Queen Mary
University of London

DR ADRIAN BEVAN

MULTIVARIATE ANALYSIS AND ITS USE IN HIGH ENERGY PHYSICS

4) OPTIMISATION

Lectures given at the department of Physics at CINVESTAV, Instituto Politécnico Nacional, Mexico City
28th August - 3rd Sept 2018

LECTURE PLAN

- Introduction
- Hyper parameter optimisation
  - Supervised learning
  - Loss functions
  - Gradient descent
  - Stochastic learning
  - Batch learning
- Overtraining
  - Training validation
  - Dropout for deep networks
  - Weight regularisation for neural networks
  - Cross Validation methods
- Summary
- Suggested reading

INTRODUCTION

- We initialise the weights of a neural network or other MVA algorithm randomly.
- The determination a set of optimal weights requires some heuristic algorithm and some figure of merit.
  - The algorithm is the optimisation method (typically derived from gradient descent).
  - The figure of merit is called the loss or cost function and can take many forms.
- The optimisation process for machine learning algorithms is similar to optimisation of a X^2 or -lnL in a fit where one minimises the X^2 or -lnL as the figure of merit, with respect to the model parameters.

HYPERPARAMETER OPTIMISATION

- Models have hyperparameters (**HPs**) that are required to fix the response function(*).
- The set of HPs forms a hyperspace.
- The purpose of optimisation is to select a point in hyperspace that optimises the performance of the model using some figure of merit (**FOM**).
- The figure of merit is called the cost or loss function.
  - c.f. least squares regression or a χ^2 or likelihood fit.

(*) Minimisation problems related to likelihood fitting often split the hyper-parameters into parameters of interest (e.g. physical quantities) and other nuisance parameters that are not deemed to be interesting. Machine learning model parameters are the equivalent of nuisance parameters in the language of likelihood fits. e.g. see the book by Edwards, *Likelihood* (1992), John Hopkins Uni Press.

HYPERPARAMETER OPTIMISATION

- Consider a perceptron with N inputs.
  - This has N+1 HPs: N weights and a bias:

$$y = f\left(\sum_{i=1}^{N} w_i x_i + \theta\right)$$
$$= f(w^T x + \theta)$$

  - For brevity the bias parameter is called a weight from the perspective of HP optimisation.

HYPERPARAMETER OPTIMISATION

- Consider a neural network with an N dimensional input feature space, M perceptrons on the input layer and 1 output perceptron.
  - This has $M(N+1) + (M+1)$ HPs.
- For an MLP with one hidden layer of K perceptrons:
  - This has $M(N+1) + K(M+1) + (K+1)$ HPs.
- For an MLP with two hidden layers of K and L perceptrons, respectively:
  - This has $M(N+1) + K(M+1) + L(K+1) +(L+1)$ HPs.
- and so on.

HYPERPARAMETER OPTIMISATION

- Neural networks have a lot of HPs. Deep networks, especially CNNs can have *millions* of HPs to optimise.
  - Requires appropriate computing resource.
  - Requires appropriately efficient methods for HP optimisation.
  - What is acceptable for an optimisation of 10 or 100 HPs will not generally scale well to a problem with 10^3-10^6 HPs.
- The more HPs to determine the more data is required to obtain a generalisable solution for the HPs.
  - By generalisable we mean that the model defined using a set of HPs will have reproducible behaviour when presented with unseen data.
  - When this is not the case we have an overtrained model - one that has learned statistical fluctuations in our training set.

HYPERPARAMETER OPTIMISATION: SUPERVISED LEARNING

- The type of machine learning we are using is referred to as supervised learning.
  - We present the algorithm with known (labeled) samples of data, and optimise the HPs in order to minimise the loss function.
  - The loss function is a function of:
    - labels (e.g. signal = +1, background = 0 or -1 depending on convention/choice of loss function)
    - hyper parameters (i.e. weights, biases, etc.)
    - activation functions
    - architecture of the network (arrangement of perceptrons)

HYPERPARAMETER OPTIMISATION: LOSS FUNCTIONS

- There are a number of different types of loss (cost) function that are commonly used.
- These different figures of merit are measures of how well a model performs at ensuring the weights provide a reasonable output response.
- Loss functions can have additional terms added to them (e.g. weight regularisation for overtraining and for constructing adversarial networks).
- Three common loss functions are described in the following.

HYPERPARAMETER OPTIMISATION: LOSS FUNCTIONS

- L2-norm loss:
  - This is like a X^2 term, but without the error normalisation and a factor of 1/2.

$$\varepsilon = \sum_{i=1}^{N} \frac{1}{2}(y_i - t_i)^2$$

N = number of examples

y_i = Model output for the ith example

t_i = True target type for the ith example (label values)

  - The L1 norm loss function is as above, without the factor of 1/2 or square.

HYPERPARAMETER OPTIMISATION: LOSS FUNCTIONS

- Mean Square Error (MSE) loss:
  - Very similar to the L2 norm loss function; just normalise the L2 norm loss by the number of training examples to compute an average.

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)^2$$

N = number of examples

y_i = Model output for the ith example

t_i = True target type for the ith example (label values)

HYPERPARAMETER OPTIMISATION: LOSS FUNCTIONS

- Cross Entropy:
  - This loss function is inspired by the likelihood for observing either target value $P(t|x) = y^t(1-y)^{(1-t)}$
  - From the likelihood L of observing the training data set we can compute the -lnL as

$$\varepsilon = -\sum_{i=1}^{n} [t^n \ln y^n + (1-t^n)\ln(1-y^n)]$$

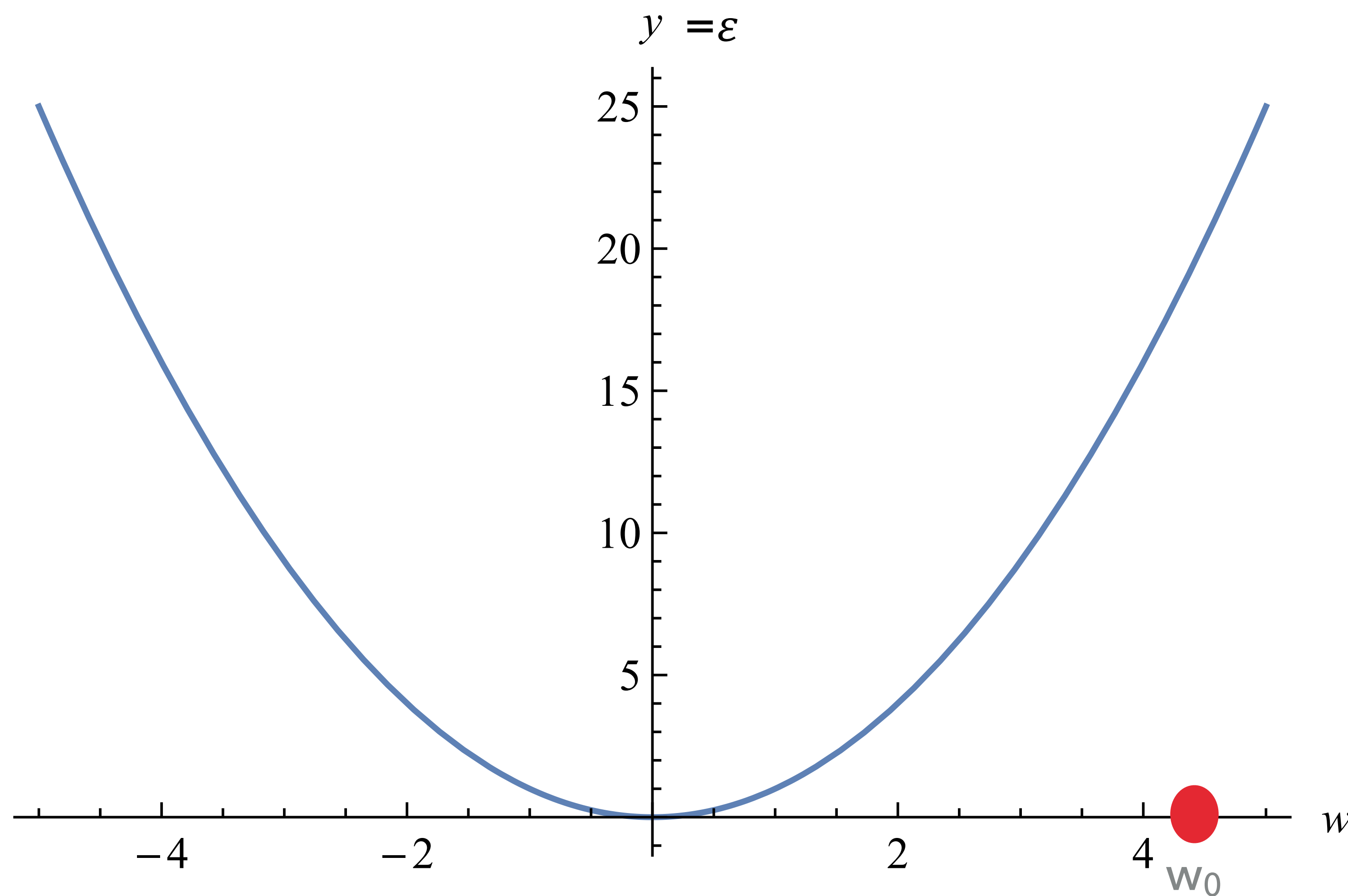
n = number of examples

t = target type (0 or 1) depending on example (label values)

y = output prediction of model

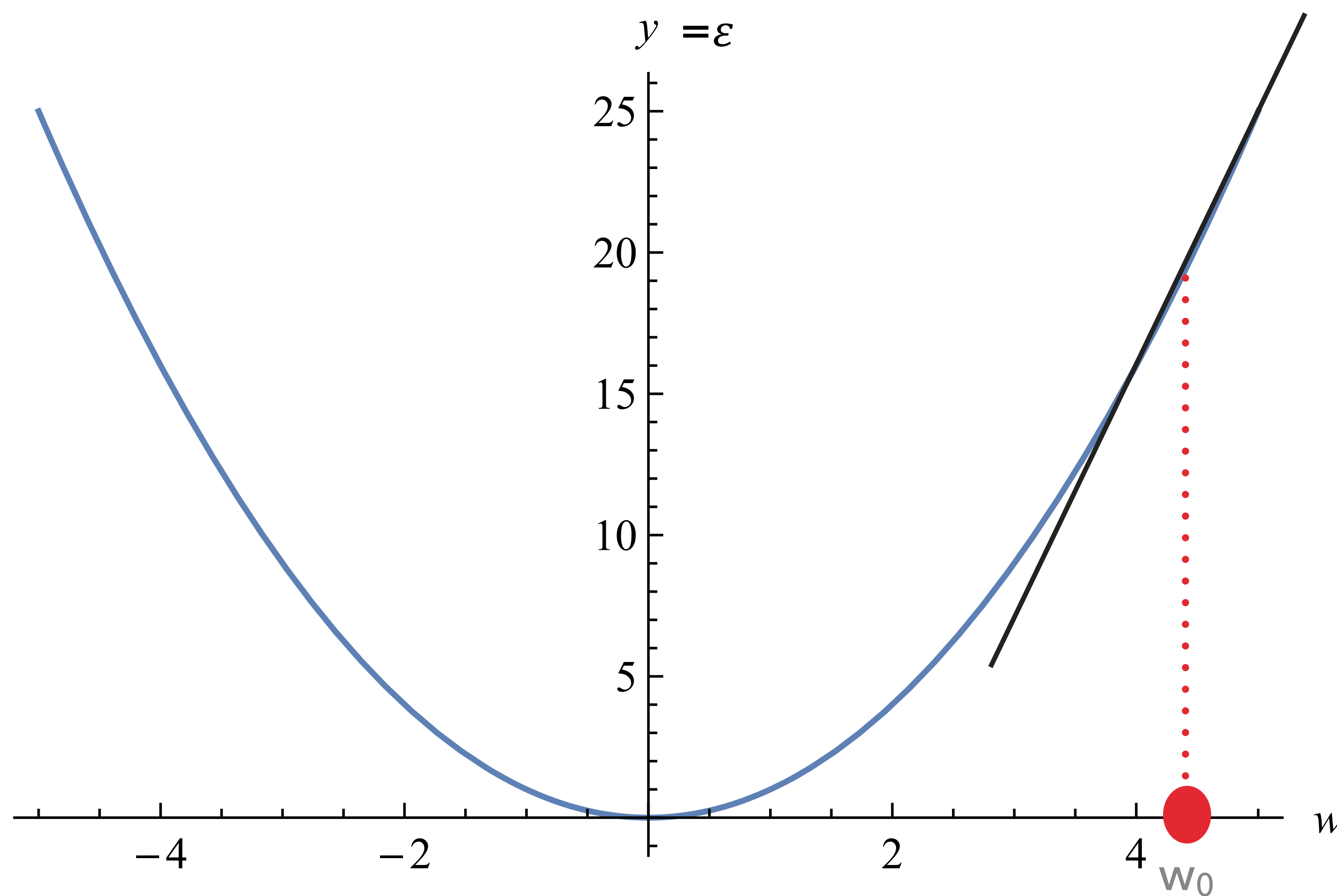
HYPERPARAMETER OPTIMISATION: GRADIENT DESCENT

- Newtonian gradient descent is a simple concept.
  - Guess an initial value for the weight parameter: w_0.



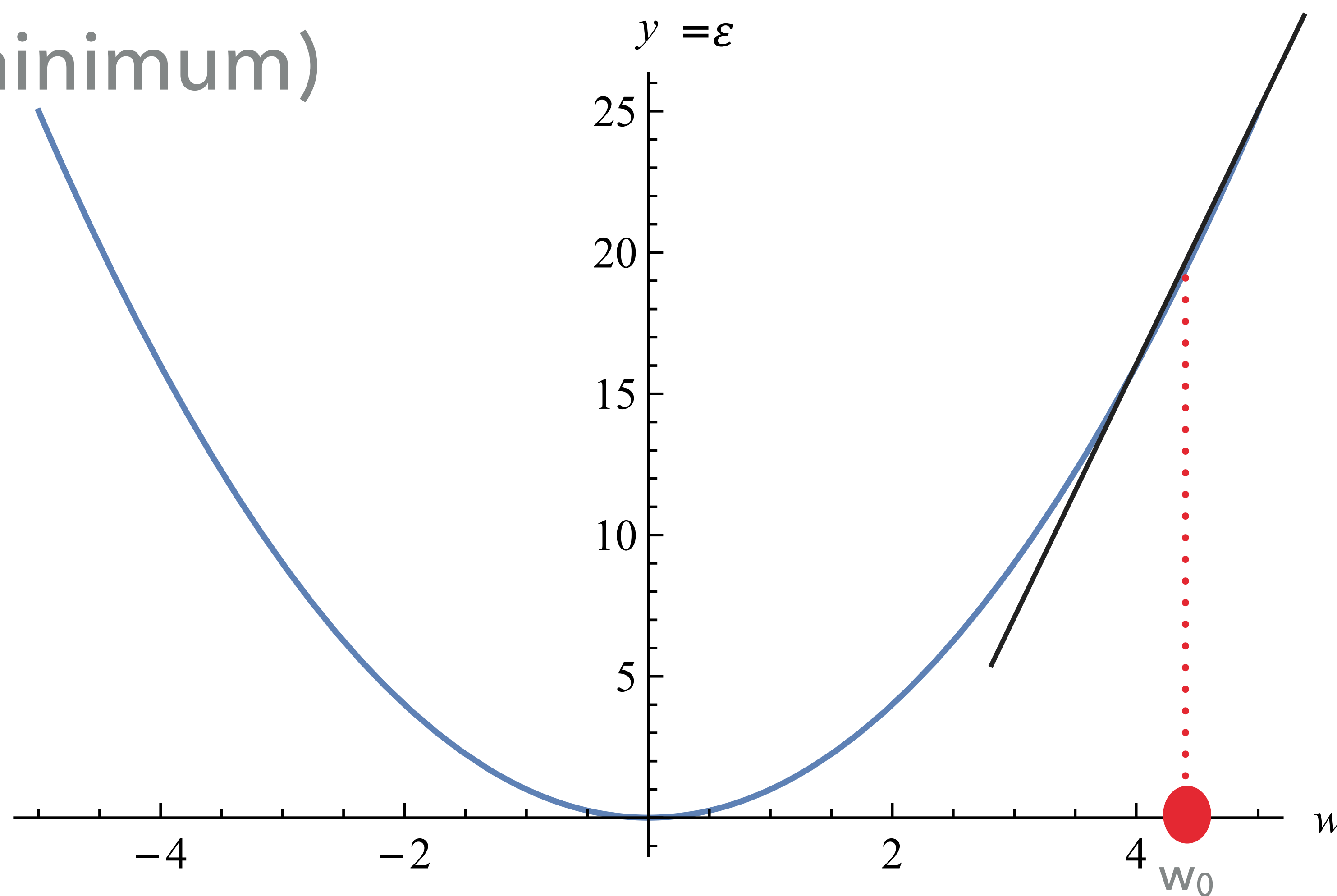
HYPERPARAMETER OPTIMISATION: GRADIENT DESCENT

- Newtonian gradient descent is a simple concept.
  - Estimate the gradient at that point (tangent to the curve)



HYPERPARAMETER OPTIMISATION: GRADIENT DESCENT

- Newtonian gradient descent is a simple concept.
  - Compute Δw such that Δy is negative (to move toward the minimum)



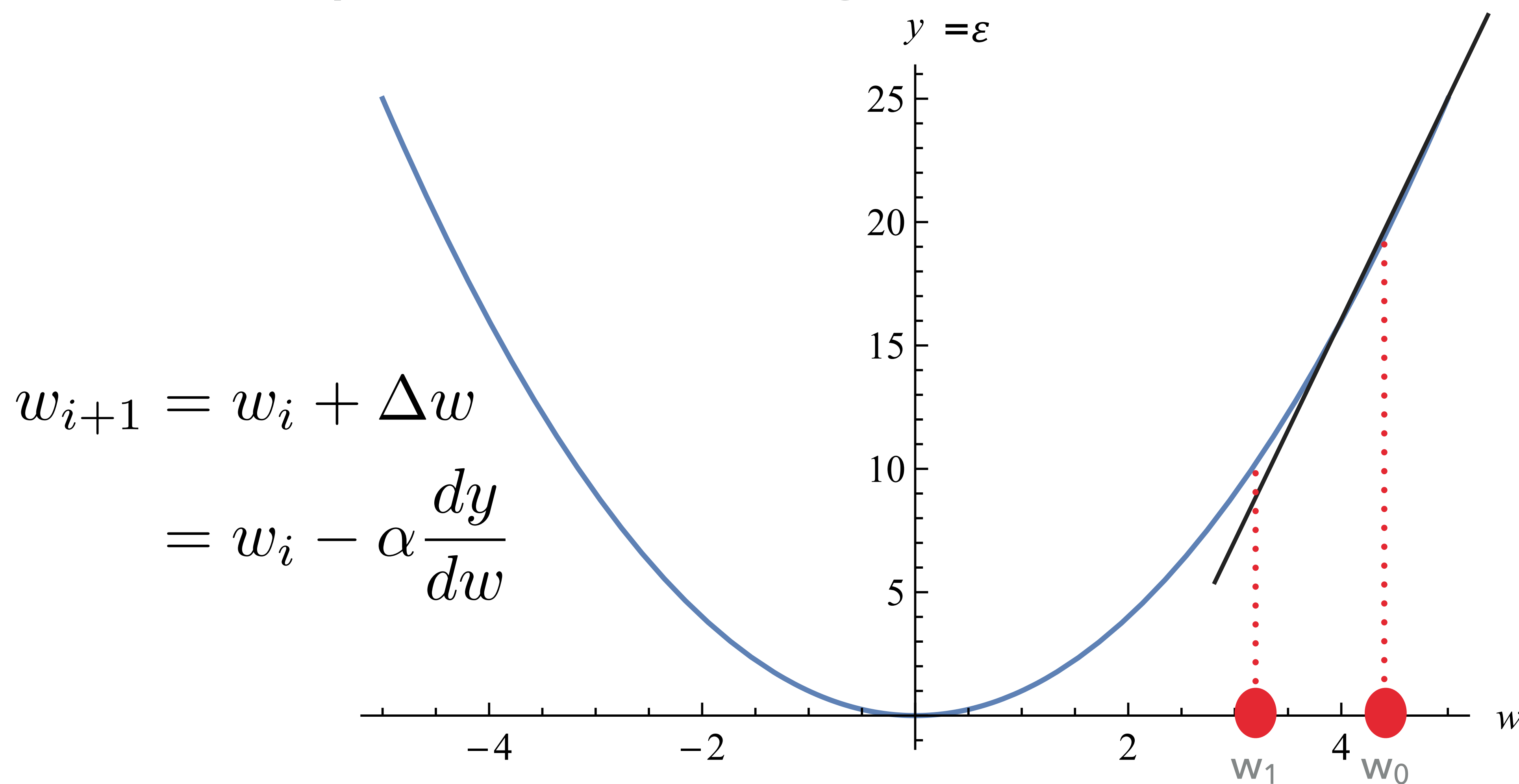
$$\Delta y = \Delta w \frac{dy}{dw}$$
$$= -\alpha \left(\frac{dy}{dw}\right)^2$$

α is the learning rate: a small positive number

Choose $\Delta w = -\alpha \frac{dy}{dw}$ to ensure Δy is always negative.

HYPERPARAMETER OPTIMISATION: GRADIENT DESCENT

- Newtonian gradient descent is a simple concept.
  - Compute a new weight value: $w_1 = w_0 + \Delta w$



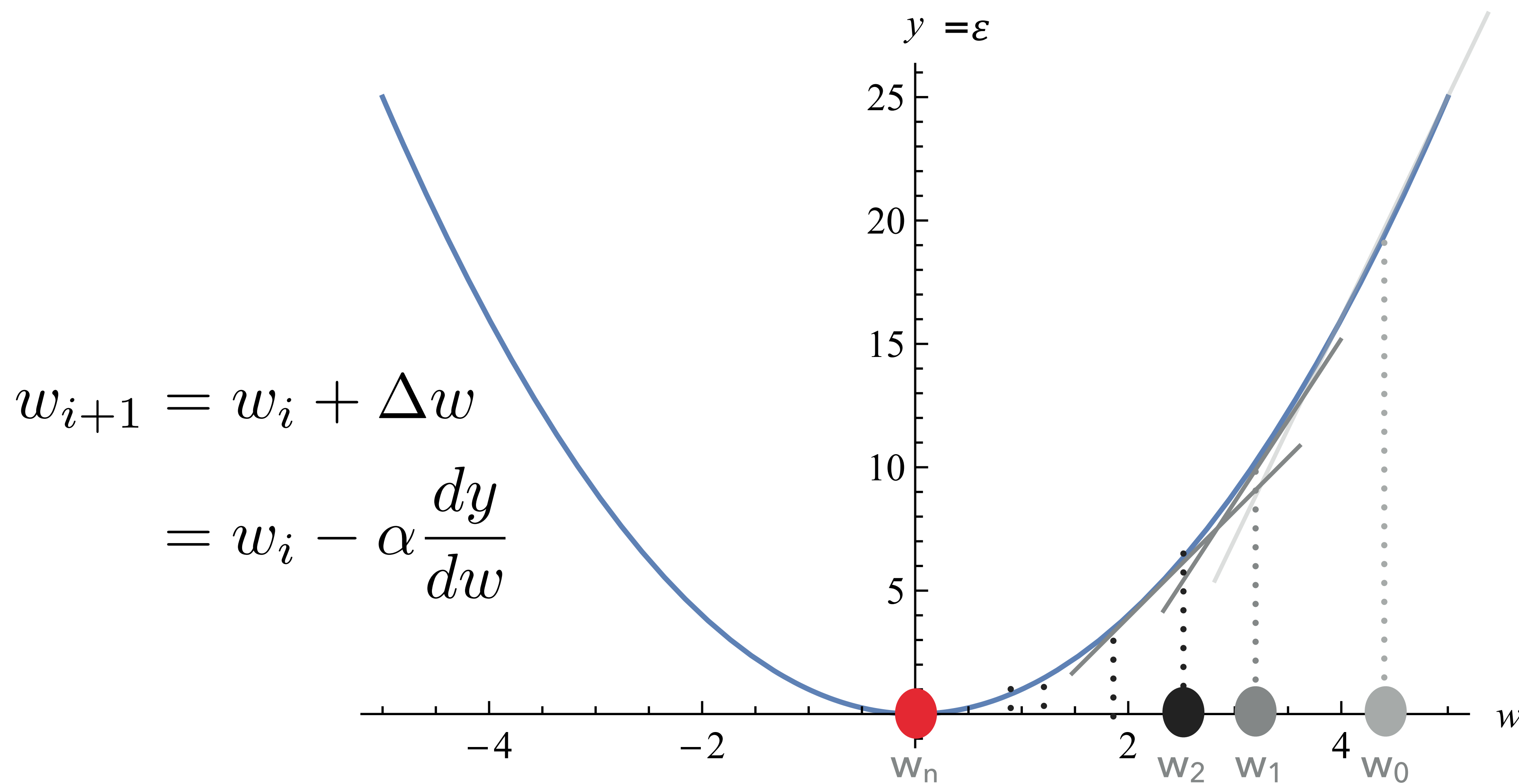
$$\Delta y = \Delta w \frac{dy}{dw}$$
$$= -\alpha \left(\frac{dy}{dw}\right)^2$$

α is the learning rate: a small positive number

Choose $\Delta w = -\alpha \frac{dy}{dw}$ to ensure Δy is always negative.

HYPERPARAMETER OPTIMISATION: GRADIENT DESCENT

- Newtonian gradient descent is a simple concept.
  - Repeat until some convergence criteria is satisfied.



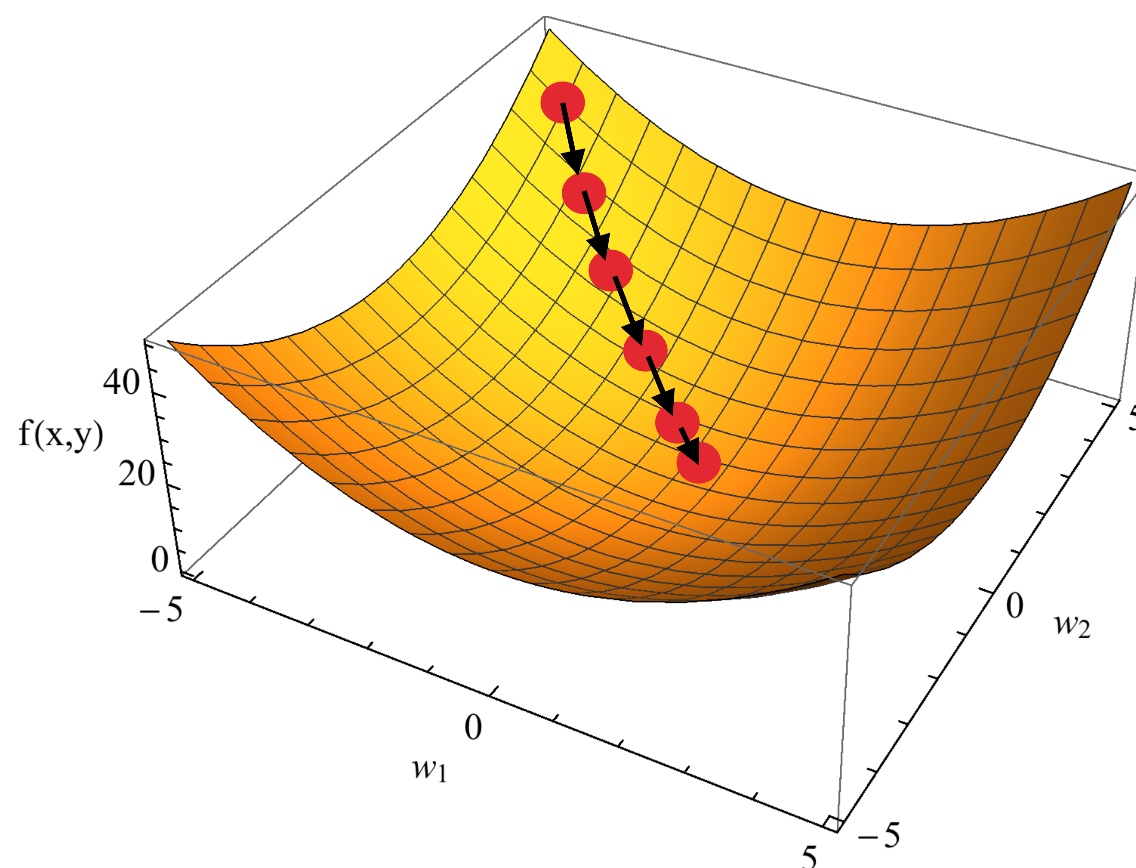
$$\Delta y = \Delta w \frac{dy}{dw}$$
$$= -\alpha \left(\frac{dy}{dw} \right)^2$$

α is the learning rate: a small positive number

Choose $\Delta w = -\alpha \frac{dy}{dw}$ to ensure Δy is always negative.

HYPERPARAMETER OPTIMISATION: GRADIENT DESCENT

- We can extend this from a one parameter optimisation to a 2 parameter one, and follow the same principles, now in 2D.



- The successive points $\underline{w}_{i+1}$ can be visualised a bit like a ball rolling down a concave hill into the region of the minimum.

HYPERPARAMETER OPTIMISATION: GRADIENT DESCENT

- In general for an n-dimensional hyperspace of hyper parameters we can follow the same brute force approach using:

$$\begin{aligned}\Delta y &= \Delta w \nabla y \\ &= -\alpha \left[\left(\frac{dy}{dw_{1,i}} \right)^2 + \left(\frac{dy}{dw_{2,i}} \right)^2 + \ldots \left(\frac{dy}{dw_{n,i}} \right)^2 \right]\end{aligned}$$

- where

$$\begin{aligned}w_{i+1} &= w_i + \Delta w \\ &= w_i - \alpha \nabla y \\ &= w_i - \alpha \left[\left(\frac{dy}{dw_{1,i}} \right)^2 + \left(\frac{dy}{dw_{2,i}} \right)^2 + \ldots \left(\frac{dy}{dw_{n,i}} \right)^2 \right]\end{aligned}$$

HYPERPARAMETER OPTIMISATION: BACK PROPAGATION

- The delta-rule or back propagation method is used for NNs to determine the weights; based on gradient descent.

- For a regularisation problem* we use the L2 norm loss function (with or without the factor of 1/2):

$$\varepsilon = \sum_{i=1}^{N} \frac{1}{2}(y_i - t_i)^2$$

y_i is the model output for example x_i

t_i is the corresponding label for the ith example

- We can compute the derivative of the ε with respect to the weights as:

$$\frac{\partial \varepsilon_i}{\partial w} \text{ and } \frac{\partial \varepsilon_i}{\partial \theta}$$

Derivatives depend on the activation function(s) used in the model y(x)

*Either this or cross entropy are used for the classification problems.

HYPERPARAMETER OPTIMISATION: BACK PROPAGATION

- The parameters w and θ are updated using:

$$w^{r+1} = w^r - \alpha \sum_{i=1}^{N} \frac{\partial \varepsilon_i}{\partial w}$$

$$\theta^{r+1} = \theta^r - \alpha \sum_{i=1}^{N} \frac{\partial \varepsilon_i}{\partial \theta}$$

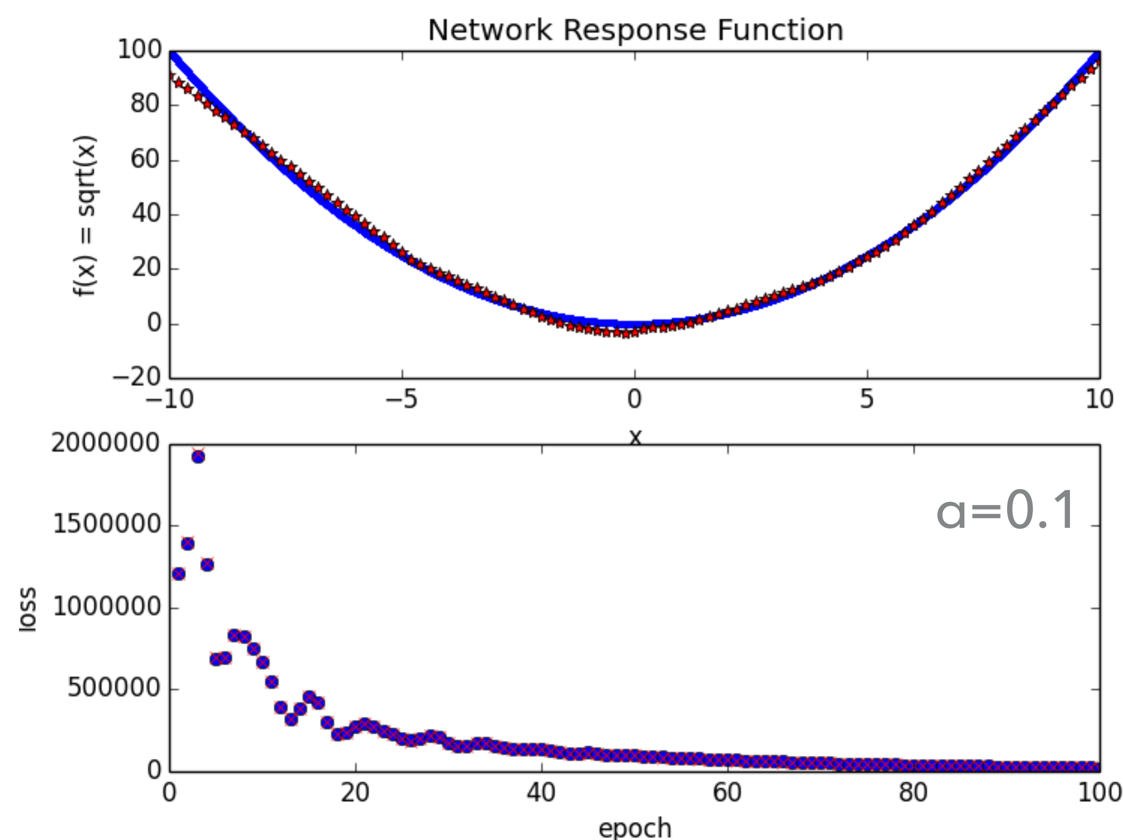
where α is the small positive learning rate

- The derivatives can be re-written in terms of the "errors" on the weights, and the errors on w and θ can be related to each other.
- Back propagation involves:
  - A forward pass where weights are fixed and the model predictions are made*.
  - This is followed by the backward pass where the errors on the bias parameters are computed and used to determine the errors on the weights. These in turn are then used to update the HPs from epoch r to epoch r+1.

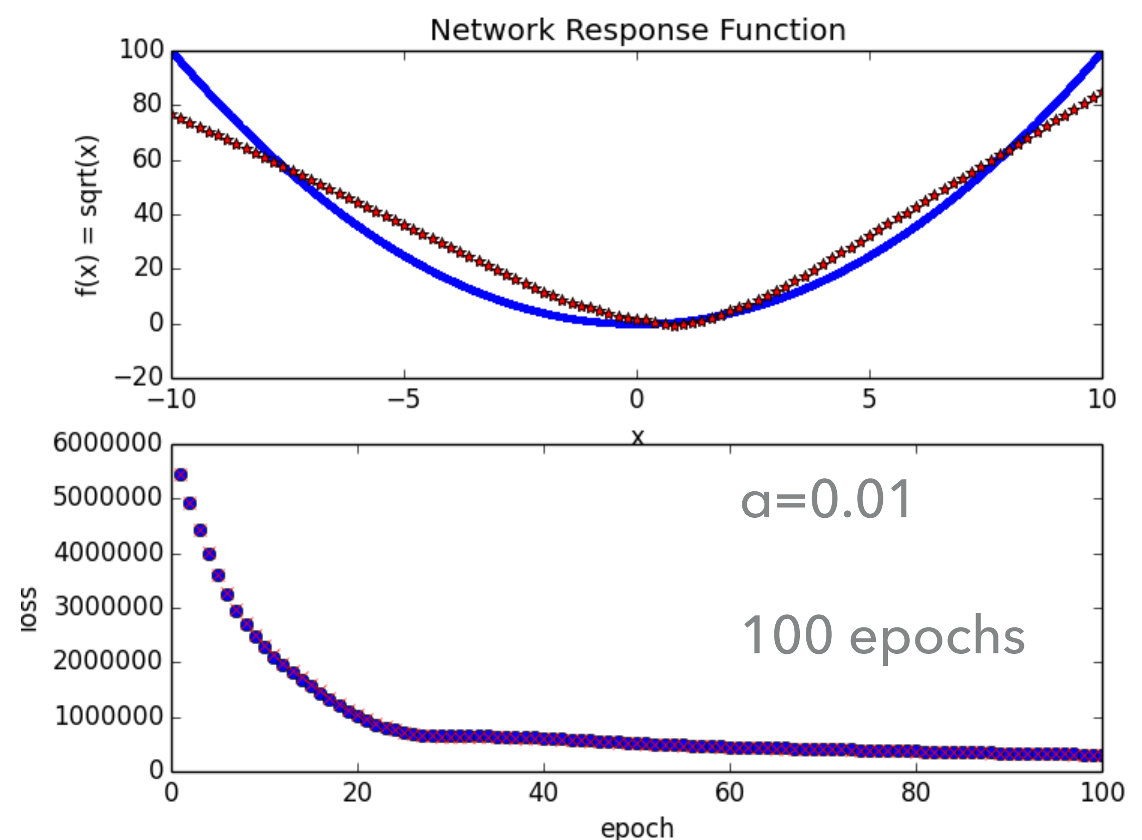
*Weights are randomly initialised for the whole network.

HYPERPARAMETER OPTIMISATION

- Consider the examples of an MLP to approximate the function $f(x) = x^2$ and how the convergence depends on the learning rate.



Too large a learning rate and the optimisation does not always lead to an improved cost; the small change approximation breaks down.

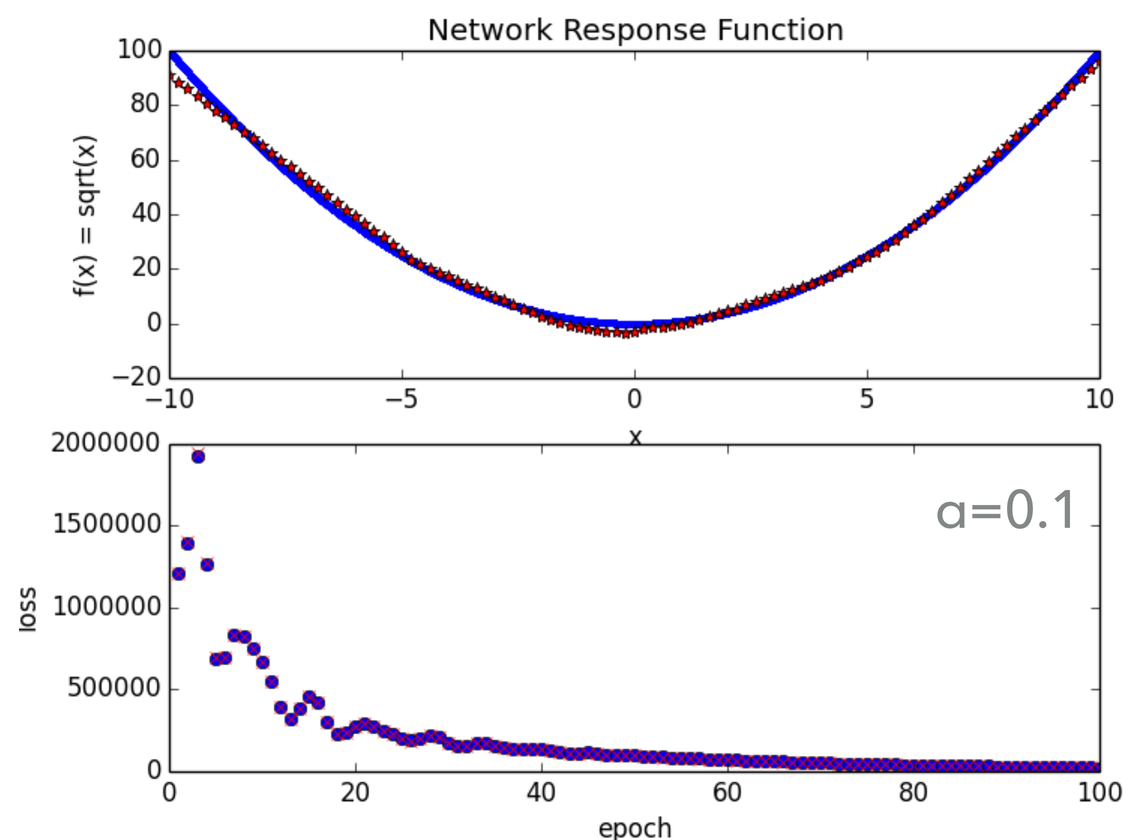


A small learning rate means the convergence takes much longer (a x10 reduction in the learning rate will require a x10 increase in optimisation steps)

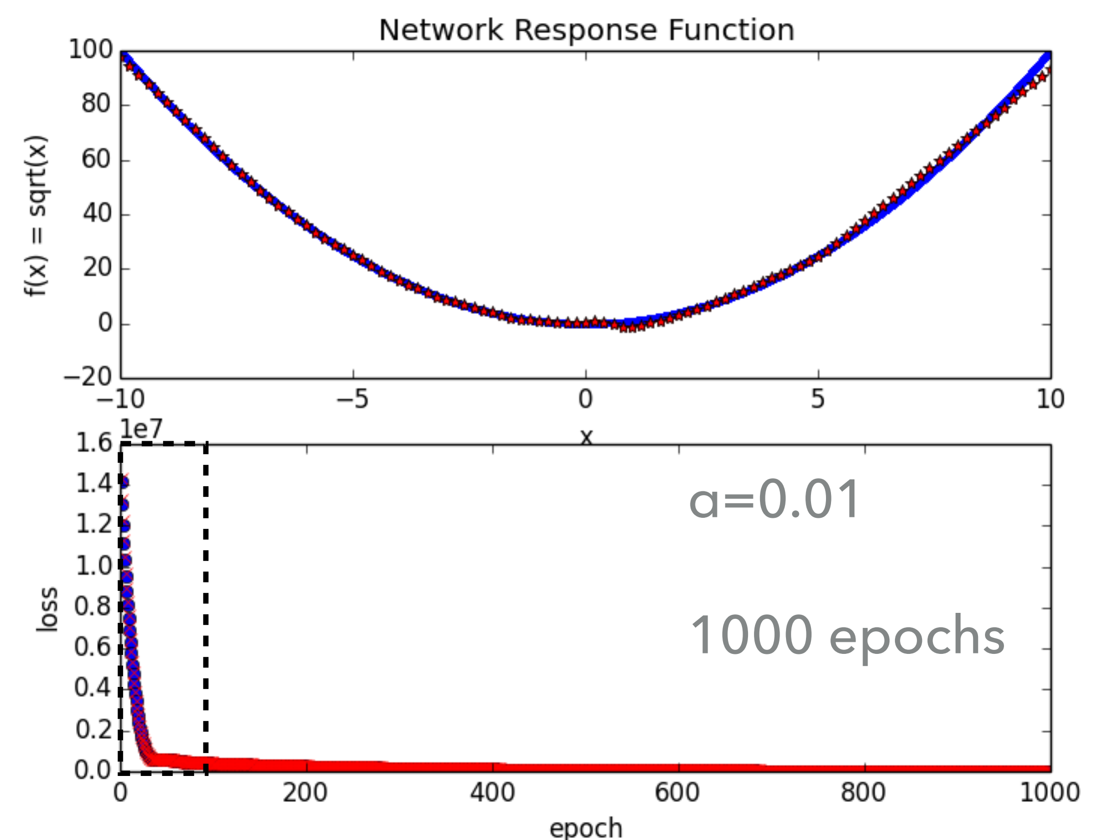
This example uses an L2 loss function and is a neural network with 1 input, a single layer of 50 nodes and 1 output; implemented in TensorFlow.

HYPERPARAMETER OPTIMISATION

- Consider the examples of an MLP to approximate the function $f(x) = x^2$ and how the convergence depends on the learning rate.



Too large a learning rate and the optimisation does not always lead to an improved cost; the small change approximation breaks down.



A small learning rate means the convergence takes much longer (a x10 reduction in the learning rate will require a x10 increase in optimisation steps)

This example uses an L2 loss function and is a neural network with 1 input, a single layer of 50 nodes and 1 output; implemented in TensorFlow.

HYPERPARAMETER OPTIMISATION: STOCHASTIC LEARNING[1]

Advantages of Stochastic Learning

1. Stochastic learning is usually *much* faster than batch learning.
2. Stochastic learning also often results in better solutions.
3. Stochastic learning can be used for tracking changes.

- Data are inherently noisy.
- Individual training examples can be used to estimate the gradient.
- Training examples tend to cluster, so processing a batch of training data, one example at a time results in sampling the ensemble in such a way to have faster optimisation performance.
- Noise in the data can help the optimisation algorithm avoid getting locked into local minima.
- Often results in better optimisation performance than batch learning.

[1] LeCun et al., Efficient BackProp, Neural Networks Tricks of the Trade, Springer 1998

HYPERPARAMETER OPTIMISATION: BATCH LEARNING[1]

Advantages of Batch Learning

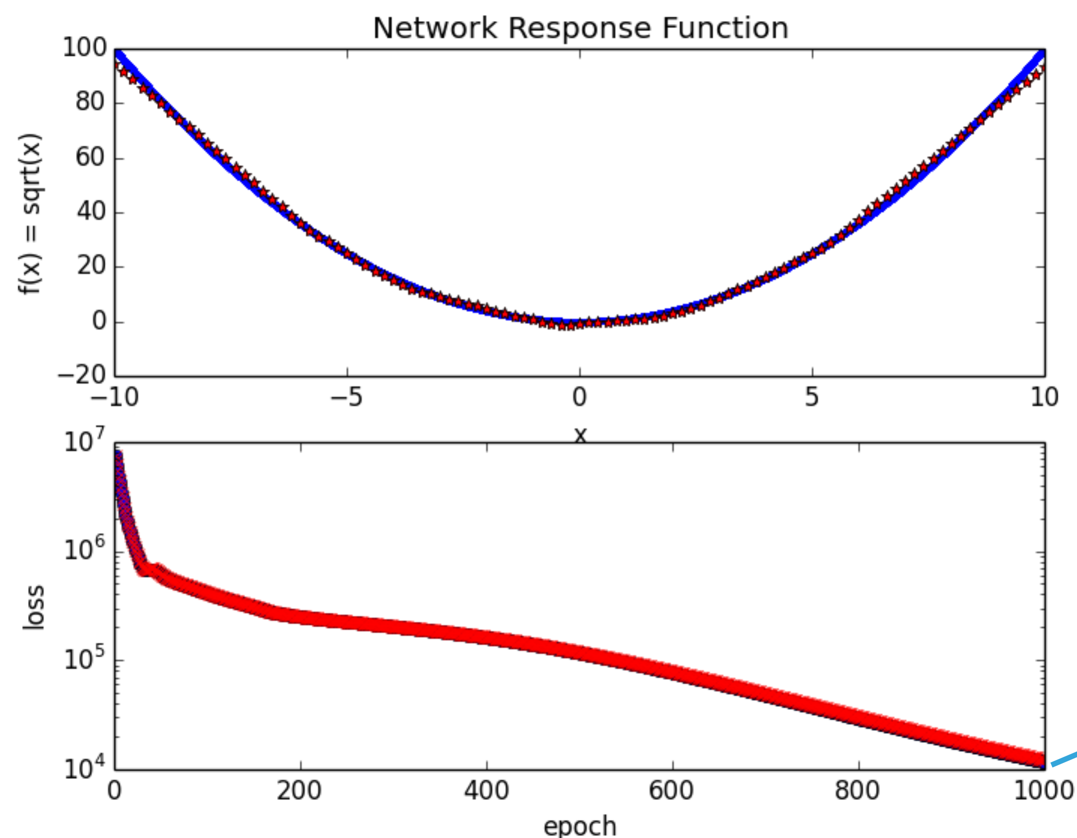
1. Conditions of convergence are well understood.
2. Many acceleration techniques (e.g. conjugate gradient) only operate in batch learning.
3. Theoretical analysis of the weight dynamics and convergence rates are simpler.

- Data are inherently noisy.
- Can use a sample of training data to estimate the gradient for minimisation (see later) to minimise the effect of this noise.
  - The sample of data is used to obtain a better estimate the gradient
  - This is referred to as batch learning.
  - Can use mini-batches of data to speed up optimisation, which is motivated by the observation that for many problems there are clusters of similar training examples.

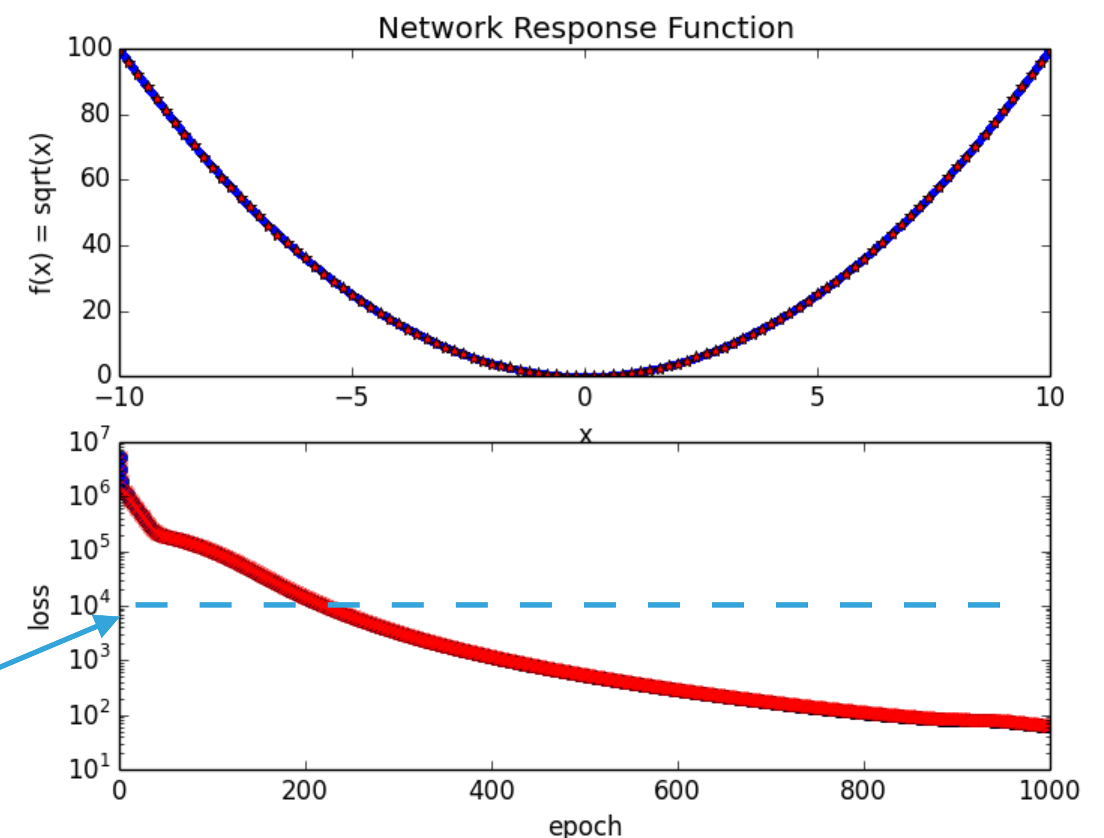
[1] LeCun et al., Efficient BackProp, Neural Networks Tricks of the Trade, Springer 1998

HYPERPARAMETER OPTIMISATION: BATCH LEARNING

- Returning to the example $f(x) = x^2$;
  - optimising on 1/4 of the data at a time (4 batches) leads to accelerated optimisation relative to optimising on all the data each epoch.



Training with all the data and a learning rate of 0.01.



Batch training (4 batches) with all the data and a learning rate of 0.01.

This example uses an L2 loss function and is a neural network with 1 input, a single layer of 50 nodes and 1 output; implemented in TensorFlow.

GRADIENT DESCENT: REFLECTION

- For a problem with a parabolic minimum and an appropriate learning rate, α, to fix the step size, we can guarantee convergence to a sensible minimum in some number of steps.
  - If we translate the distribution to a fixed scale, then all of a sudden we can predict how many steps it will take to converge to the minimum from some distance away from it for a given α.
  - If the problem hyperspace is not parabolic, this becomes more complicated.

[1] LeCun et al., Efficient BackProp, Neural Networks Tricks of the Trade, Springer 1998

GRADIENT DESCENT: REFLECTION

- Based on the underlying nature of the gradient descent optimisation algorithm family, being derived to optimise a parabolic distribution, ideally we want to try and standardise the input distributions to a neural network.

  - Use a unit Gaussian as a standard e.g.:
    - maps x to $x' = (x-\mu)/\sigma$;
    - Scale x' to the range [-1, 1].

  - The transformed data inputs will be scale invariant in the sense that HPs such as the learning rate will be come general, rather than problem (and therefore scale) dependent.

  - If we don't do this the optimisation algorithm will work, but it may take longer to converge to the minimum, and could be more susceptible to divergent behaviour.

[1] LeCun et al., Efficient BackProp, Neural Networks Tricks of the Trade, Springer 1998

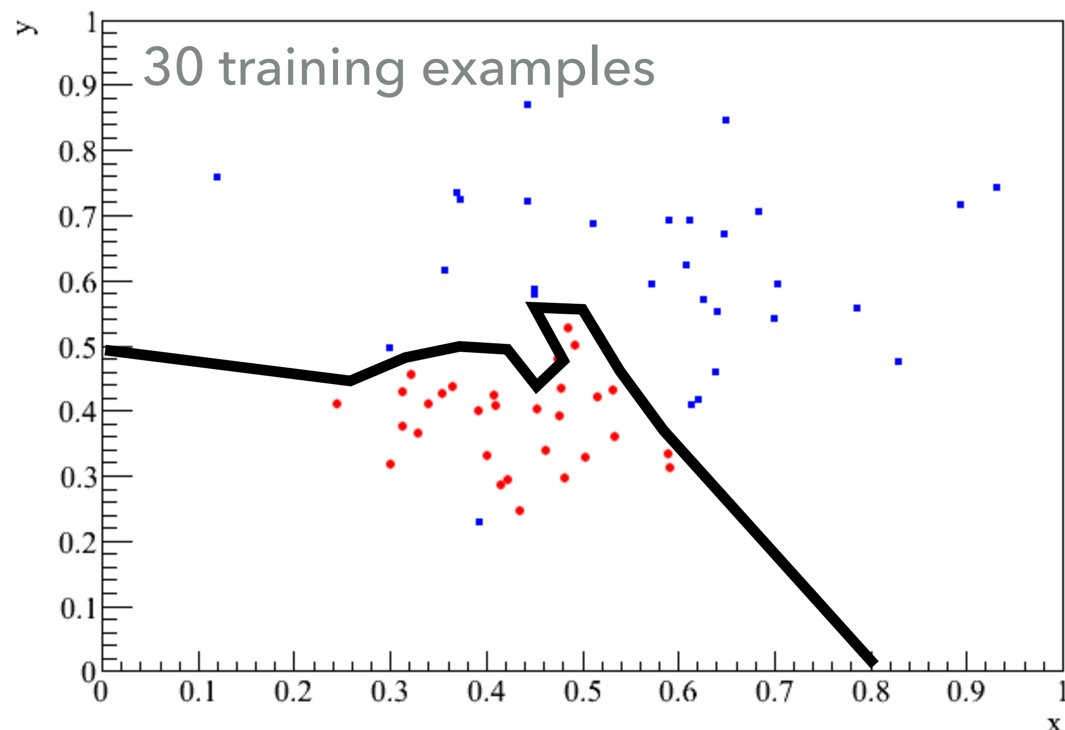
OVERTRAINING

- Data are noisy.
- Optimisation can result in learning the noise in the training data.
- Overtraining is the term given to learning the noise, and this can be mitigated in a number of different ways:
    - Using more training data (not always possible).
    - Checking against different data sets to identify the onset of learning noise.
    - Changing the network configuration when training (dropout).
    - Weight regularisation (large weights are penalised in the cost).
- None of these methods guarantees that you avoid over training.

OVERTRAINING

- A model is over fitted if the HPs that have been determined are tuned to the statistical fluctuations in the data set.

- Simple illustration of the problem:



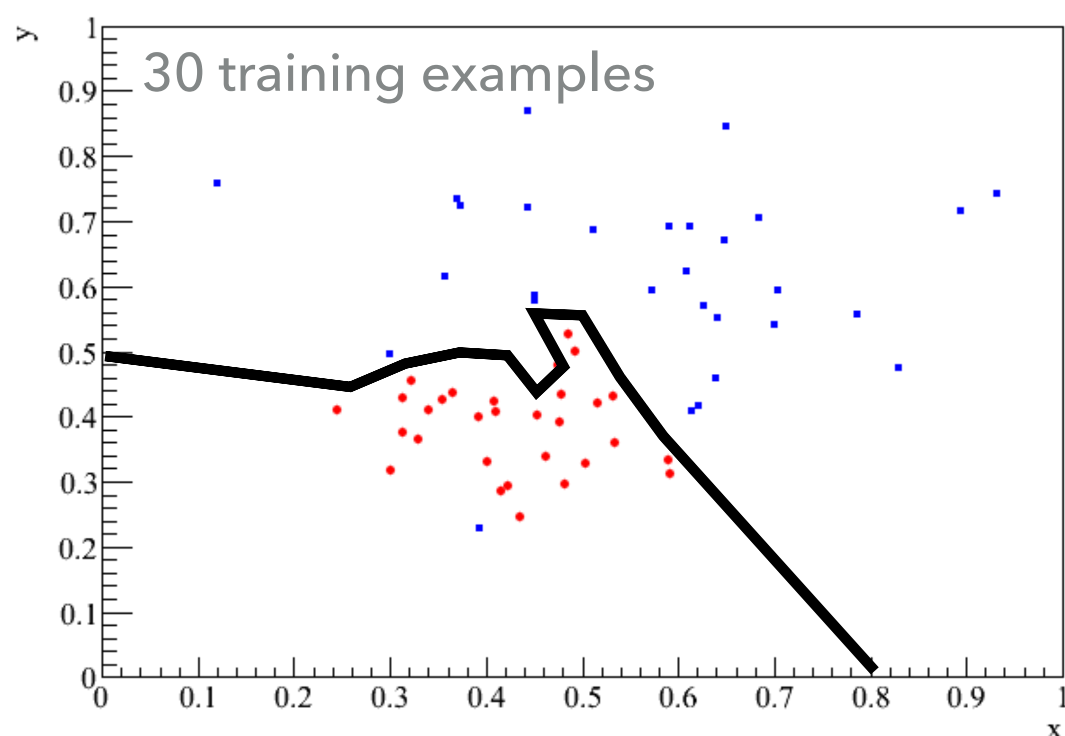
The decision boundary selected here does a good job of separating the red and blue dots.

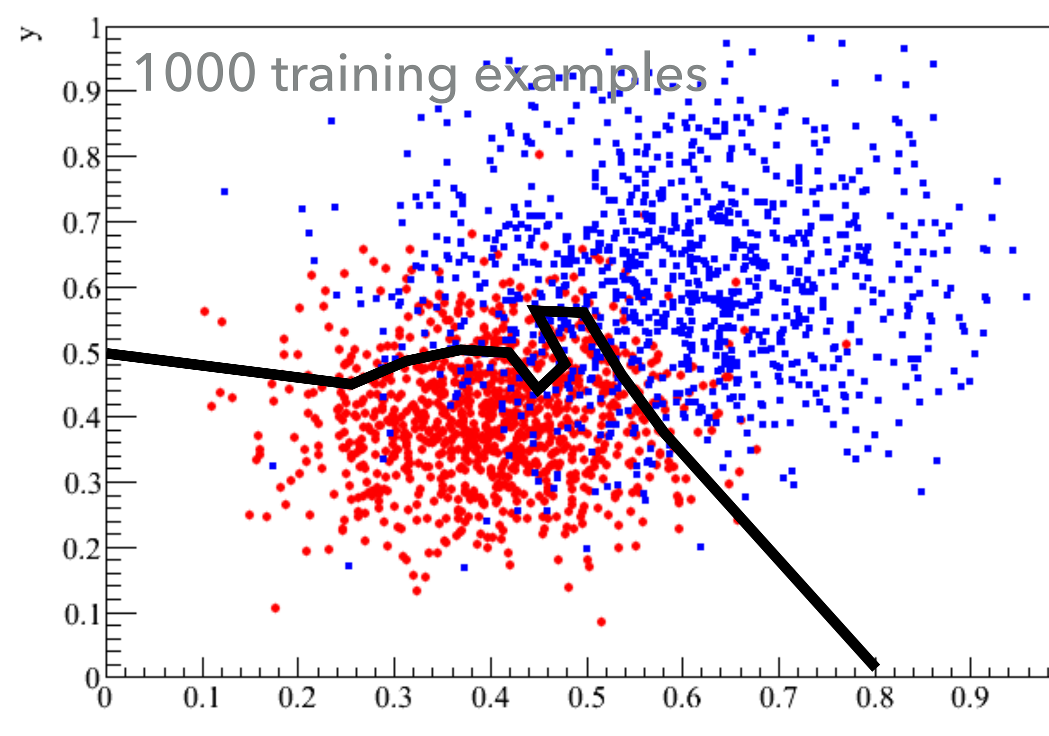
Boundaries like this can be obtained by training models on limited data samples. The accuracies can be impressive.

But would the performance be as good with a new, or a larger data sample?

OVERTRAINING

- A model is over fitted if the HPs that have been determined are tuned to the statistical fluctuations in the data set.
- Simple illustration of the problem:





Increasing to 1000 training examples we can see the boundary doesn't do as well. This illustrates the kind of problem encountered when we overfit HPs of a model.

OVERTRAINING: TRAINING VALIDATION

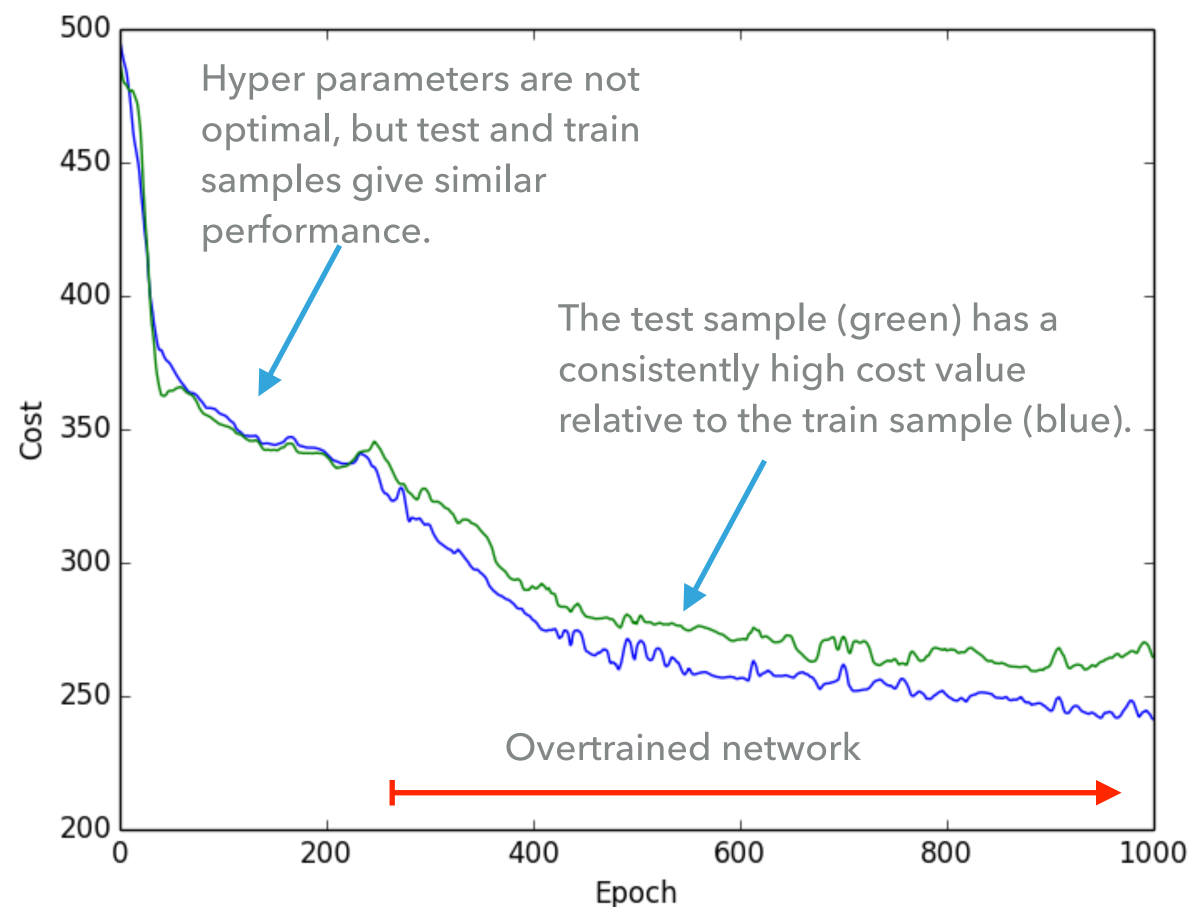
- One way to avoid tuning to statistical fluctuations in the data is to impose a training convergence criteria based on a data sample independent from the training set: a validation sample.
  - Use the cost evaluated for the training and validation samples to check to see if the HPs are over trained.
  - If both samples have similar cost then the model response function is similar on two statistically independent samples.
  - If the samples are large enough then one could reasonably assume that the response function would then be general when applied to an unseen data sample.
- "large enough" is a model and problem dependent constraint.

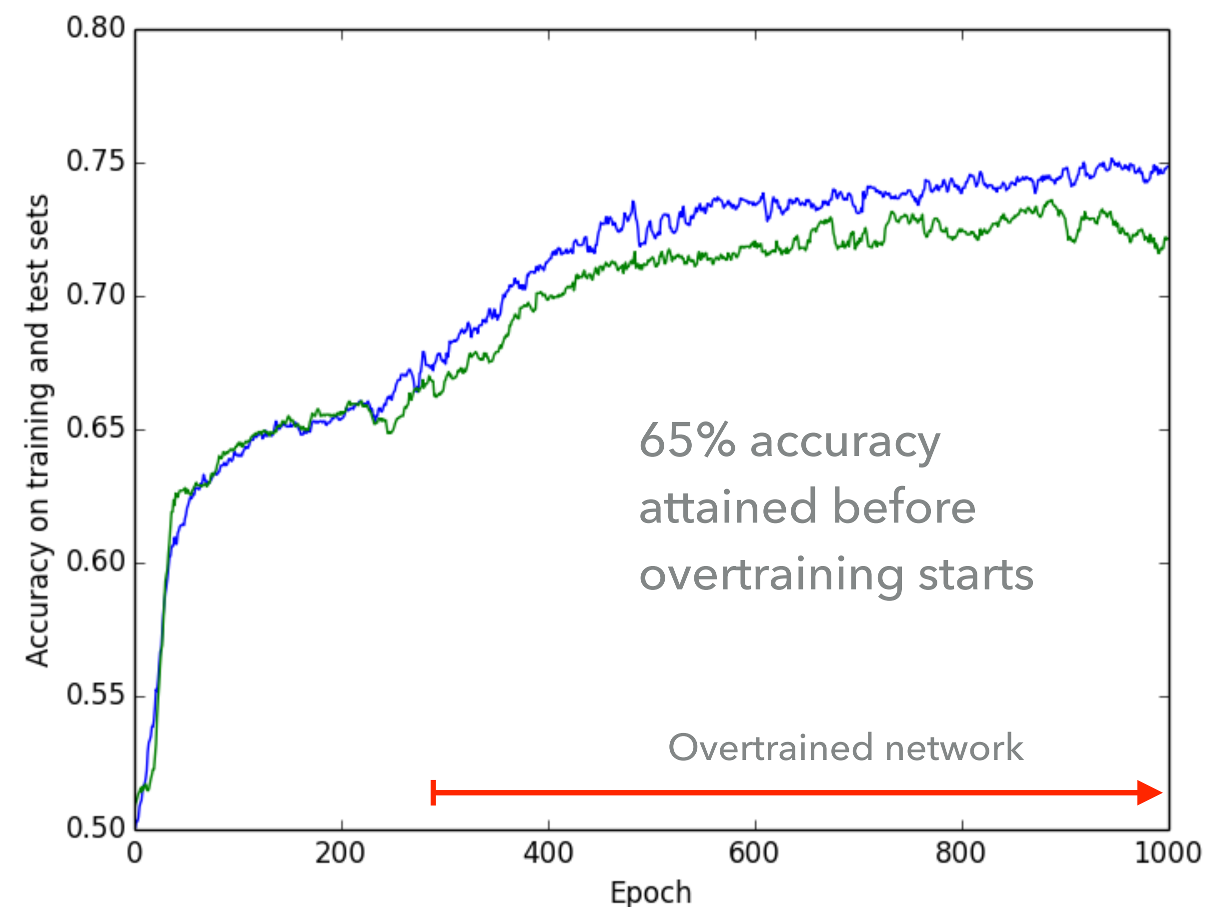
OVERTRAINING: TRAINING VALIDATION

- Training convergence criteria that could be used:
  - Terminate training after N_{epochs}
  - Cost comparison:
    - Evaluate the performance on the training and validation sets.
    - Compare the two and place some threshold on the difference $\Delta cost < \delta_{cost}$
  - Terminate the training when the gradient of the cost function with respect to the weights is below some threshold.
  - Terminate the training when the Δcost starts to increase for the validation sample.

OVERTRAINING: TRAINING VALIDATION: EXAMPLE HIGGS KAGGLE DATA

- This example shows the Higgs Kaggle ($H \rightarrow \tau\tau$) with an overtrained neural network.

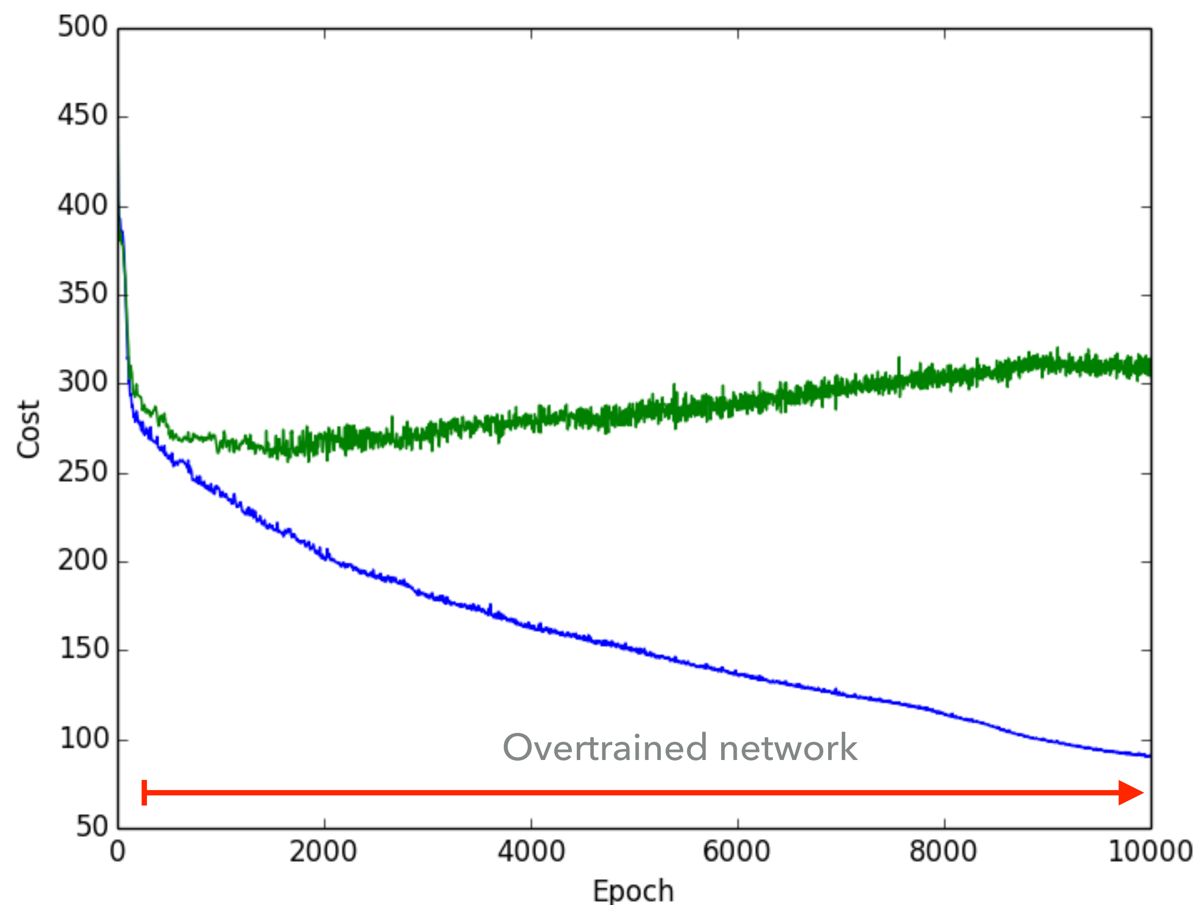


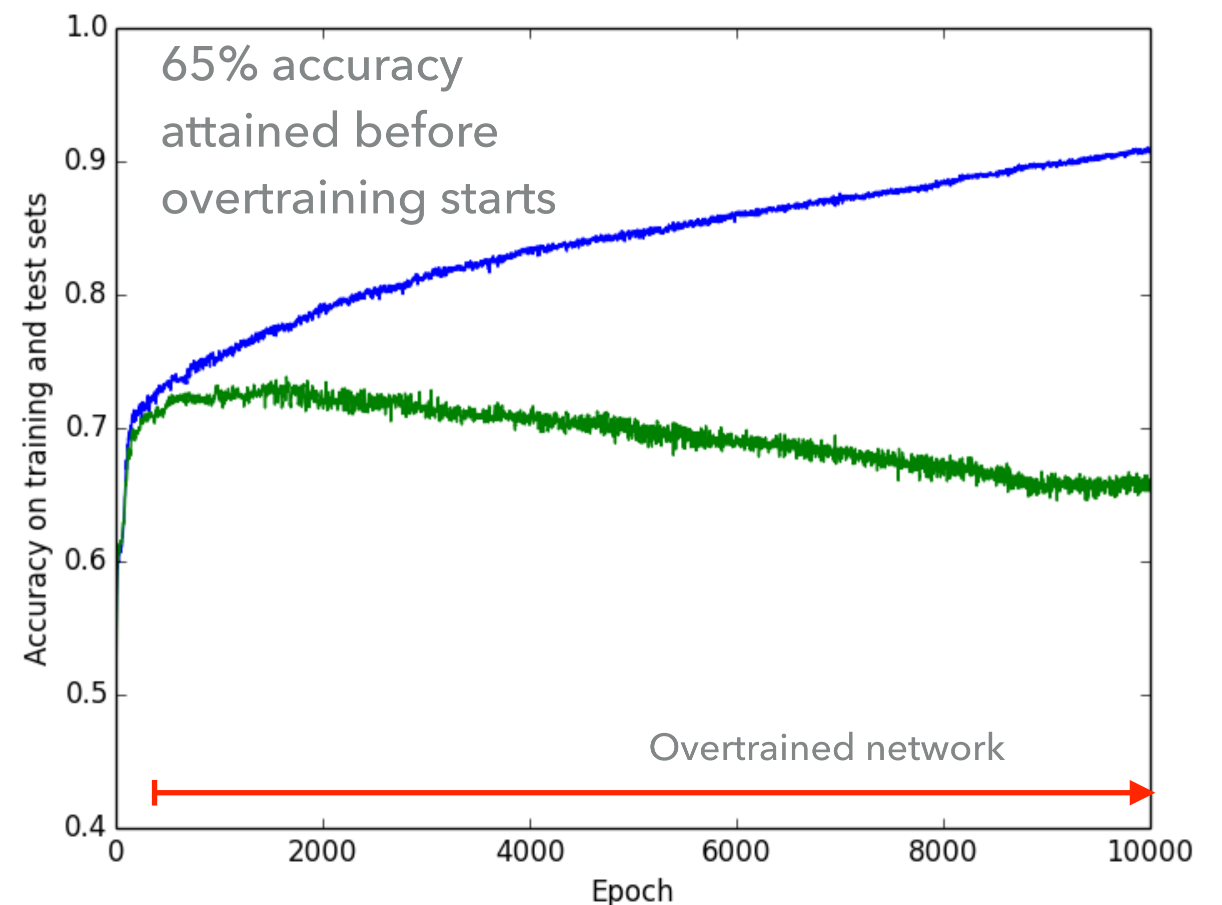


This example uses all features of the data set, but only 2000 test/train events with a learning rate of 0.001 and dropout is not being used. The network has a single layer with 256 nodes and a single output to classify if a given event is signal or background There is no batch training used for this example.

OVERTRAINING: TRAINING VALIDATION: EXAMPLE HIGGS KAGGLE DATA

- Continuing to train an over-trained network does not resolve the issue - the network remains over-trained.





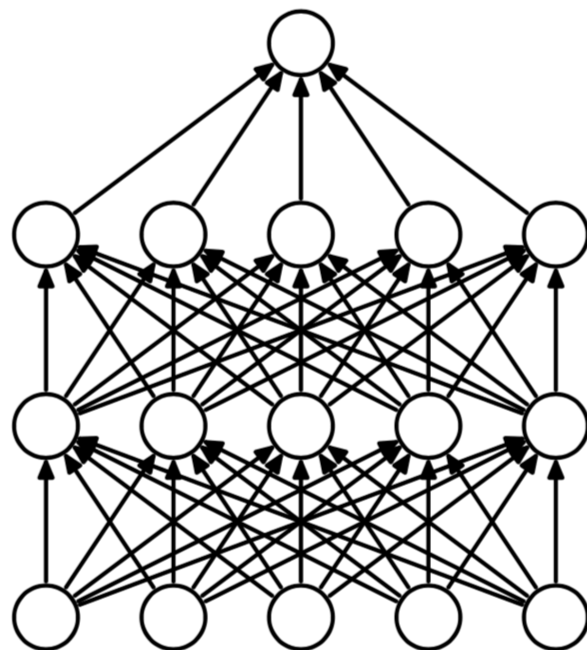
In this case the level of overtraining continues to increase and the difference in the performance of the train and test samples in terms of accuracy increases. After 10000 training cycles we have an accuracy of over 93% for the train sample, but less than 70% for the test sample. This is not a good configuration to use on unseen data as the outcome is unpredictable.

For this example we should terminate after about 200 epochs, while the test/train performance remain similar and have an accuracy of 70%. Other network configurations may be better.

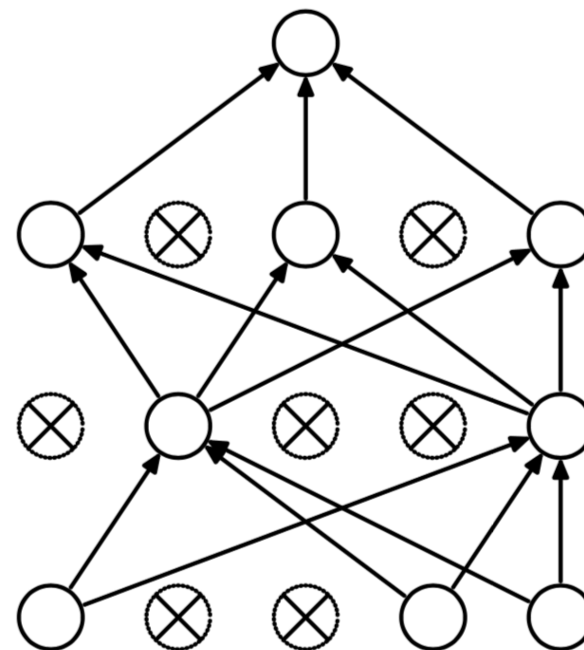
See https://pprc.qmul.ac.uk/~bevan/teaching/PML.html for example code for this problem

OVERTRAINING: DROPOUT FOR DEEP NETWORKS

- A pragmatic way to mitigate overfitting is to compromise the model randomly in different epochs of the training by removing units from the network.



(a) Standard Neural Net



(b) After applying dropout.

Dropout is used during training; when evaluating predictions with the validation or unseen data the full network of Fig. (a) is used.

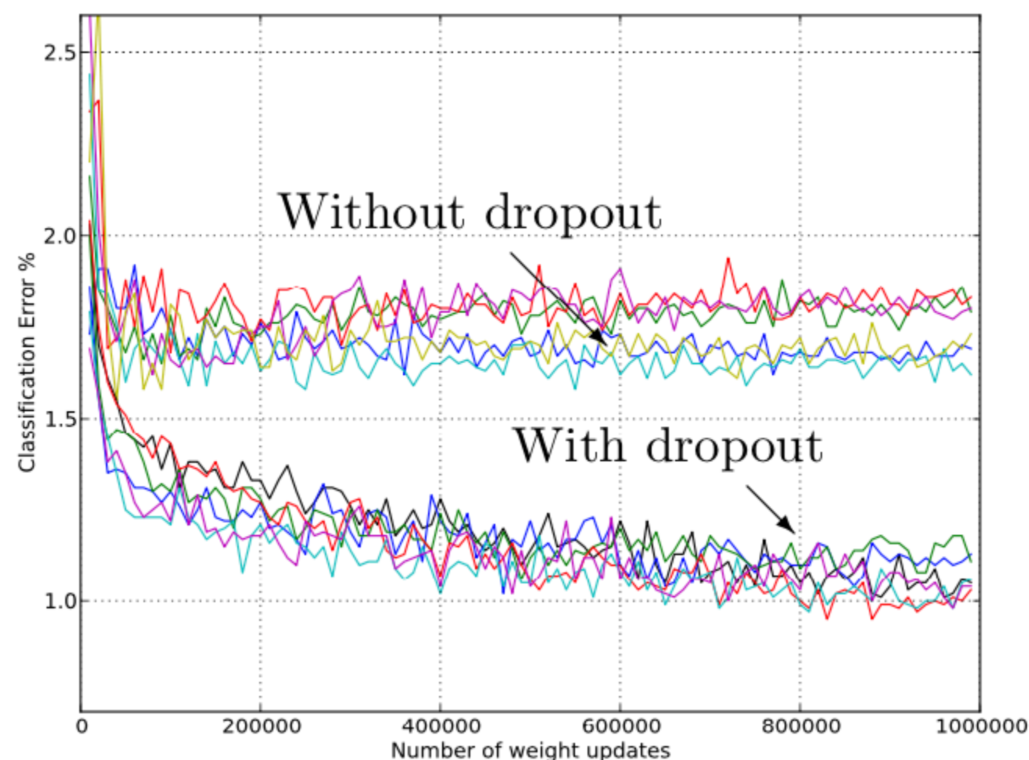
- That way the whole model will be effectively trained on a sub-sample of the data in the hope that the effect of statistical fluctuations will be limited.

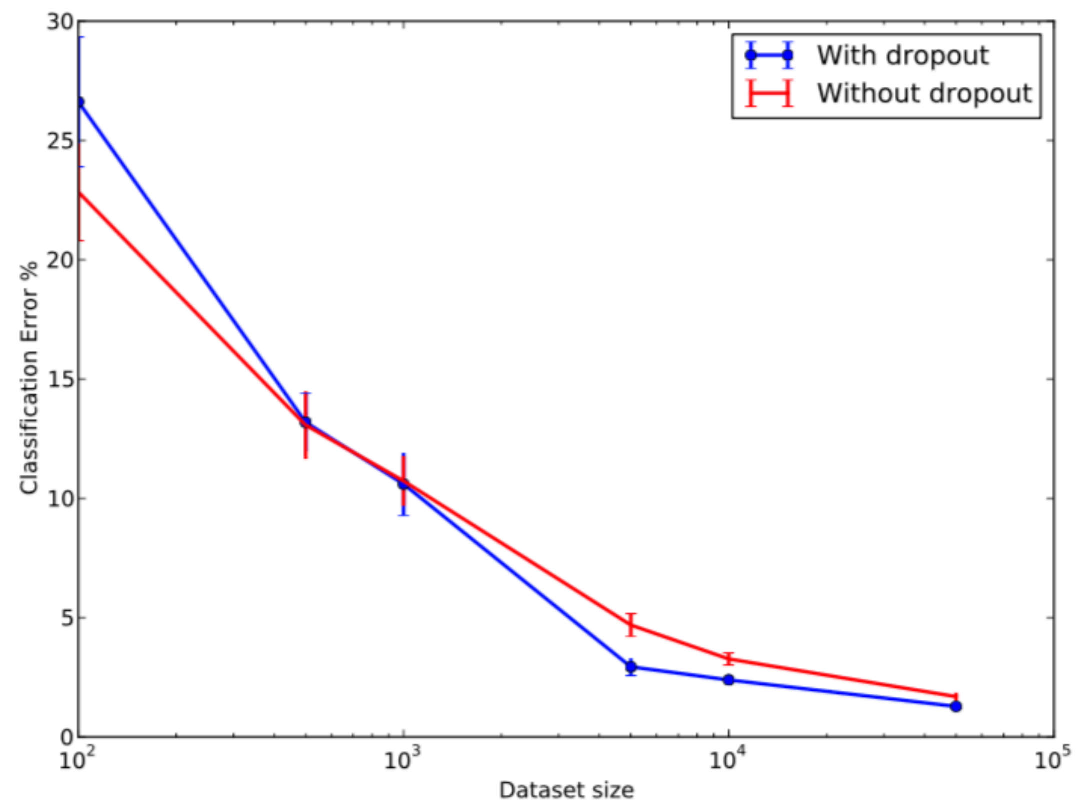
- This does not remove the possibility that a model is overtrained, as with the previous discussion HP generalisation is promoted by using this method.

Srivastava et al., [J. Machine Learning Research 15 (2014) 1929-1958]

OVERTRAINING: DROPOUT FOR DEEP NETWORKS

- A variety of architectures has been explored with different training samples (see Ref [1] for details).



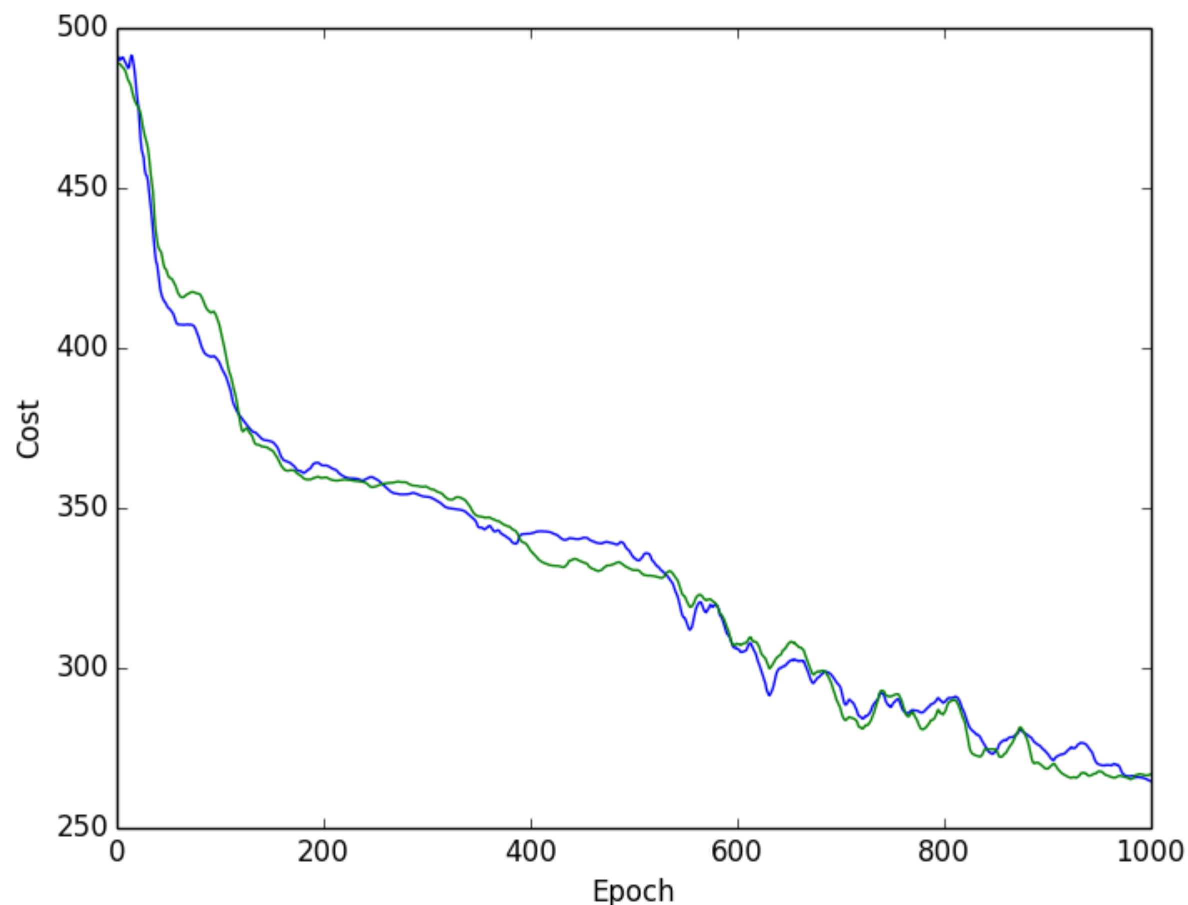


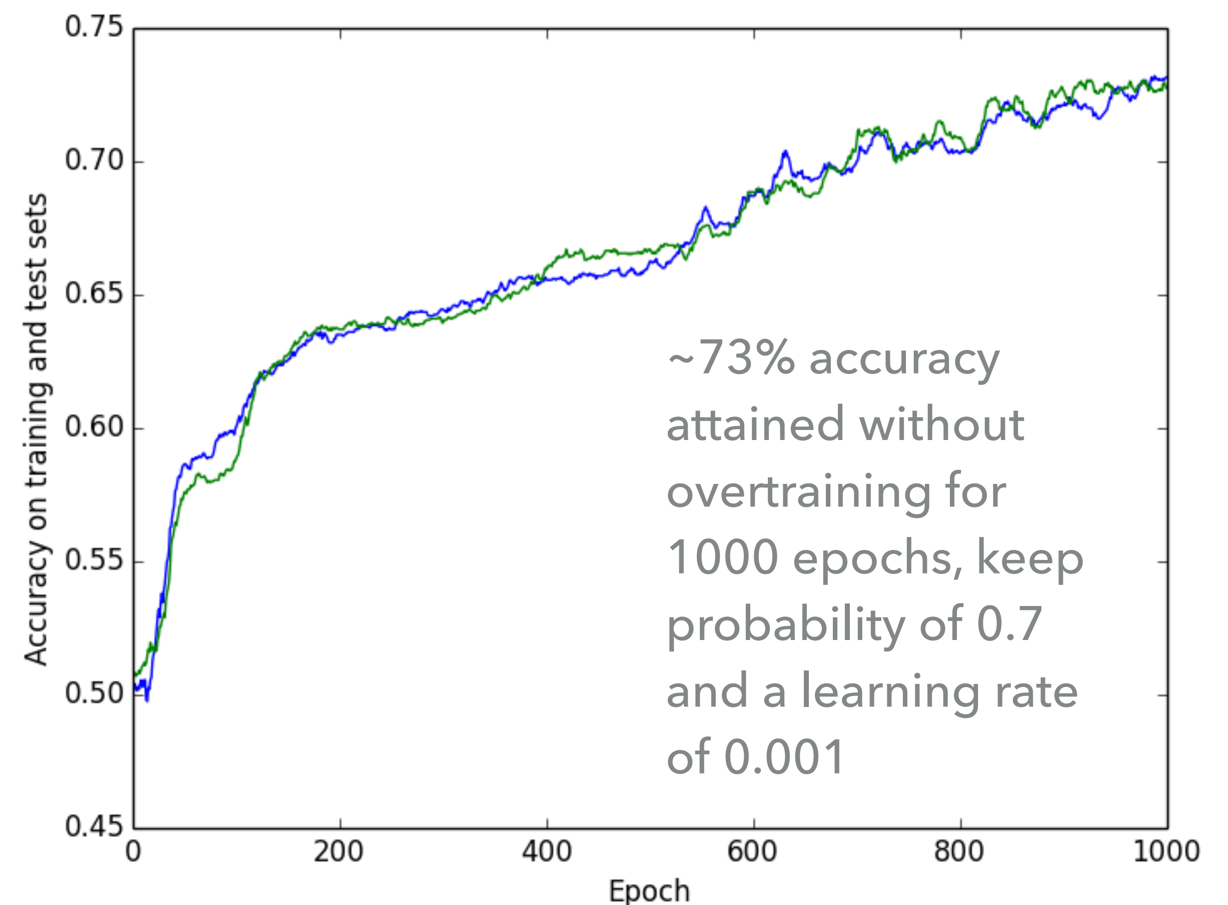
- Dropout can be detrimental for small training samples, however in general the results show that dropout is beneficial.
- For deep networks or typical training samples O(500) examples or more this technique is expected to be beneficial.

[1] Srivastava et al., J. Machine Learning Research 15 (2014) 1929-1958

OVERTRAINING: DROPOUT: EXAMPLE HIGGS KAGGLE DATA

- Changing the drop out keep probability from 0.9 to 0.7 stops the network becoming overtrained in the first 1000 epochs.





- A better accuracy is attained for this network using dropout; above 70%.

See https://pprc.qmul.ac.uk/~bevan/teaching/PML.html for example code for this problem

OVERTRAINING: WEIGHT REGULARISATION FOR NEURAL NETWORKS

- Weight regularisation involves adding a penalty term to the loss function used to optimise the HPs of a network.

- This term is based on the sum of the weights w_i (including bias parameters) in the network and takes the form:

$$\lambda \sum_{i=\forall weights} w_i$$

This is the L1 norm regularisation term.

- The rationale is to add an additional cost term to the optimisation coming from the complexity of the network.

- The performance of the network will vary as a function of λ.

- To optimise a network using weight regularisation it will have to be trained a number of times in order to identify the value corresponding to the min(cost) from the set of trained solutions.

See Ch. 9 of Bishop's Neural Network for Pattern Recognition
Loshchilov, Frank Hutter, arXiv:1711.05101

OVERTRAINING: WEIGHT REGULARISATION FOR NEURAL NETWORKS

- Weight regularisation involves adding a penalty term to the loss function used to optimise the HPs of a network.
- This term is based on the sum of the weights w_i (including bias parameters) in the network and takes the form:

$$\lambda \sum_{i=\forall, weights} w_i^2$$

This variant is the L2 norm regularisation term; also know as weight decay regularisation.

- The rationale is to add an additional cost term to the optimisation coming from the complexity of the network.
- The performance of the network will vary as a function of λ.
- To optimise a network using weight regularisation it will have to be trained a number of times in order to identify the value corresponding to the min(cost) from the set of trained solutions.

See Ch. 9 of Bishop's Neural Network for Pattern Recognition
Loshchilov, Frank Hutter, arXiv:1711.05101

OVERTRAINING: WEIGHT REGULARISATION FOR NEURAL NETWORKS

- For example we can consider extending an MSE cost function to allow for weight regularisation. The MSE cost is given by:

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)^2$$

- To allow for regularisation we add the sum of weights term:

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)^2 + \lambda \sum_{i=\forall, weights} w_i^2$$

- This is a simple modification to make to the NN training process that adds a penalty for the inclusion of non-zero weights in the network.

See Ch. 9 of Bishop's Neural Network for Pattern Recognition
Loshchilov, Frank Hutter, arXiv:1711.05101

OVERTRAINING: WEIGHT REGULARISATION FOR NEURAL NETWORKS

- The optimisation process needs to be run for each value of λ in order to choose the best (least cost) solution that is not overtrained.
- This means we have to train the network many times, with different values of λ when using regularisation. e.g. for the Kaggle example with 1000 epochs we have:

No regularisation; best cost → 0.0

"Best" performance for this sampling of λ; but similar outputs obtained for a range of trainings → 0.2

λ	cost: train / test	accuracy: train / test
0.0	284.9 / 291.2	0.711 / 0.710
0.1	622.9 / 632.2	0.722 / 0.711
0.2	781.5 / 791.8	0.725 / 0.714
0.3	907.3 / 915.8	0.717 / 0.710
1.0	1987.7 / 1986.8	0.714 / 0.712
2.0	3801.1 / 3795.7	0.651 / 0.658
10.0	17555 / 17562	0.674 / 0.666

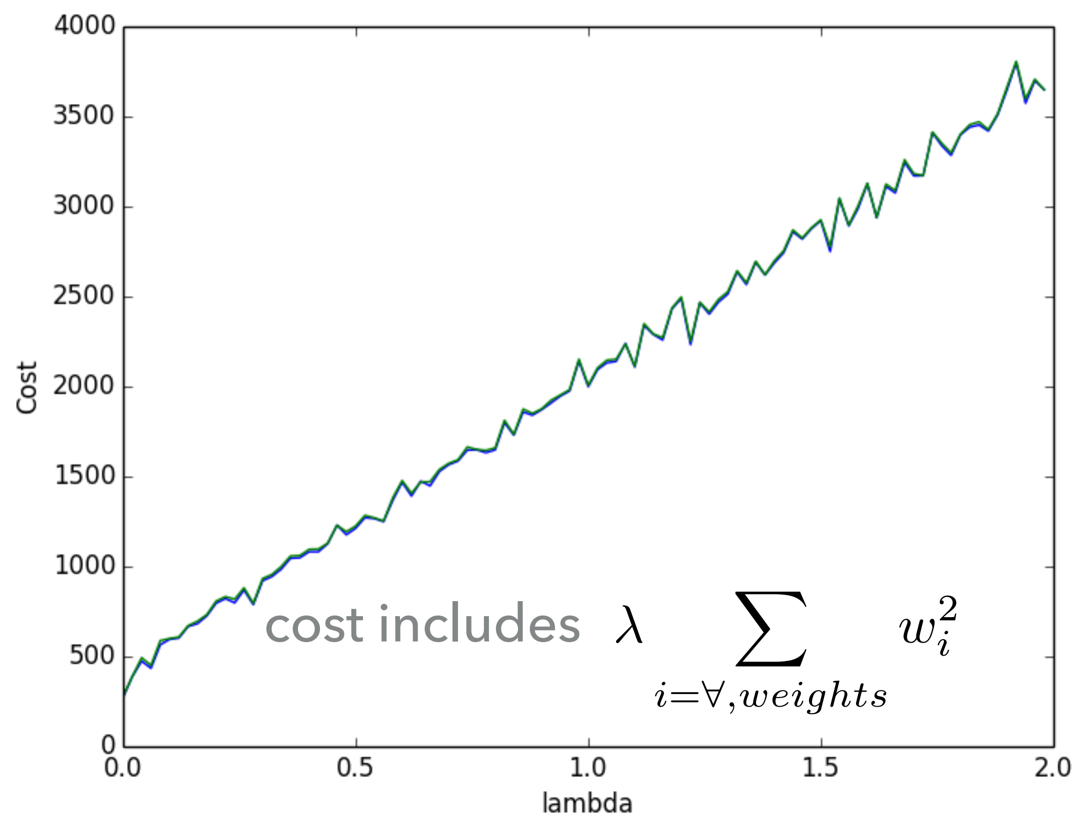
Increasing cost of weights ↓

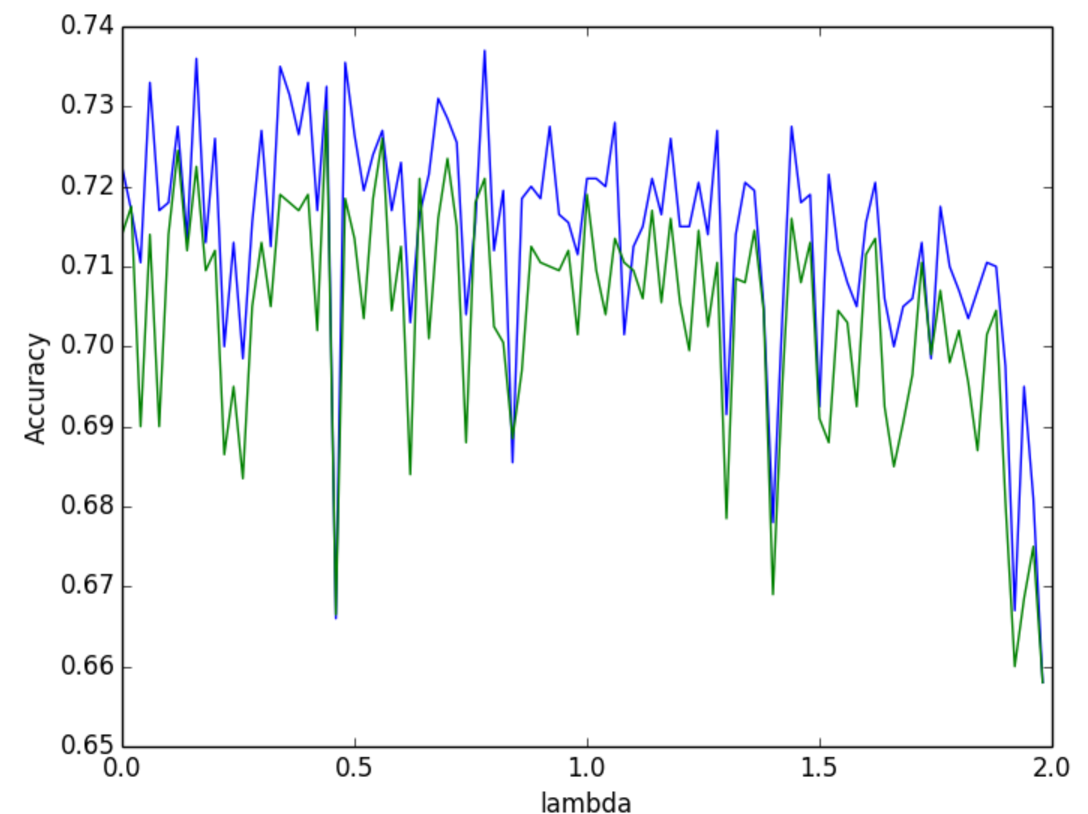
See Ch. 9 of Bishop's Neural Network for Pattern Recognition
Loshchilov, Frank Hutter, arXiv:1711.05101

OVERTRAINING: WEIGHT REGULARISATION FOR NEURAL NETWORKS

- The optimisation process needs to be run for each value of λ in order to choose the best (least cost) solution that is not overtrained.

- This means we have to train the network many times, with different values of λ when using regularisation. e.g. for the Kaggle example with 1000 epochs we have:





- Accuracy falls off as lambda increases, but there is little dependence for this example.

See Ch. 9 of Bishop's Neural Network for Pattern Recognition
Loshchilov, Frank Hutter, arXiv:1711.05101

OVER FITTING: CROSS VALIDATION

- An alternative way of thinking about the problem is to assume that the response function of the model will have some bias and some variance.
  - The bias will be irreducible and mean that the predictions made will have some systematic effect related to the average output value.
  - The variance will depend on the size of the training sample, and we would like to know what this is.

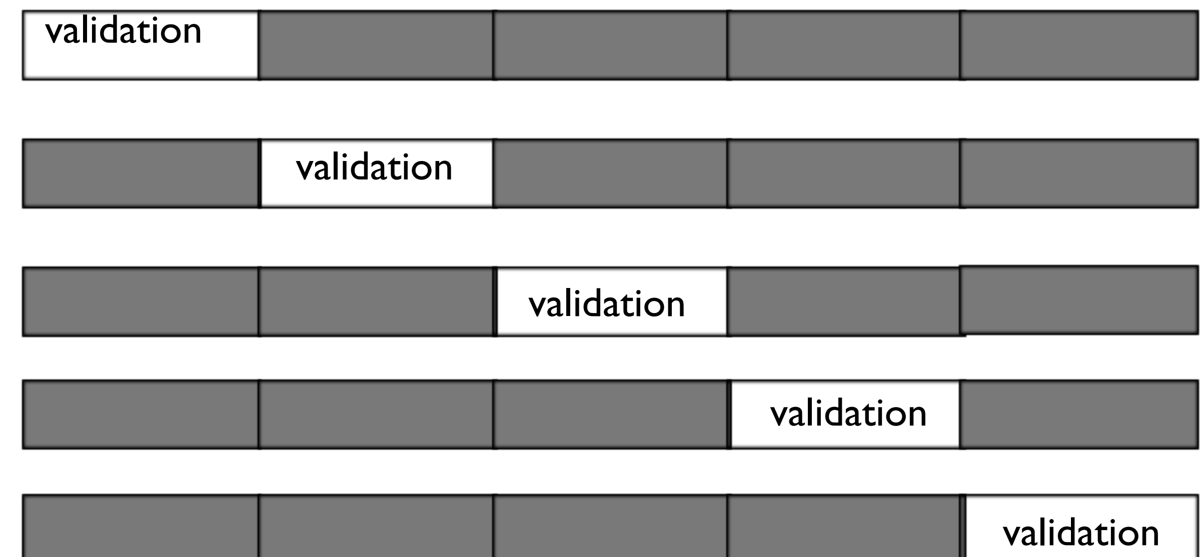
We can use cross validation to estimate the prediction error.

Any prediction bias can be measured using control samples.

Geisser, S. (1975). The predictive sample reuse method with applications. J. Amer. Statist. Assoc., 70:320–328.
For a review of cross validation see: S. Arlot and A. Celisse, Statistics Surveys Vol. 4 4079 (2010).

OVER FITTING: CROSS VALIDATION

- Application of this concept to machine learning can be seen via k-fold cross validation and its variants*
  - Divide the data sample for training and validation into k equal sub-samples.
  - From these one can prepare k sets of validation samples and residual training samples.
  - Each set uses all examples; but the training and validation sub-sets are distinct.

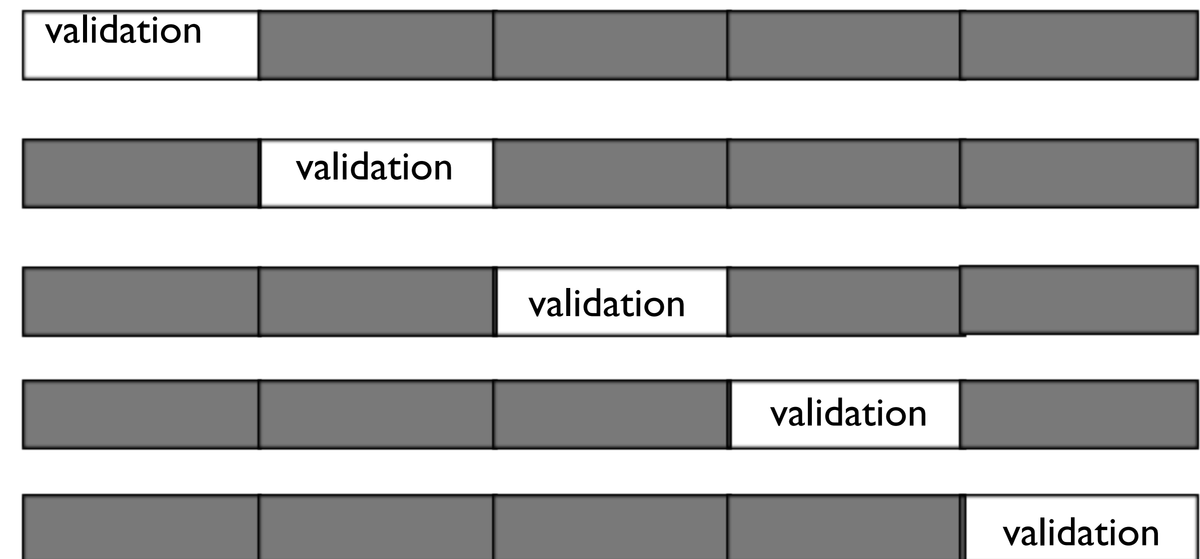


*Variants include the extremes of leave 1 out CV and Hold out CV as well as leave p-out CV. These involve reserving 1 example, 50% of examples and p examples for testing, and the remainder of data for training, respectively.

Geisser, S. (1975). The predictive sample reuse method with applications. J. Amer. Statist. Assoc., 70:320–328.
For a review of cross validation see: S. Arlot and A. Celisse, Statistics Surveys Vol. 4 4079 (2010).

OVER FITTING: CROSS VALIDATION

- Application of this concept to machine learning can be seen via k-fold cross validation and its variants*
  - One can then train the data on each of the k training sets, validating the performance of the network on the corresponding validation set.
  - We can measure the model error on the validation set.
  - The model prediction error is the average error obtained from the k folds on their corresponding validation sets.
  - If desired we could combine the k models to compute some average that would have a prediction performance that is more robust than any of the individual folds.



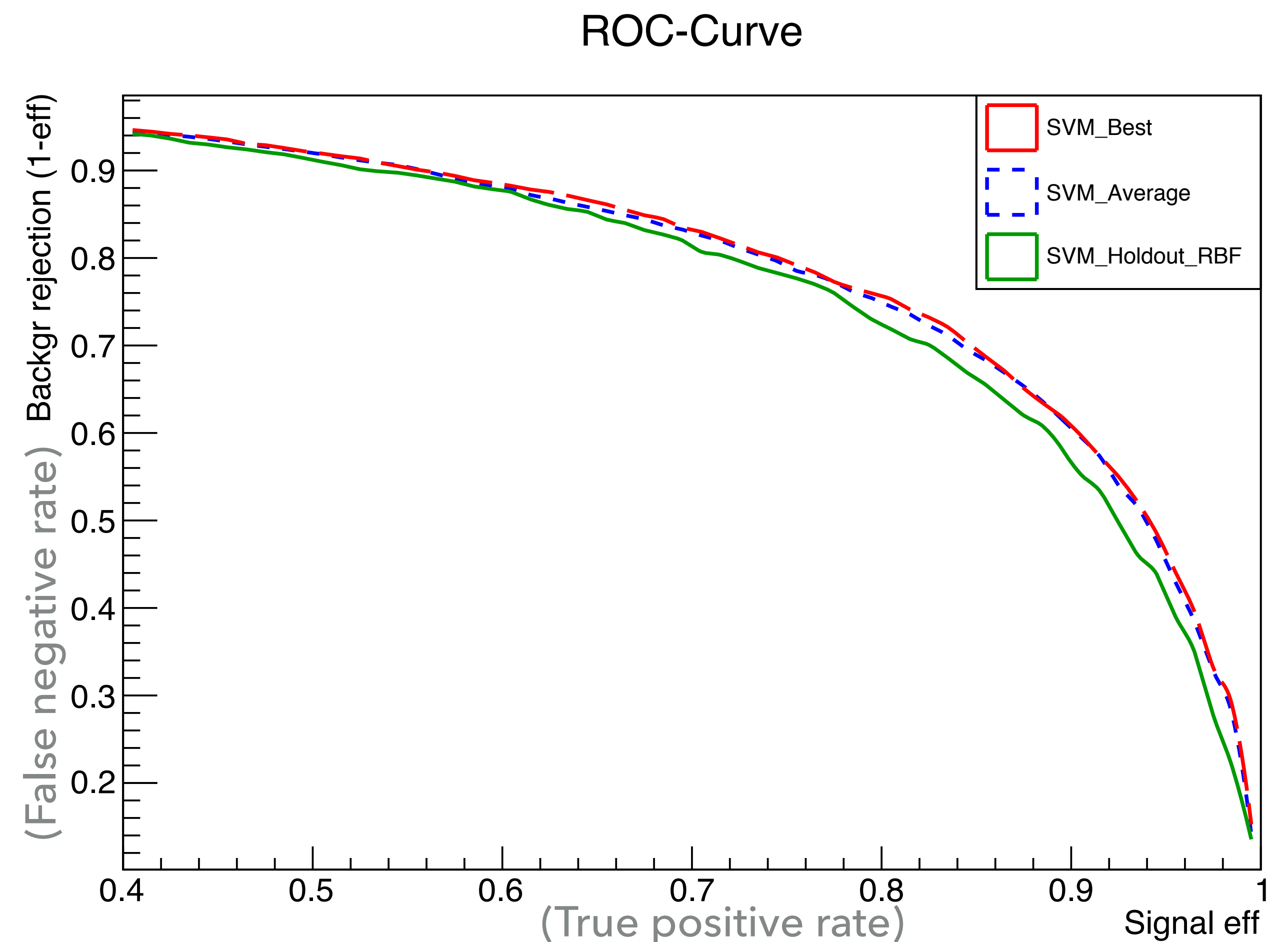
*Variants include the extremes of leave 1 out CV and Hold out CV as well as leave p-out CV. These involve reserving 1 example, 50% of examples and p examples for testing, and the remainder of data for training, respectively.

Geisser, S. (1975). The predictive sample reuse method with applications. J. Amer. Statist. Assoc., 70:320–328.
For a review of cross validation see: S. Arlot and A. Celisse, Statistics Surveys Vol. 4 4079 (2010).

OVER FITTING: CROSS VALIDATION

- Application of this concept to machine learning can be seen via k-fold cross validation and its variants*

- The ensemble of response function outputs will vary in analogy with the spread of a Gaussian distribution.
- This results in family of ROC curves; with a representative performance that is neither the best or worst ROC.
- The example shown is for a Support Vector Machine, with the best average and holdout ROC curves to indicate some sense of spread.



*Variants include the extremes of leave 1 out CV and Hold out CV as well as leave p-out CV. These involve reserving 1 example, 50% of examples and p examples for testing, and the remainder of data for training, respectively.

Geisser, S. (1975). The predictive sample reuse method with applications. J. Amer. Statist. Assoc., 70:320–328.
For a review of cross validation see: S. Arlot and A. Celisse, Statistics Surveys Vol. 4 4079 (2010).

SUMMARY

- Hyper-parameter optimisation does not guarantee robust solution.
  - Minimisation of different cost functions will allow us to determine "optimial" hyper-parameters for our model.
  - When the model starts to learn statistical fluctuations in the training sample we should stop the training to avoid overtraining the data (i.e. to avoid learning the statistical fluctuations in a given sample).
- Methods exist to mitigate overtraining, however none of them guarantee a generalised solution that is robust from overtraining [dropout, regularisation, cross validation].
    - The only exception to this is to supply an effectively infinite sample of training data for a given problem.

SUGGESTED READING (HEP RELATED)

- There is not much HEP-specific literature regarding HPs optimisation.
  - Minuit is used to optimise parameters in TMVA along with other algorithms. See the Minuit user guide and TMVA for more details. This is based on the DFP method of variable metric minimisation.
  - Logarithmic grid searches are also discussed for specific algorithms with a few HPs (see SVM notes).

SUGGESTED READING (NON-HEP)

- There are a few backup slides on the ADAM optimiser that performs well with a number of deep learning problems that you might wish to look at.
- In addition the following books discuss the issue of parameter optimisation:
  - Bishop, Neural Networks for Pattern Recognition, Oxford University Press (2013)
  - Cristianini and Shawe-Taylor, Support Vector Machines and other kernel-based learning methods, Cambridge University Press (2014)
  - Goodfellow, Deep Learning, MIT Press (2016)
  - MacKay's Information Theory, Inference and Learning algorithms, Cambridge University Press (2011)

APPENDIX

- The following slides contain additional information that may be of interest.

MULTIPLE MINIMA

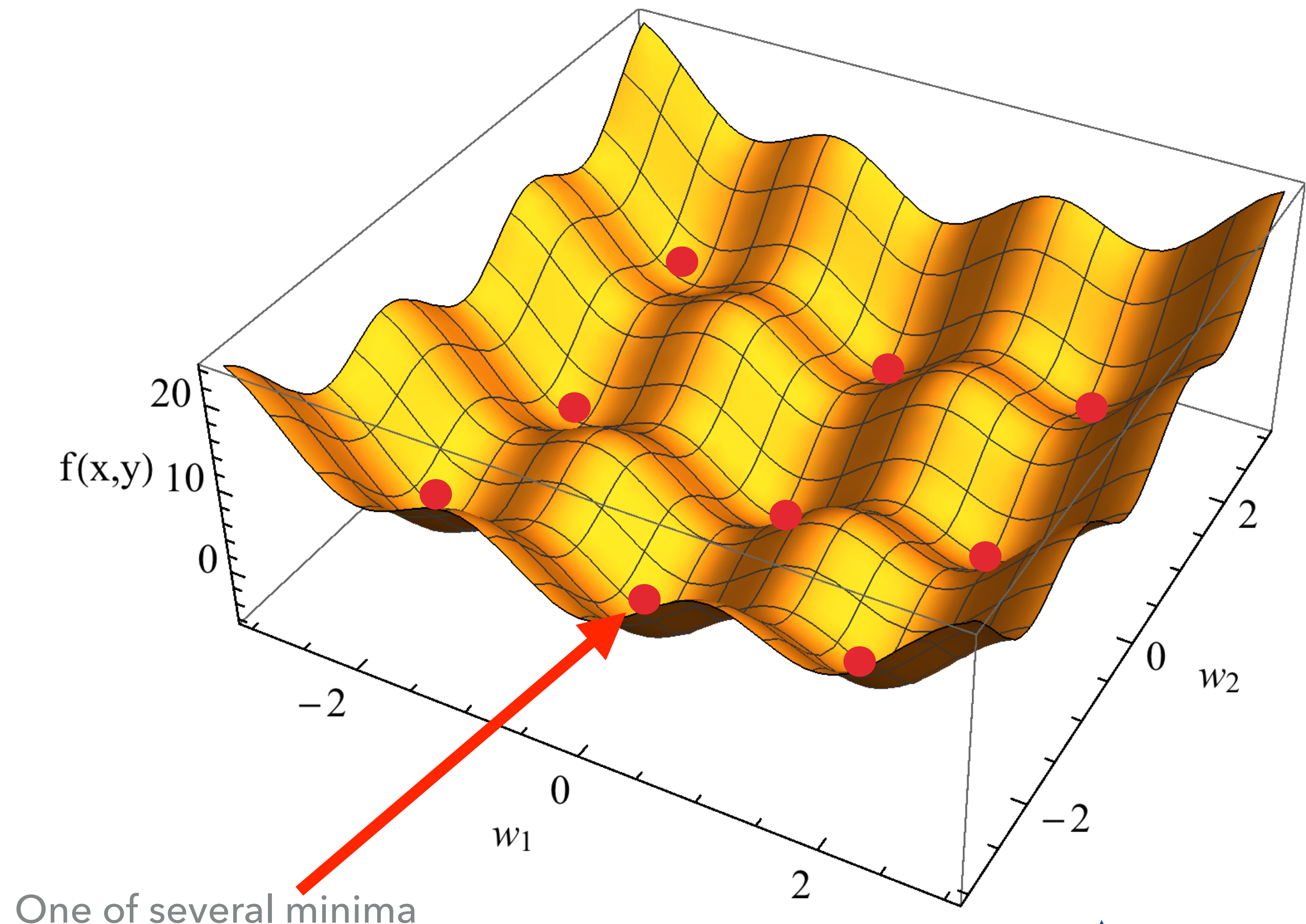
- Often more complication hyperspace optimisation problems are encountered, where there are multiple minima.

The gradient descent minimisation algorithm is based on the assumption that there is a single minimum to be found.

In reality there are often multiple minima.

Sometimes the minima are degenerate, or near degenerate.

How do we know we have converged on the global minimum?



One of several minima

MULTIPLE MINIMA

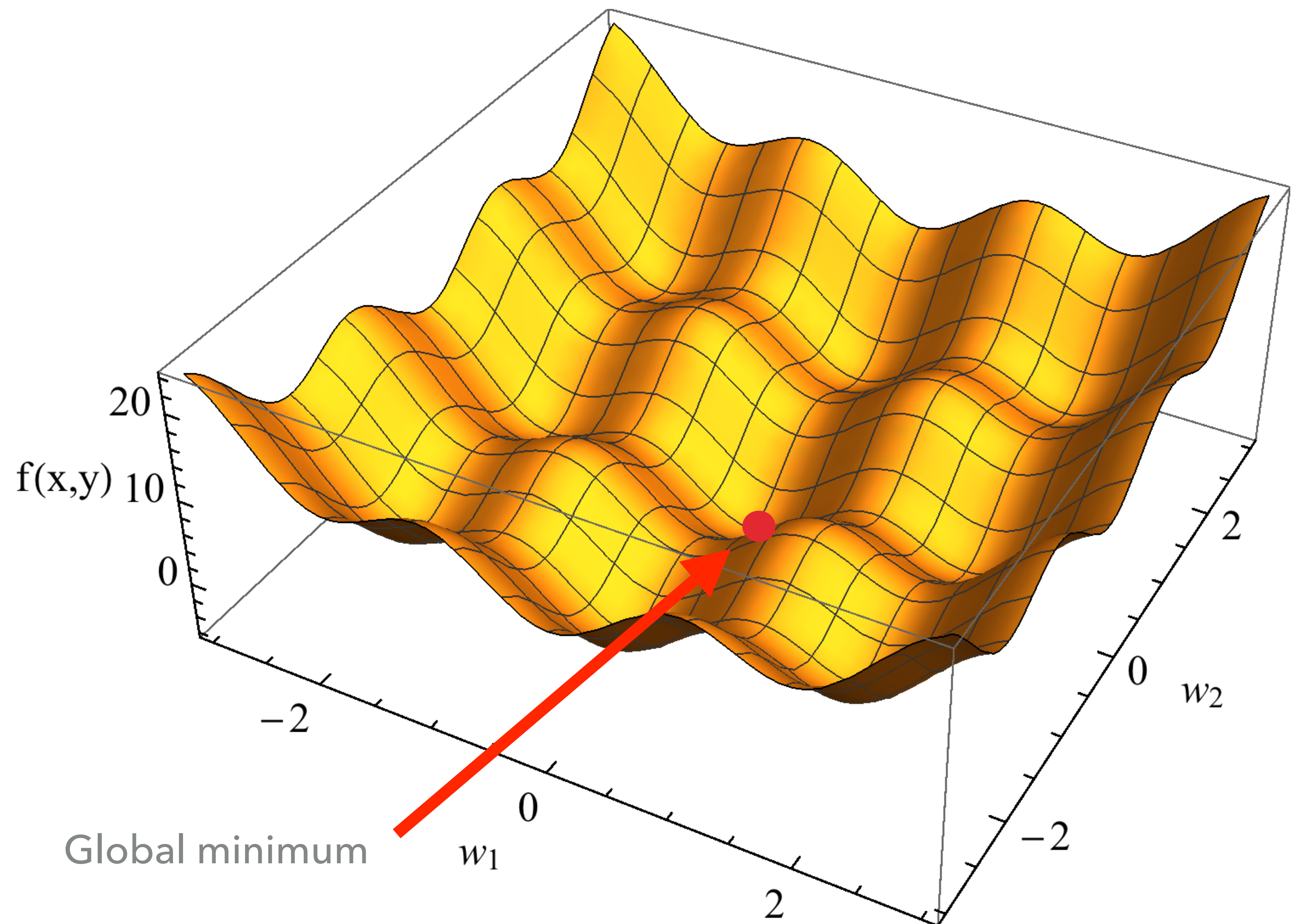
- Often more complication hyperspace optimisation problems are encountered, where there are multiple minima.

The gradient descent minimisation algorithm is based on the assumption that there is a single minimum to be found.

In reality there are often multiple minima.

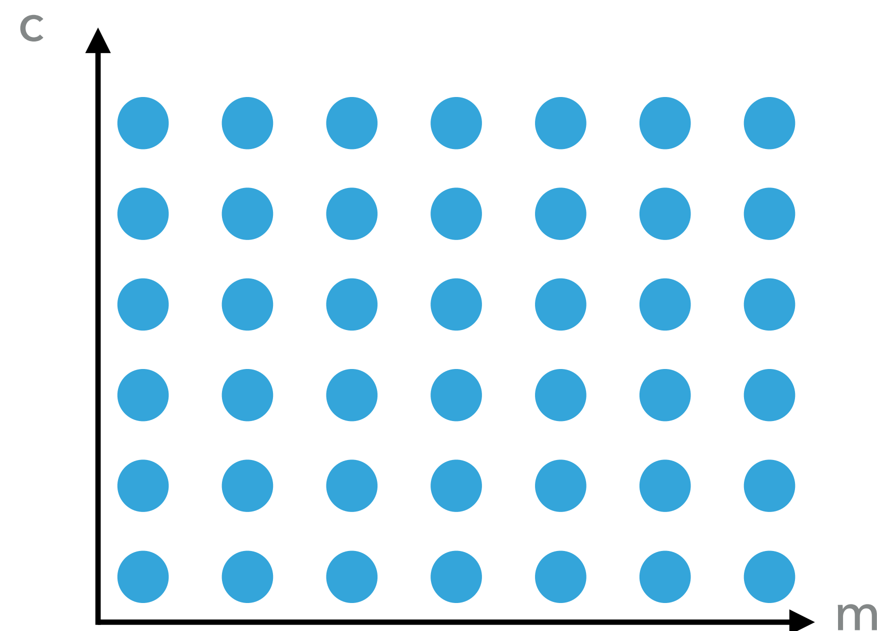
Sometimes the minima are degenerate, or near degenerate.

How do we know we have converged on the global minimum?



GRID SEARCHES

- Just as we can scan through a parameter in order to minimise a likelihood ratio, we can scan through a HP to observe how the loss function changes.
- For simple models we can construct a 2D grid of points in m and c.
- Evaluating the loss function for each point in the 2D sample space we can construct a grid from which to select the minimum value.
- The assumption here is that our grid spacing is sufficient for the purpose of optimising the problem.

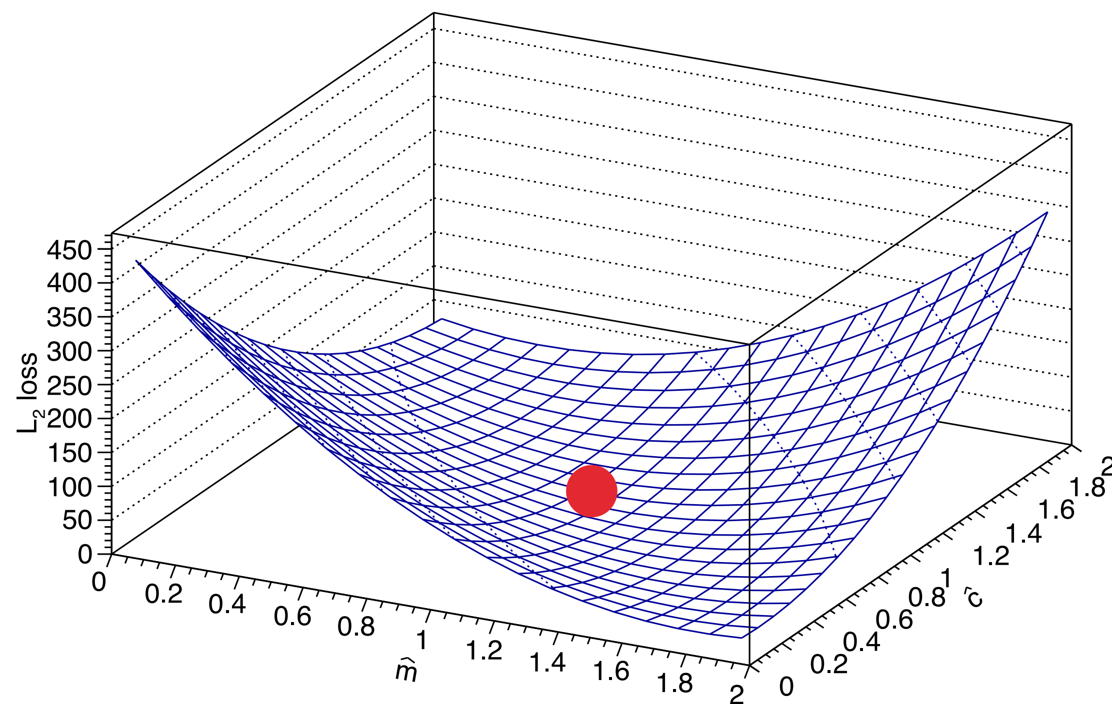


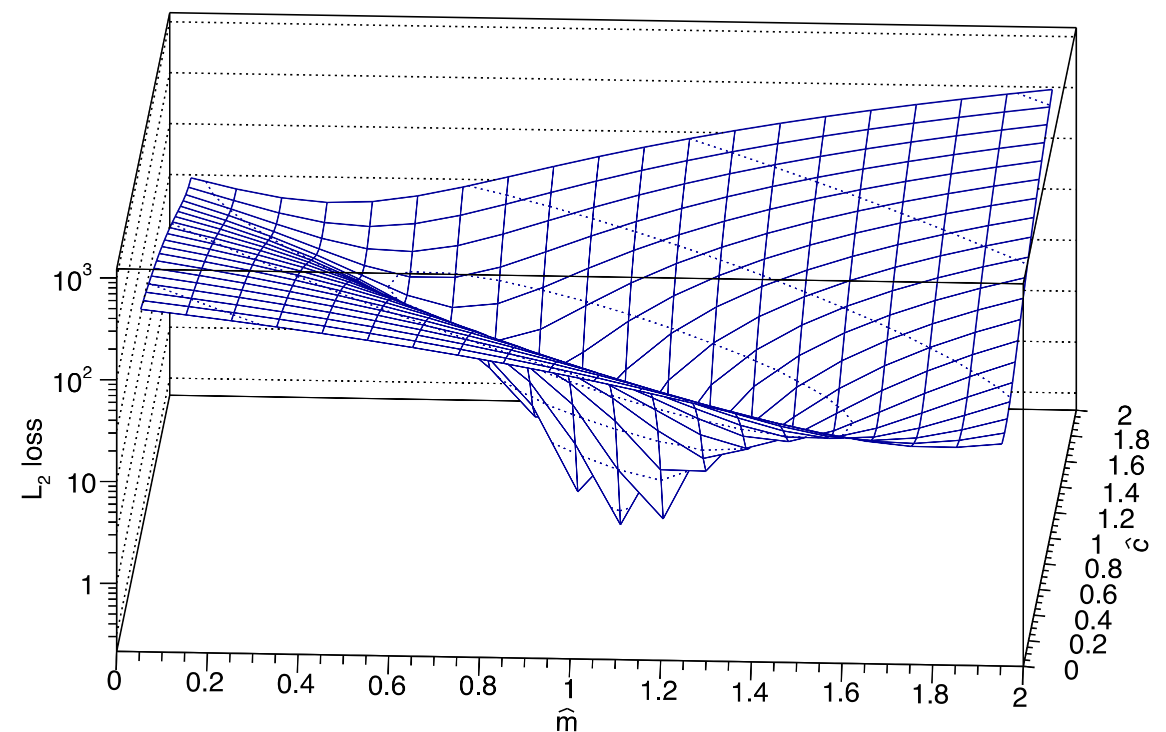
This type of parameter search is often used for support vector machine HPs (kernel function parameters and cost).

This method does not scale to large numbers of parameters.

GRID SEARCHES

- e.g. consider a linear regression study optimising the parameters for the model y=mx+c
- The loss function for this problem results in a "valley" as m and c are anti-correlated parameters in this 2D hyperspace.





- The contours of the loss function show a minimum, but this is selected from a discrete grid of points (need to ensure grid spacing is sufficient for your needs).

ADAM OPTIMISER

- This is a stochastic gradient descent algorithm.
- Consider a model $f(\theta)$ that is differentiable with respect to the HPs θ so that:
  - the gradient $g_t = \nabla f_t(\theta_{t-1})$ can be computed.
  - t is the training epoch
  - m_t and v_t are biased values of the first and second moment
  - m_t-hat and v_t-hat are bias corrected estimator of the moments
  - Some initial guess for the HP is taken: θ_0, and the HPs for a given epoch are denoted by θ_t
  - α is the step size
  - β_1 and β_2 are exponential decay rates of moving averages.

Kingma and Ba, Proc. 3rd Int Conf. Learning Representations, San Diego, 2015

ADAM OPTIMISER

- ADAptive Moment estimation based on gradient descent.

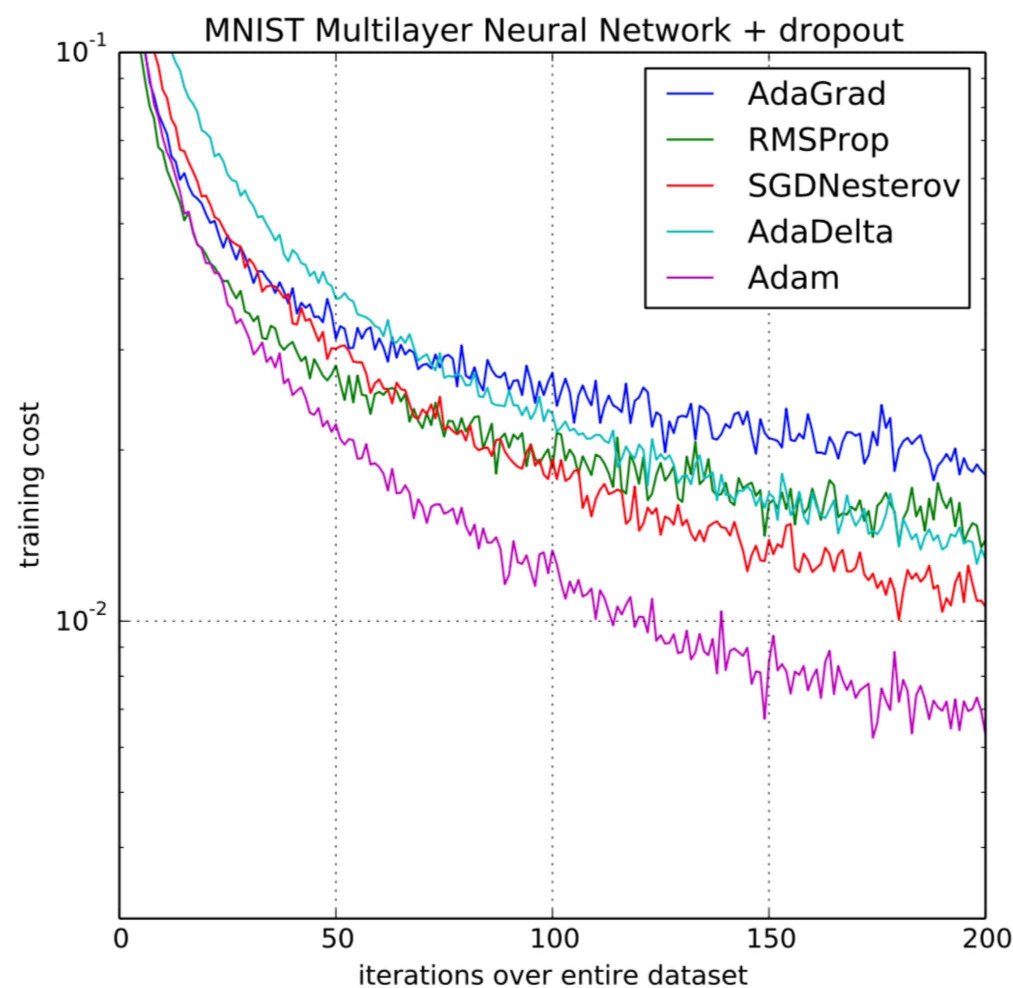
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

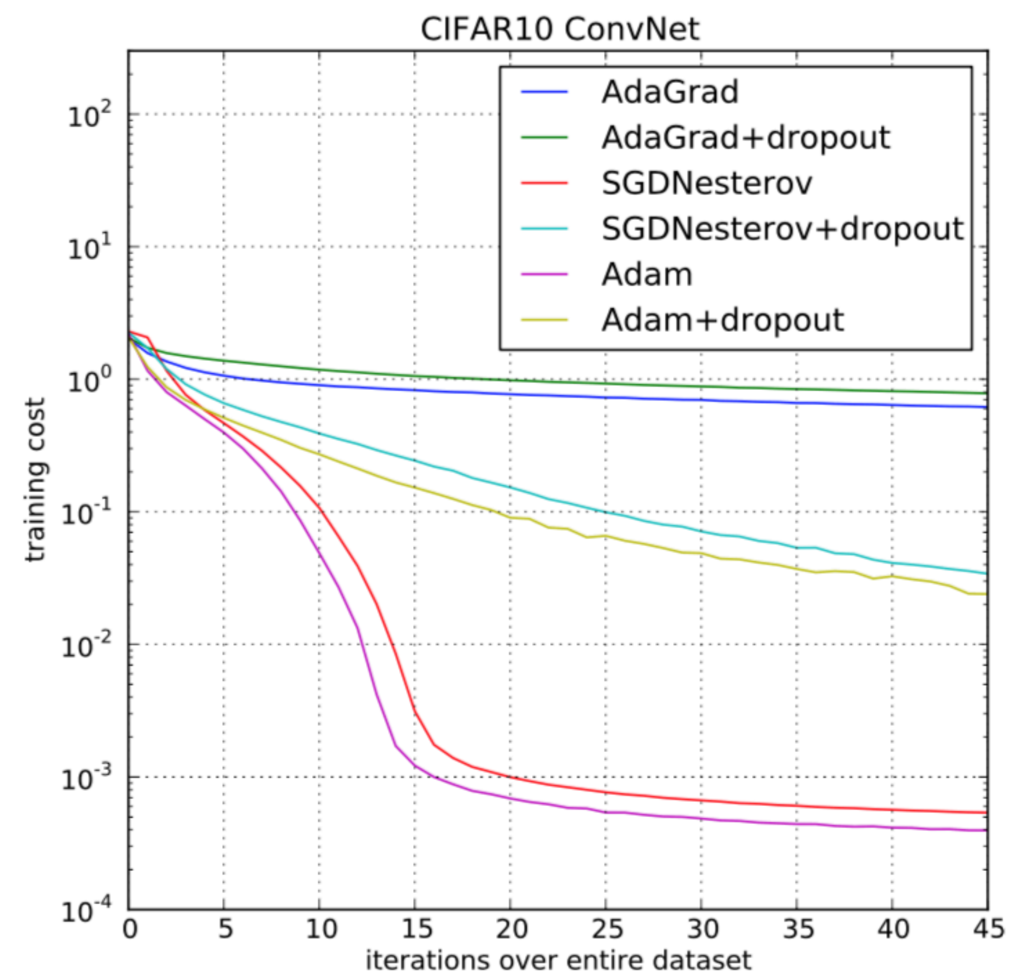
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Kingma and Ba, Proc. 3rd Int Conf. Learning Representations, San Diego, 2015

ADAM OPTIMISER

- Benchmarking performance using MNIST and CFAR10 data indicates that Adam with dropout minimises the loss function compared with other optimisers tested.





- Faster drop off in cost, and lower overall cost obtained.

Kingma and Ba, [Proc. 3rd Int Conf. Learning Representations, San Diego, 2015]

DAVIDON-FLETCHER-POWELL

- This is a variable metric minimisation algorithm (1959, 1963) is described in wikipedia as:

> The DFP formula is quite effective, but it was soon superseded by the BFGS formula, which is its dual (interchanging the roles of y and s).

- This is used in high energy physics via the package MINUIT (FORTRAN) and Minuit2 (C++) implementations.
  - The standard tools that are used for data analysis in HEP have these implementations available, and while the algorithm may no longer be optimal, it is still deemed good enough by many for the optimisation tasks.
    - Robust / reliable minimisation.
    - Underlying method derived assuming parabolic minima
    - Understood and trusted by the HEP community.

https://ntrs.nasa.gov/search.jsp?R=19760017876
https://www.osti.gov/servlets/purl/4222000

DAVIDON-FLETCHER-POWELL

- This is a variable metric minimisation algorithm (1959, 1963) is described in wikipedia as:

> The DFP formula is quite effective, but it was soon superseded by the BFGS formula, which is its dual (interchanging the roles of y and s).

- This is used in high energy physics via the package MINUIT (FORTRAN) and Minuit2 (C++) implementations.

This is mentioned **only** because it is the dominant algorithm used in particle physics at this time.

Almost all minimisation problems are solved using this algorithm, where the dominant use is for maximum likelihood and χ^2 fit minimisation problems.

The number of HPs required to solve those problems is small (up to a few hundred) in comparison with the numbers required for neural networks (esp. deep learning problems).

If you don't (intend to) work in this field, you can now forget you heard about this algorithm.

https://ntrs.nasa.gov/search.jsp?R=19760017876
https://www.osti.gov/servlets/purl/4222000