Lecture 5: Training in Practice

Program for Today

Date	Topic	Tutorial
Sep 17	Intro & class description	Linear Algebra in a nutshell + prob and stat
Sep 24	Basic of machine learning + Dense NN	*Basic jupyter + DNN on mnist (give jet dnn as homework)*
Oct 1	Convolutional NN	*Convolutional NNs with MNIST*
Oct 8	Training in practice: regularization, optimization, etc	*Practical methodology*
Oct 15	*Tensor Flow tutorial (tbc)*	
Oct 22	Recurrent NN	*Tutorial on RNNs*
Oct 29	Graph NNs	*Tutorial on Graph NNs*
Nov 5	Unsupervised learning and anomaly detection	*Autoencoders with MNIST*
Nov 12	Generative models: GANs, VAEs, etc	Normalizing flows
Nov 19	Spiking Neural Network	*Tutorial on neuromorphic chips & spiking NNs (tbc)*
Nov 26	Network compression (pruning, quantization, Knoweldge Distillation)	
Dec 3	Tutorial on hls4ml/qkeras	
Dec 10	Transformers	
Dec 17	tbd	

Regularisation

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*

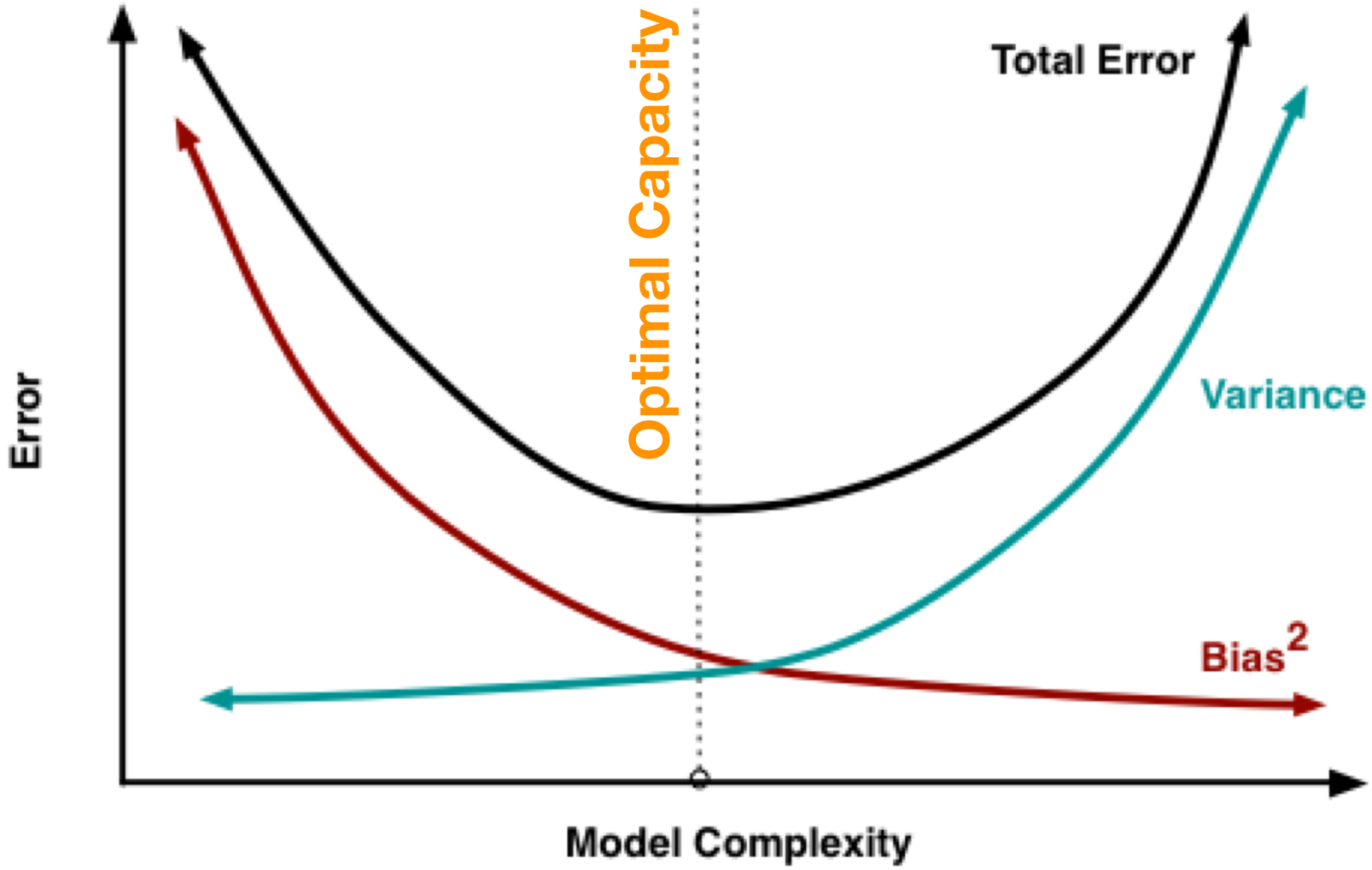
$$E[(y - h(x))^2] = E[(y - \bar{y})^2] + (\bar{y} - \bar{h}(x))^2 + E[(h(x) - \bar{h}(x))^2]$$

Noise

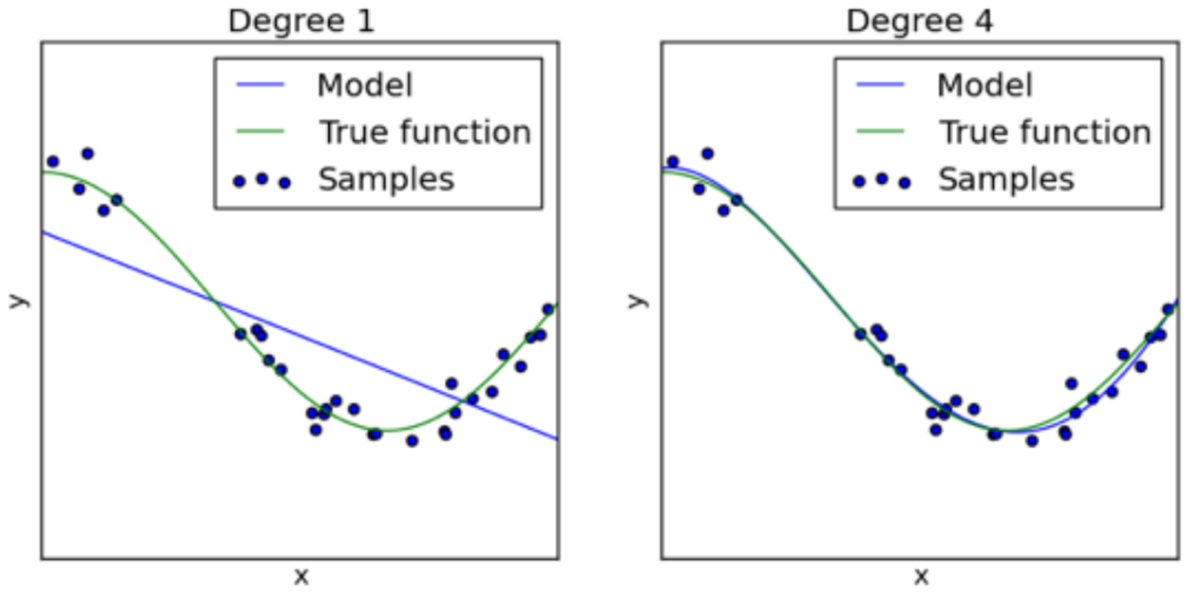
Bias Squared

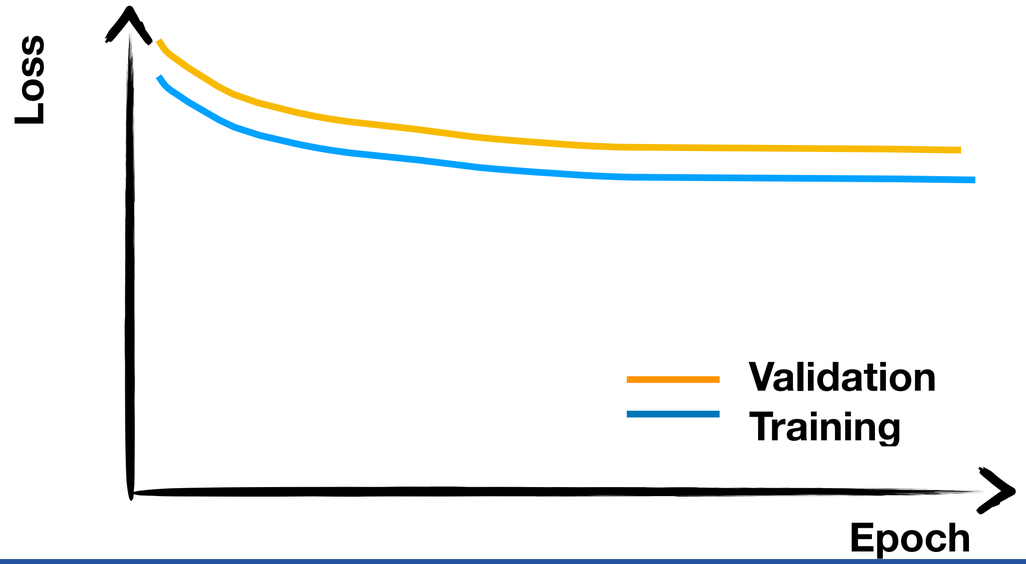
Variance

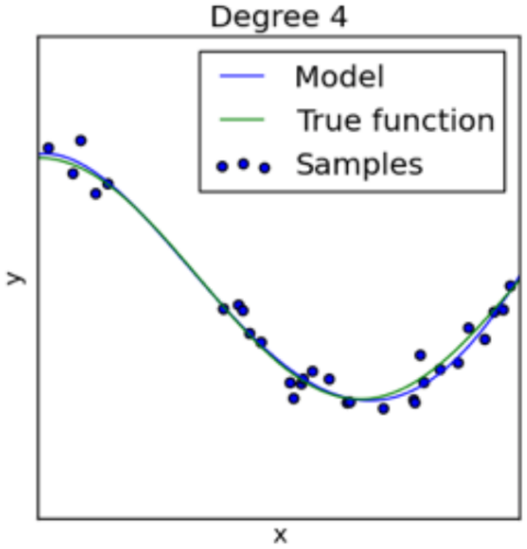
Bias vs Variance

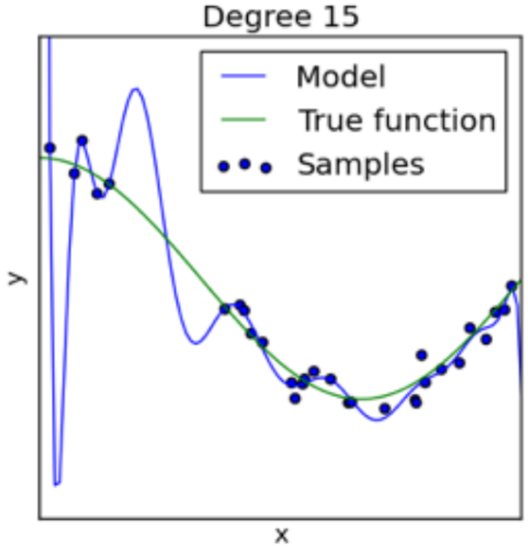


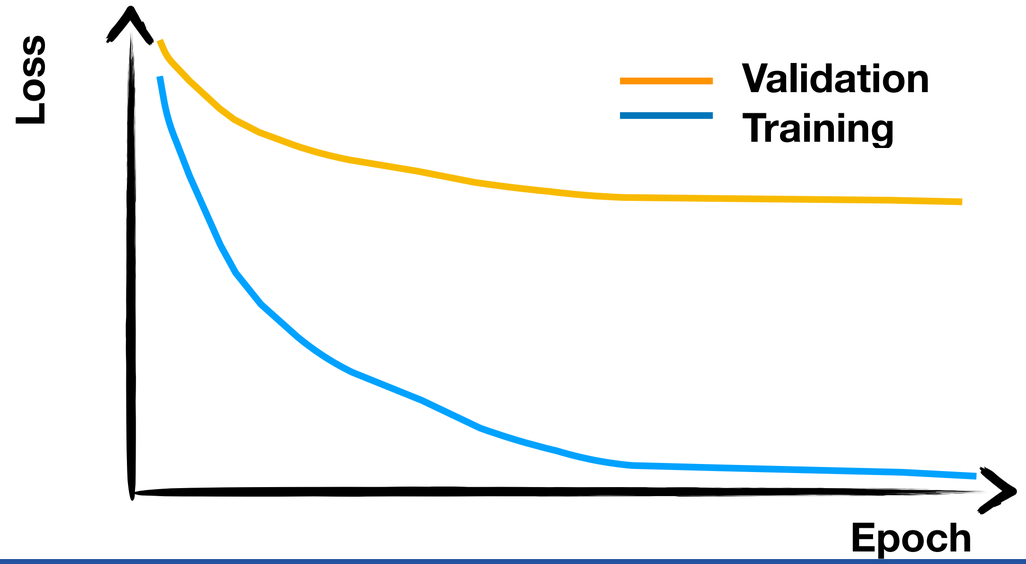
Underfitting vs Overfitting





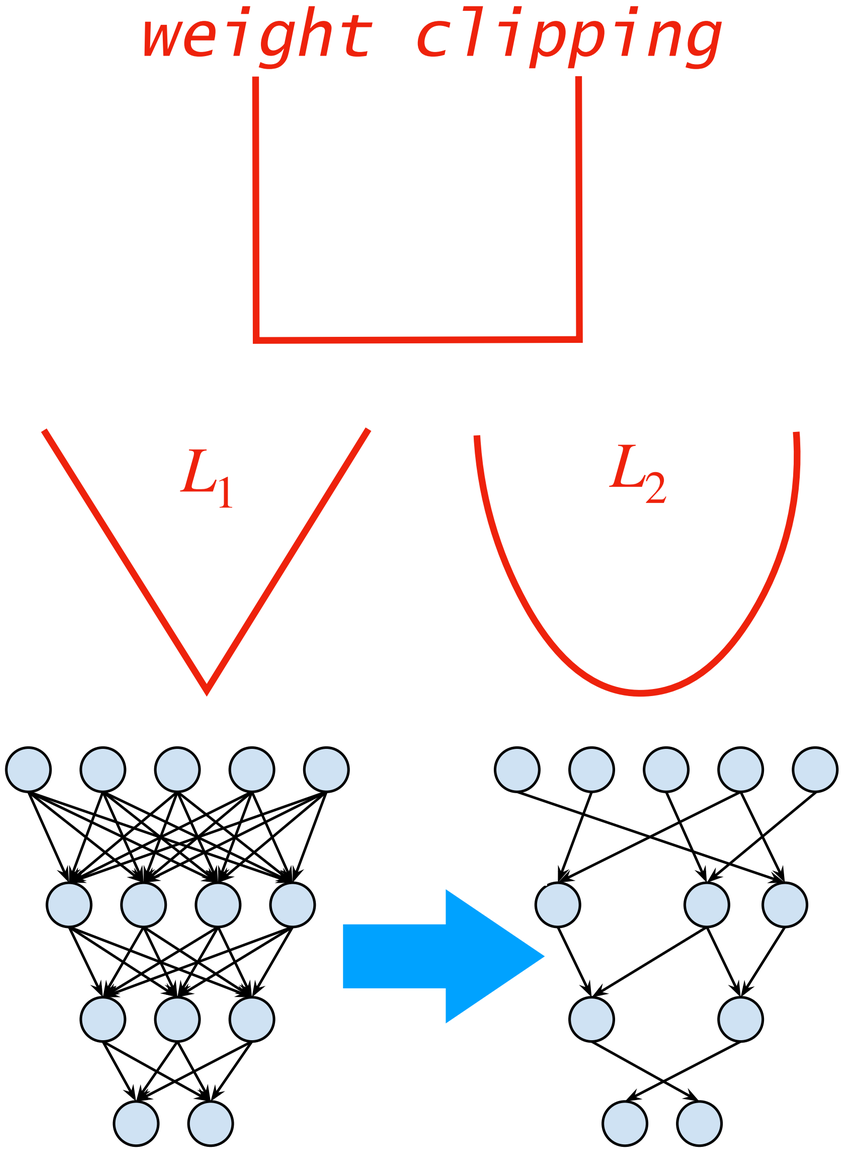






Regularisation

- A *regularisation method* is a modification of the learning algorithm that is intended to reduce its generalization error but not its training error
  - adding constraints to the parameters, e.g., *clipping* the range of values it can take
  - adding an extra term in the loss, such as L_p *regularisation*
  - modifying the network architecture, e.g., *dropout and pruning*
- Most of regularisation methods can be seen as priors on the parameters
- Sometimes they express a preference for a simpler (i.e., smaller) network architecture

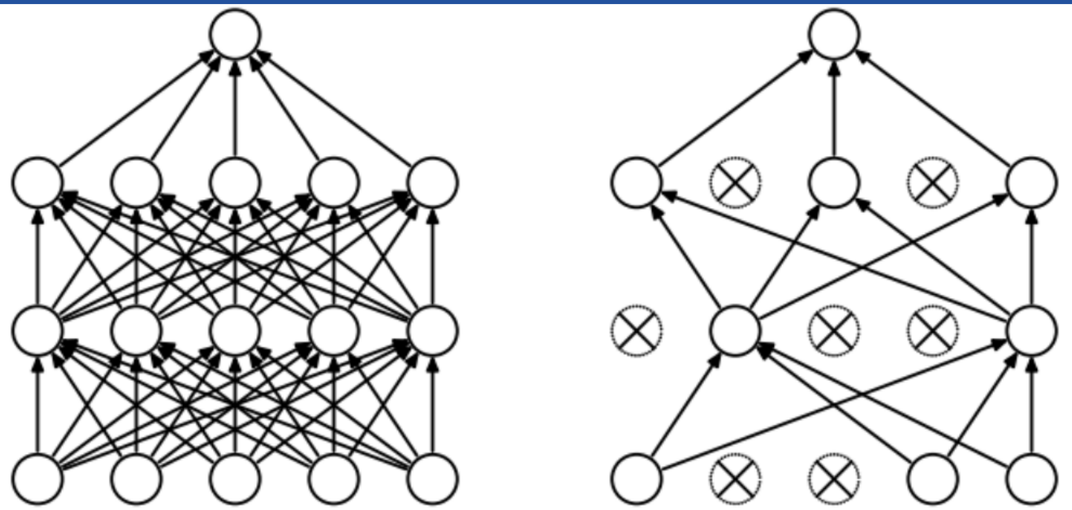


Regularisation in practice

- *If the "right model" is among the models that your function can approximate, regularisation removes unnecessary degrees of freedoms*
  - *e.g., fitting a parabola with a 4th order polynomial*
- *But this is rarely the case*
  - *The "right" model is unaccessible, and most likely not among the functions that your network can approximate*
  - *In this case, one wants the largest/deepest (i.e., most complex) model possible*
  - *Even in these cases, regularisation is required to avoid overfitting*

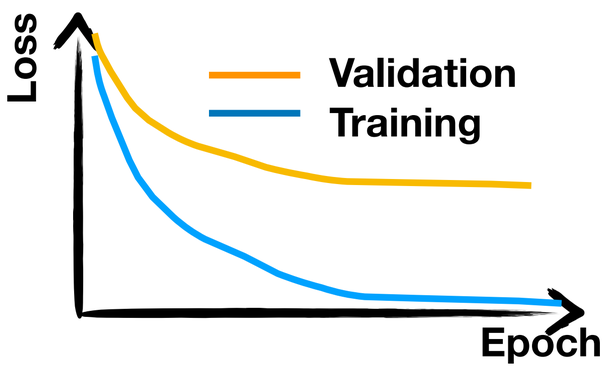
Dropout

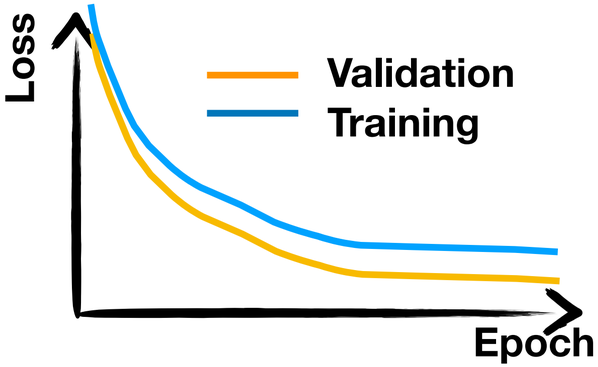
- *Dropout is one of the most popular and easy-to-apply regularisation*
  - *In Keras, it is just an extra layer you add*
- *It consists in randomly zero-ing some of the parameters during training*
  - *This is typically done by batch: when a new batch starts, all weights are back (to the values they had at the next-to-last batch) and new weights are randomly zero-ed*
  - *The final network is obtained from the optimization of an ensemble of smaller networks that share parameters*
  - *This prevents overtraining by brute force*



(a) Standard Neural Net (b) After applying dropout.

Since the network is smaller at training time, training loss can become worse than validation loss



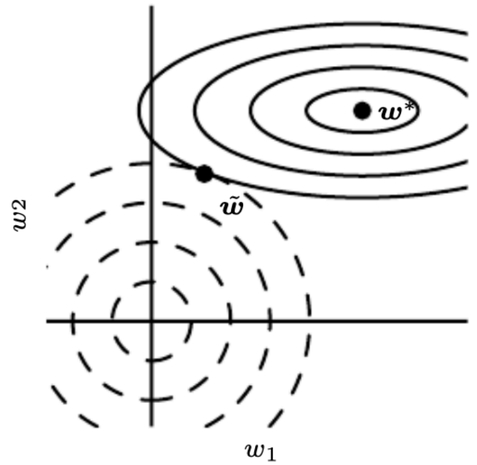


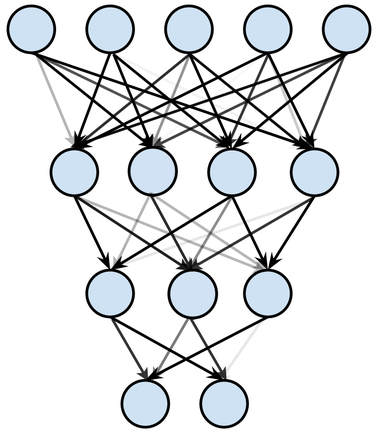
Parameter Norm penalty

- This regularisation method consists in adding a term to the loss

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha \Omega(\boldsymbol{\theta})$$

  - $\Omega(\theta)$ is some norm of the network parameters θ. For instance $L_2 = \sum_i |\theta_i|^2$
  - α in a hyperparameter that sets the relative importance of the regularisation
    - With deep networks, one often uses a different α for each layer

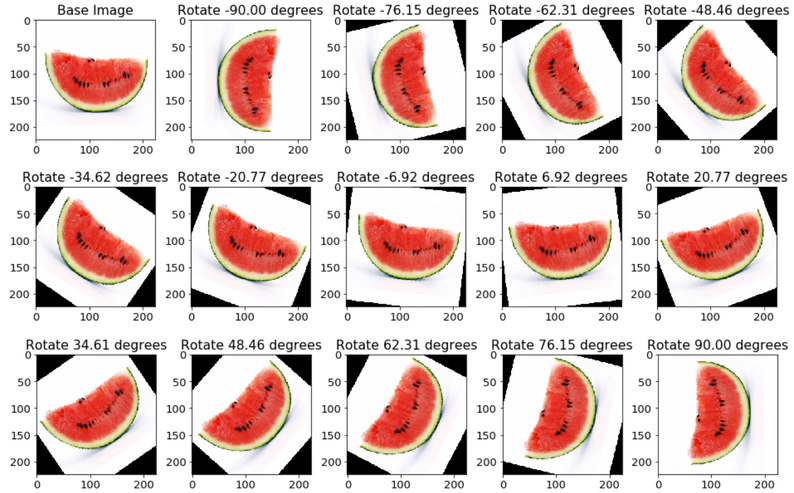


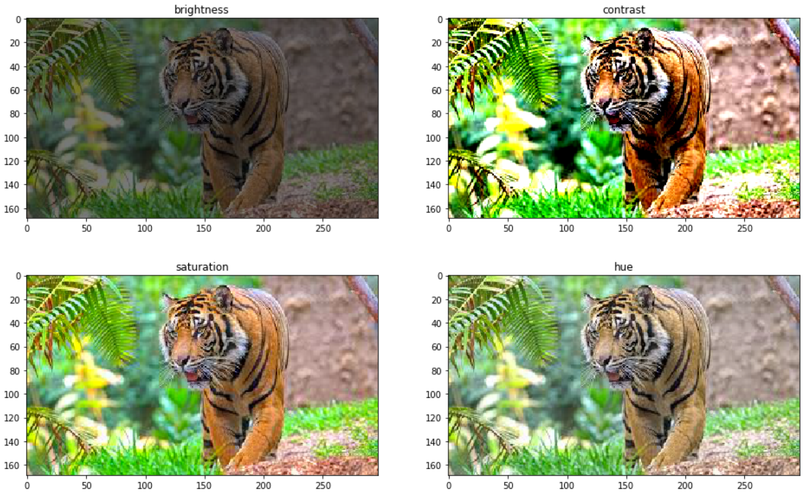


- $\Omega(\theta)$ is a prior that privileges $\theta = 0$, i.e., less degrees of freedom (easy to see with DNNs)
  - Similar to dropout in spirit, but replacing discrete yes/no switches with tunable gauges

Dataset Augmentation

- *Regularisation can also be applied on the dataset, rather than on the network or the loss function*
- *Data augmentation consists in training on more data than those specific of the task*
  - *This approach is often used in classification problems*
- *Example: with images, one can augment the data applying data transformations (translations, rotations, etc.) to the original images.*
  - *It helps even when the architecture is equivariant to that transformation*
  - *But it could generate confusion: e.g, a 180º rotate 6 looks like a 9, which is a problem for a MNIST classifier*
- *Sometimes the tranformation can applied using more complex filters applying filters*

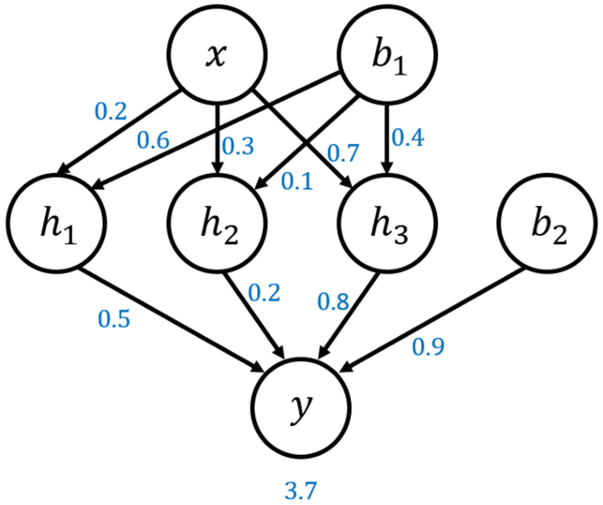




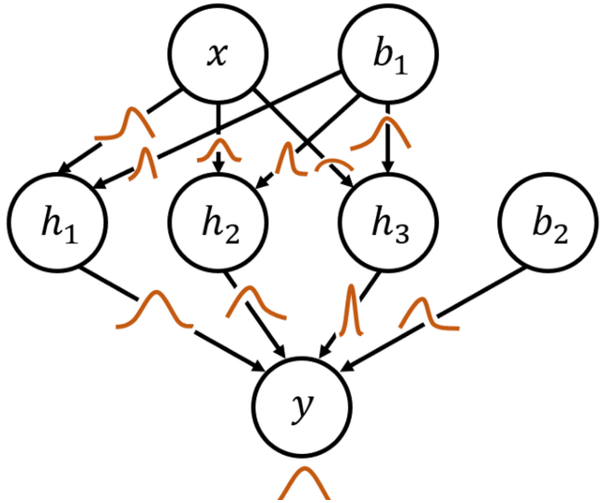
Noise on weights

- *One way to regularise a network and gain generalization power is by perturbing the weight*
  - *Replace a fixed value θ with $\hat{\theta} = \theta + \epsilon$, ϵ being some random number*
    - *Could be distributed flat in a range*
    - *Could be distributed as a Gaussian, e.g., $\mathcal{N}(0,1)$*
  - *This procedure replaces a neural network with an ensemble of networks, obtained perturbing the weight set **θ** around it's central values*
  - *These networks are called Bayesian Neural Networks*

Standard Neural Network



Bayesian Neural Network

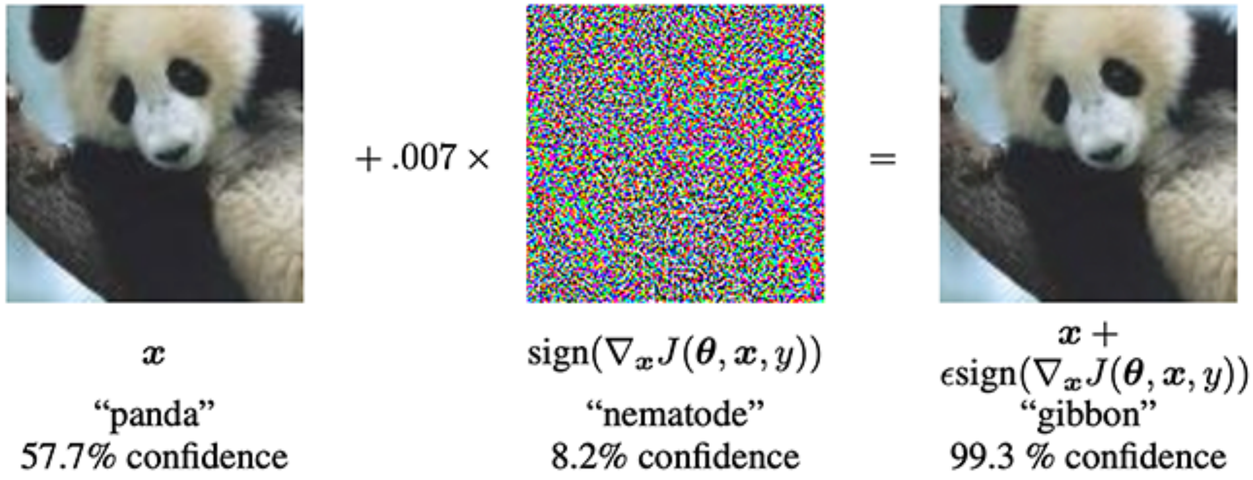


Noise on labels

- *Often, data labelling (e.g., classification done by humans) come with mistake*
  - *A network would take these mistakes as ground truth*
- *To prevent mistakes from misguiding the learning process, one can try to model the mistake probability and add it to the learning procedure*
  - *Example: trade the 0,1 "hard labels" of a binary classifier with "smooth labels"* $\epsilon, 1-\epsilon$*, with* $\epsilon \propto \mathcal{N}(0,\sigma_\epsilon)$ and $\sigma_\epsilon < 1$

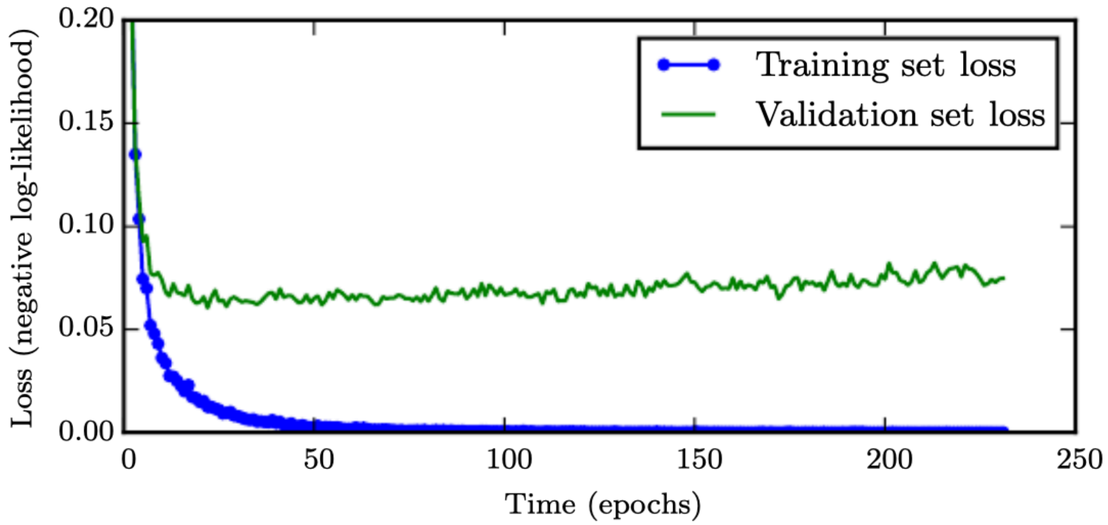
Noise on the data

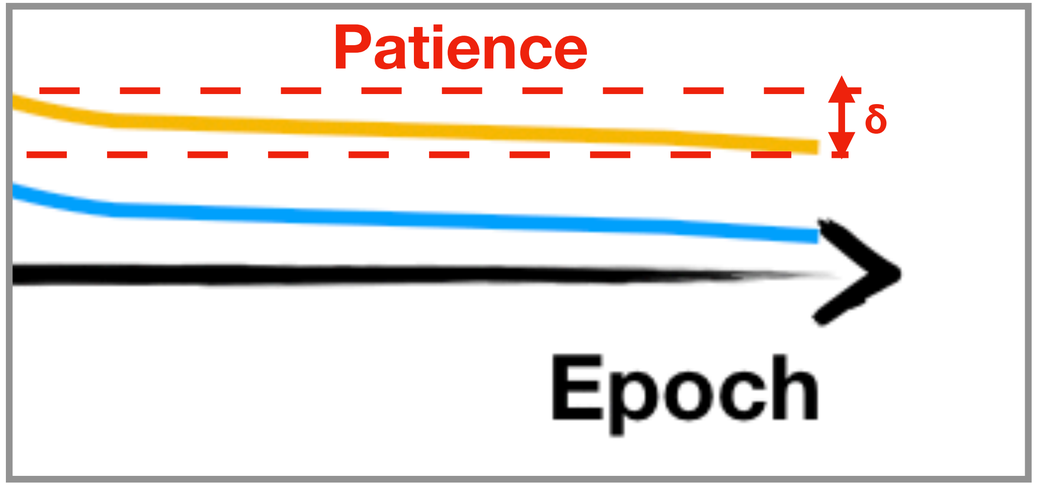
- One way to regularise a network and gain generalization power is by perturbing the data with noise
  - One can try to emulate various weather conditions (e.g., light exposure in automotive)
  - Or emulate various sound quality for speech recognition
- This is usually done applying random noise to the inputs (random numbers distributed according to some prior, e.g., a Gaussian $\mathcal{N}(0,1)$
- Sometimes one applies structured noise, e.g., noise designed to maximally confuse a classifier
  - This kind of noise is derived with so-called adversarial attacks (we will discuss them in ~ 1 month)



Early Stopping

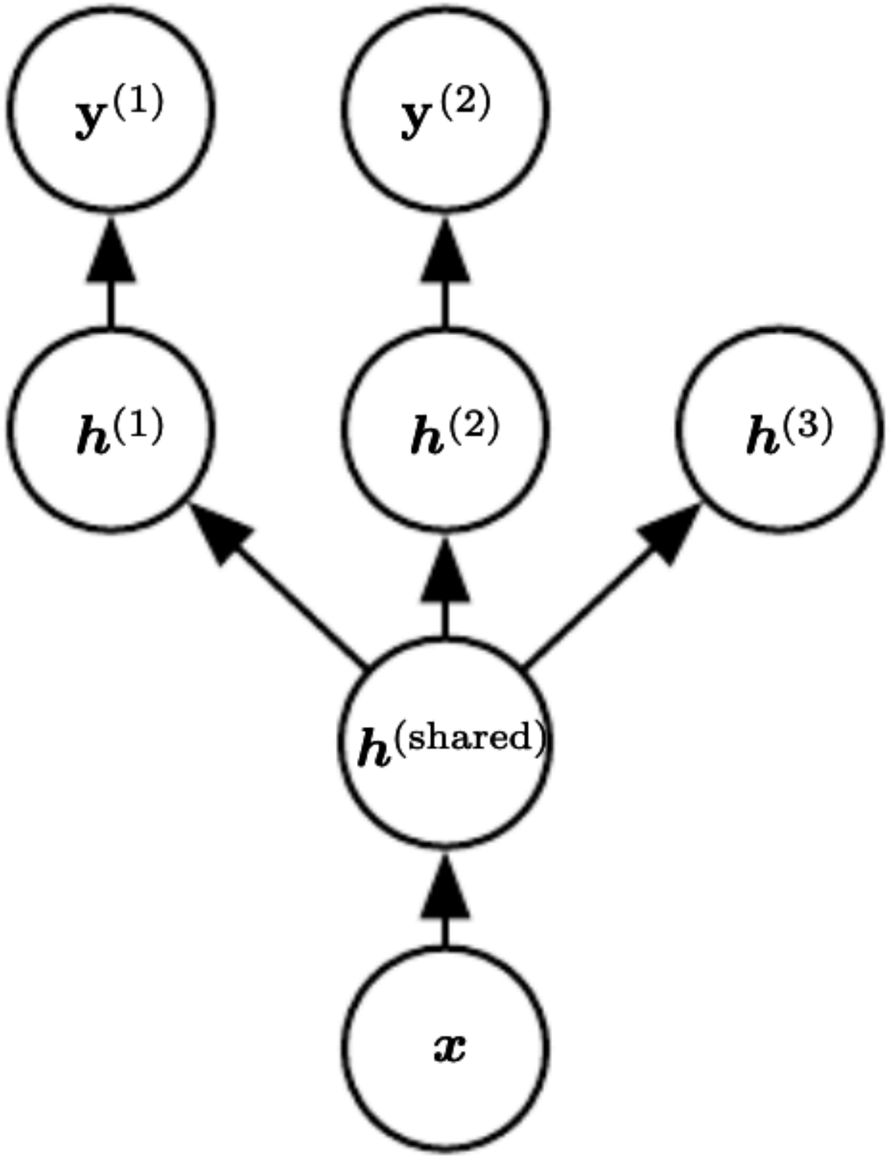
- *Sometimes the val loss grows after a little bit, while the training loss might still decrease*
  - *Having stopped the training earlier, we would have had a better generalization loss*
- *This can be done adding early stopping to the training: stop the training if after p epochs (the patience) the validation loss did not increase by more than a fixed amount δ*
  - *The number of epochs is yet another hyperparameter and early stopping is an algorithm that scans and fixes this hyperparameter*
  - *Since the outcome is a better generalization power (smaller distance between training and validation), early stopping is also a regularisation algorithm*



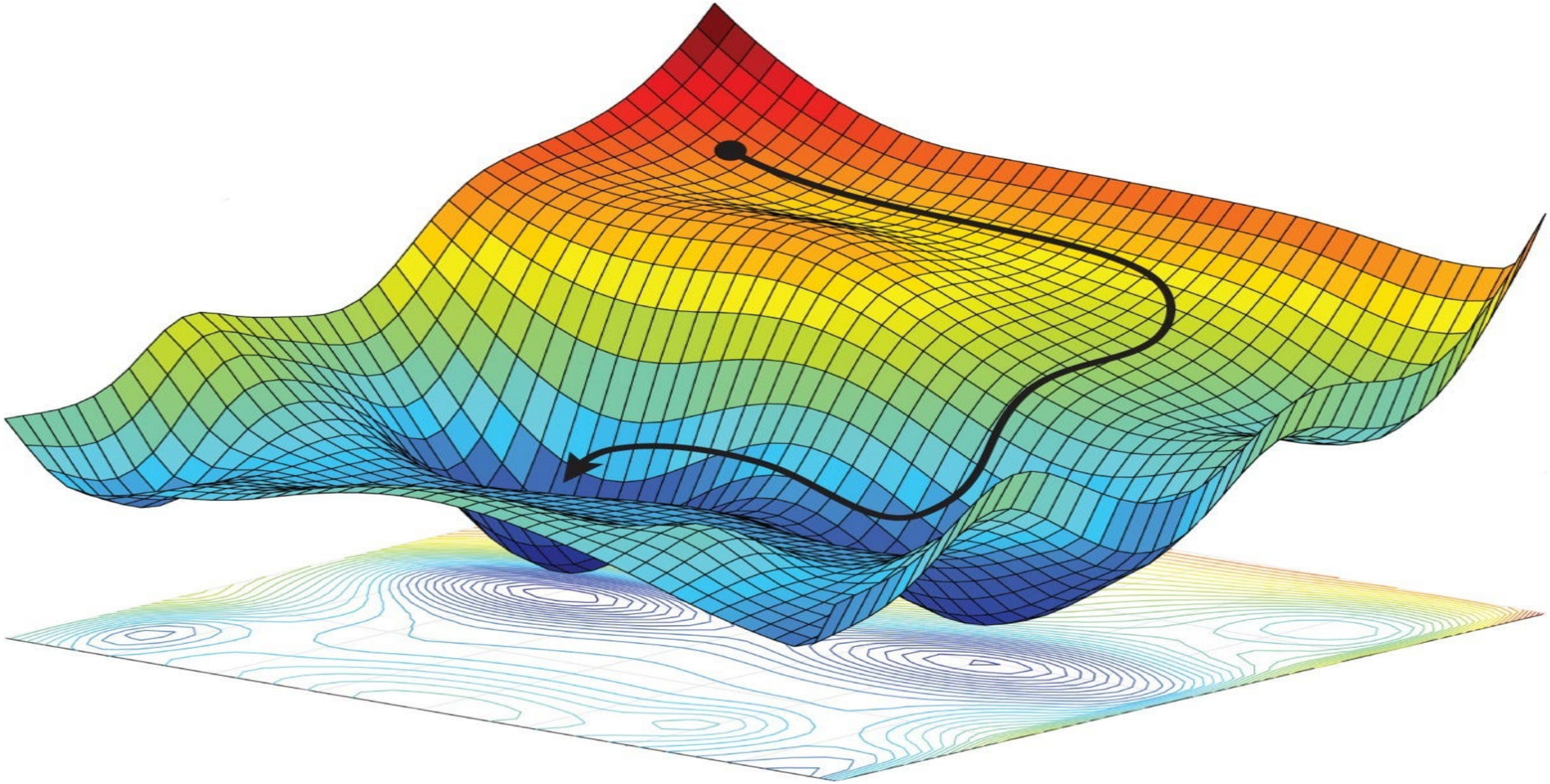


Multitask Learning

- *Sometimes, making the task come complex prevents overtraining*
- *Example: you want to classify particles*
  - *Your discrimination power changes with the energy of the particle*
  - *By learning also the particle energy, you learn this dependence and make the classification more robust*
- *This is typically realised with two DNNs sharing the same features (e.g., extracted from the same convolutional layers)*
- *It only makes sense when the two tasks are related: among the factors that explain the variations observed in the data associated with the different tasks, some are shared across two or more tasks*



Optimization



The Empirical risk

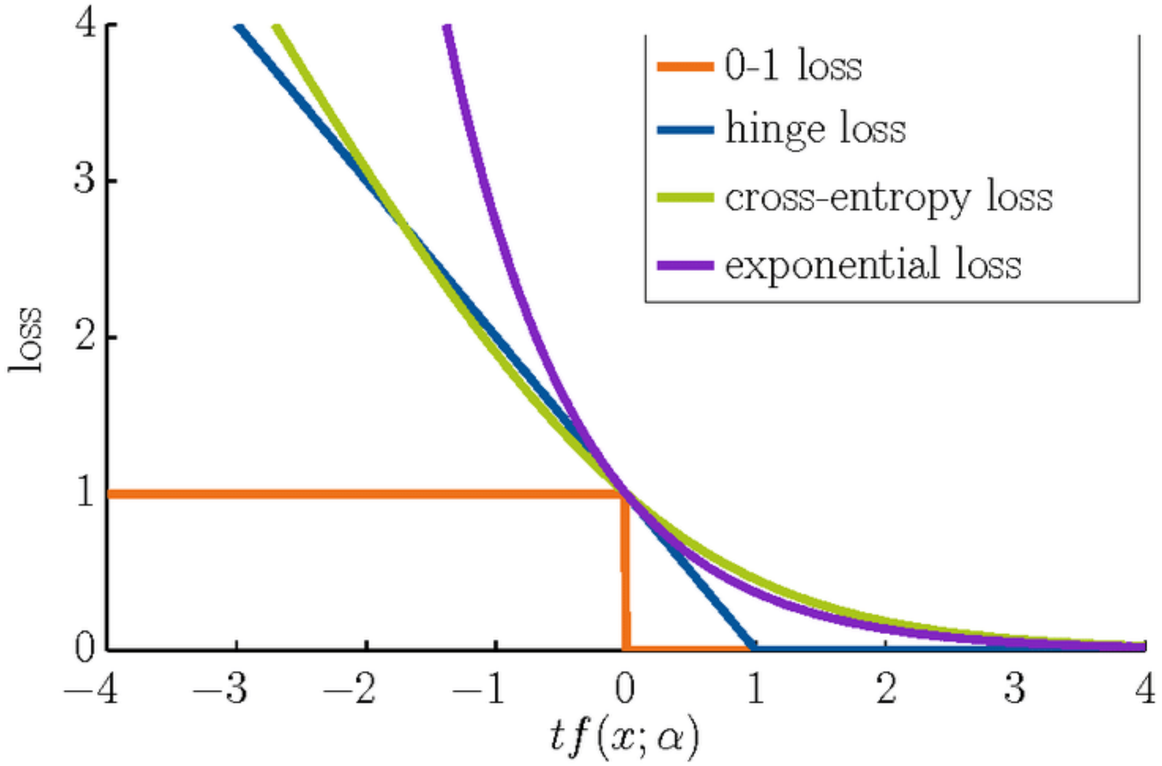
- *The goal of the optimization step is the minimisation of the expected generalization error. This quantity is known as the risk*
  - *You want to minimise the probability to make mistakes on an independent dataset (e.g., your test sample)*
- *We can only do exactly if we know the true data distribution (which we never do)*
  - *Instead, we estimate it numerically, measuring the risk on a training dataset. This is called the empirical risk*

$$\mathbb{E}_{\boldsymbol{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}(\boldsymbol{x},y)}[L(f(\boldsymbol{x};\boldsymbol{\theta}),y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),y^{(i)}),$$

Remember: Empirical risk = cost = loss = …

Surrogate loss functions

- *Sometimes problems can be intractable*
  - *e.g., if the loss is not a smooth convex function it can't be easily minimized.*
- *Example: % of correct 0-1 classification (accuracy, aka 0-1 loss) is discontinuous*
  - *Instead, the negative log likelihood is typically used, resulting in the binary cross entropy*



Example: Cross Entropy

- *Bernoulli's problem: probability of a process that can give 1 or 0*

$$\mathscr{L} = \prod_i p_i^{x_i}(1-p_i)^{1-x_i}$$

- *The corresponding likelihood is (as usual) the product of the probabilities across the events*

$$-\log \mathscr{L} = -\log[\prod_i p_i^{x_i}(1-p_i)^{1-x_i}]$$

- *Maximizing the likelihood corresponds to minimizing the -logL*

- *Minimizing the -logL corresponds to minimizing the binary cross entropy*

$$= -\sum_i \left[x_i \log p_i + (1-x_i)\log(1-p_i)\right]$$

- *We will use this expression as loss function in classification problems*

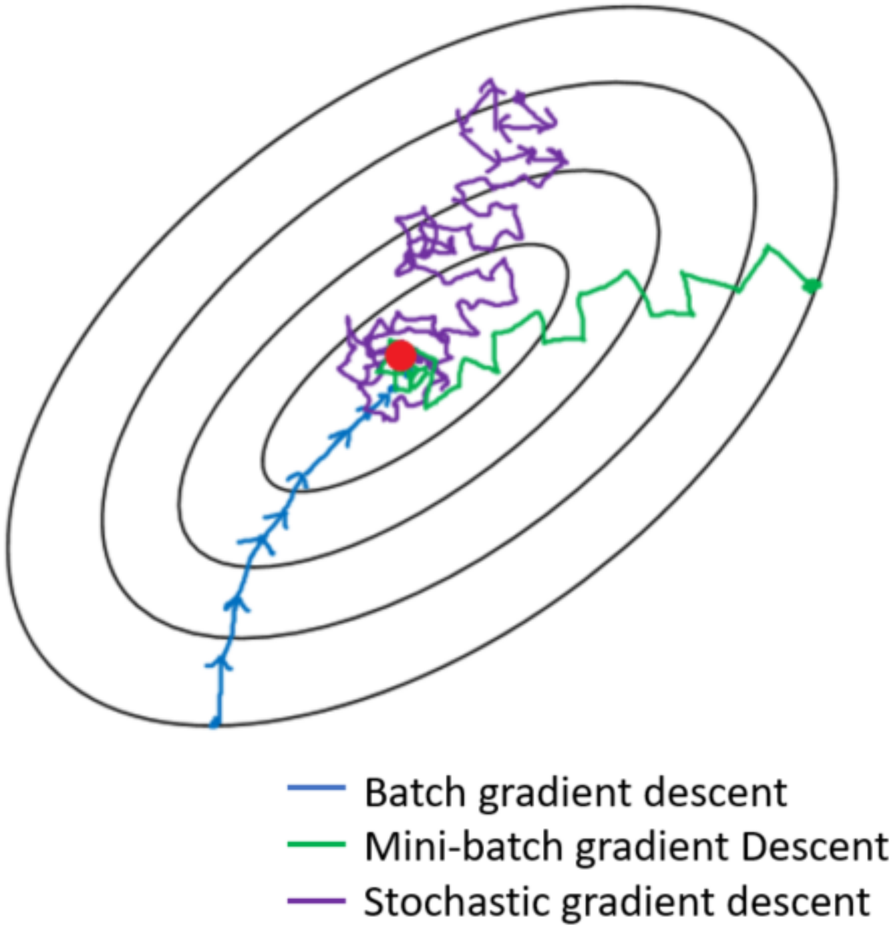
REMINDER FROM LECTURE 2

Batches and Minibatches

- *Loss is normally decomposed in a sum over the training set*
  - *Remember: the advantage of -logLik:*

$$\arg\max\left[\log\prod_i p_i\right] = \arg\min\left[-\sum\log p_i\right] = \arg\min\mathbb{E}\left[\log p\right]$$

  - *Computationally very expensive to perform the average on the whole training set (bath minimisation)*
  - *But we know that the RMS of an average is $\sigma/\sqrt{n}$, when n is the number of examples and σ is the RMS of the sample*
    - *Estimates on x10 more data are x10 more expensive but only x3.16 more precise*
  - *one can then perform single-example estimates and update the model (stochastic models), with large parallelizability at large stochasticity cost*
  - *One can instead perform the average on small portions of data (minibatch) in parallel, gaining in performance while reducing stochastic noise*



SGD Algorithm with mini batch

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$

Require: Initial parameter $\boldsymbol{\theta}$

$k \leftarrow 1$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$

$k \leftarrow k + 1$

end while

Learning rate

A few considerations

- *Larger batches provide a more accurate estimate of the gradient, but with less than linear returns*
- *Multicore architectures are usually underutilized by extremely small batches.This motivates using some absolute minimum batch size, below which there is no reduction in the time to process a minibatch*
- *If all examples in the batch are to be processed in parallel (as is typically the case), then the amount of memory scales with the batch size. For many hardware setups this is the limiting factor in batch size*
- *Some kinds of hardware achieve better runtime with specific sizes of arrays. Especially when using GPUs, it is common for power of 2 batch sizes to offer better runtime. Typical power of 2 batch sizes range from 32 to 256, with 16 sometimes being attempted for large models*
- *Small batches can offer a regularizing effect, due to the noise they add to the learning process. Generalization error is often best for a batch size of 1. Training with such a small batch size might require a small learning rate to maintain stability because of the high variance in the estimate of the gradient. The total runtime can be very high as a result of the need to make more steps, both because of the reduced learning rate and because it takes more steps to observe the entire training set*
- *NNs learning process is not an understood process. It is then difficult to predict which choice will result in a better convergence*

Parameter initialisation

- *The result of the learning process may depend on where you start from*
  - *Examples: all parameters are set to same values and two units are interchangeable (same connections, same activation, etc). They might introduce some degeneracy in the training. Better choose DIFFERENT INITIALISATIONS to break the symmetry*
    - *usually one takes random values from Gaussian or flat distribution*
      - *large values break degeneracy more*
      - *but too large values can cause numerical instabilities*
    - *The scale or the random generation is often used as a hyperparameter*

Updating the learning rate

- *Usually, the learning rate is adapted to the training history (smaller and smaller to increase accuracy once getting close to minimum)*
- *Various prescriptions:*
  - *AdaGrad: use as learning rate $\frac{\epsilon}{\delta+\sqrt{r}}$, where ϵ is the global learning rate (as in SGD), δ is a numerical constant for stability (to avoid /0) and r is the cumulative gradient*

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g} \qquad \Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta+\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g} \qquad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$$

  - *RMSPROP: as RMSPROP, but with hyperparameter ruling relative importance between cumulative gradient and last term of sum*

$$\boldsymbol{r} \leftarrow \rho\boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$$

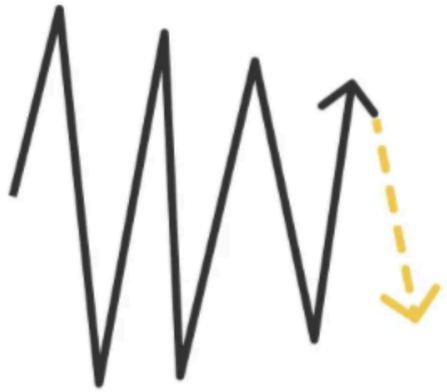
  - *Adam: Cumulative gradient as in RMSPROP, but also uses momentum for gradient update (to avoid getting stuck)*

$$\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g} \qquad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$

- *How to choose the best algorithm? Hyperparameter choice (see here)*

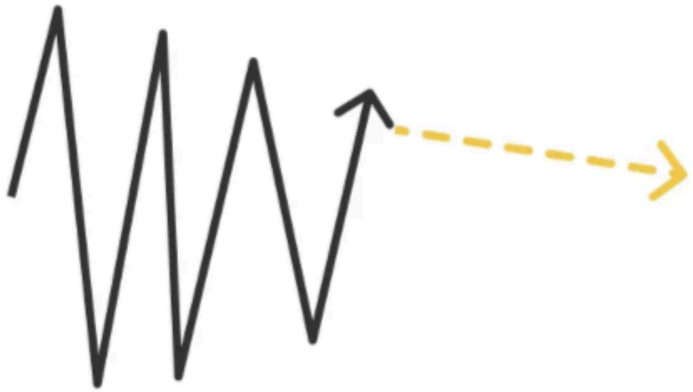
Updating the learning rate

Gradient descent



the gradient computed on the current iteration does not prevent gradient descent from oscillating in the vertical direction

Momentum



the average vector of the horizontal component is aligned towards the minimum

the average vector of the vertical component is close to 0

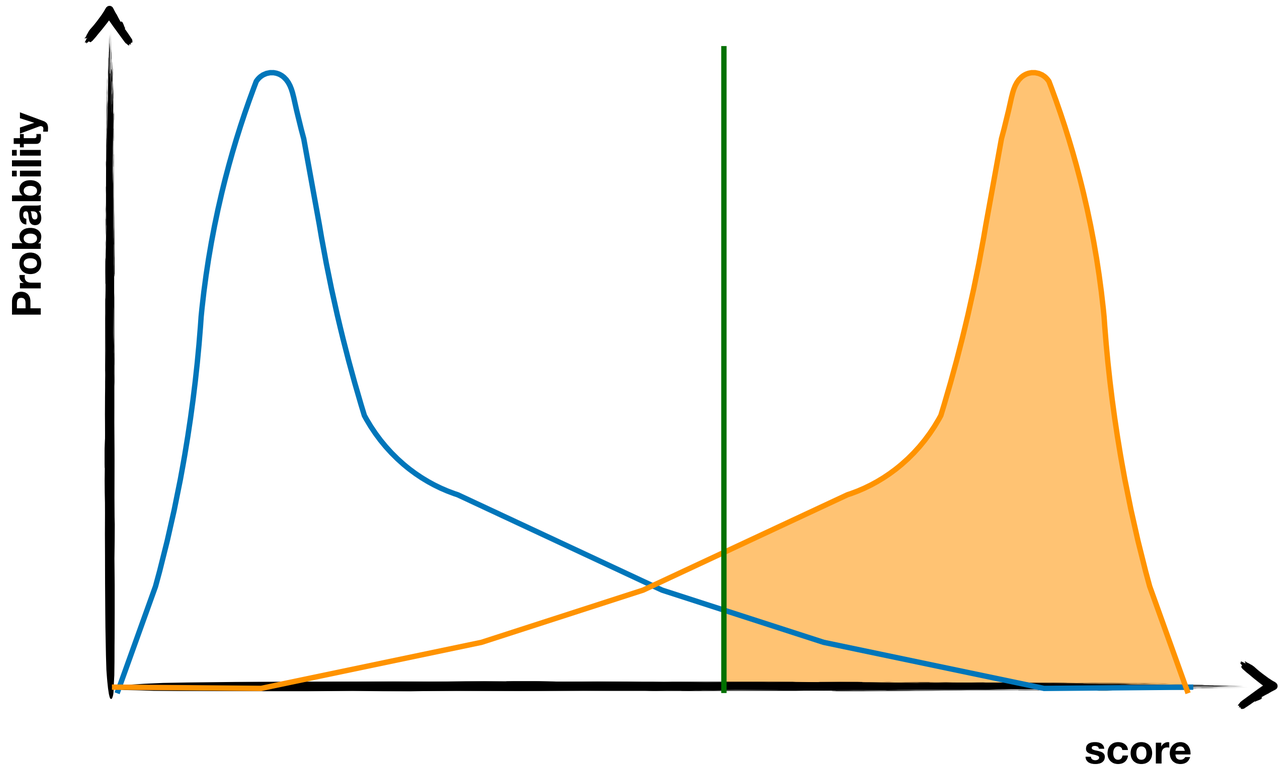
Practical Methodology

What to do in practice

- *This is not an exact science:*
  - *You can know all ingredients but need expertise to put them together*
  - *Often, you need to try things*
- *Get a working pipeline as soon as possible and try various options*
- *Record performance looking at various figures of metric*
  - *Don’t get obsessed with one (accuracy, AOC, etc)*
- *Keep in mind numerical aspects of your minimization*
  - *Avoid very large/big numbers*
  - *stay away from zero gradients*
  - *If you can’t learn, check for NaN*

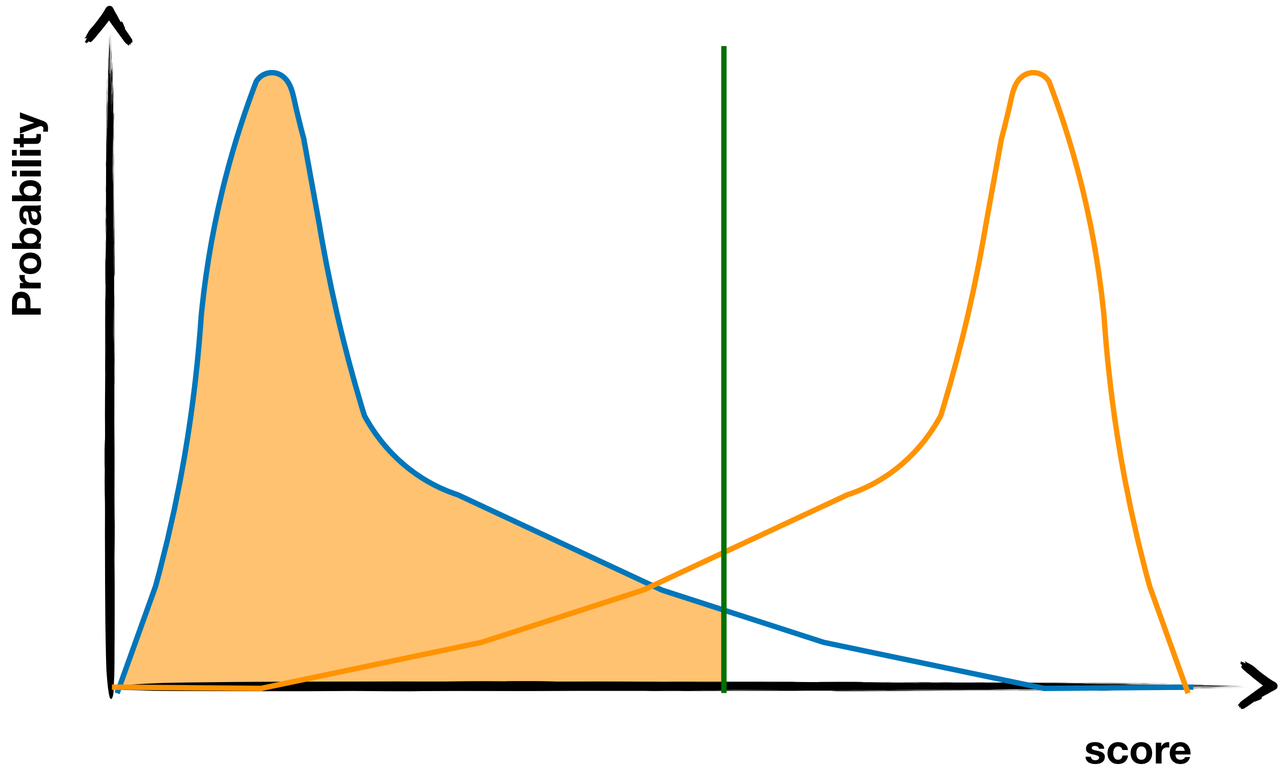
Classifier metrics

- A given threefold 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



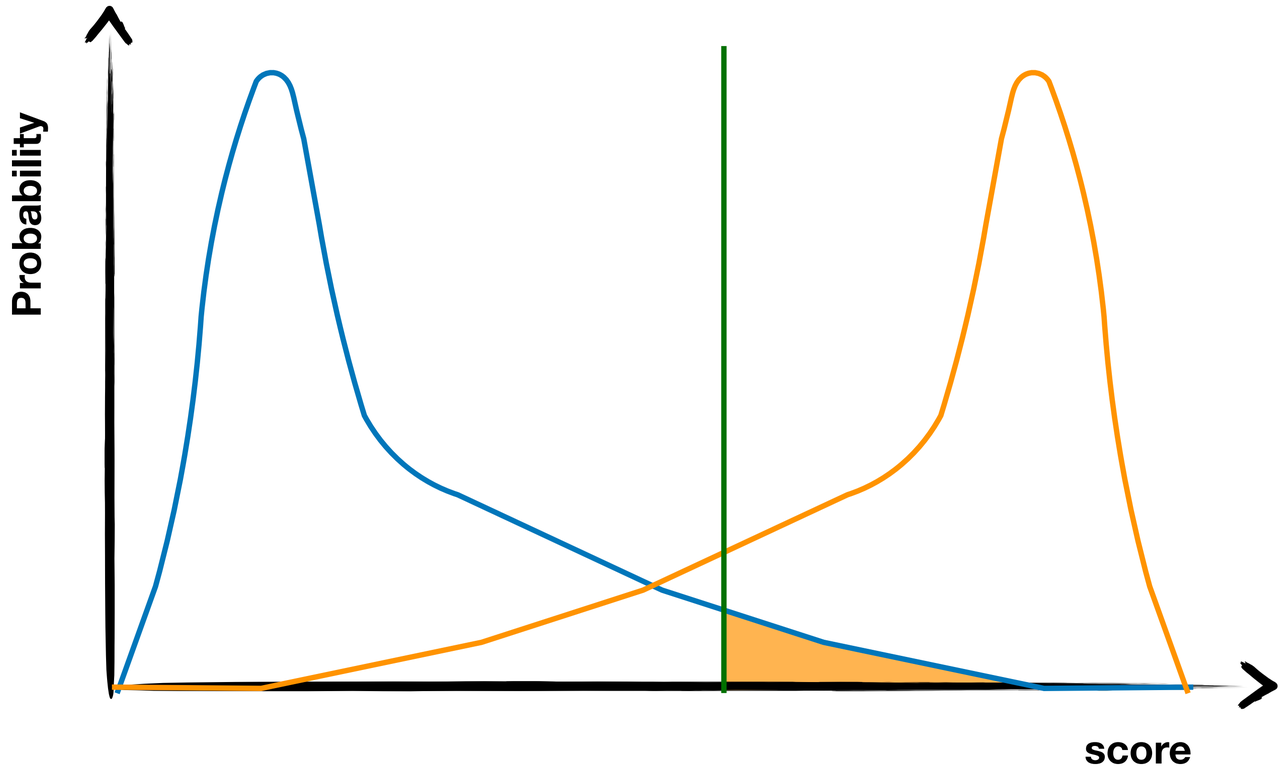
Classifier metrics

- *A given threefold 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*



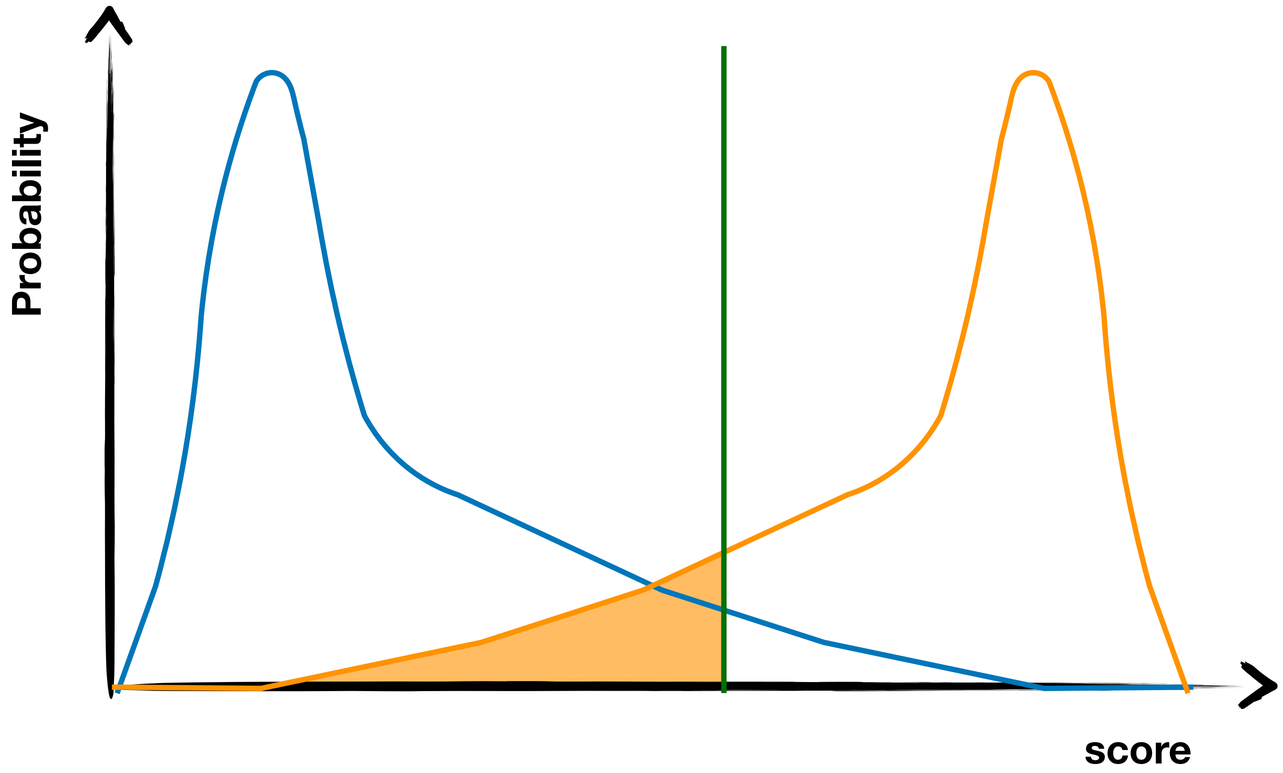
Classifier metrics

- *A given threefold 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*



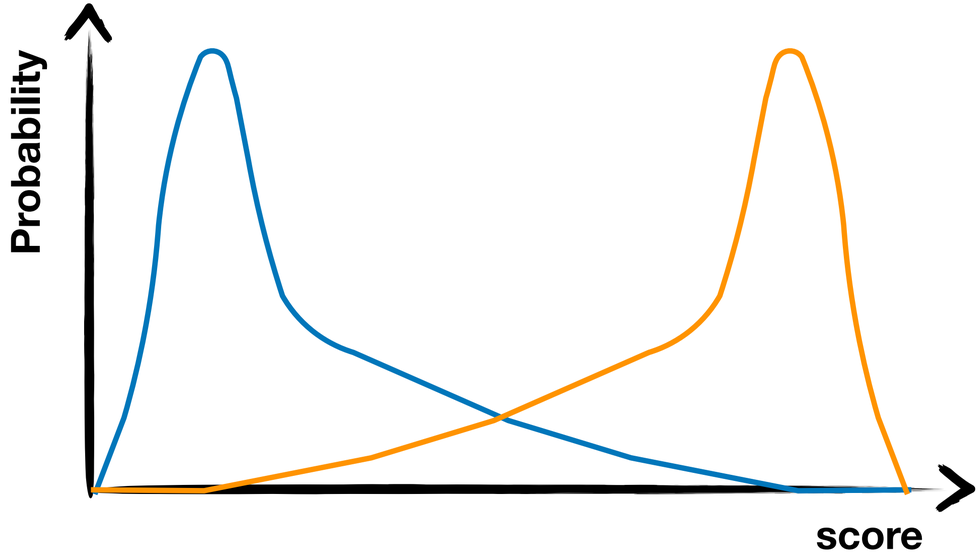
Classifier metrics

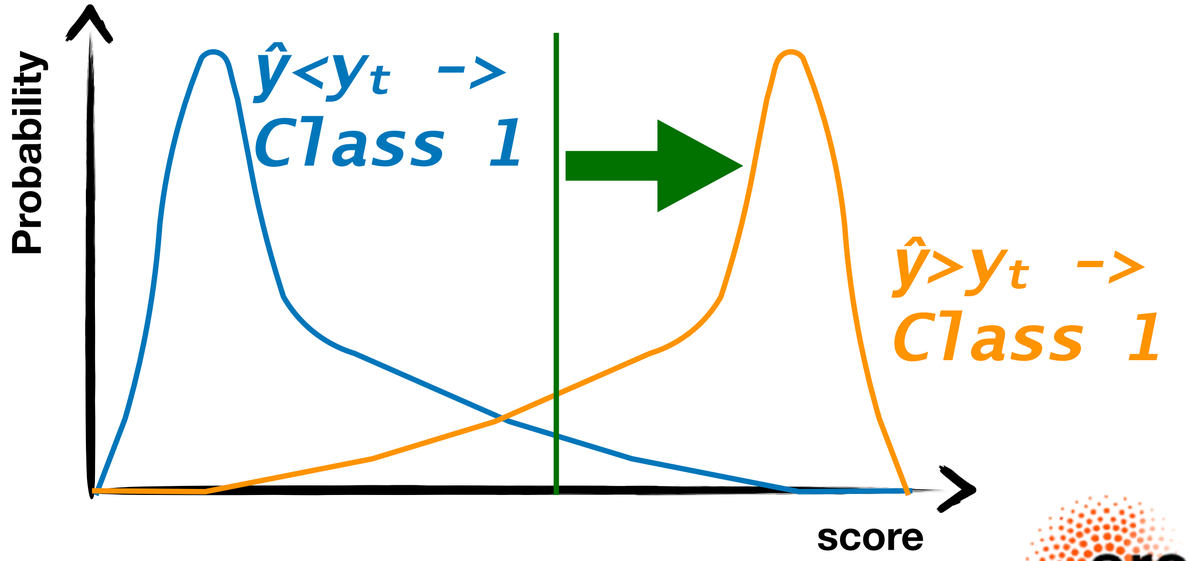
- *A given threefold defines the following qualities*
  - *True-positives: Class-1 events above the threshold*
  - *True-negatives: Class-0 events below the threshold*
  - *False-positives: Class-0 events above the threshold*
  - *False-negatives: Class-1 events below the threshold*



Classifier metrics

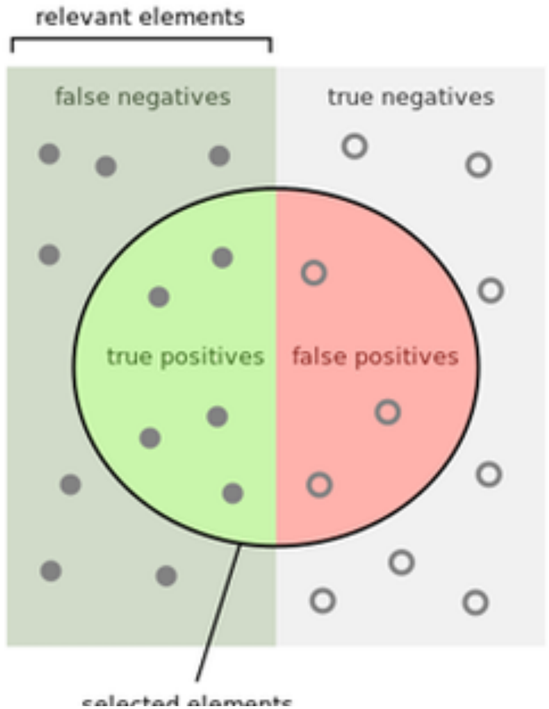
- Consider a binary classifier
- Its output $\hat{y}$ is a number in [0,1]
- If well trained, value should be close to 0 (1) for class-0 (class-1) examples
- One usually defines a threshold y_t such that:
  - $\hat{y} > y_t$ -> Class 1
  - $\hat{y} < y_t$ -> Class 0

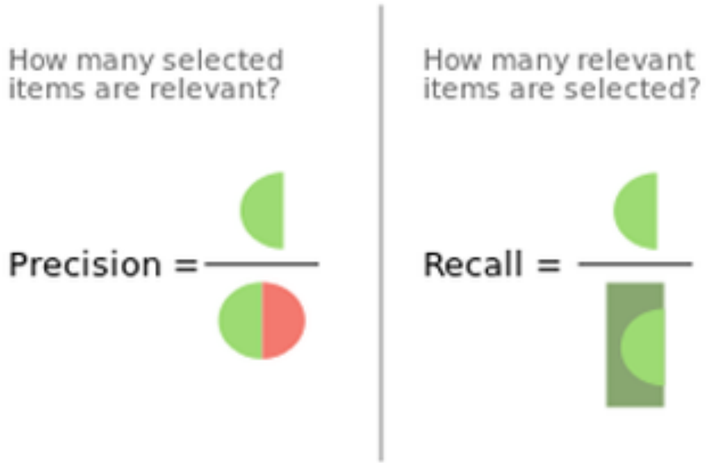




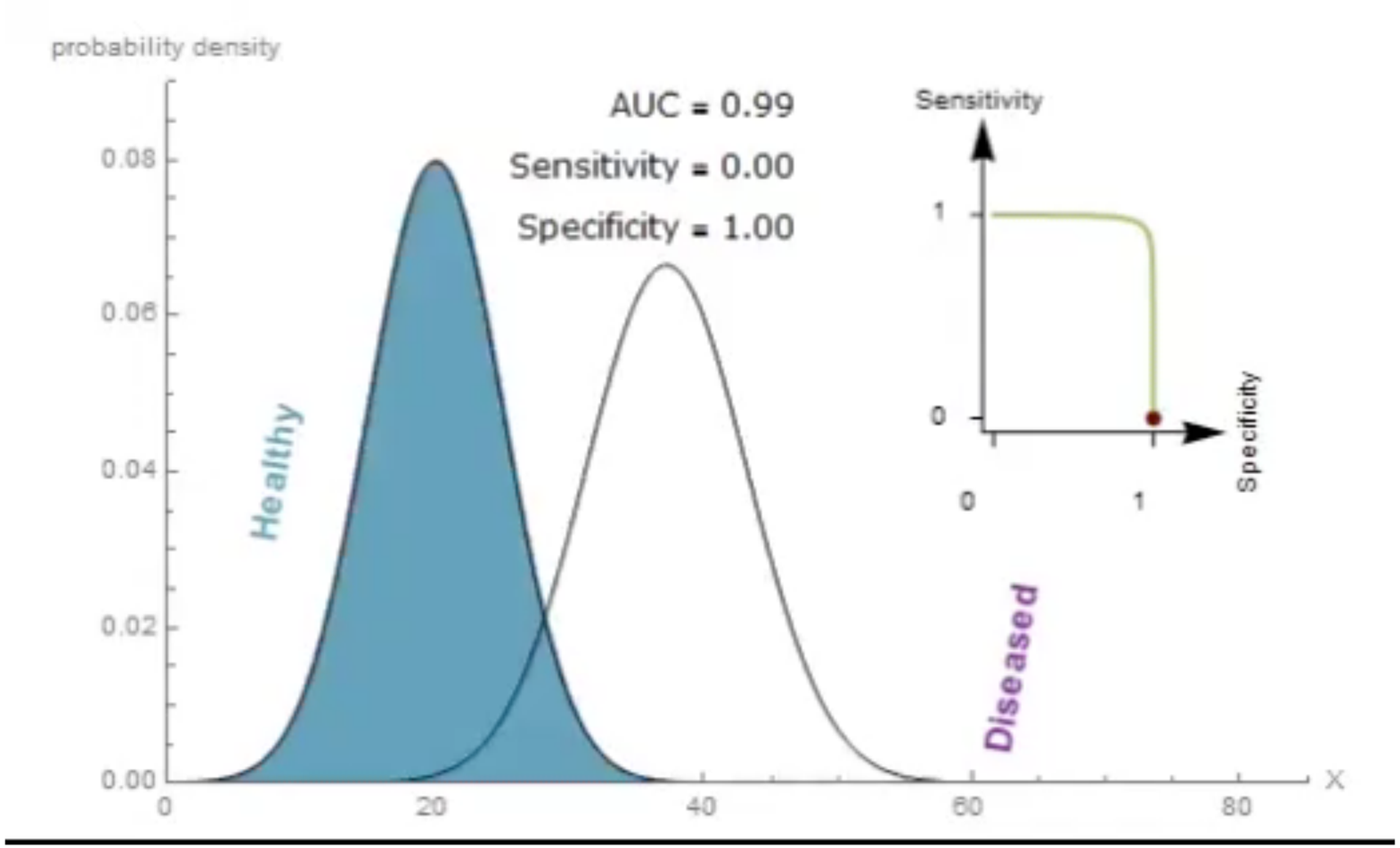
Classifier metrics

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





Receiver operating characteristic



Hyperparameter Search

- *Good choice of parameters can be done by hand, guided by your understanding of what each parameter does and how they interplay*

Hyperparameter	Increases capacity when...	Reason	Caveats
Number of hidden units	increased	Increasing the number of hidden units increases the representational capacity of the model.	Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model.
Learning rate	tuned optimally	An improper learning rate, whether too high or too low, results in a model with low effective capacity due to optimization failure.	
Convolution kernel width	increased	Increasing the kernel width increases the number of parameters in the model.	A wider kernel results in a narrower output dimension, reducing model capacity unless you use implicit zero padding to reduce this effect. Wider kernels require more memory for parameter storage and increase runtime, but a narrower output reduces memory cost.
Implicit zero padding	increased	Adding implicit zeros before convolution keeps the representation size large.	Increases time and memory cost of most operations.
Weight decay coefficient	decreased	Decreasing the weight decay coefficient frees the model parameters to become larger.	
Dropout rate	decreased	Dropping units less often gives the units more opportunities to "conspire" with each other to fit the training set.	

Hyperparameter Search

- *Good choice of parameters can be done by large-scale calculations, repeating the training multiple times*
  - *Scan the space with a regular grid or with a random scan*
  - *Define the tuning as a minimisation process itself (e.g., with Bayesian optimization or other regression techniques)*