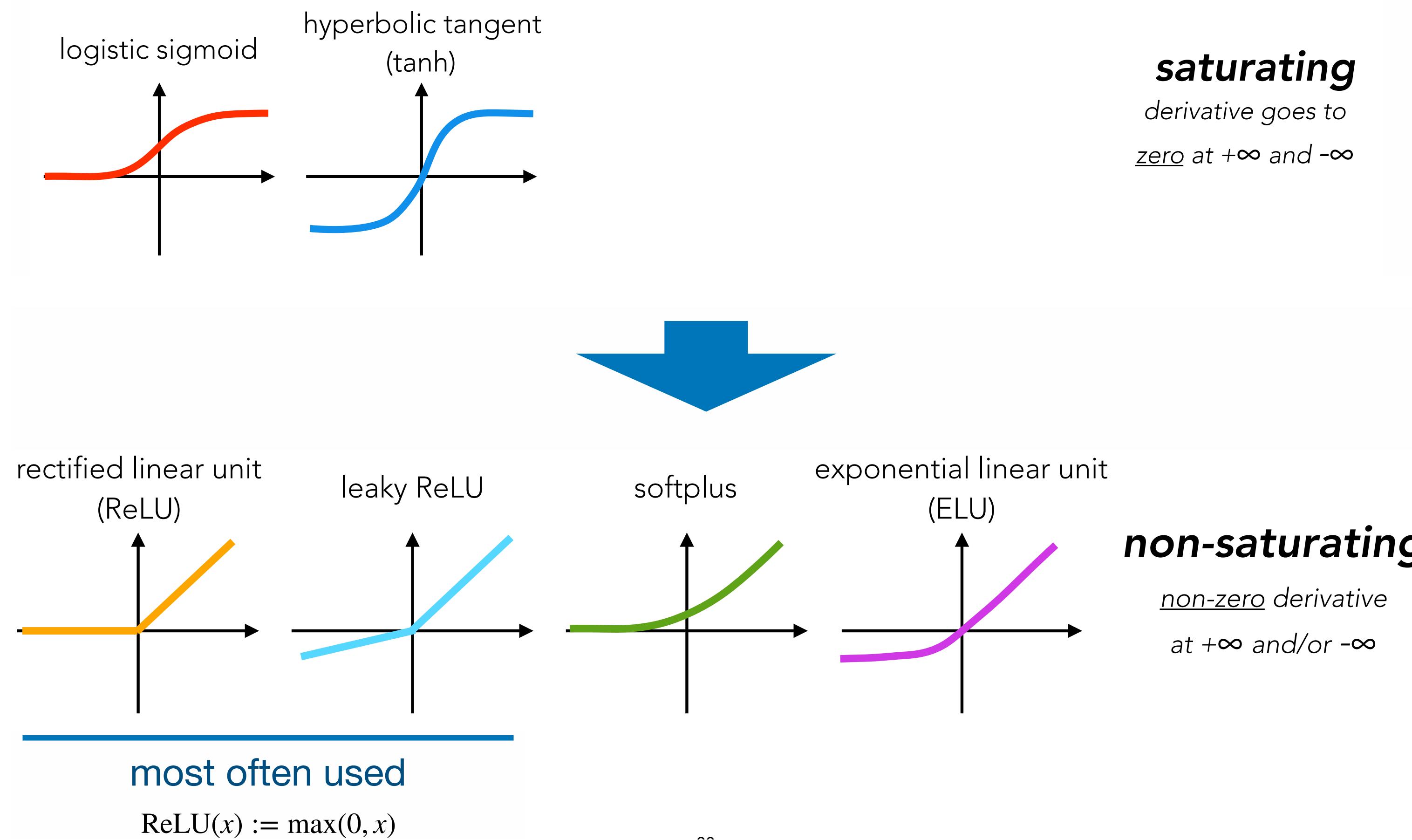
- It is good practice to give normalized inputs to a layer
- With all inputs having the same order of magnitude, all weights are equal important in the gradient
- Prevents explosion of the loss function
- This can be done automatically with BatchNormalization
- non-learnable shift and scale parameters, adjusted batch by batch



Non-linearities

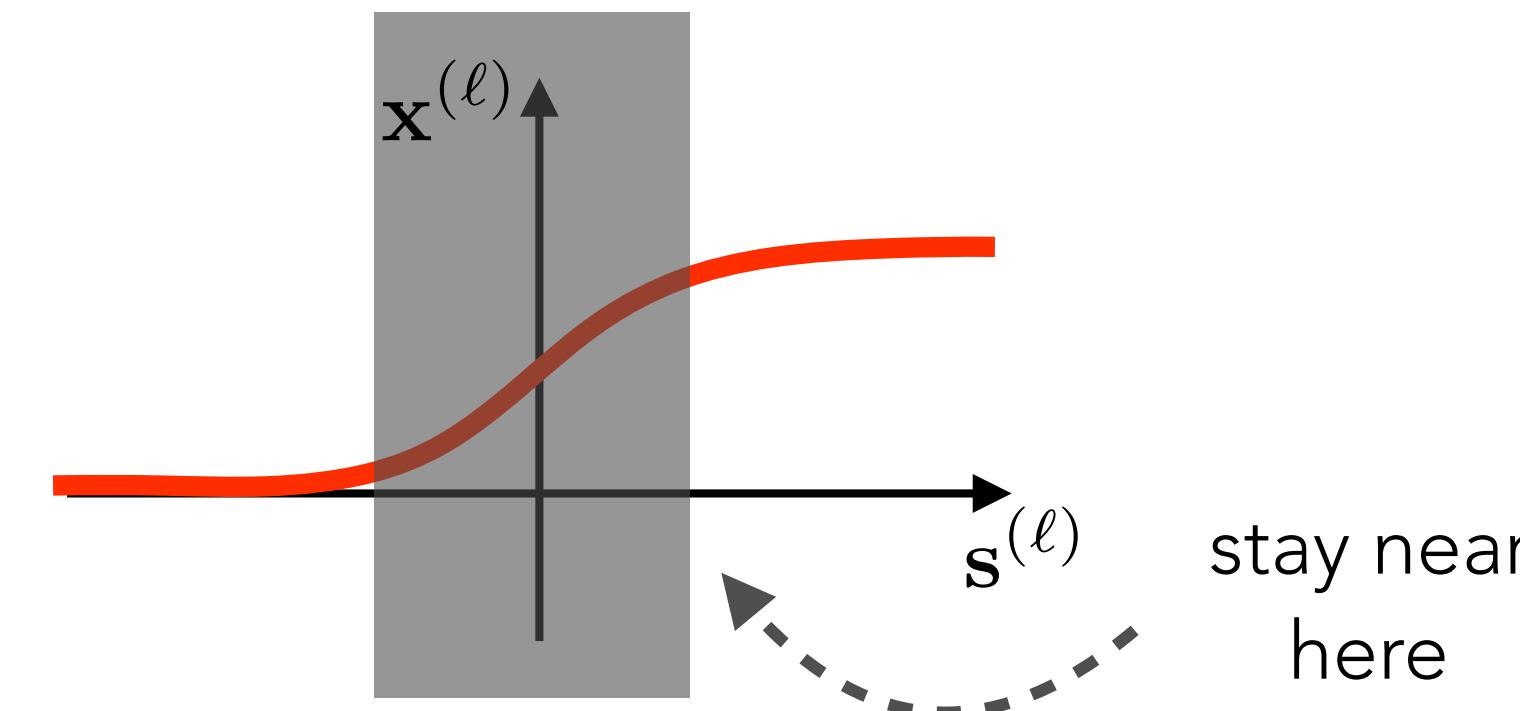
Vanishing gradient:

- Some activation functions have small derivatives almost everywhere
- In backpropagation, the product of many small terms goes to zero! → Gradient “vanishes”



Batch normalisation

Want to keep inputs within range of the non-linearity function!



We can normalize the activations before applying the non-linearity

$$s \leftarrow \frac{s - \text{shift}}{\text{scale}}$$

Array Manipulation

- Dense NN architectures can be made more complex

- Multiple inputs

- Multiple outputs

- Different networks branches

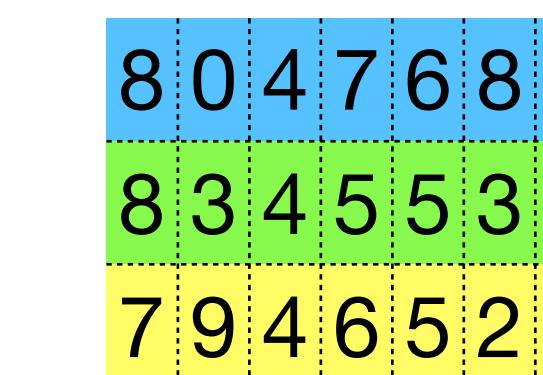
- This is possible thanks to layer-manipulation layers

- Add, Subtract, etc.

- Concatenation

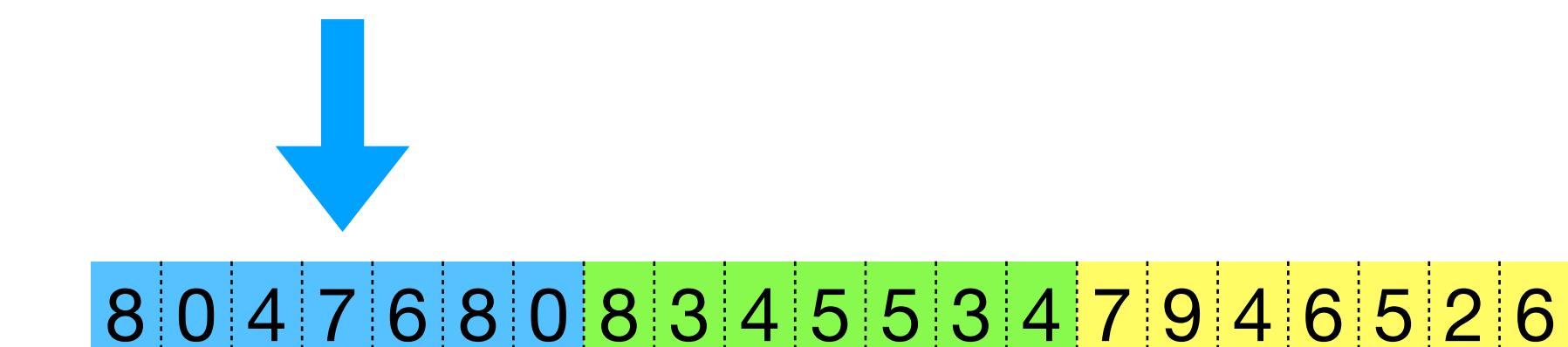
- Flattening

- All these operations are usually provided with NN training libraries



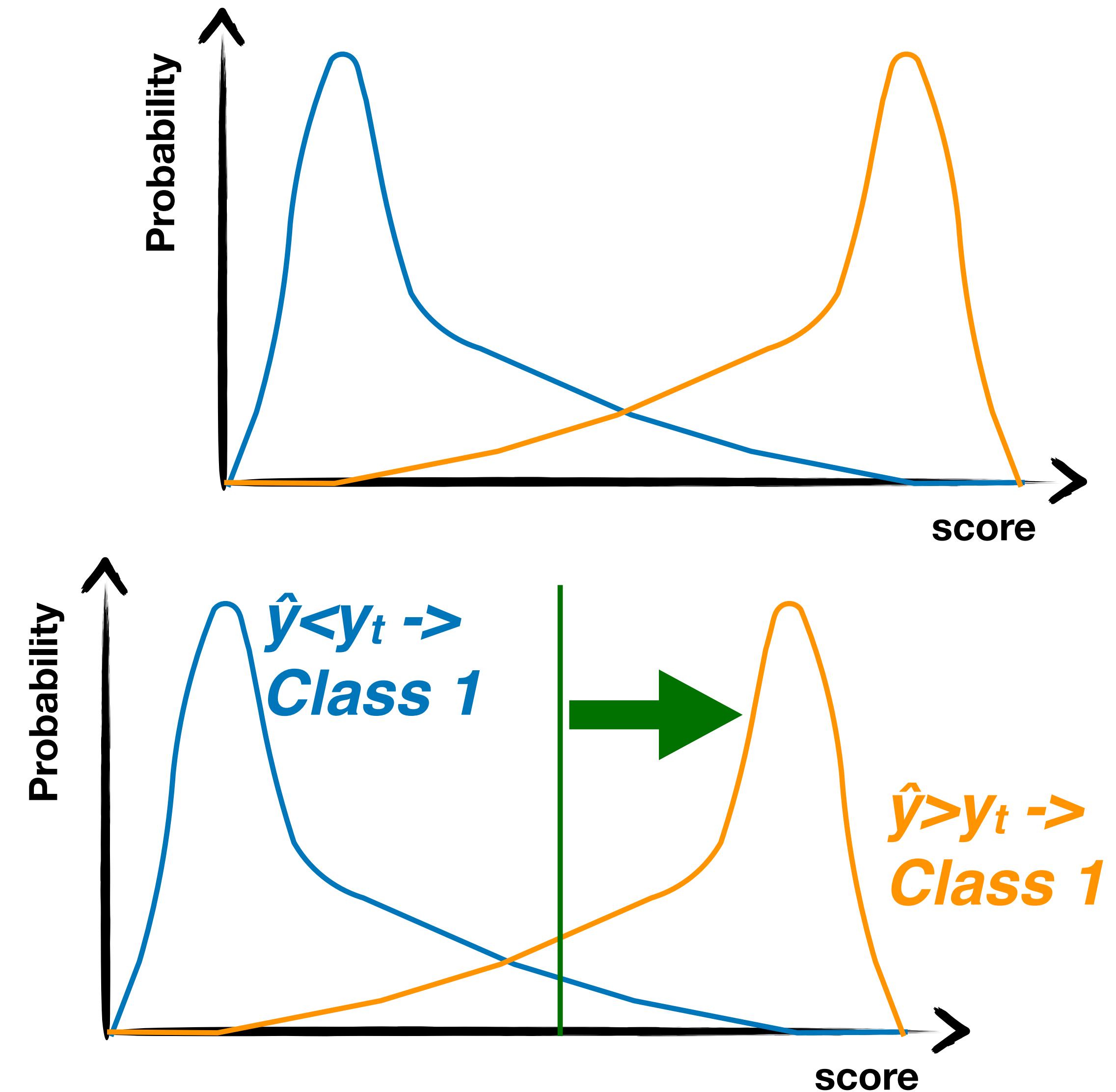
A large blue arrow pointing downwards, positioned next to the word "Flattening".

Flattening



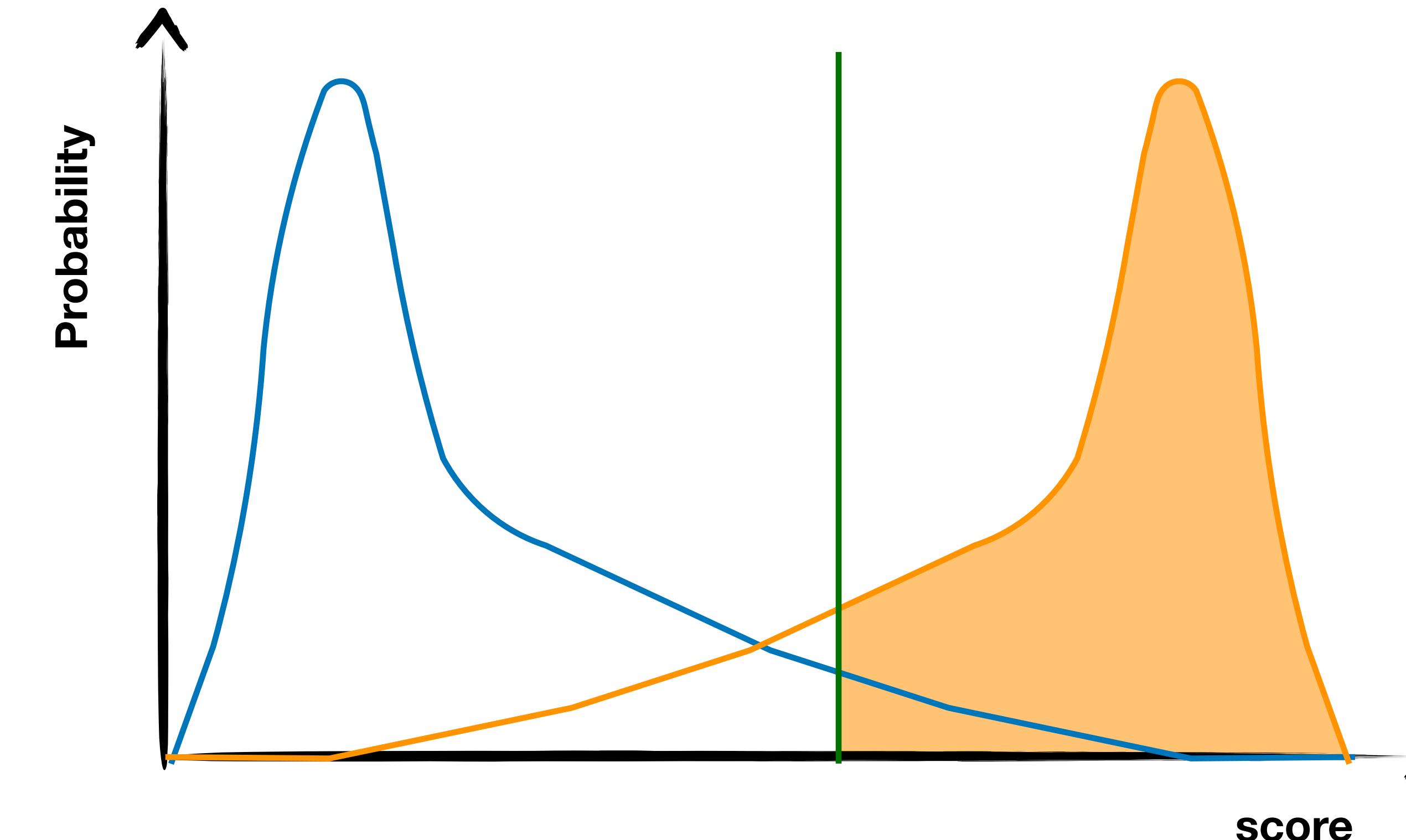
Classifier metrics

- Consider a binary classifier
- Its output \hat{y} is a number in $[0, 1]$
- If well trained, value should be close to 0 (1) for class-0 (class-1) examples
- One usually defines a threshold y_t such that:
 - $\hat{y} > y_t \rightarrow \text{Class 1}$
 - $\hat{y} < y_t \rightarrow \text{Class 0}$



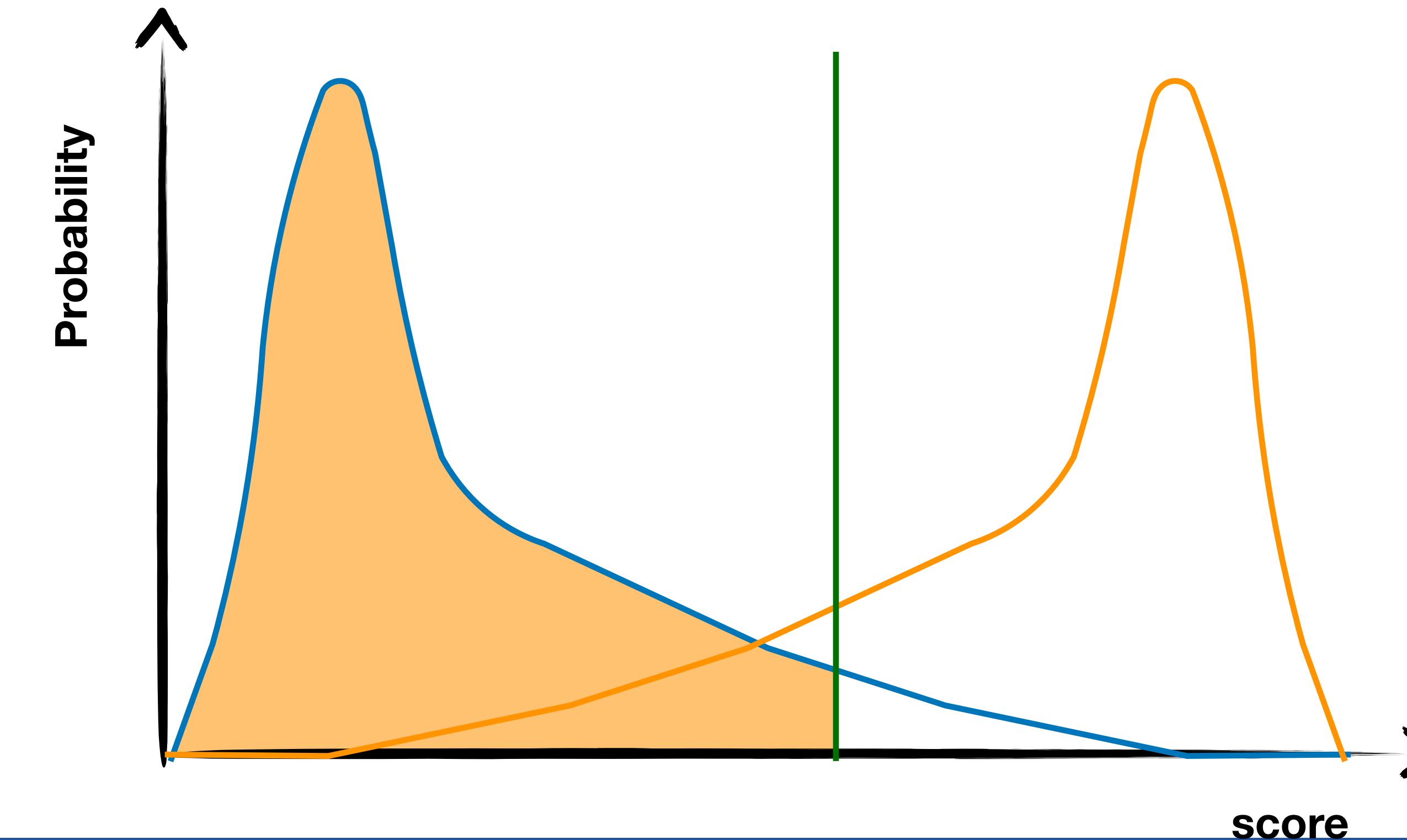
Classifier metrics

- A given threshold defines the following qualities
 - True-positives: Class-1 events above the threshold
 - True-negatives: Class-0 events below the threshold
 - False-positives: Class-0 events above the threshold
 - False-negatives: Class-1 events below the threshold



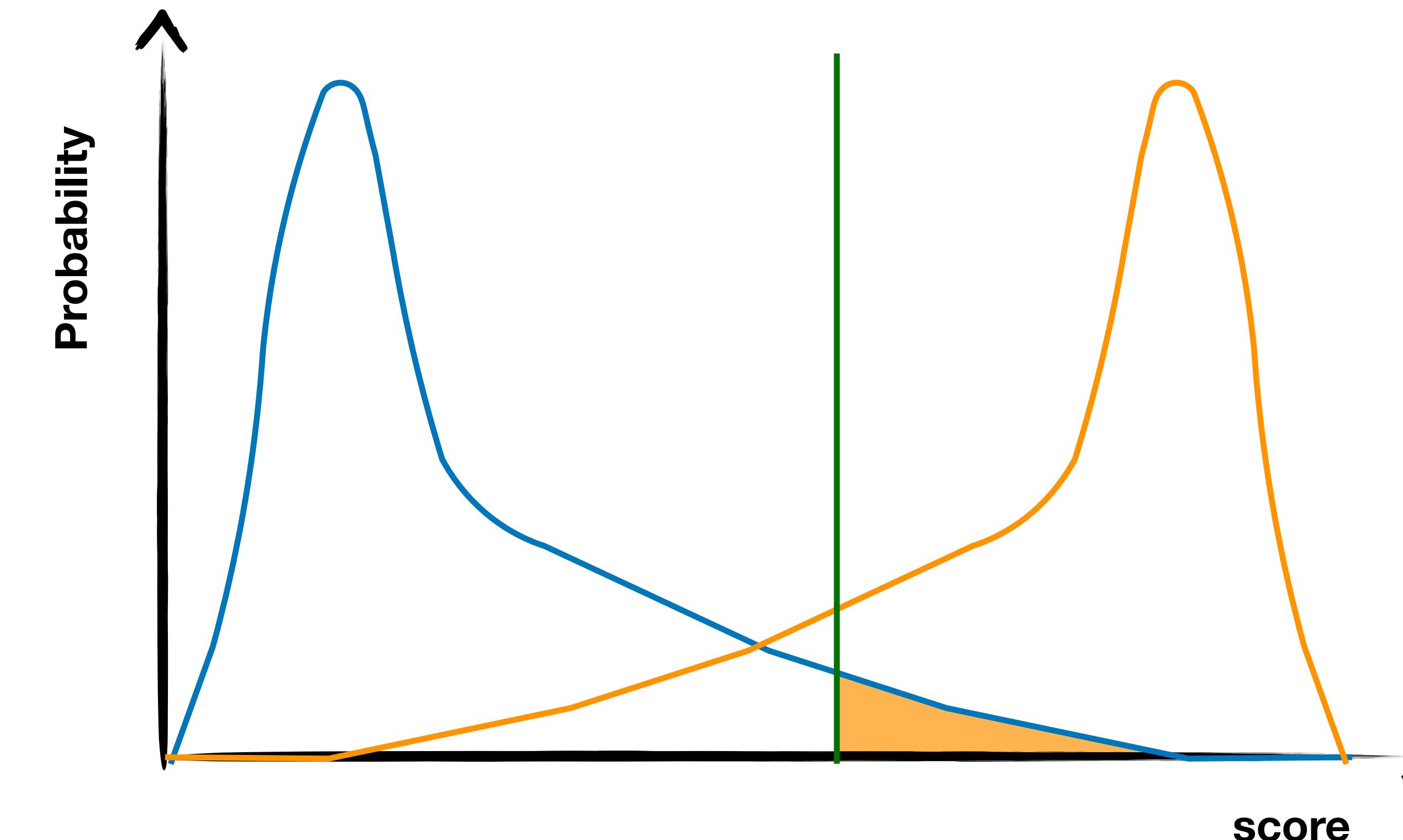
Classifier metrics

- A given threshold defines the following qualities
- True-positives: Class-1 events above the threshold
- True-negatives: Class-0 events below the threshold
- False-positives: Class-0 events above the threshold
- False-negatives: Class-1 events below the threshold



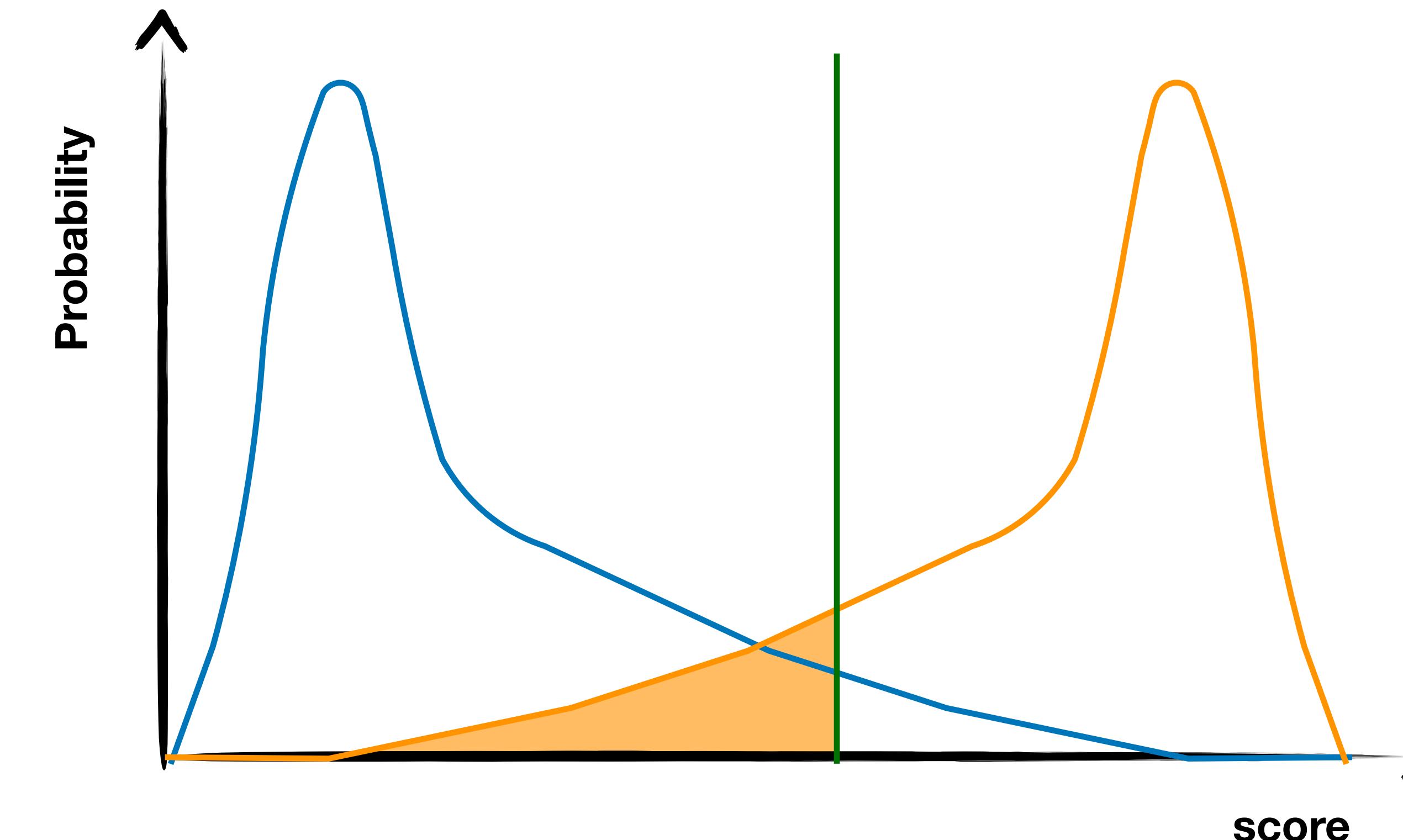
Classifier metrics

- A given threshold defines the following qualities
- True-positives: Class-1 events above the threshold
- True-negatives: Class-0 events below the threshold
- False-positives: Class-0 events above the threshold
- False-negatives: Class-1 events below the threshold



Classifier metrics

- A given threshold defines the following qualities
- True-positives: Class-1 events above the threshold
- True-negatives: Class-0 events below the threshold
- False-positives: Class-0 events above the threshold
- False-negatives: Class-1 events below the threshold

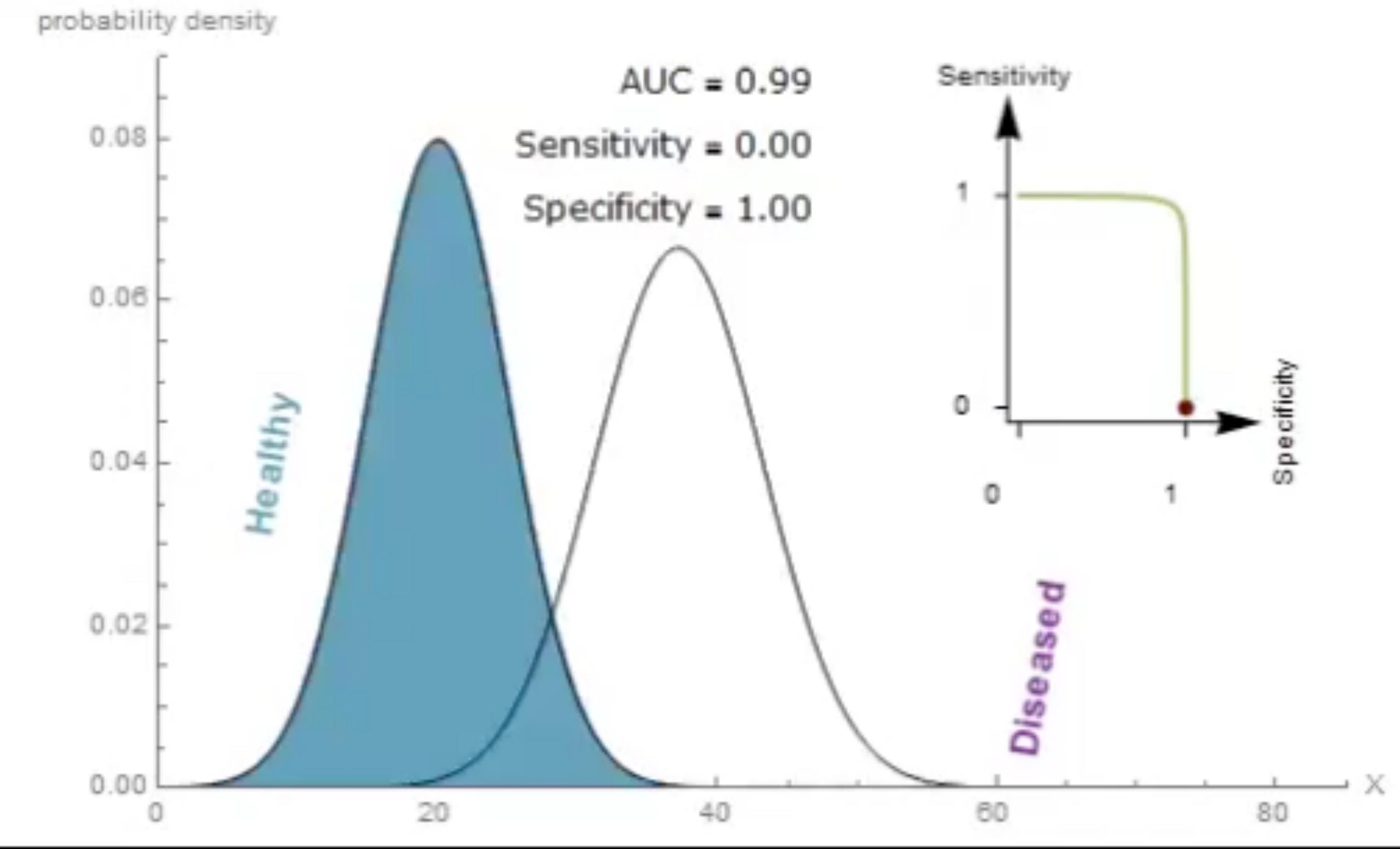




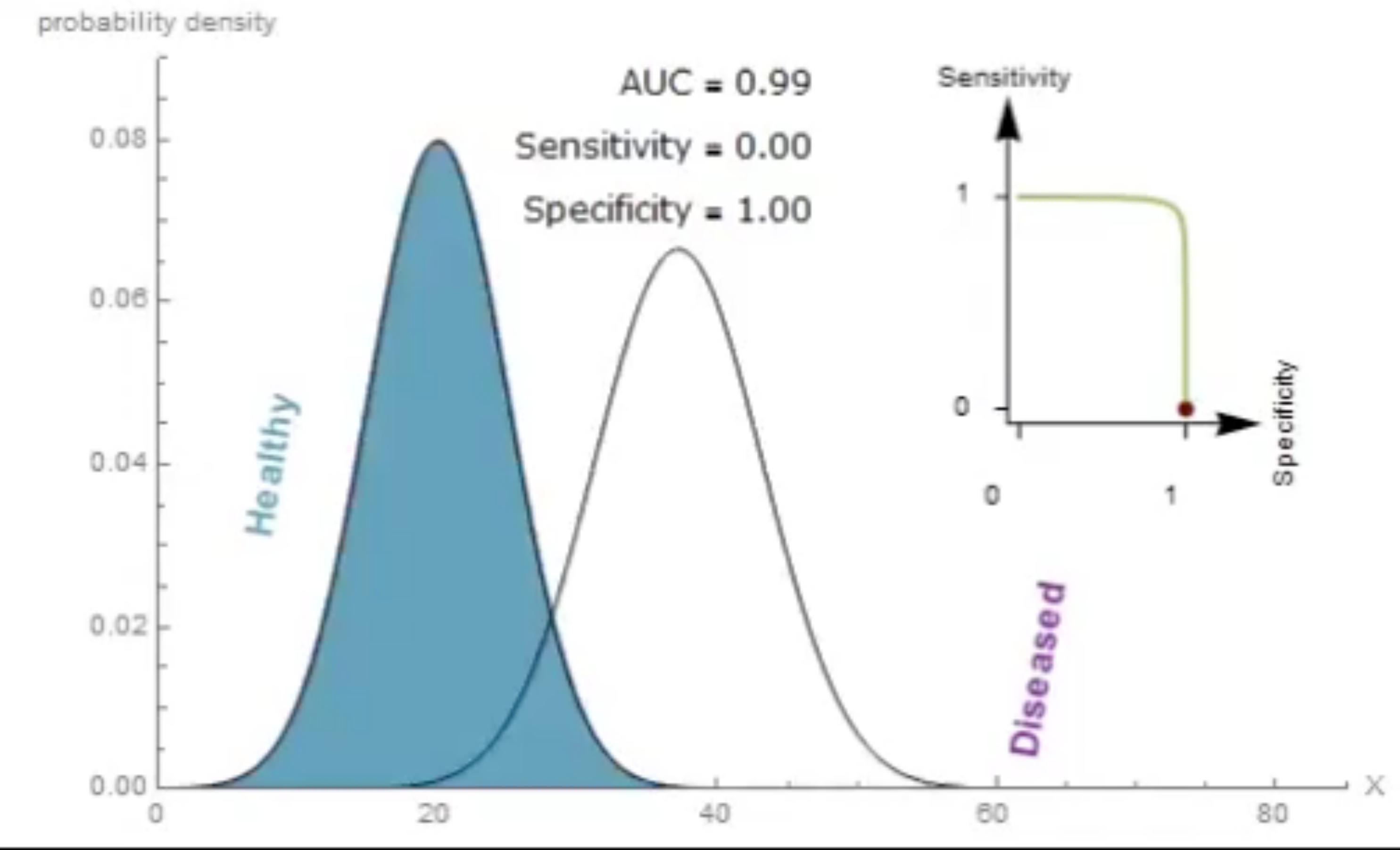
Classifier metrics

- *Starting ingredients are true positive (TP) and true negative (TN) rates*
- *Accuracy: $(TP+TN)/Total$*
 - *The fraction of events correctly classified*
- *Sensitivity: $TP/(Total \text{ positive})$*
 - *AKA signal efficiency in HEP*
- *Specificity: $TN/(Total \text{ negative})$*
 - *AKA mistag rate in HEP*

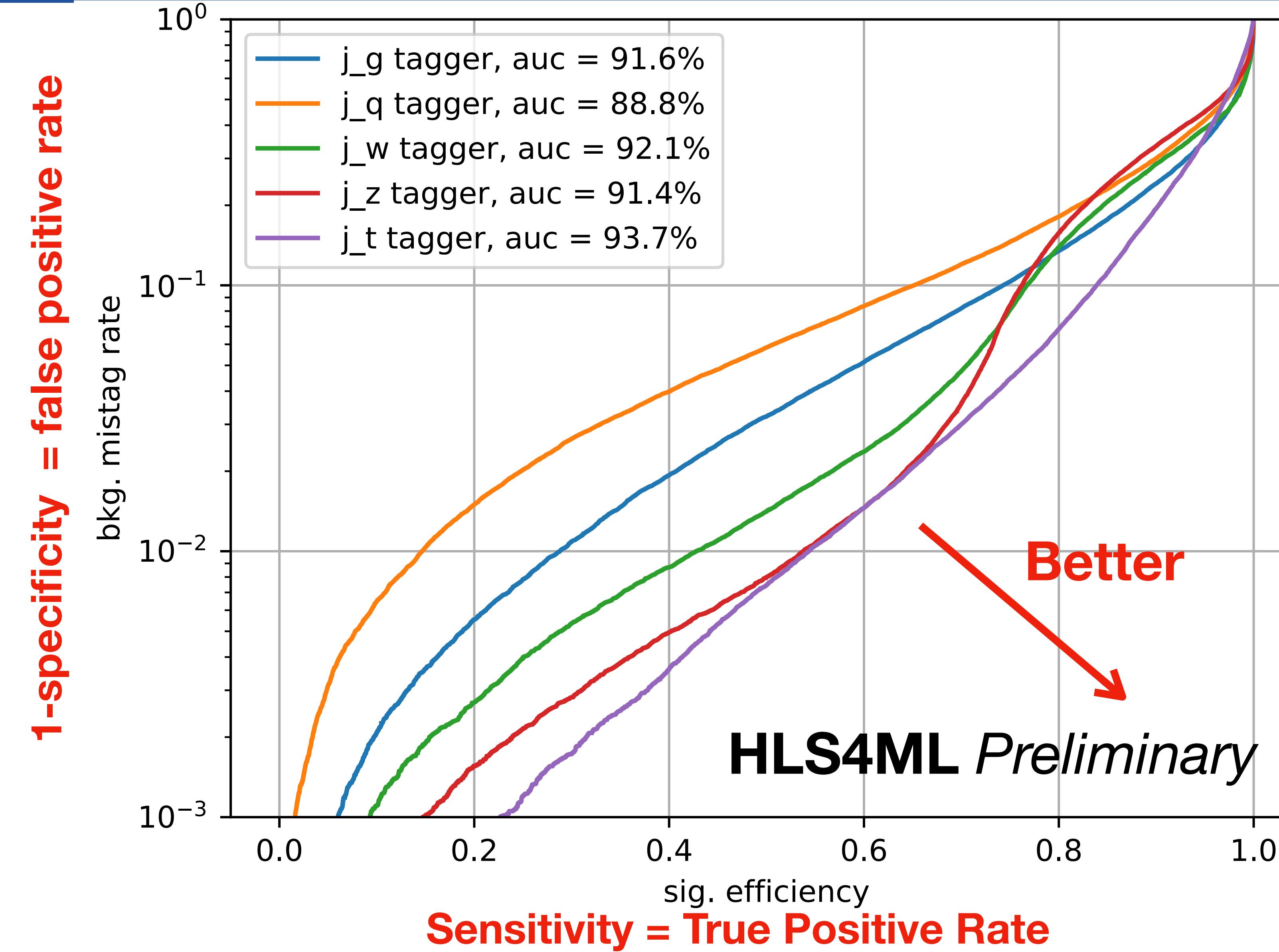
Receiver operating characteristic



Receiver operating characteristic



ROC curve



Confusion matrix

- A very effective visual tool to understand where your classifier works and where it doesn't
- Fill a matrix A_{ij} the fraction of examples of class C_i classified as class C_j
- Ideally, one would like to see the identity matrix
- In practise, one can look for large off-diagonal elements to see which classification is problematic

		Confusion Matrix			Total Predicted
		Actual Label			
Predicted Label	Actual Label	A	B	C	
	A	856 28.98%	58 1.96%	130 4.4%	1044 35.34%
B	B	0	765 25.90%	136 4.6%	901 30.5%
C	C	69 2.34%	33 1.12%	907 30.7%	1009 34.16%
Total Actual		925 31.31%	856 28.98%	1173 39.71%	2954 100%

Training Libraries

- Many solutions exist. Most popular softwares live in a python ecosystem
- Google's TensorFlow
- Facebook's Pytorch
- Apache MXnet
- All of them integrated in a data science ecosystem
- with numpy, scikit, etc.
- Convenient libraries built on top, with pre-coded ingredients
- Keras for TF (this is what we will be using)



TensorFlow



NumPy

Summary

- *Training a network requires work*
- *To prepare the data (dataset split, normalization, etc)*
- *To optimize the training efficiency (early stopping, regularization, etc.)*
- *To perform post-training diagnostic: training convergence, bias vs variance, accuracy, confusion*
- *Things might not work out of the box, but insight on pre-training, training and post-training techniques make your work easier*



Setting up COLAB

https://github.com/pierinim/tutorials/blob/master/GGI_Jan2021/Lecture1/Tutorial_Colab.pdf



Tutorial

← → ⌂ colab.research.google.com ☆ 5 | ↴ Relaunch to update :

TriggerShift Useful tools Trigger Jet substructure Zenseact Reads AXOL1TL FastML IEEE Offline Trigger Shift Scouting AXOL1TL ETH stuff Teaching Great talks Google Colab

Welcome To Colab File Edit View Insert Runtime Tools Help

Table of contents + Code + Text Copy to Drive Connect Gemini ^

Getting started Data science Machine learning More Resources Featured examples + Section

Examples Recent Google Drive GitHub Upload

Enter a GitHub URL or search by organization or user //github.com/pierinim/tutorials/tree/master/UZH/tutorials Include private repos

Repository: pierinim/tutorials Branch: master

Path

SMARTHEP/HandsOn2_CNN.ipynb

SMARTHEP/HandsOn2_RNN.ipynb

TCO/jupyter/ExploreData.ipynb

TCO/jupyter/Pion_Electron_CNN_Classifier.ipynb

UZH/tutorials/Tutorial1_DNN.ipynb

+ New notebook Cancel

Gemini Chat

This tutorial will guide you through the process of opening a Jupyter notebook from a GitHub repository directly within Google Colab. We'll start by cloning a repository containing several notebooks related to particle physics analysis.

1. Open the Colab interface and click on the "File" menu.

2. Select "Open Notebook" from the dropdown menu.

3. In the "Open notebook" dialog, enter the GitHub URL of the repository: `//github.com/pierinim/tutorials/tree/master/UZH/tutorials`. You can also specify a branch if needed.

4. Click the "Search" button to find the notebooks.

5. Once the notebooks are listed, click on one to open it in a new tab.

6. You can now run the code cells and experiment with the data analysis.

7. When you're done, don't forget to save your work back to Google Drive!



Tutorial

 Tutorial1_DNN.ipynb Share Settings User Picture

File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

RAM Disk Gemini ▾

☰

🔍

Muliclass classifier on IRIS dataset

{x}

16s

▶ # Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import roc_curve, auc, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import label_binarize

0s [2] # Load the Iris dataset
iris = load_iris()
X = iris.data # Features
y = iris.target # Labels

0s [3] # One-hot encode the labels: use (1,0,0), (0,1,0), (0,0,1)
y = to_categorical(y)

0s [6] print(y.shape)
print(X.shape)

→ (150, 3)
(150, 4)

↔ # Split data into training (60%), test (20%), and validation (20%)

Tutorial

The Iris Dataset

This data set consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray

The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.

The below plot uses the first two features. See [here](#) for more information on this dataset.

```
# Code source: Gaël Varoquaux
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause
```



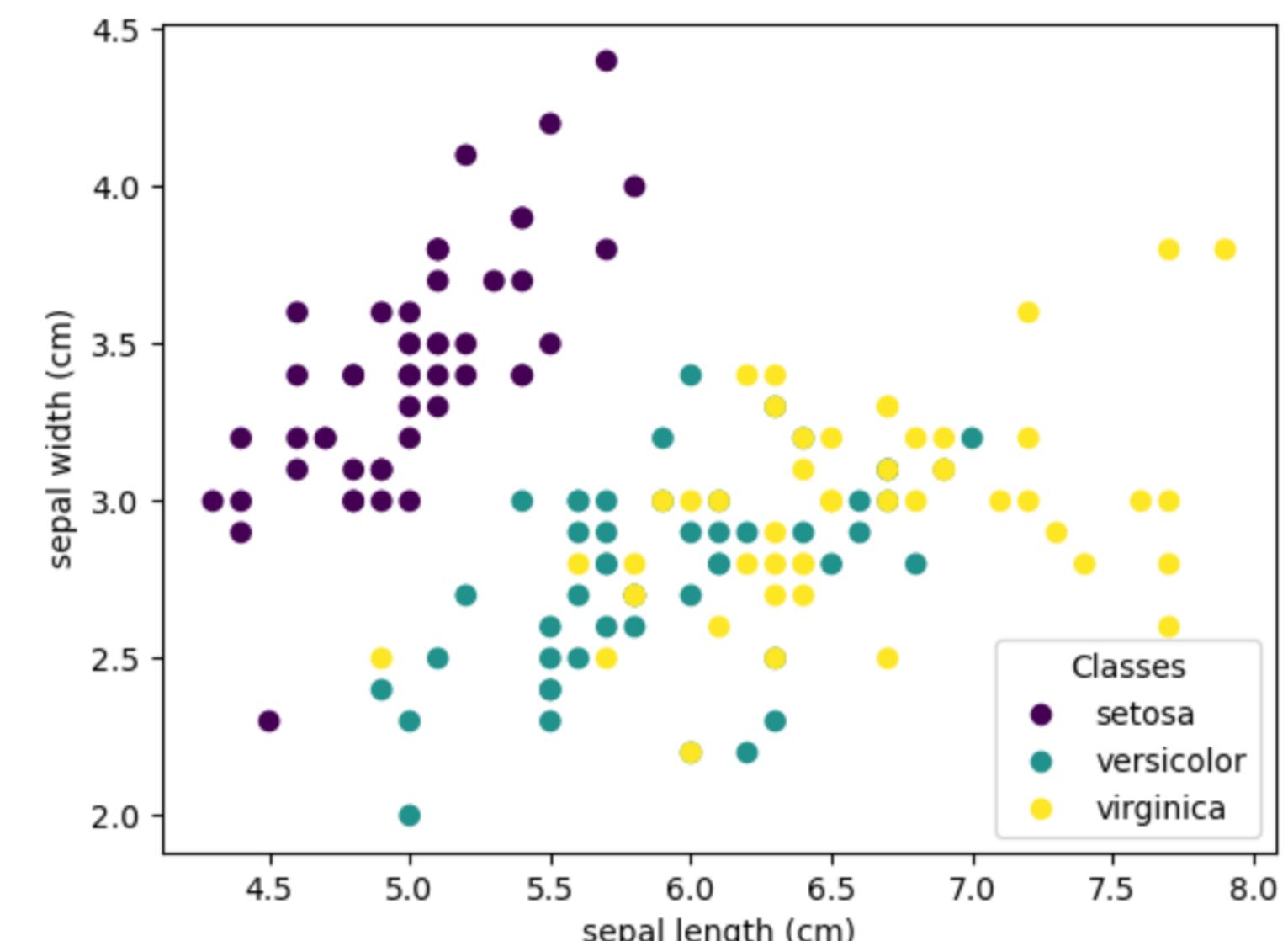
Iris Versicolor

Iris Setosa

Iris Virginica

Loading the iris dataset

```
from sklearn import datasets
iris = datasets.load_iris()
```



https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html



Tutorial

```
[12] # Add layers
from tensorflow.keras.layers import Input
model.add(Input(shape=X_train.shape[1],))
model.add(Dense(16, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(16, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax')) # Output layer

→ /usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do n
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```



Tutorial

Data Dictionary		
Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

In this challenge, we ask you to build a predictive model that answers the question: "what sorts of people were more likely to survive?" using passenger data (ie name, age, gender, socio-economic class, etc).

<https://www.kaggle.com/c/titanic/overview>



Tutorial

In this challenge, we ask you to build a predictive model that answers the question: "what sorts of people were more likely to survive?" using passenger data (ie name, age, gender, socio-economic class, etc).

Data Dictionary		
Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

<https://www.kaggle.com/c/titanic/overview>



Tutorial

In this challenge, we ask you to build a predictive model that answers the question: "what sorts of people were more likely to survive?" using passenger data (ie name, age, gender, socio-economic class, etc).

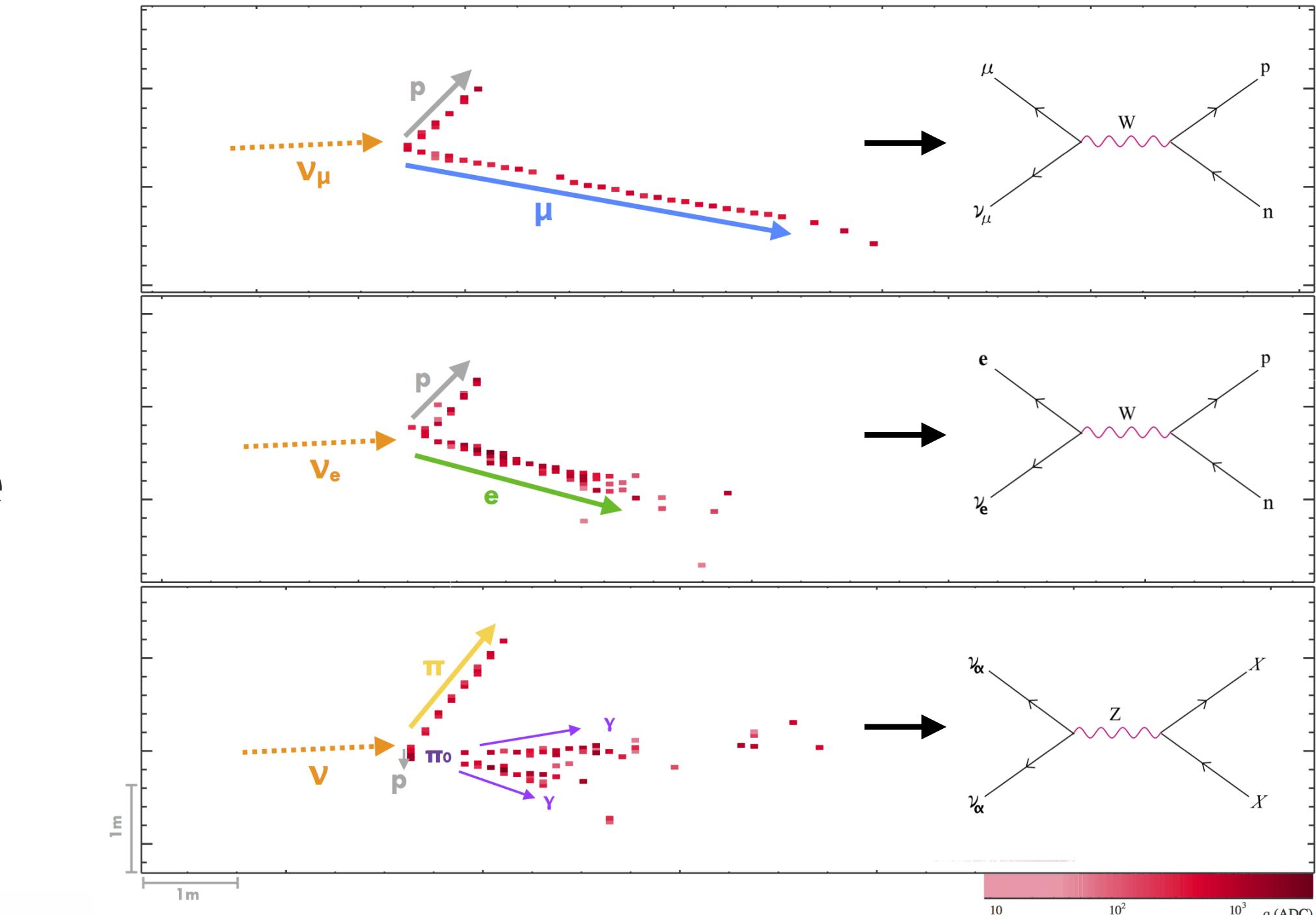
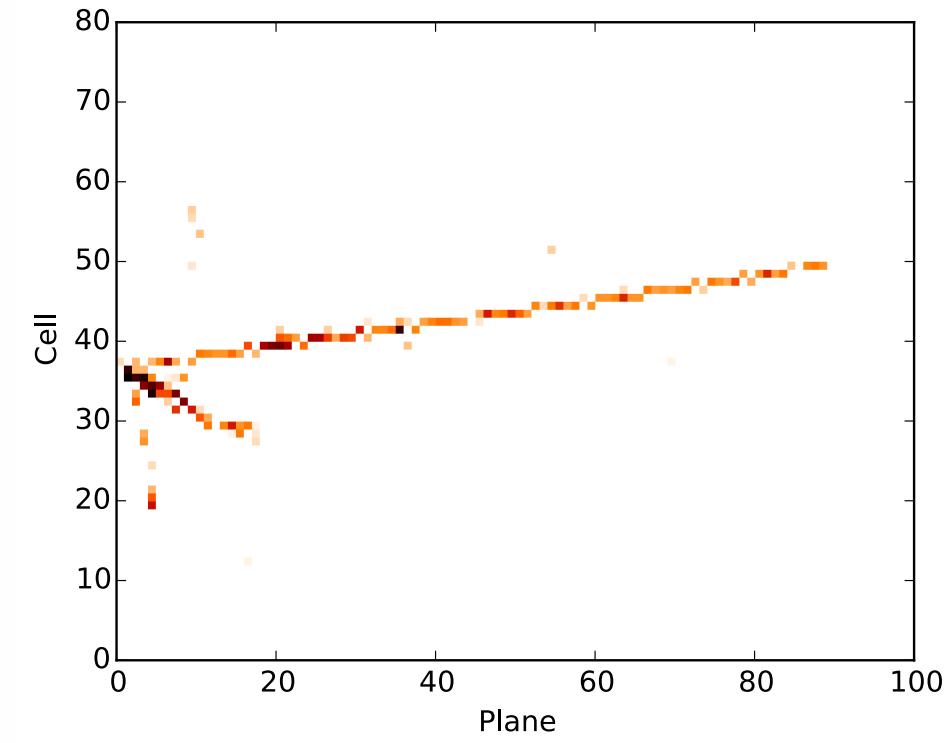
Data Dictionary		
Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

<https://www.kaggle.com/c/titanic/overview>

Losses

MSE is a standard loss when predicting a continuous value
(e.g regression)

What about classification?
Labels (y_i) 1 for correct class, 0 for the rest



$$w^T x = \begin{bmatrix} w_1^T x \\ w_2^T x \\ w_3^T x \end{bmatrix}$$

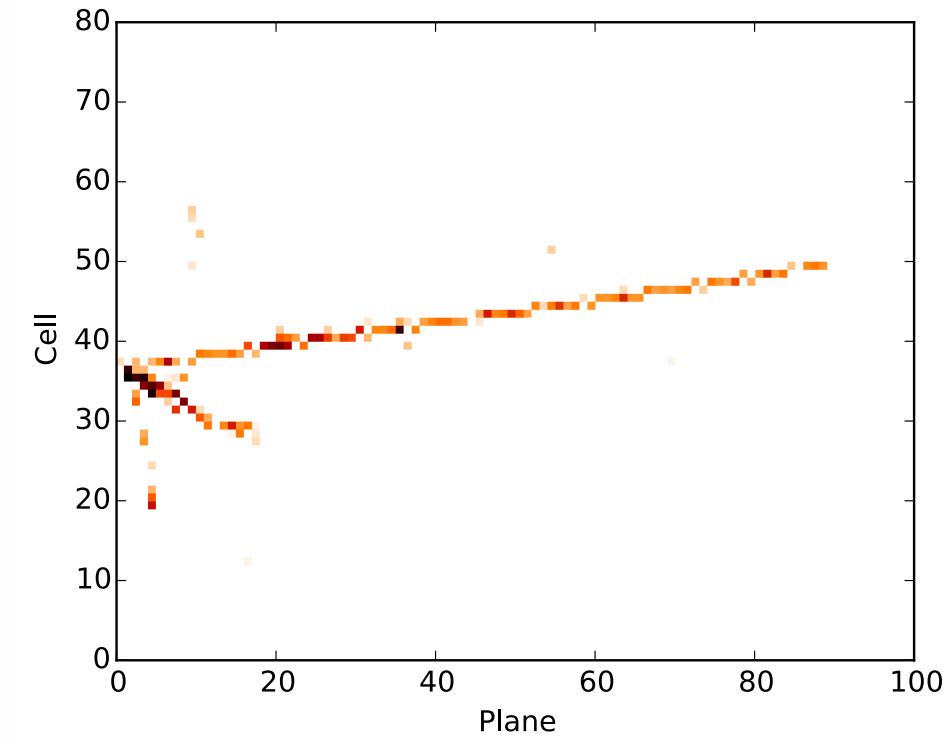
→

- $w_1^T x$ ← ν_μ CC score
- $w_2^T x$ ← ν_e CC score
- $w_3^T x$ ← NC score

Softmax

Apply an activation function to final output layer (in our case (3,1) for each interaction type):

- should push each value to be between 0 and 1, and
- should make sure the sum (in our case, across all 3) is equal to 1



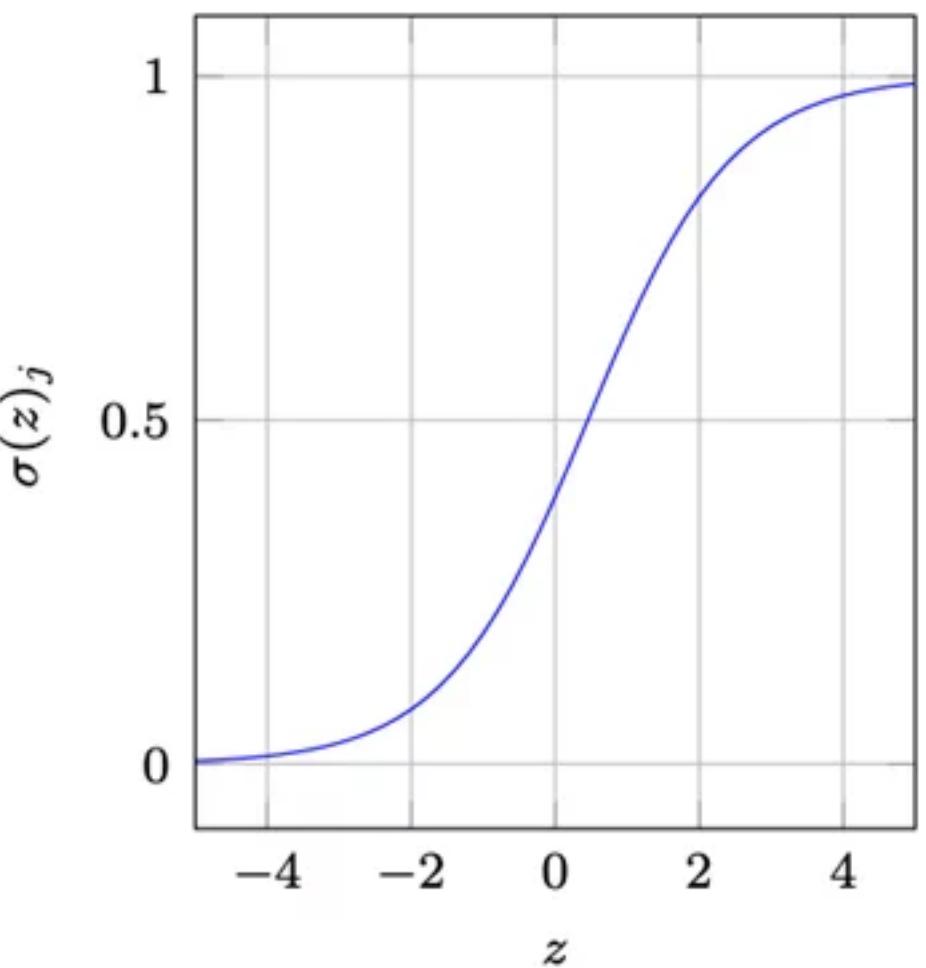
Apply some activation function here!

$$w^T x = \begin{bmatrix} w_1^T x \\ w_2^T x \\ w_3^T x \end{bmatrix}$$

w = vector containing weights

ν_μ CC score
 ν_e CC score
NC score

Softmax



$$\text{softmax} \left(\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_K \end{bmatrix} \right) = \frac{1}{\sum_{k=1}^K \exp(a_k)} \begin{bmatrix} \exp(a_1) \\ \exp(a_2) \\ \vdots \\ \exp(a_K) \end{bmatrix}$$

Probability of each class $\in \{0,1\}$

These are often called the logits Normalisation factor to make sum of probabilities=1

$$\begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \xrightarrow{\text{Softmax}} \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix}$$

Softmax

$$\text{softmax} \left(\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_K \end{bmatrix} \right) = \frac{1}{\sum_{k=1}^K \exp(a_k)} \begin{bmatrix} \exp(a_1) \\ \exp(a_2) \\ \vdots \\ \exp(a_K) \end{bmatrix}$$

Probability of each class $\in \{0,1\}$

These are often called the logits

Normalisation factor to make sum of probabilities=1

In contrast to other activation functions (like ReLu), softmax takes a vector as input and has a vector as output (since it needs to normalise across classes)

- argmax of output is the model class!
- Why exp? Because it hugely increase the probability of the **biggest** score and decrease the probability of the lower scores

Crossentropy loss

Given y , try to make probability for corresponding entry in output vector as high as possible:

$$\hat{y} = \begin{bmatrix} 0.9 \\ 0.02 \\ 0.08 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

How? Cross-entropy. The cross-entropy loss for a softmax output layer is given by

$$L(y, \hat{y}) = - \sum_{i=1}^n y_i \log(\hat{y}_i),$$

y : True probability distribution (one-hot encoded)

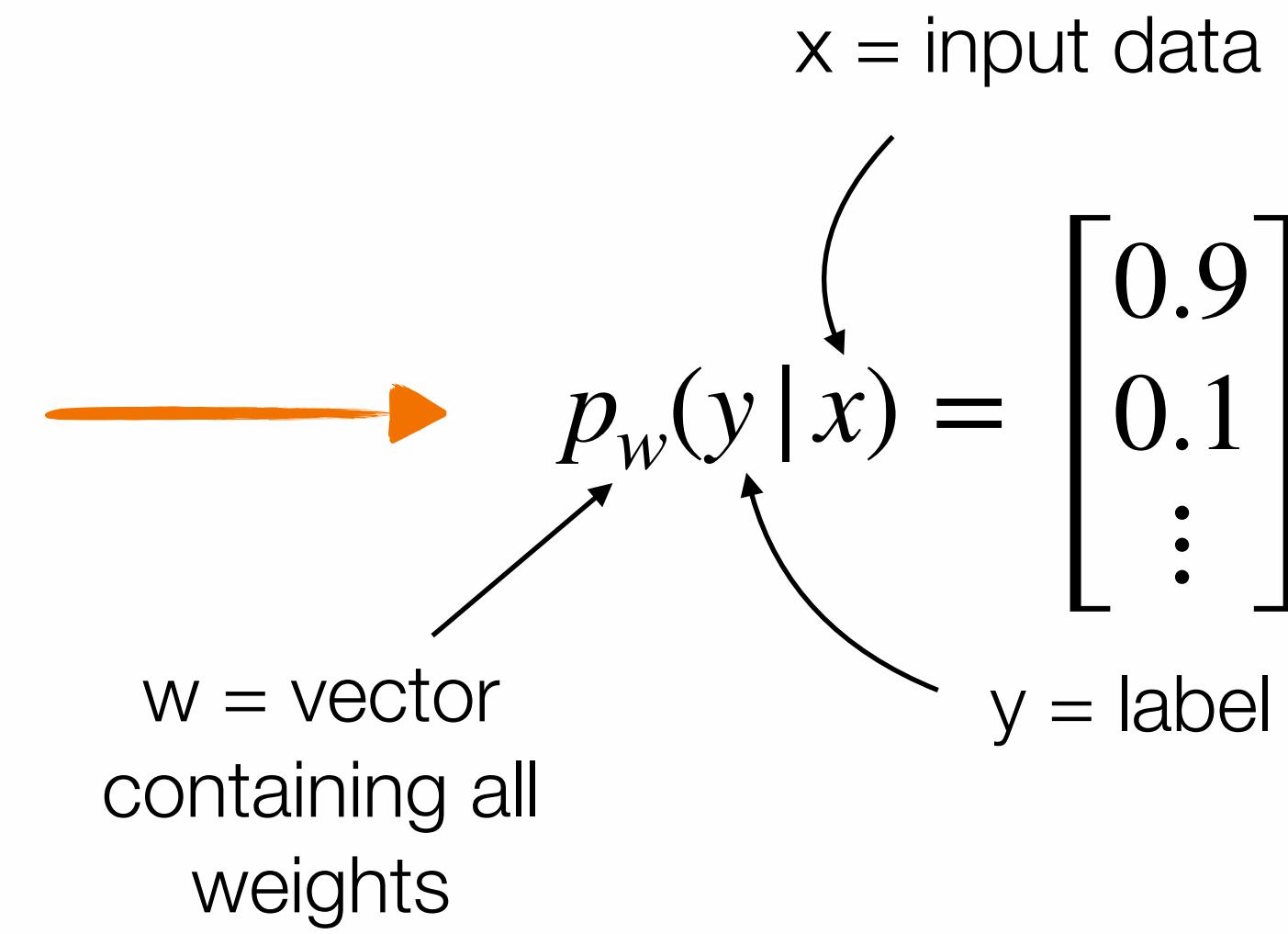
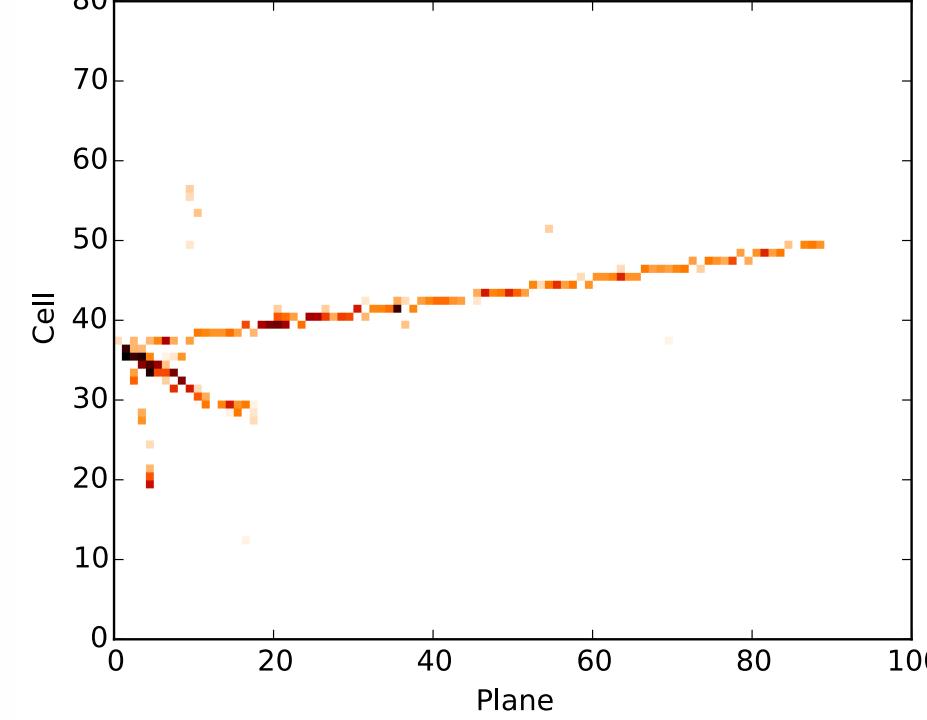
\hat{y} : Predicted probability distribution (softmax output)

n : Number of classes

All terms with 0 in y , will be zero! Only term left is “correct class” term, in our case $-\hat{y}_1 \log y_1$

Gradient descent wants to make this term small, must make y_1 as large as possible (can be at most 1)!

Training



Train such that w^* maximise the log likelihood of the data under our model:

$$\mathcal{L}_{cross-entropy}(w) = -\frac{1}{N} \sum_{n=1}^N \log \hat{P}(Y = y_n | X = x_n) = -\frac{1}{N} \sum_{n=1}^N \log \frac{e^{f(x_n; w)_{y_n}}}{\sum_z e^{f(x_n; w)_z}}$$

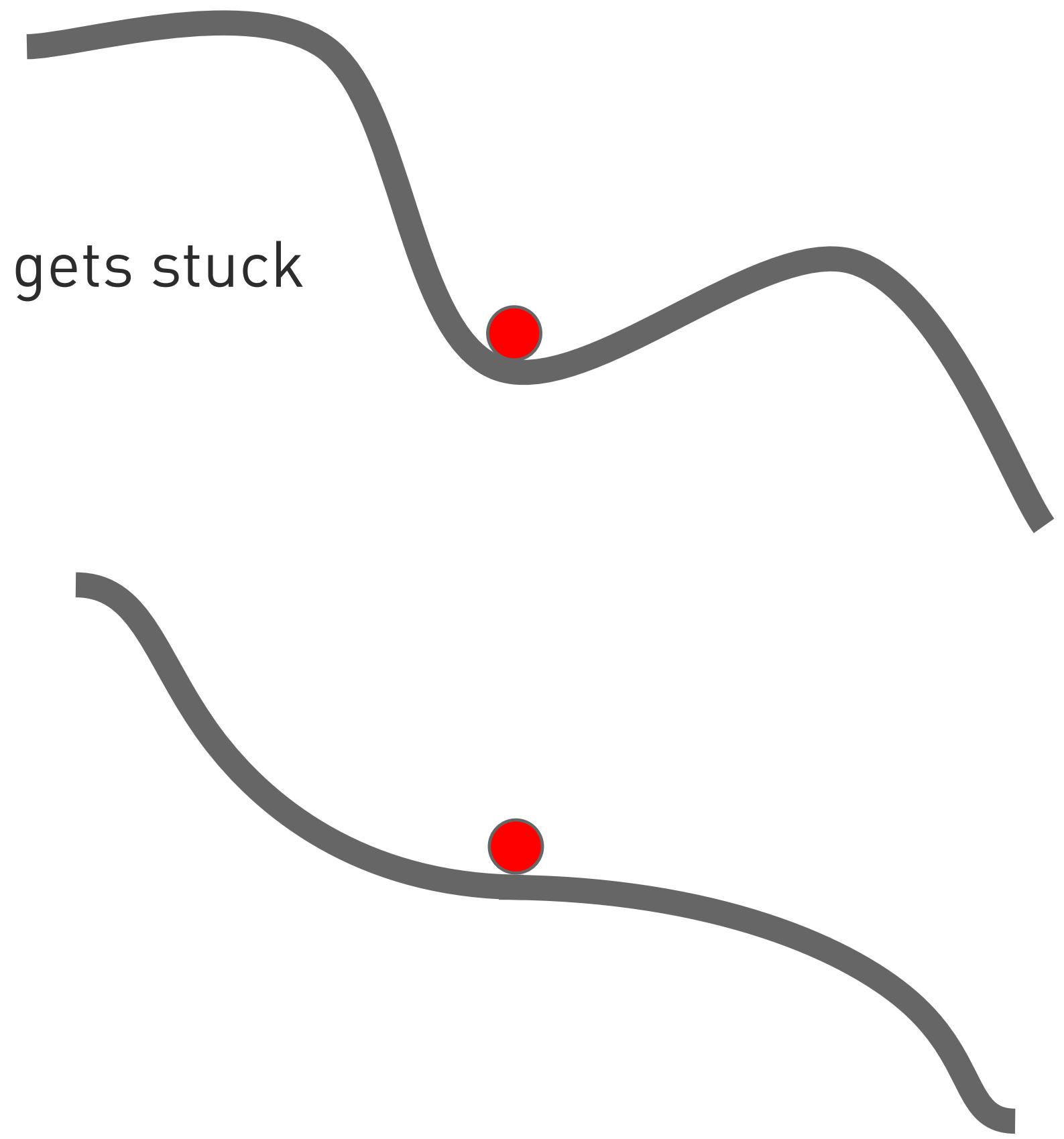
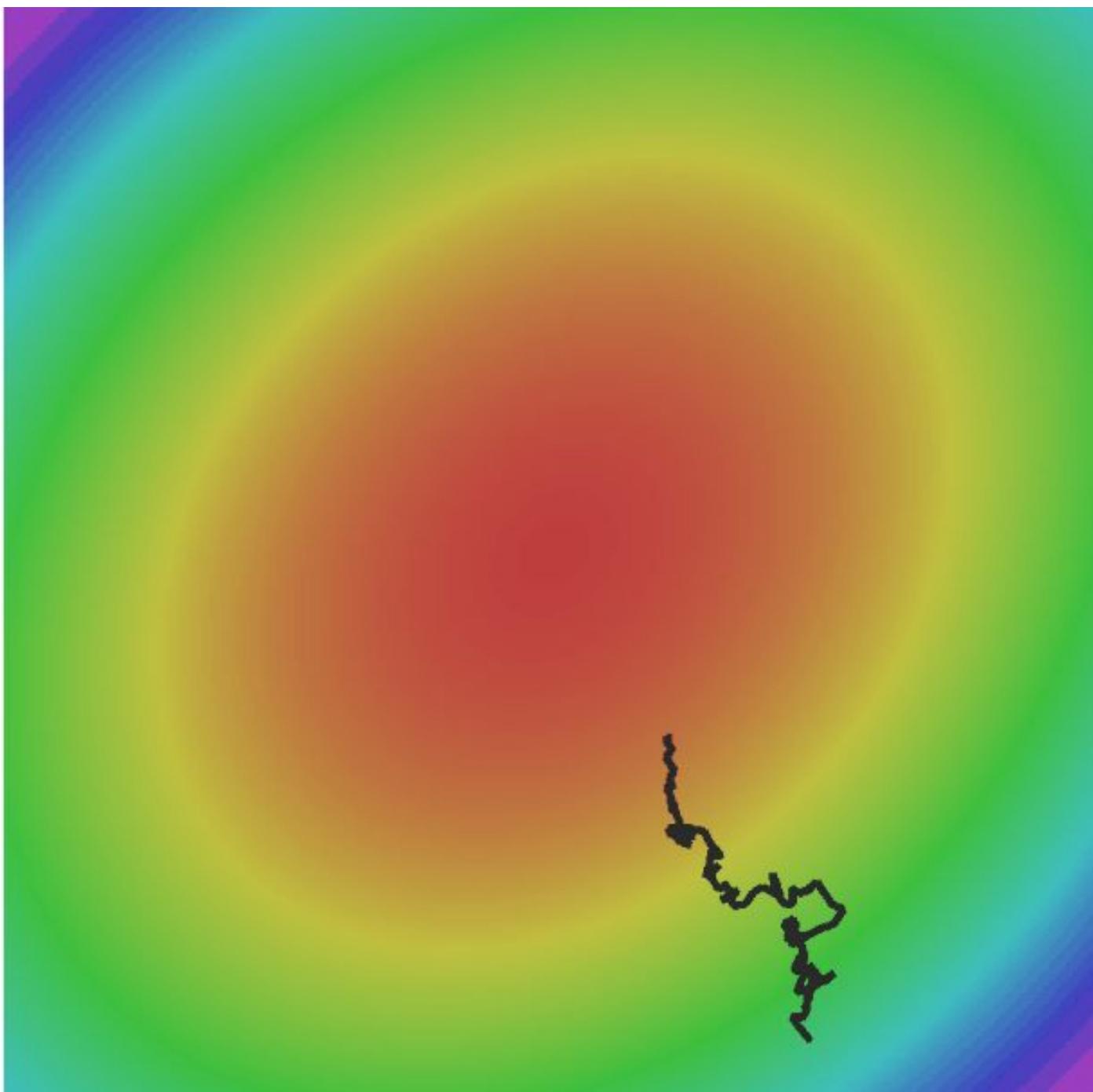
(Negative log-likelihood loss, or cross-entropy loss)

Optimisers

What if the loss function has local minima or saddle point?

- Zero gradient, function has a gradient descent local minima or gets stuck
- Saddle points much more common in high dimension

Our gradients come from mini batches so they can be noisy!

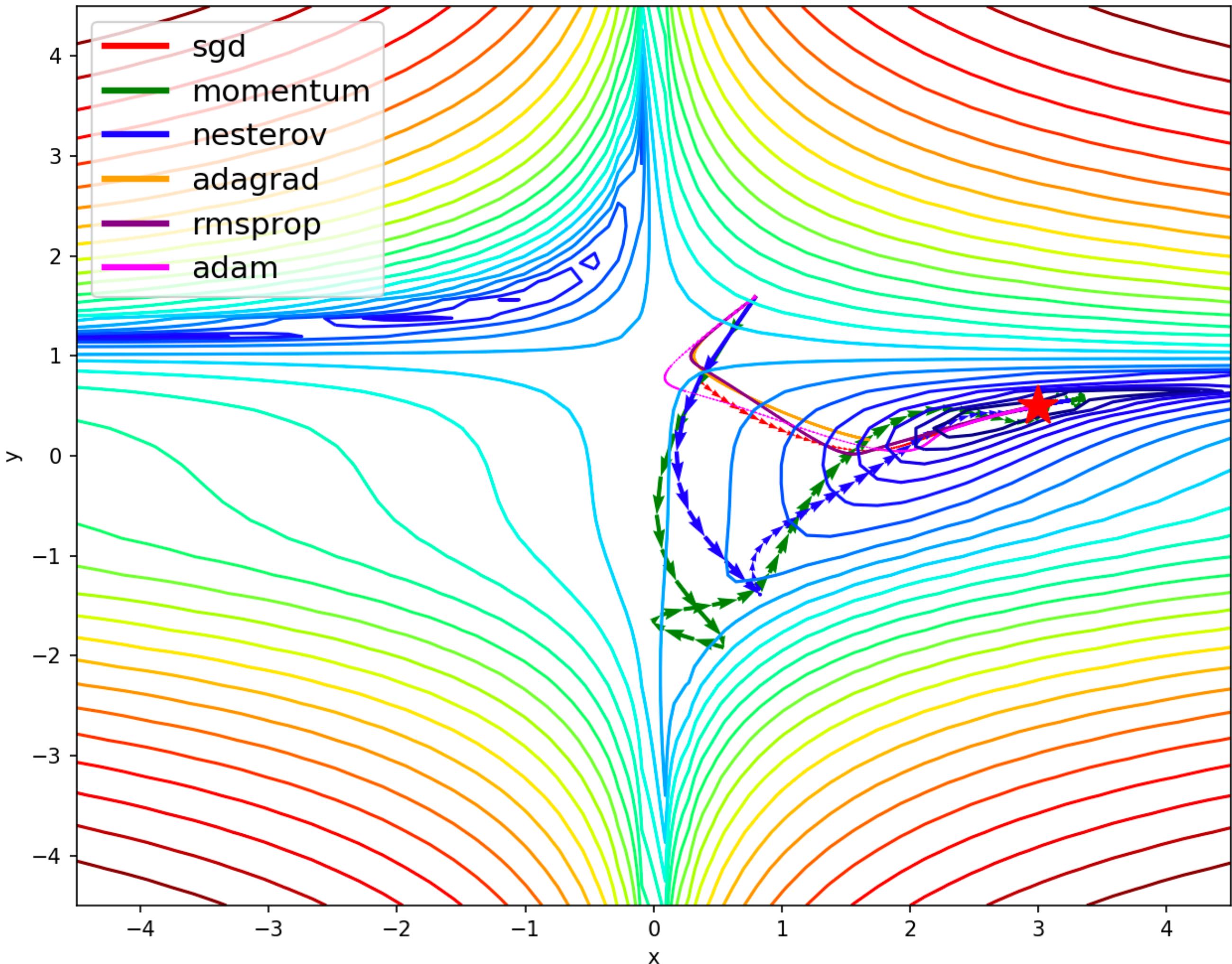


Optimisers

Several alternative optimisation algorithms to stochastic gradient descent exists

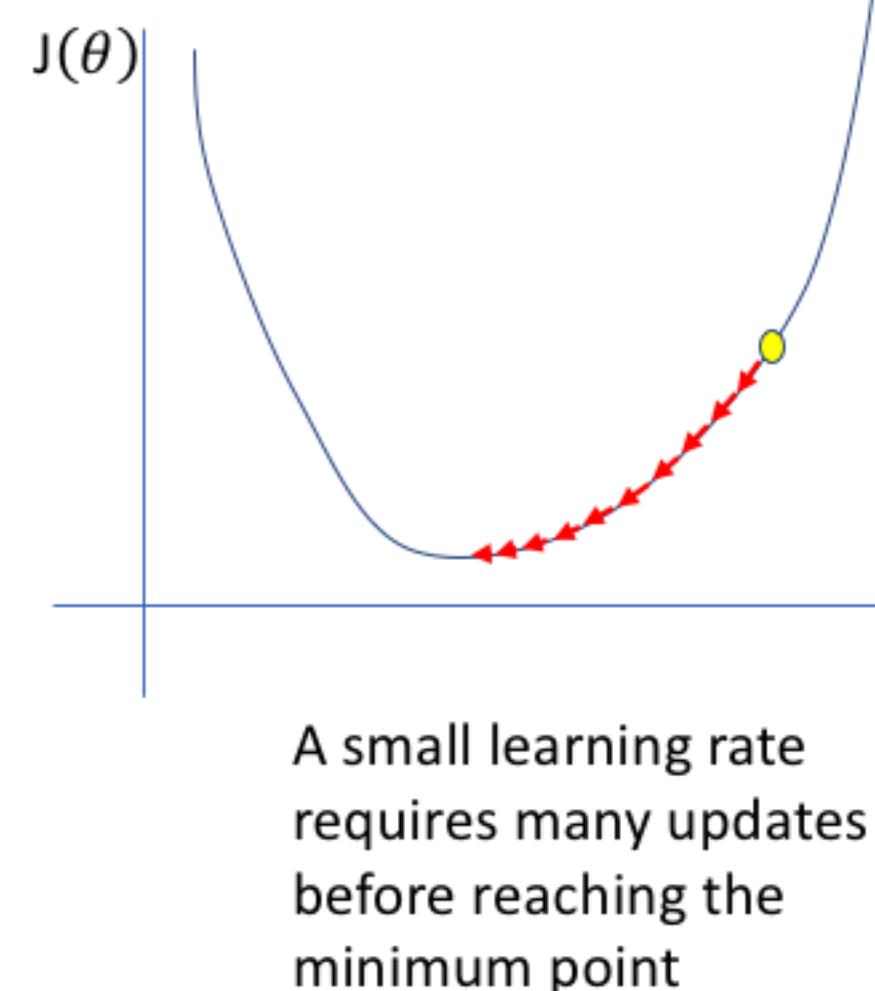
- AdaGrad
- RMSProp
- Adam (per parameter lr)

All based on estimating moments of the gradients and/or iteratively changing learning rate

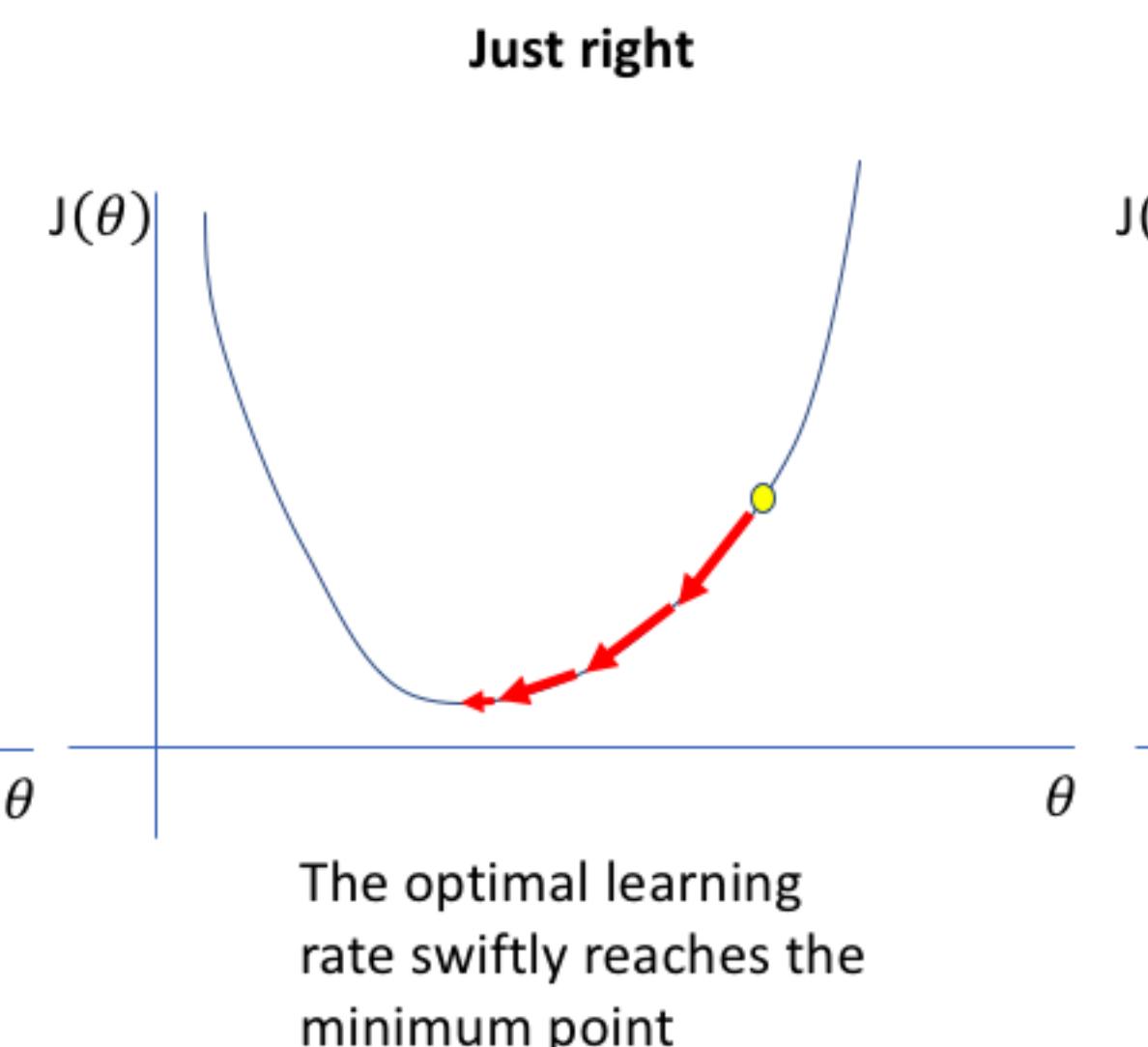


Other (popular) options

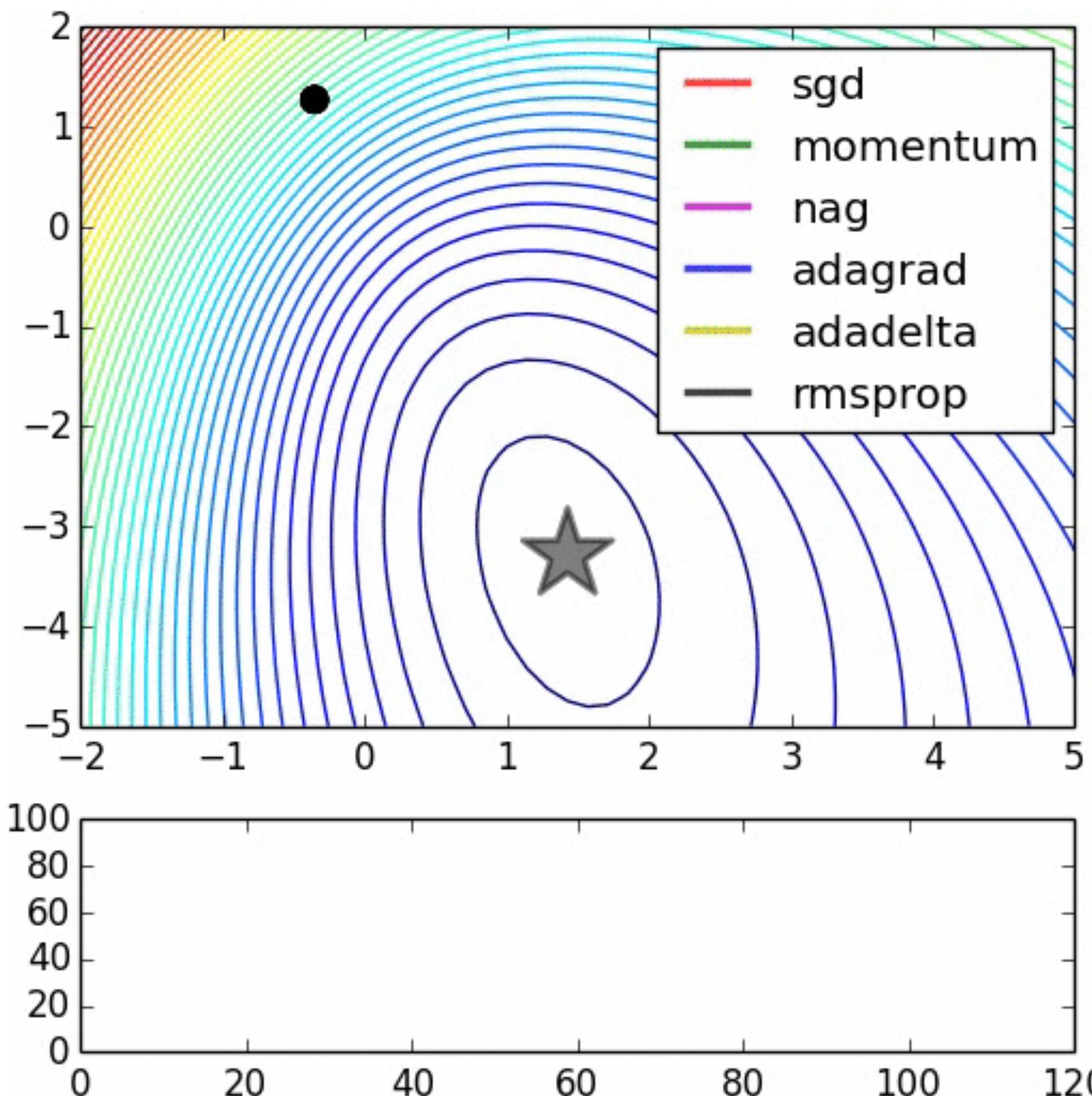
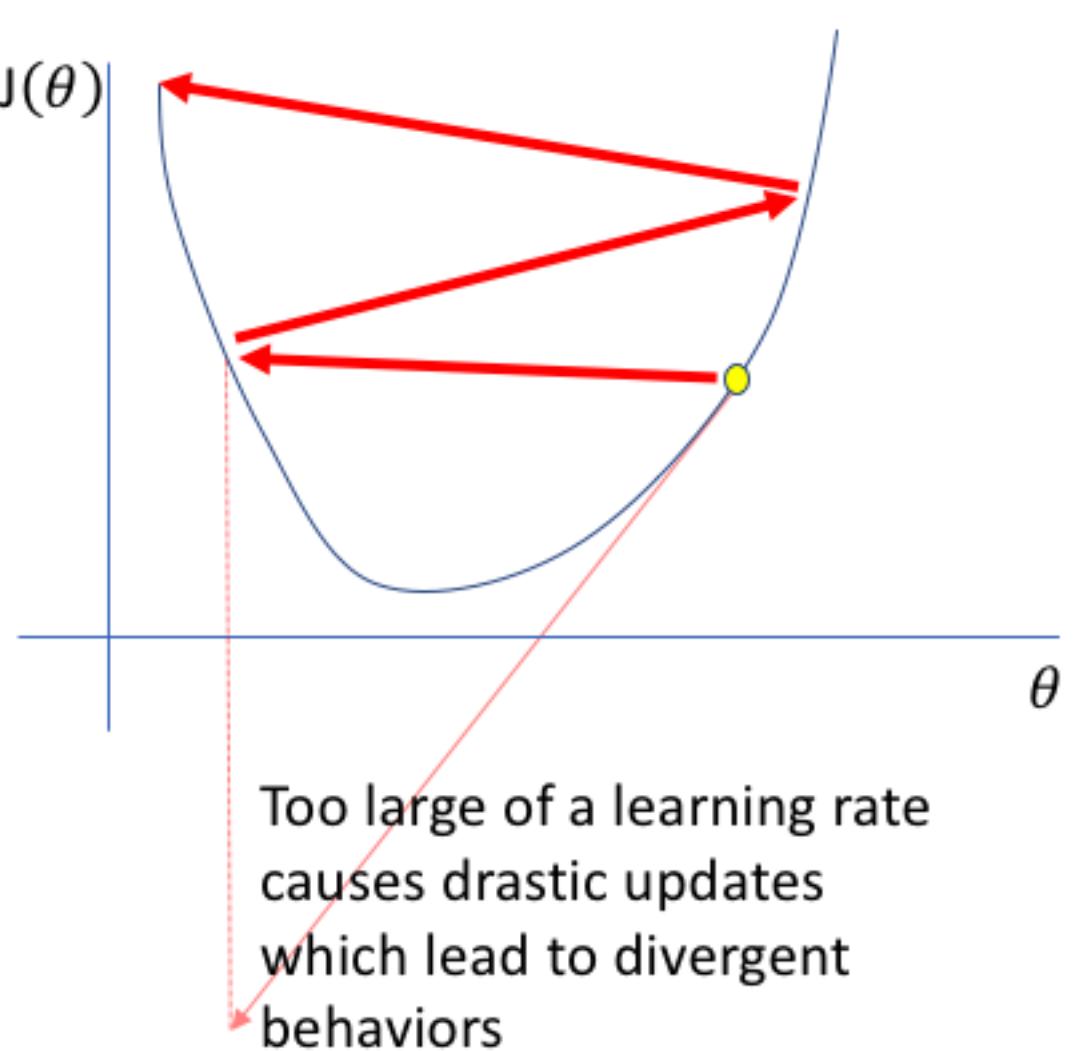
Too low



Just right

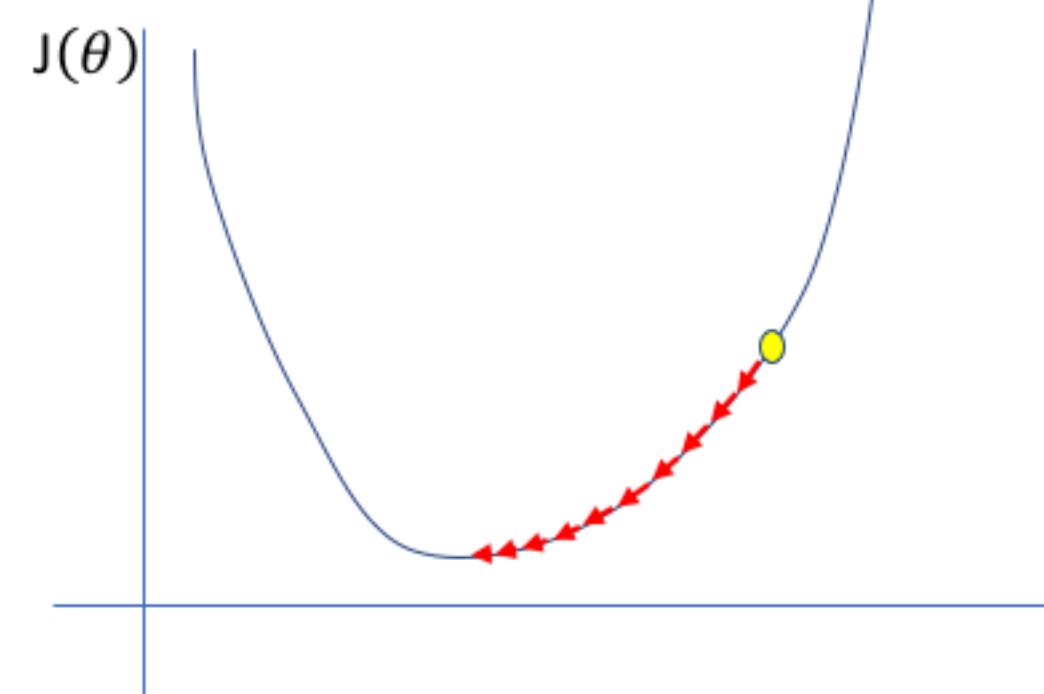


Too high



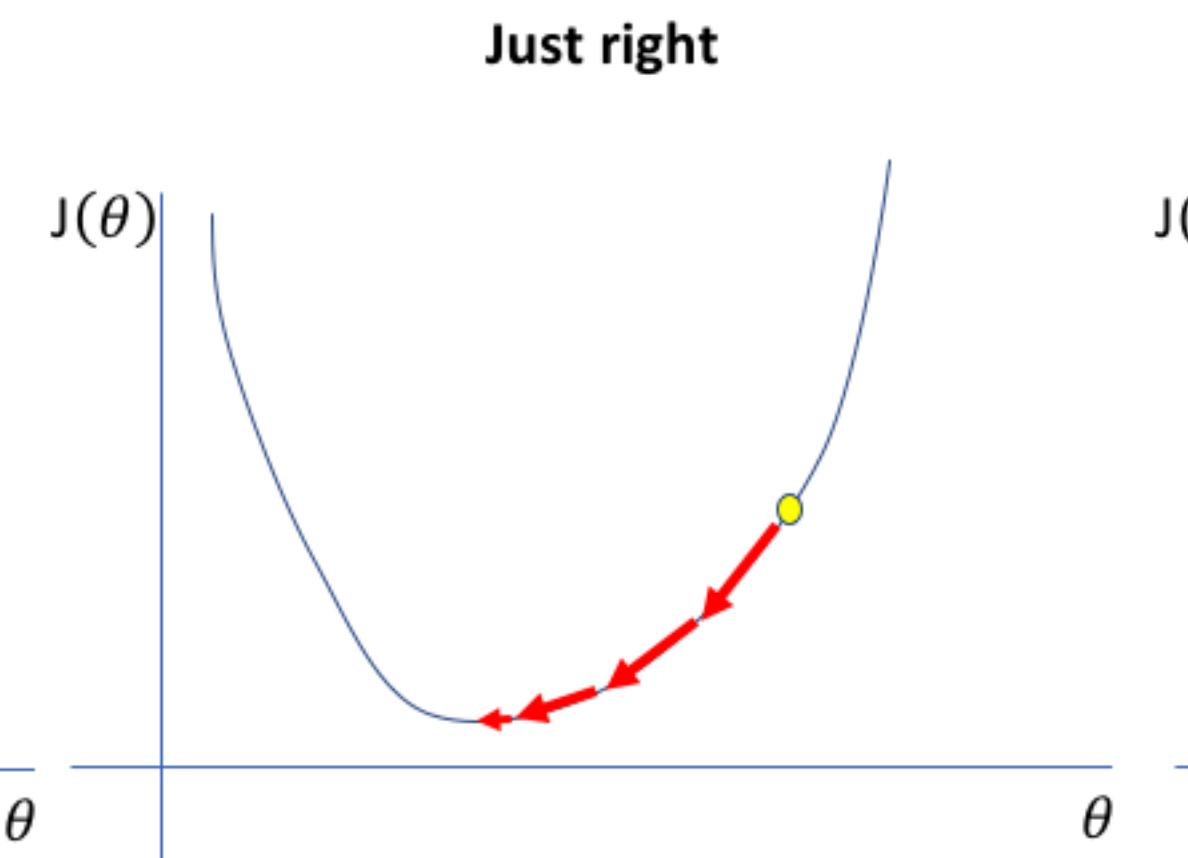
Other (popular) options

Too low



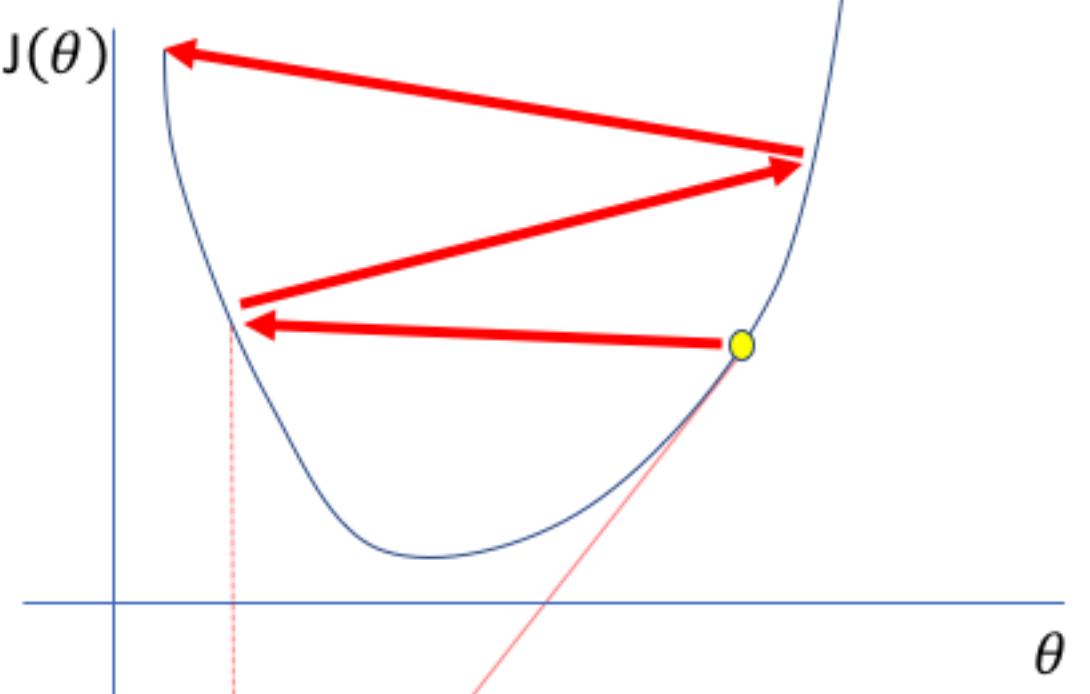
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

