Queen Mary
University of London

ADRIAN BEVAN

GRADNET MACHINE LEARNING AND AI WORKSHOP: INTRODUCTION TO PYTHON AND TENSOFLOW

NEURAL NETWORKS (NNs)

LECTURE PLAN

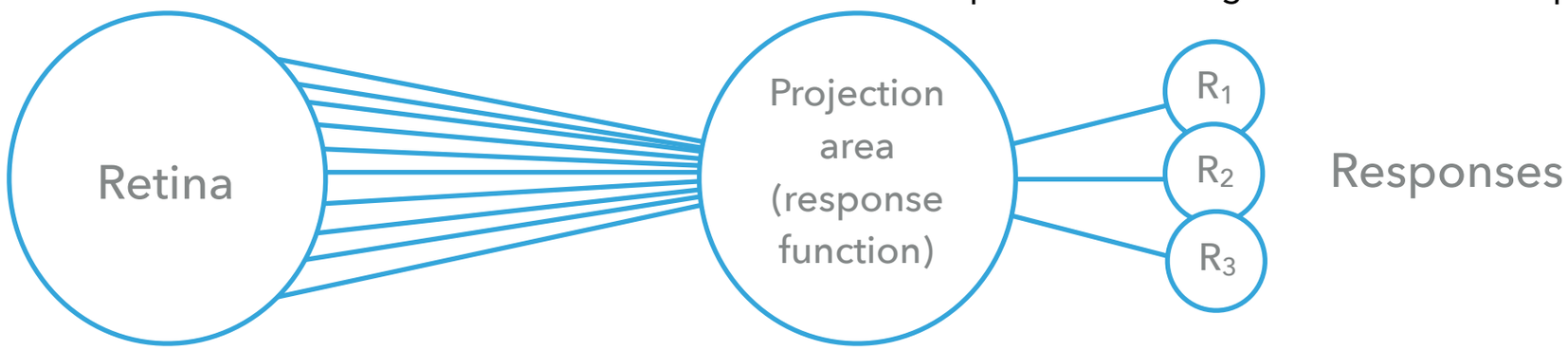
- Perceptrons
- Activation functions
- Artificial Neural Network
- Multilayer Perceptrons
- Training
- Summary

Note that this has been written to support a TensorFlow-based tutorial, and where appropriate there are some TensorFlow related remarks made.

PERCEPTRONS

- Rosenblatt[1] coined the concept of a perceptron as a probabilistic model for information storage and organisation in the brain.
  - Origins in trying to understand how information from the retina is processed.

Simplified view of Fig 1 from Rosenblatt's paper.



  - Start with inputs from different cells.
  - Process those data: "if the sum of excitatory or inhibitory impulse intensities is either equal to or greater than the threshold (θ) … then the A unit fires".
- This is an all or nothing response-based system.

[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

PERCEPTRONS

- This picture can be generalised as follows:
  - Take some number, n, of input features
  - Compute the sum of each of the features multiplied by some factor assigned to it to indicate the importance of that information.
  - Compare the sum against some reference threshold.
  - Give a positive output above some threshold.

[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

PERCEPTRONS

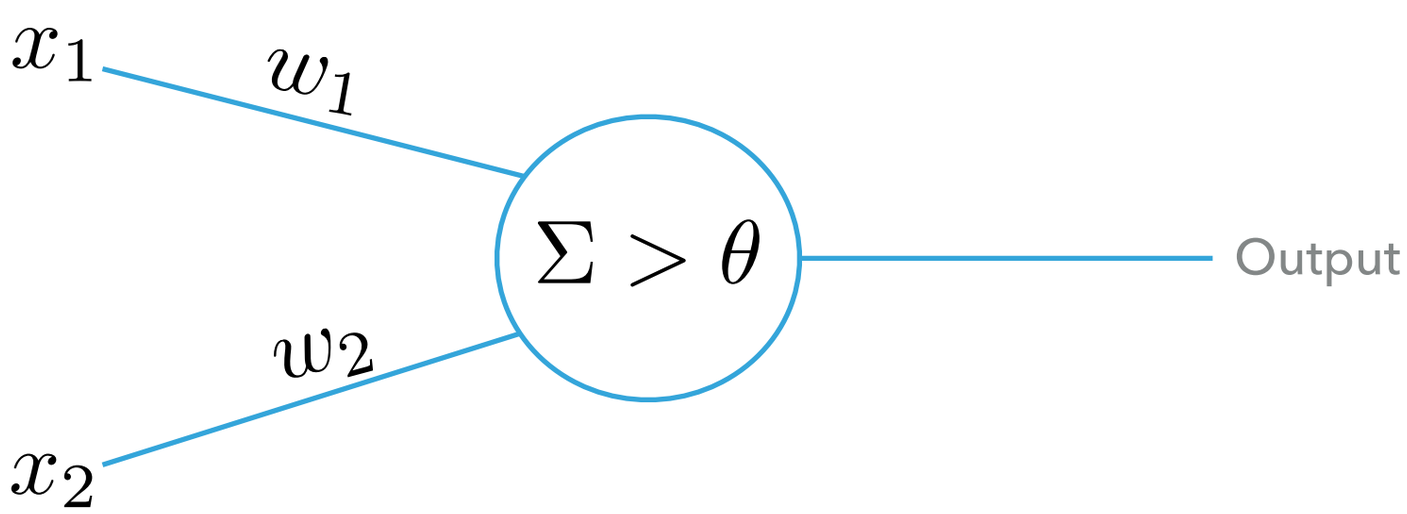
- Illustrative example:
  - Consider a measurement of two quantities x_1, and x_2.
  - Based on these measurements we determine if the perceptron is to give an output (value = 1) or not (value = 0).

$$\begin{matrix} w_1 x_1 \\ + \\ w_2 x_2 \end{matrix} \quad = \quad \begin{cases} 0 \\ 1 \end{cases}$$

[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

PERCEPTRONS

- Illustrative example:
  - Consider a measurement of two quantities x_1, and x_2.
  - Based on these measurements we determine if the perceptron is to give an output (value = 1) or not (value = 0).



[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

PERCEPTRONS

- Illustrative example:
  - Consider a measurement of two quantities x_1, and x_2.
  - Based on these measurements we determine if the perceptron is to give an output (value = 1) or not (value = 0).

If $w_1 x_1 + w_2 x_2 > \theta$

Output = 1

else

Output = 0

[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

PERCEPTRONS

- Illustrative example:
  - Consider a measurement of two quantities x_1, and x_2.
  - Based on these measurements we determine if the perceptron is to give an output (value = 1) or not (value = 0).

If $w_1 x_1 + w_2 x_2 > \theta$

Output = 1

else

Output = 0

This is called a binary activation function

[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

PERCEPTRONS

- Illustrative example:

- Decision is made on x_2

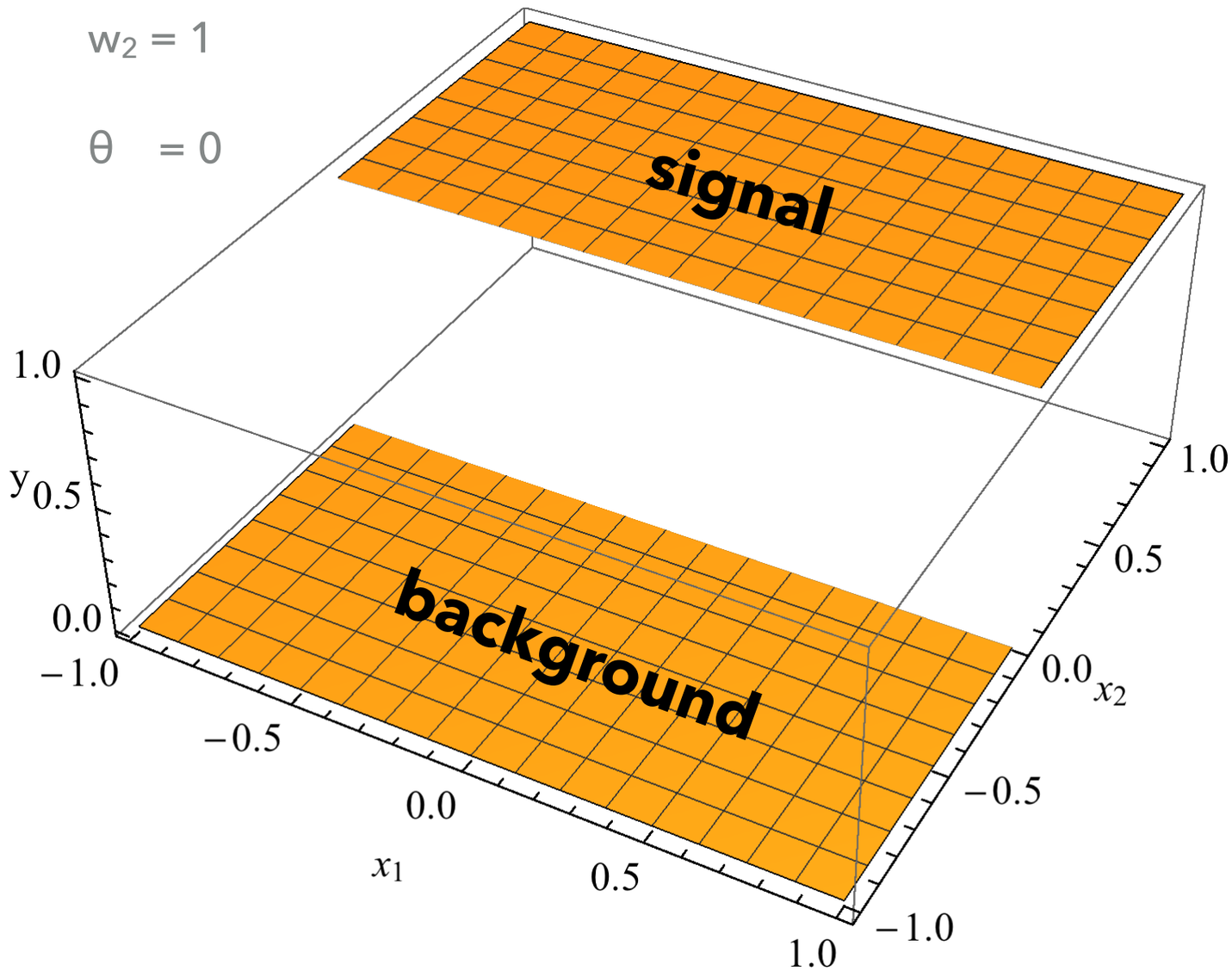
- Output value is either 1 or 0 as some $f(x_1, x_2)$ that depends on the values of w_1, w_2 and θ.

In particle physics we often use machine learning to suppress background. Here y=1 corresponds to signal and y=0 corresponds to background.

$w_1 = 0$

$w_2 = 1$

$\theta = 0$



Queen Mary University of London

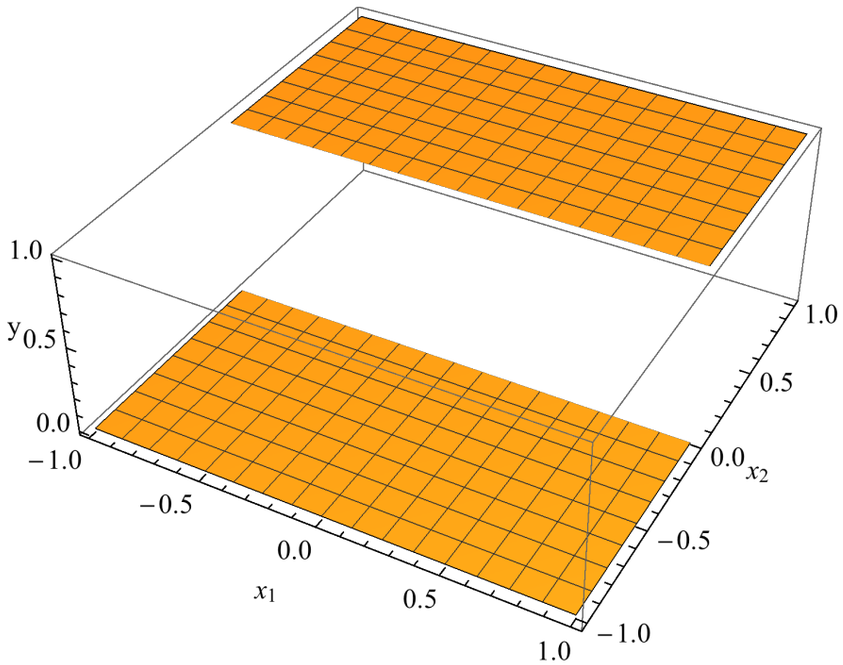
A. Bevan

[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

PERCEPTRONS

- Illustrative examples:

Shift decision plane away from origin

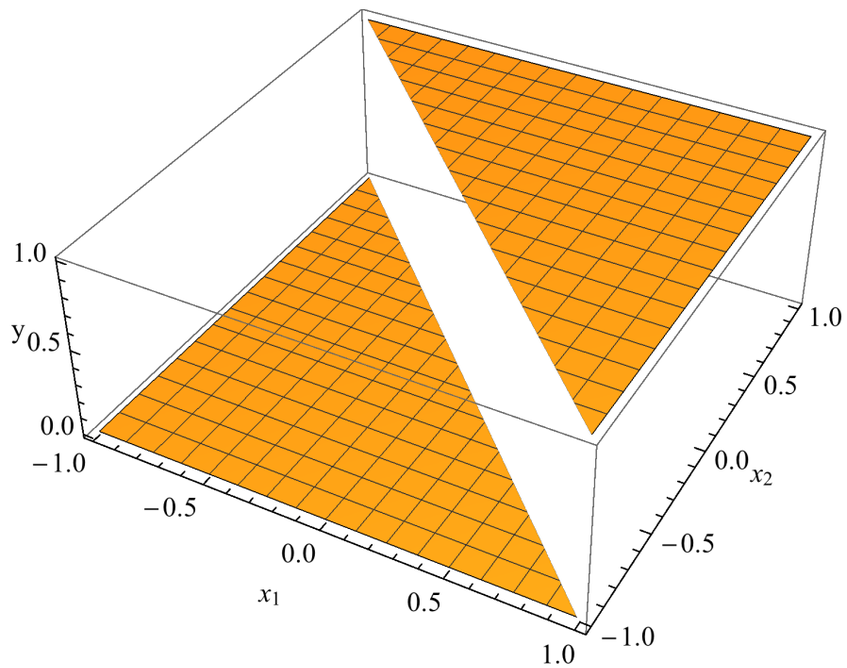


$w_1 = 0$

$w_2 = 1$

$\theta = 0$

Baseline for comparison, decision only on value of x_2

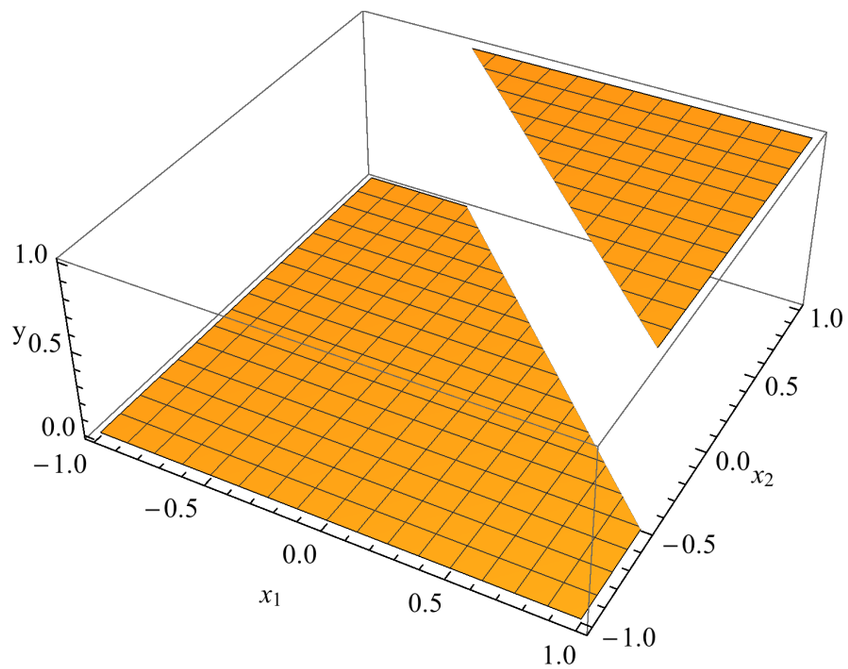


$w_1 = 1$

$w_2 = 1$

$\theta = 0$

Rotate decision plane in (x_1, x_2)



$w_1 = 1$

$w_2 = 1$

$\theta = 0.5$

Queen Mary University of London

A. Bevan

[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

PERCEPTRONS

- Illustrative example:
  - Consider a measurement of two quantities x_1, and x_2.
  - Based on these measurements we determine if the perceptron is to give an output (value = 1) or not (value = 0).
- We can generalise the problem to N quantities as

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

[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

PERCEPTRONS

- Illustrative example:
  - Consider a measurement of two quantities x_1, and x_2.
  - Based on these measurements we determine if the perceptron is to give an output (value = 1) or not (value = 0).
- We can generalise the problem to N quantities as

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

The argument is just the same functional form of Fisher's discriminant.

[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

PERCEPTRONS

- The problem of determining the weights remains (we will discuss optimisation later on).
- For now assume that we can use some heuristic to choose weights that are deemed to be “optimal” for the task of providing a response given some input data example.

[1] F. Rosenblatt, Psych. Rev. **65** p386-408, 1958.

ACTIVATION FUNCTIONS

- The binary activation function of Rosenblatt is just one type of activation function.
  - This gives an all or nothing response.

- It can be useful to provide an output that is continuous between these two extremes.
  - For that we require additional forms of activation function.

ACTIVATION FUNCTIONS

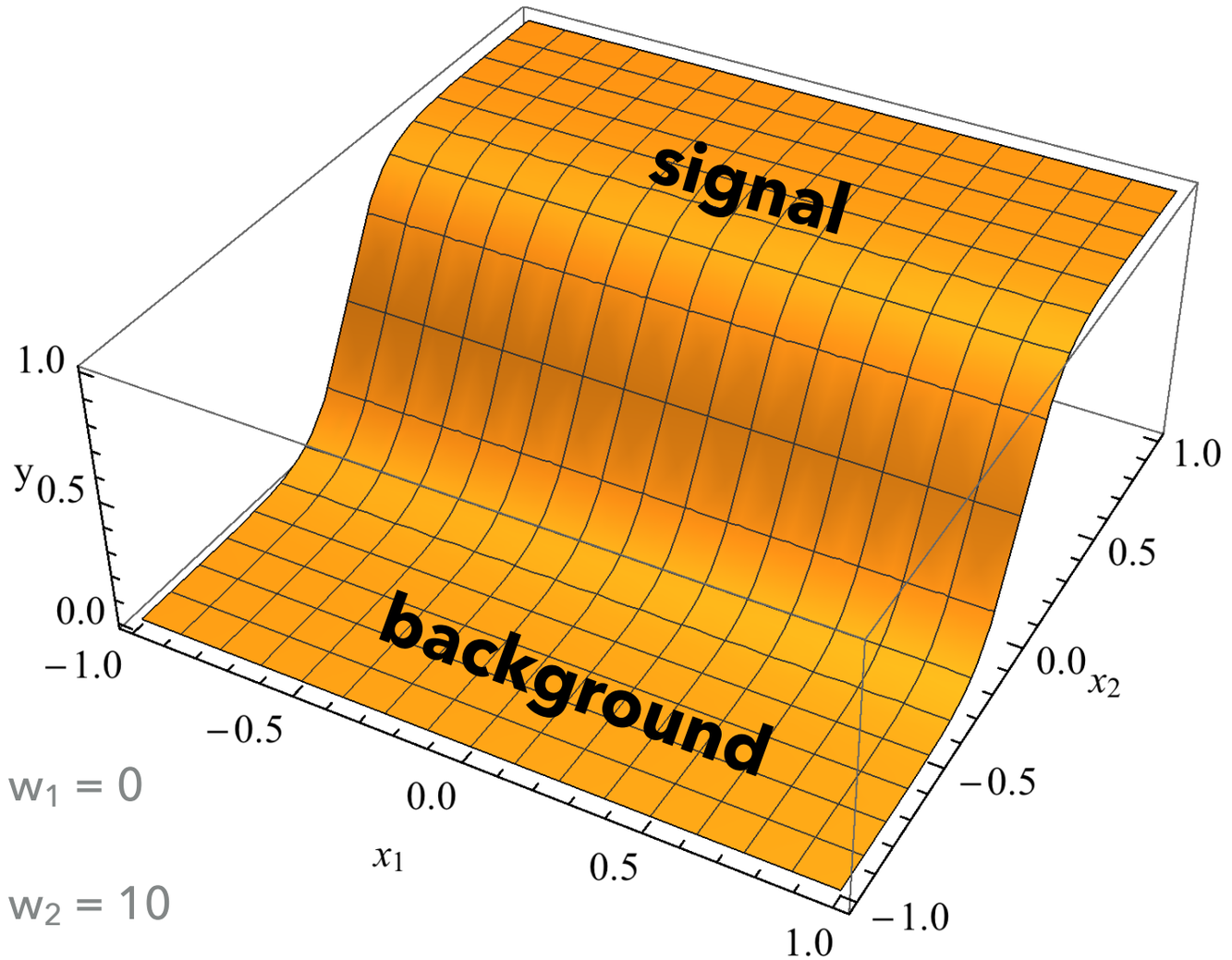
- TensorFlow has the following activation functions (see `tf.nn.ACTIVATIONFUNCTION`) e.g.
  - relu (covered here)
  - leaky_relu (covered here)
  - relu6
  - crelu
  - elu
  - selu
  - softplus
  - softsign
  - dropout
  - bias_add
  - sigmoid (covered here)
  - tanh (covered here)

ACTIVATION FUNCTIONS: LOGISTIC (OR SIGMOID)

- A common activation function used in neural networks:

$$y = \frac{1}{1 + e^{w^T x + \theta}}$$

$$= \frac{1}{1 + e^{(w_1 x_1 + w_2 x_2 + \theta)}}$$



$w_1 = 0$

$w_2 = 10$

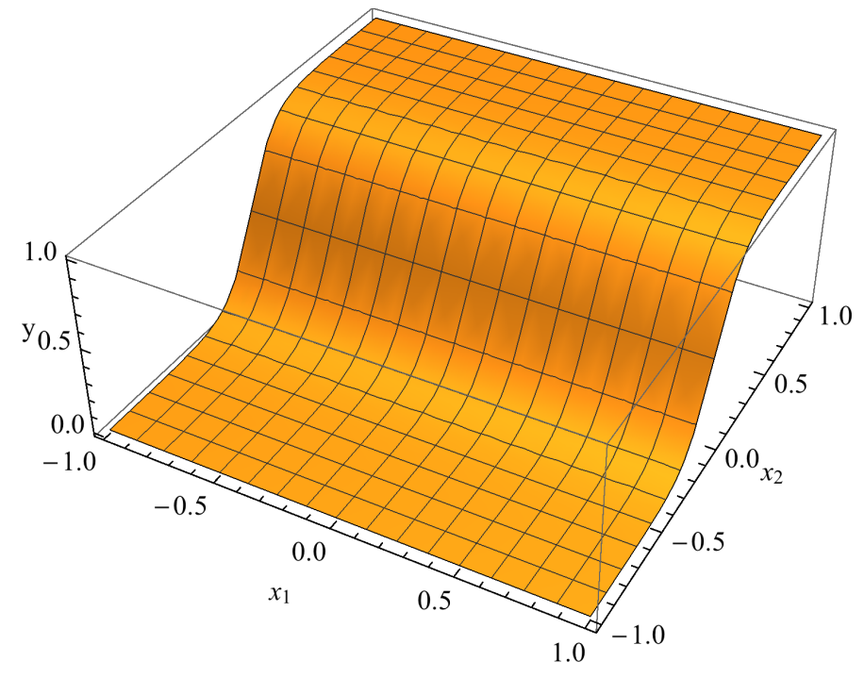
$\theta = 0$

Queen Mary University of London

ACTIVATION FUNCTIONS: LOGISTIC (OR SIGMOID)

$$\frac{1}{1+e^{(w_1x_1+w_2x_2+\theta)}}$$

- A common activation function used in neural networks:

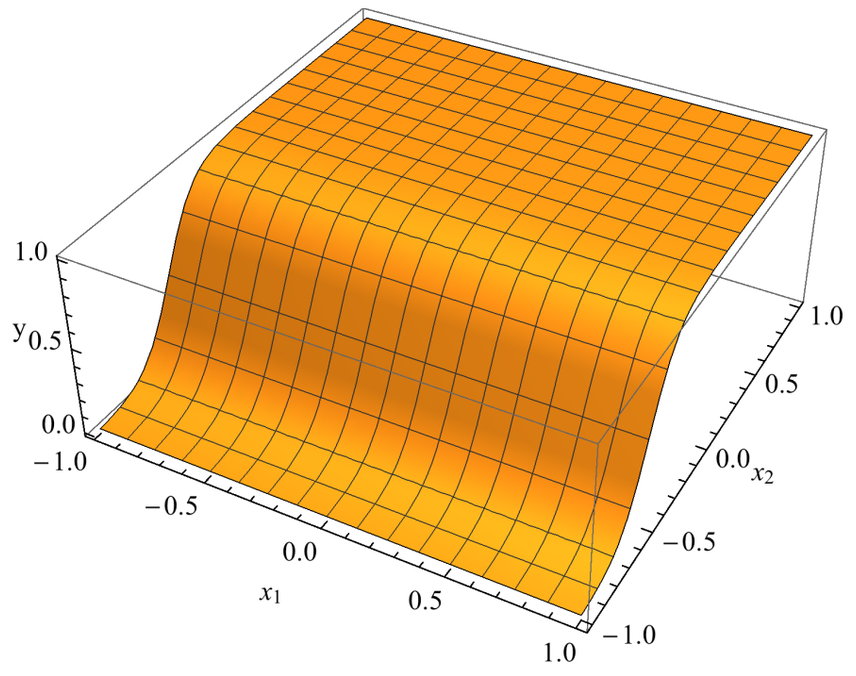


$w_1 = 0$

$w_2 = 10$

$\theta = 0$

Baseline for comparison, decision only on value of x_2

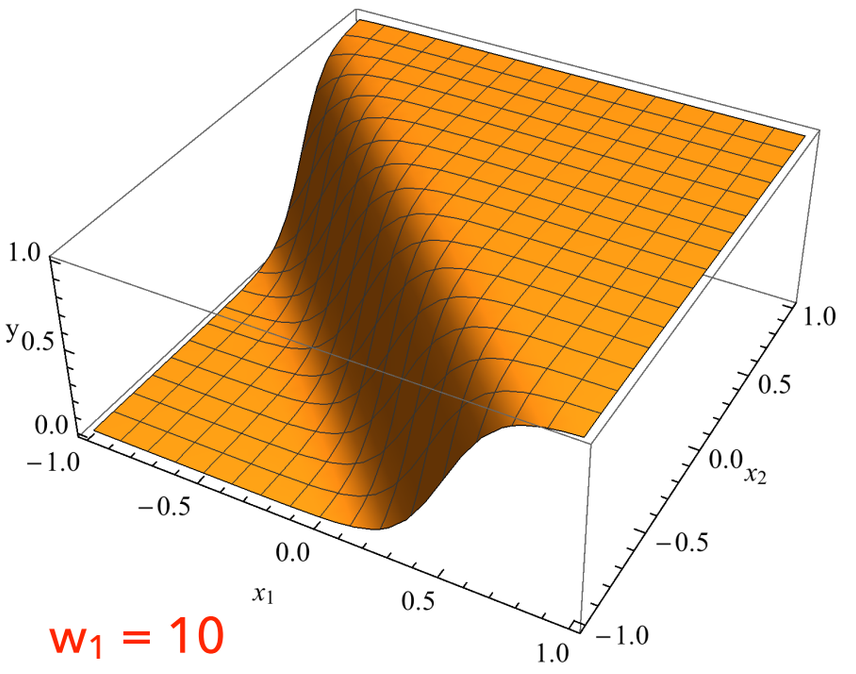


$w_1 = 0$

$w_2 = 10$

$\theta = -5$

Offset from zero using θ



$w_1 = 10$

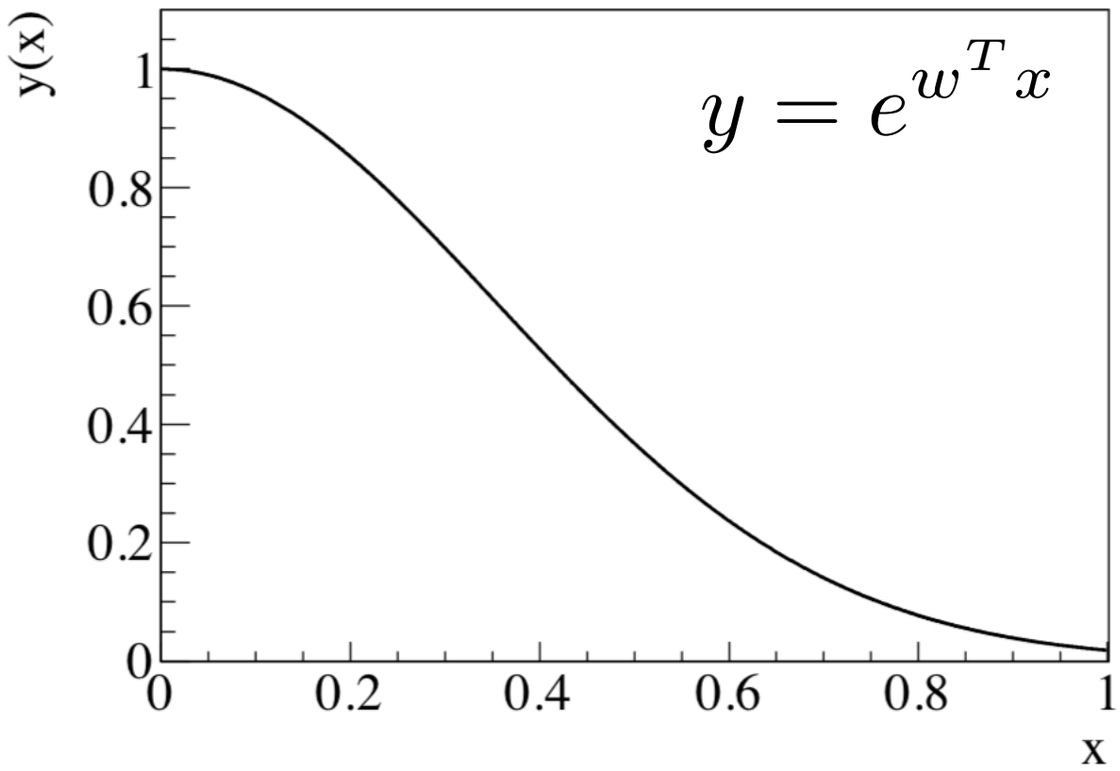
$w_2 = 10$

$\theta = -5$

rotate “decision boundary” in (x_1, x_2)

ACTIVATION FUNCTIONS: RADIAL BASIS FUNCTION

- A common activation function used in neural networks:



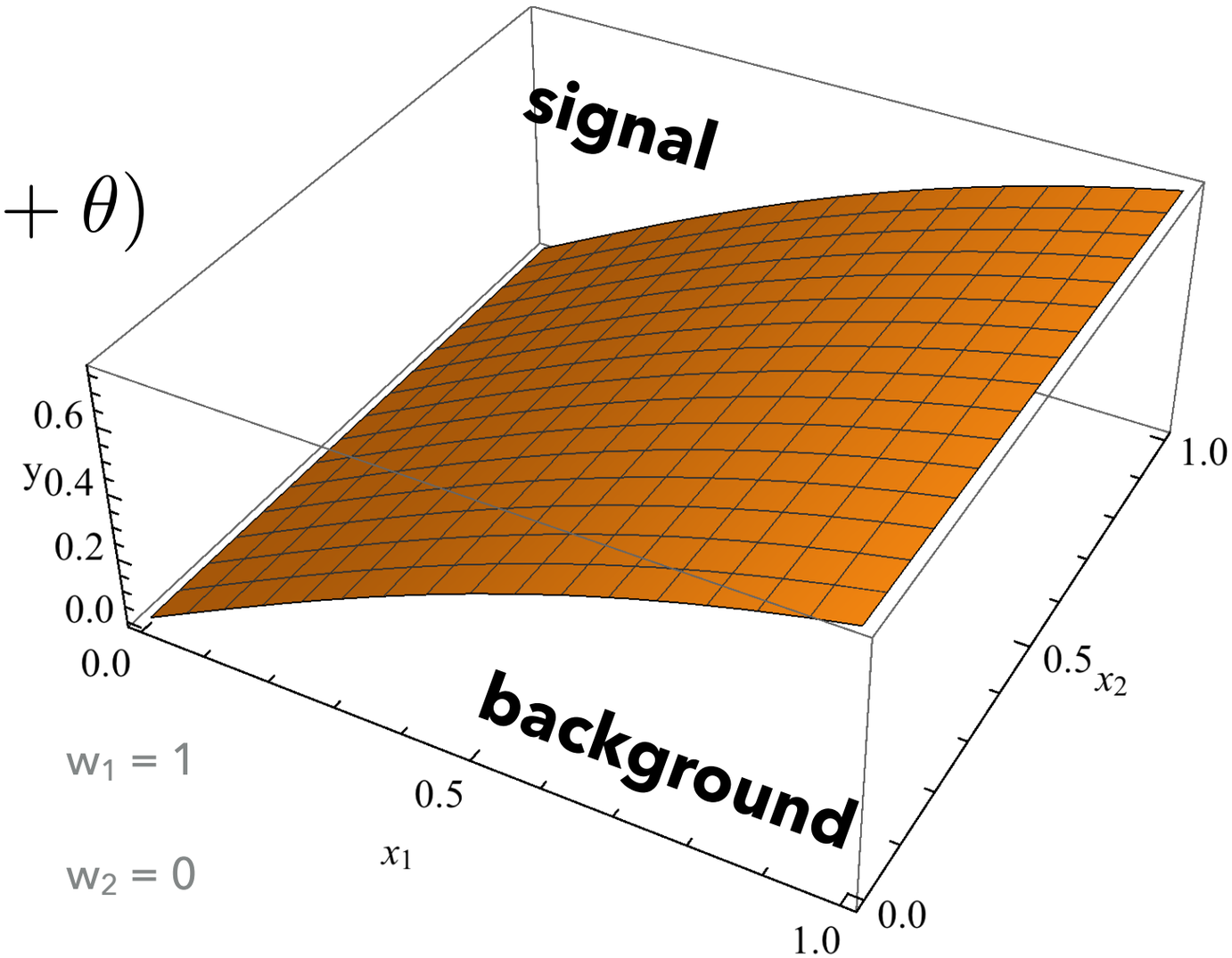
- Very similar to the hyperbolic tan in the way that y varies from 0 to 1.

ACTIVATION FUNCTIONS: HYPERBOLIC TANGENT

- A common activation function used in neural networks:

$$y = \tanh(w^T x + \theta)$$
$$= \tanh(w_1 x_1 + w_2 x_2 + \theta)$$

(Often used with θ = 0)



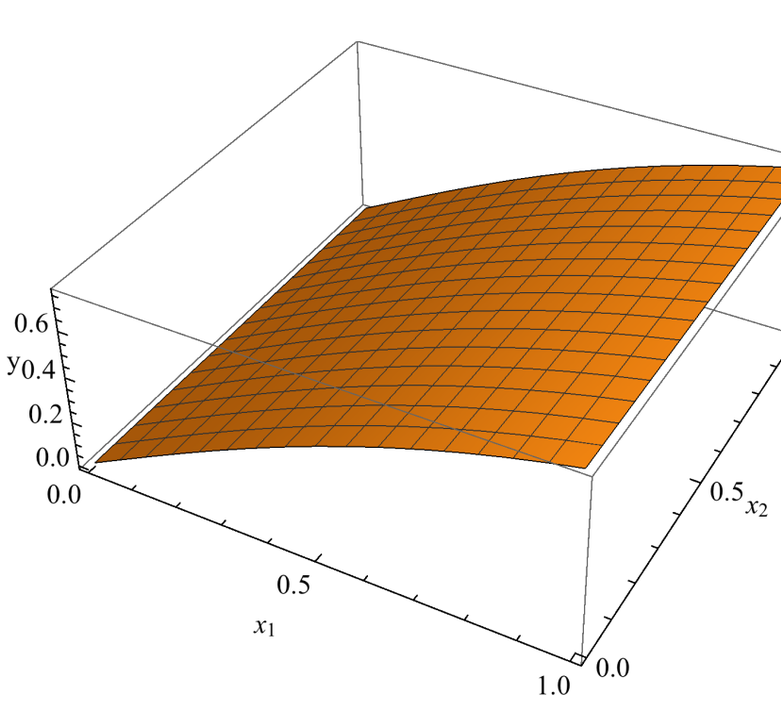
$w_1 = 1$

$w_2 = 0$

$\theta = 0$

ACTIVATION FUNCTIONS: HYPERBOLIC TANGENT $\tanh(w_1 x_1 + w_2 x_2 + \theta)$

- A common activation function used in neural networks:

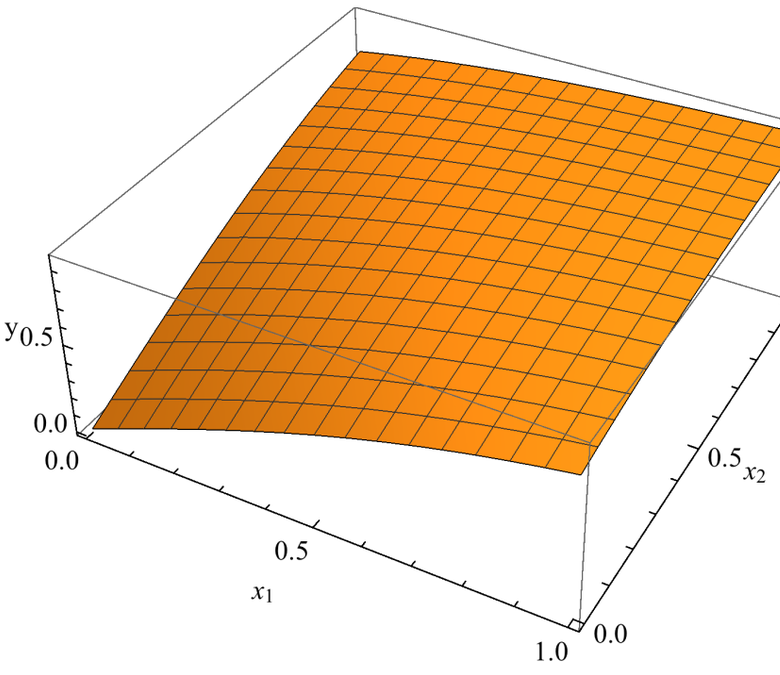


$w_1 = 1$

$w_2 = 0$

$\theta = 0$

Baseline for comparison, decision only on value of x_1

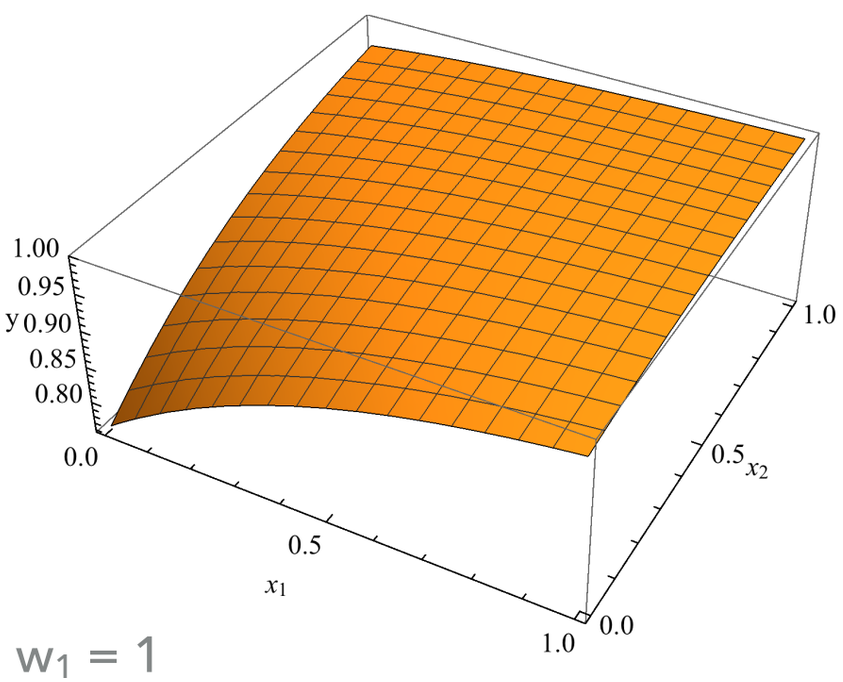


$w_1 = 1$

$w_2 = 1$

$\theta = 0$

rotate “decision boundary” in (x_1, x_2)



$w_1 = 1$

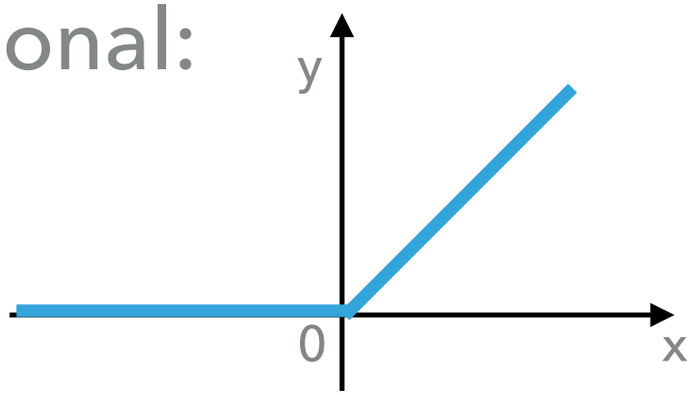
$w_2 = 1$

$\theta = -1$

Offset (vertically) from zero using θ

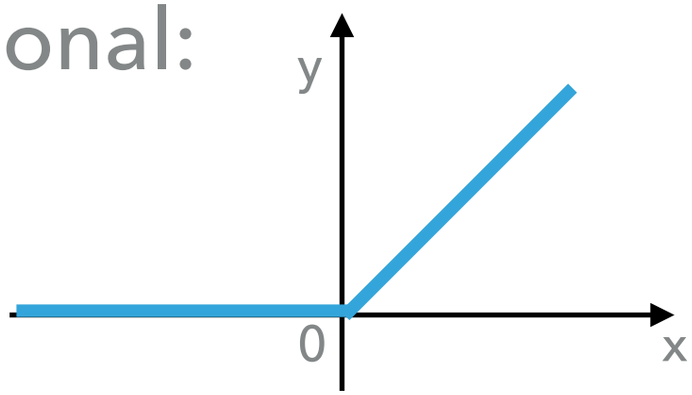
ACTIVATION FUNCTIONS: RELU

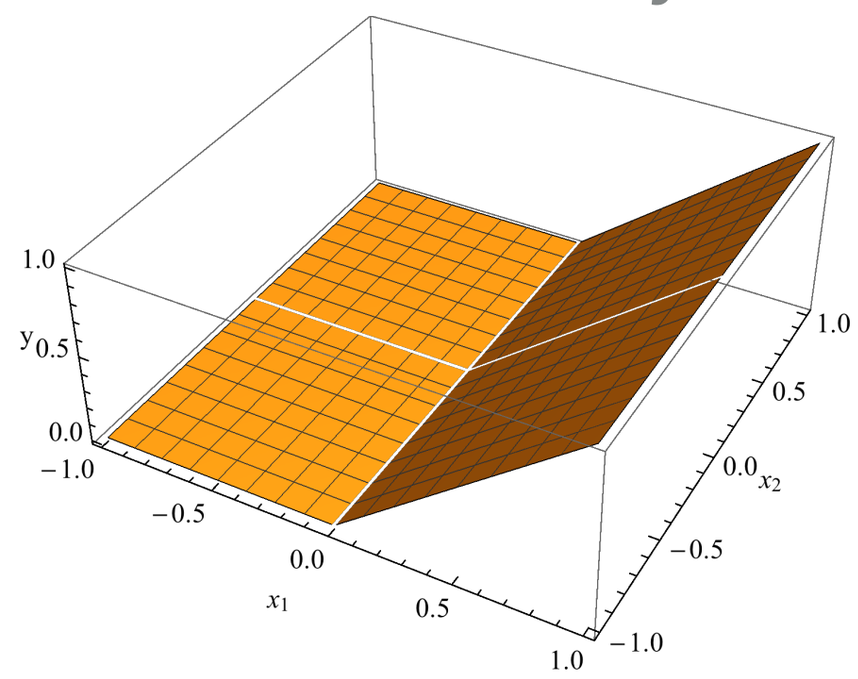
- The **Re**ctified **L**inear **U**nit activation function is commonly used for CNNs. This is given by a conditional:
  - If (x < 0) y = 0
  - otherwise y = x



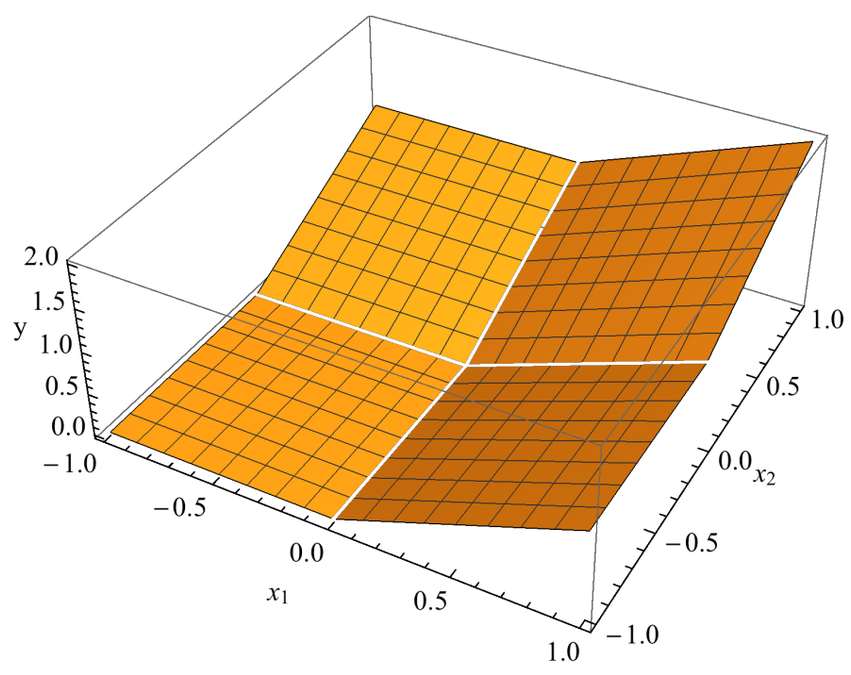
ACTIVATION FUNCTIONS: RELU

- The **Re**ctified **L**inear **U**nit activation function is commonly used for CNNs. This is given by a conditional:
  - If (x < 0) y = 0
  - otherwise y = x

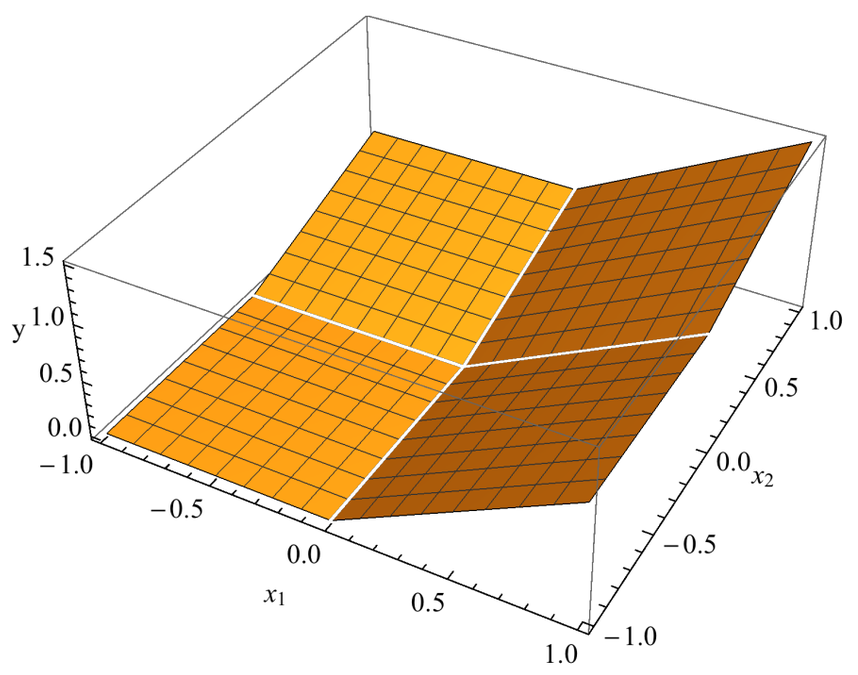




$w_1 = 1, w_2 = 0$



$w_1 = 1, w_2 = 1$



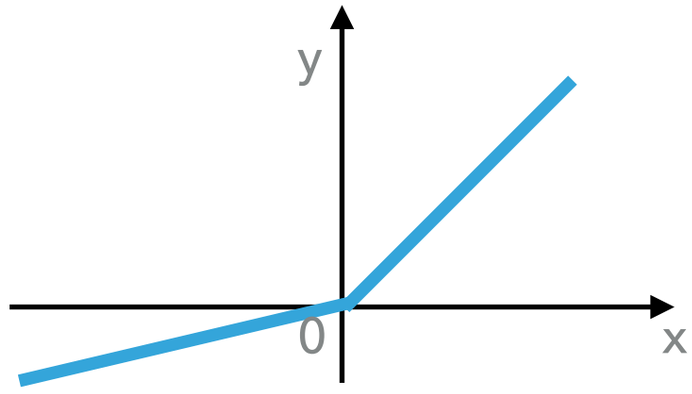
$w_1 = 1, w_2 = 0.5$

Importance of features in the perceptron still depend on weights as illustrated in these plots.

Queen Mary University of London

ACTIVATION FUNCTIONS: PRELU VARIANT

- The ReLU activation function can be modified to avoid gradient singularities.
- This is the **PReLU** or **Leaky ReLU** activation function
  - If (x < 0) y = a*x
  - otherwise y = x



- Collectively we can write the (P)ReLU activation function as

$$f(x) = \max(0, x) + a \min(0, x)$$

- Can be used effectively for need CNNs (more than 8 convolution layers), whereas the ReLU activation function can have convergence issues for such a configuration[2].
- If a is small (0.01) it is referred to as a leaky ReLU function[1]. The default implementation in TensorFlow has a=0.2[3].

[1] Maas, Hannun, Ng, ICML2013.
[2] He, Zhang, Ren and Sun, arXiv:1502.01852
[3] See https://github.com/tensorflow/tensorflow/blob/r1.8/tensorflow/python/ops/nn_ops.py

ACTIVATION FUNCTIONS: RELU

- Performs better than a sigmoid for a number of applications[1].
  - Weights for a relu are typically initialised with a truncated normal, OK for shallow CNNs, but there are convergence issues with deep CNNs when using this initialisation approach[1].

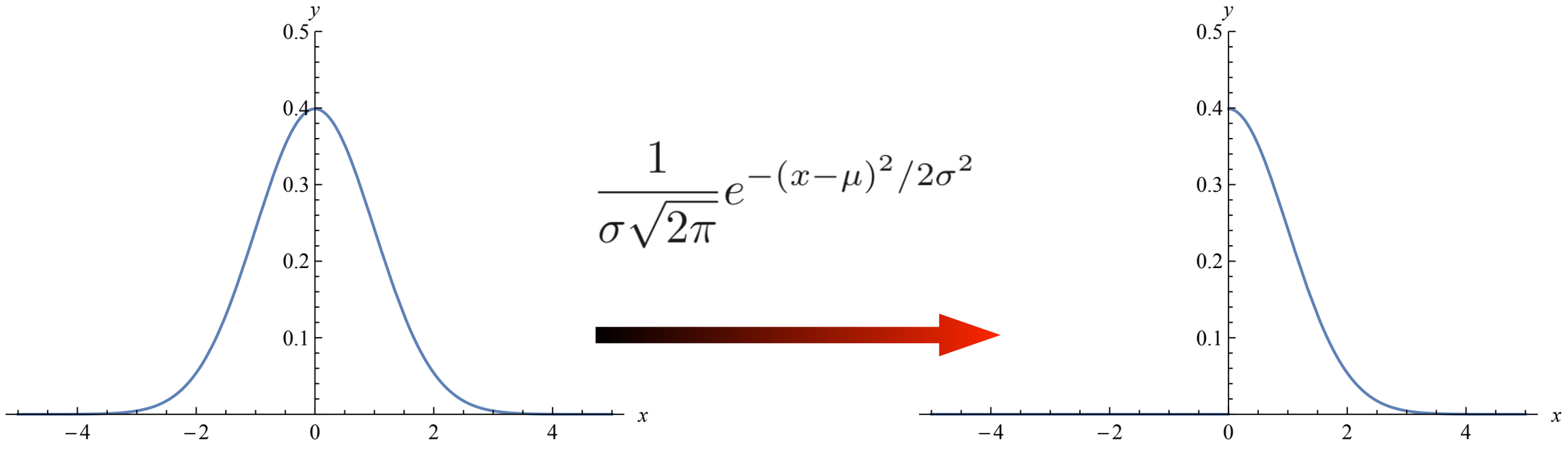
```
initial = tf.truncated_normal(shape, stddev=0.1)
```

- Other initialisation schemes have been proposed to avoid this issue for deep CNNs (more than 8 conv layers) as discussed in Ref [2].

[1] Maas, Hannun, Ng, ICML2013.
[2] He, Zhang, Ren and Sun, arXiv:1502.01852

ACTIVATION FUNCTIONS: RELU

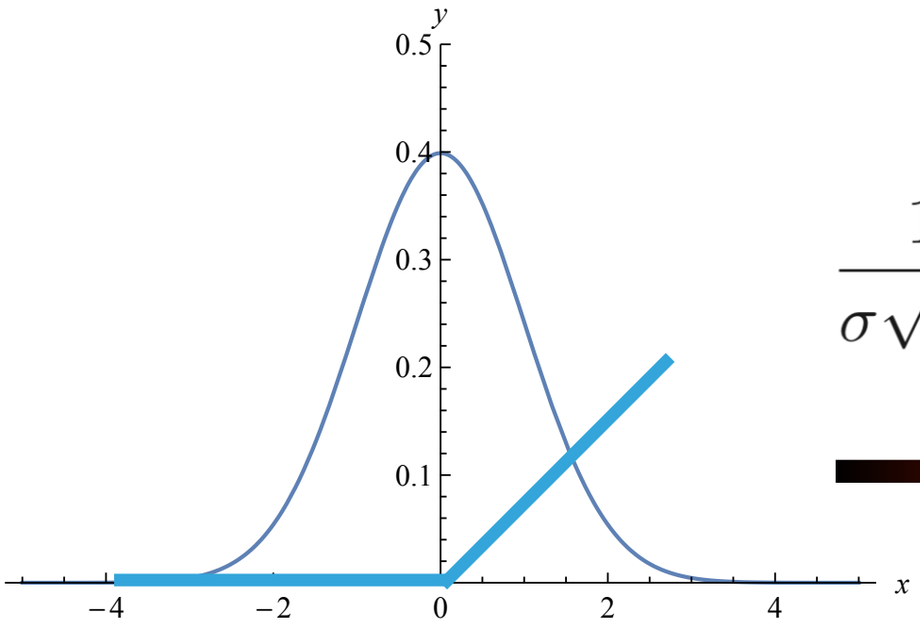
- N.B. Gradient descent optimisation algorithms will not change the weights for an activation function if the initial weight is set to zero.
  - This is why a truncated normal is used for initialisation, rather than a Gaussian that has x∈[-∞, ∞].



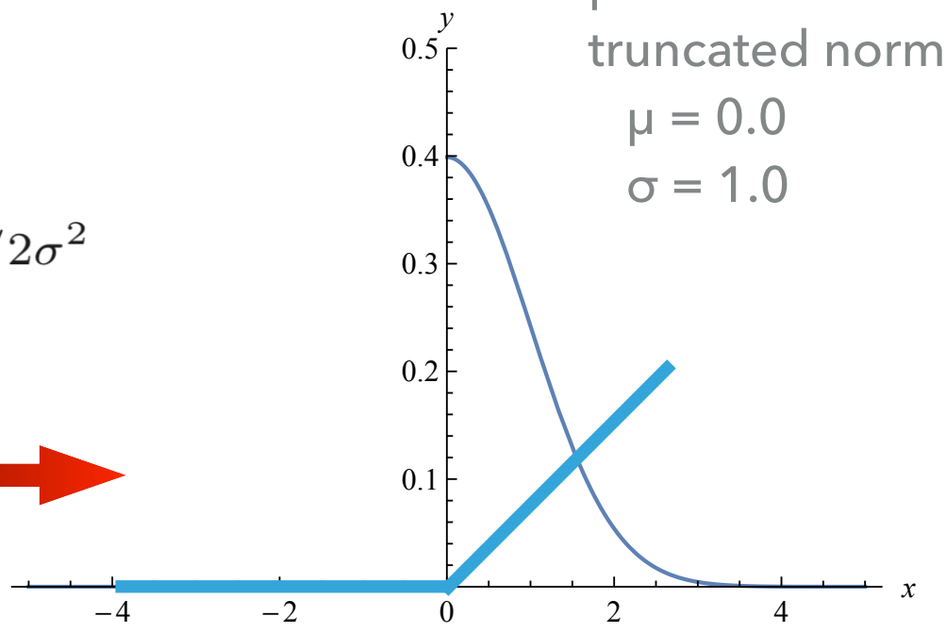
[1] Maas, Hannun, Ng, ICML2013.
[2] He, Zhang, Ren and Sun, arXiv:1502.01852

ACTIVATION FUNCTIONS: RELU

- N.B. Gradient descent optimisation algorithms will not change the weights for an activation function if the initial weight is set to zero.
  - This is why a truncated normal is used for initialisation, rather than a Gaussian that has x∈[-∞, ∞].



$$\frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$



TensorFlow default parameters for the truncated normal are:

μ = 0.0

σ = 1.0

[1] Maas, Hannun, Ng, ICML2013.

[2] He, Zhang, Ren and Sun, arXiv:1502.01852

ACTIVATION FUNCTIONS: DATA PREPARATION

- Input features are arbitrary; whereas activation functions have a standardised input domain of [-1, 1] or [0, 1].
  - Limits the range with which we have to adjust hyper-parameters to find an optimal solution.
  - Avoids large or small hyper-parameters.
- Other algorithms have more stringent requirements for data-preprocessing when being fed into them.
  - All these points indicate that we need to prepare data appropriately before feeding it into a perceptron, and hence network.

ACTIVATION FUNCTIONS: DATA PREPARATION

- Input features are arbitrary; whereas activation functions have a standardised input domain of [-1, 1] or [0, 1].
  - We can map our input feature space onto a standardised domain that matches some range that matches that of the activation function.
  - Saves work for the optimiser in determining hyper-parameters.
  - Standardises weights to avoid numerical inaccuracies; and set common starting weights.

- e.g.
  - having an energy or momentum measured in units of 10^{12} eV, would require weights $O(10^{-12})$ to obtain an O(1) result for $w_i x_i$.
  - Mapping eV $\mapsto$TeV would translate 10^{12} eV $\mapsto$ 1TeV, and allow for O(1) weights leading to an O(1) result for $w_i x_i$.
  - Comparing weights for features that are standardised allows the user to develop an intuition as to what the corresponding activation function will look like.

ACTIVATION FUNCTIONS: DATA PREPARATION

- A good paper to read on data preparation is [1]. This includes the following suggestions:
  - Standardising input features onto [-1, 1] results in faster optimisation using gradient descent algorithms.
  - Shift the features to have a mean value of zero.
  - It is also possible to speed up optimisation by de-correlating input variables[1].
  - Having done this one can also scale the features to have a similar variance.

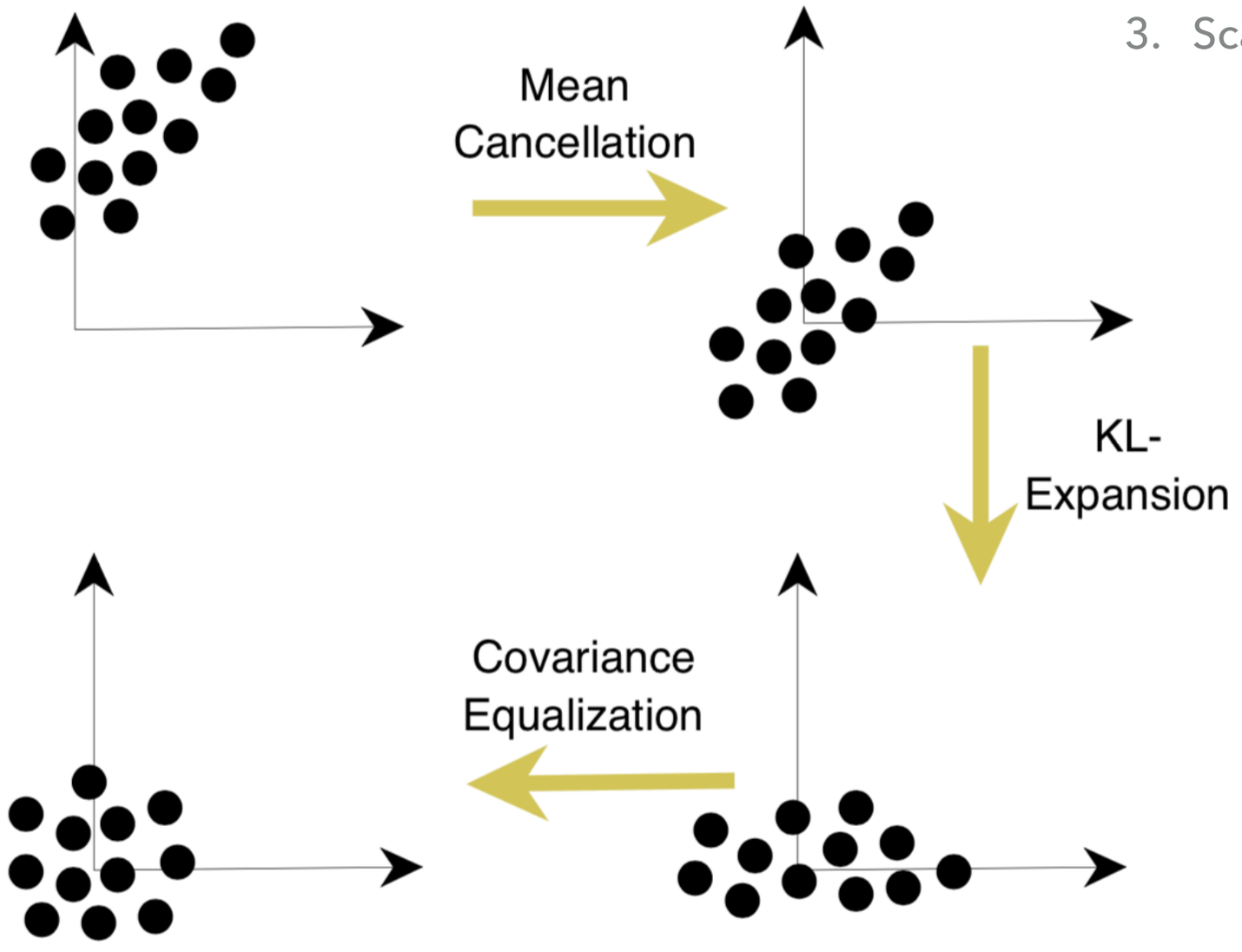
[1]Decorrelation of features is not essential assuming a sufficiently general optimisation algorithm is being used. The rationale is that if one can decorrelate features then we have to minimise the cost as a function of weights for one feature at a time, rather than the whole N dimensional feature space. See equivalent example with the BDT in TMVA.

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

ACTIVATION FUNCTIONS: DATA PREPARATION

- e.g.

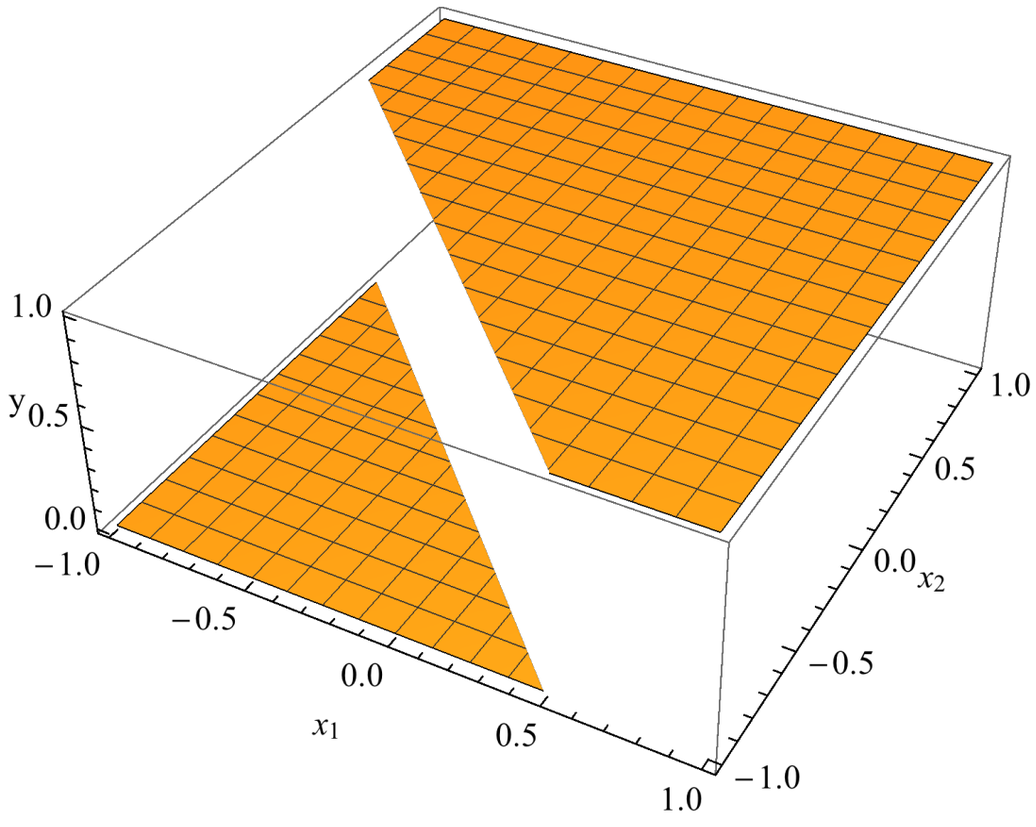
1. Shift the distribution to have a zero mean
2. Decorrelate input features
3. Scale to match covariance of features.



[1] LeCun et al., Efficient BackProp, Neural Networks Tricks of the Trade, Springer 1998 (Fig. 3)

ARTIFICIAL NEURAL NETWORKS (ANNs)

- A single perceptron can be thought of as defining a hyperplane that separates the input feature space into two regions.



A binary threshold activation function is an equivalent algorithm to cutting on a fisher discriminant to distinguish between types of training example.

$$\mathcal{F} = w^T x + \beta$$

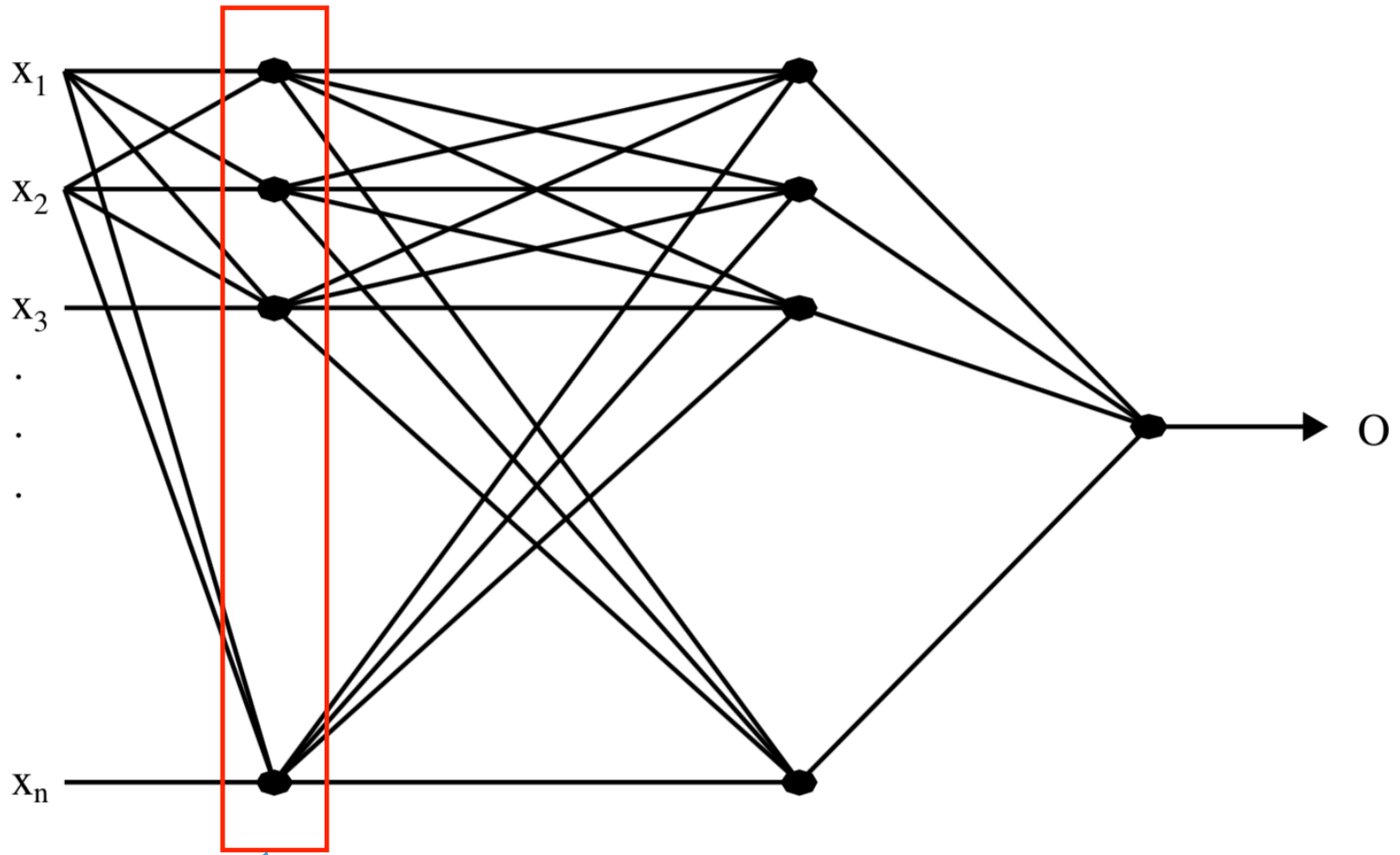
The only real difference is the heuristic used to determine the weights.

ARTIFICIAL NEURAL NETWORKS (ANNs)

- A single perceptron can be thought of as defining a hyperplane that separates the input feature space into two regions.
  - This is a literal illustration for the binary threshold perceptron.
  - The other perceptrons discussed have a gradual transition from one region to the other.
- We can combine perceptrons to impose multiple hyperplanes on the input feature space to divide the data into different regions.
- Such a system is an artificial neural network. There are various forms of ANNs; in HEP this is usually synonymous with a multi-layer perceptron (MLP).
  - An MLP has multiple layers of perceptrons; the outputs of the first layer of perceptrons are fed into a subsequent layer, and so on. Ultimately the responses of the final layer are brought together to compute an overall value for the network response.

MULTILAYER PERCEPTRONS

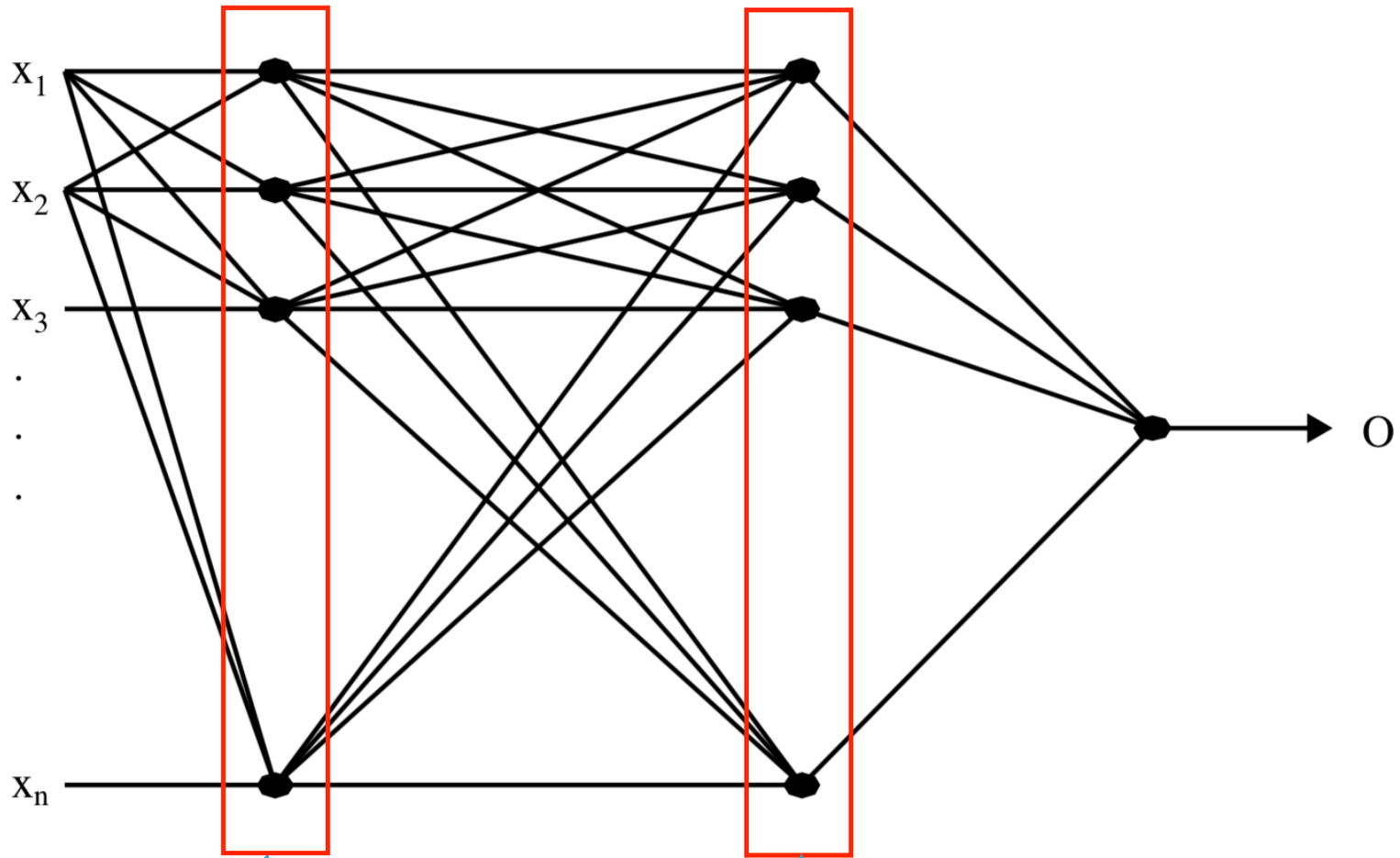
- Illustrative example: Input data example: $x = \{x_1, x_2, x_3, \ldots, x_n\}$



Input layer of n perceptrons; one for each dimension of the input feature space

MULTILAYER PERCEPTRONS

- Illustrative example: Input data example: $x = \{x_1, x_2, x_3, \ldots, x_n\}$

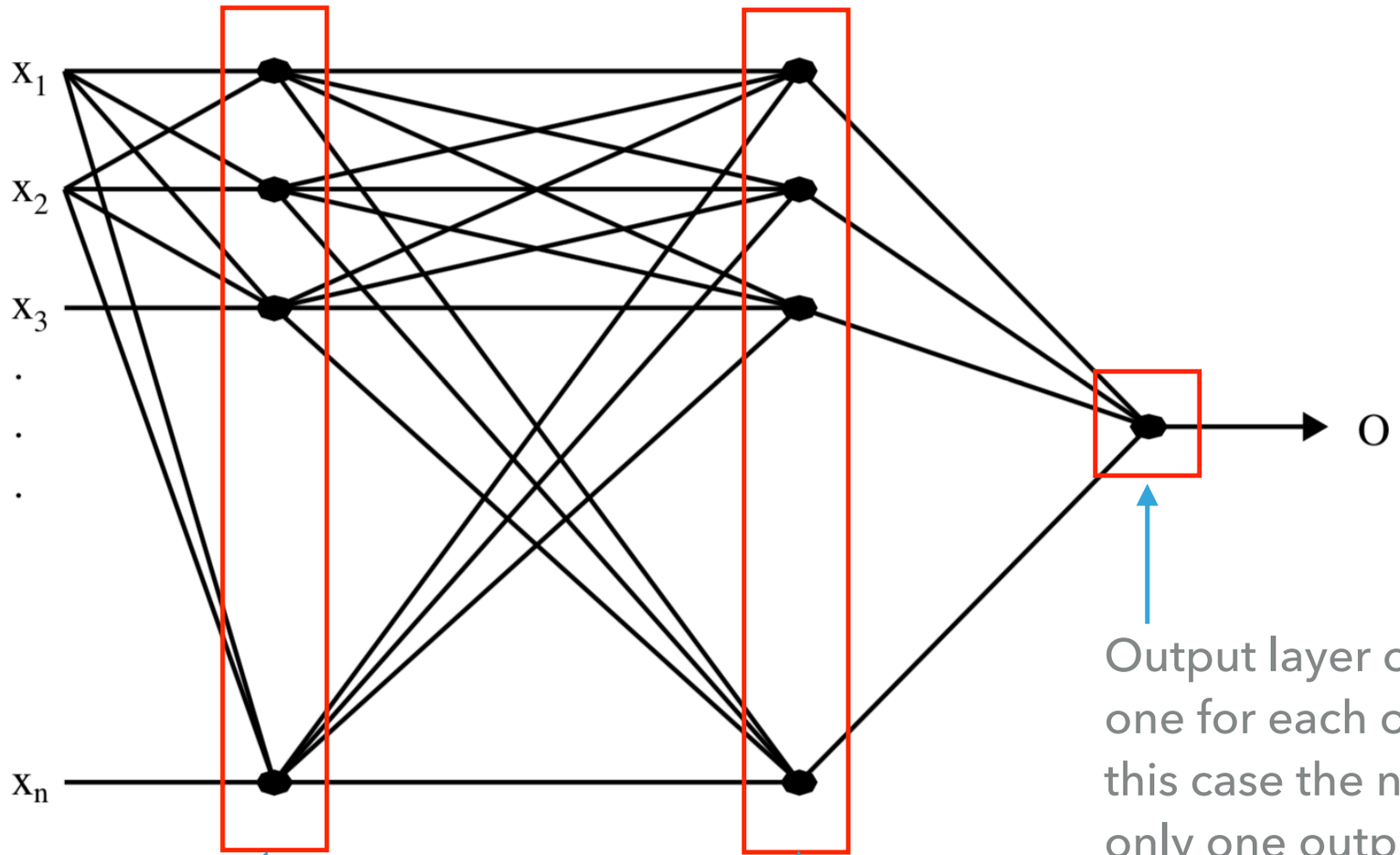


Input layer of n perceptrons; one for each dimension of the input feature space

Hidden layer of some number of perceptrons, M; at least one for each dimension of the input feature space.

MULTILAYER PERCEPTRONS

- Illustrative example: Input data example: $x = \{x_1, x_2, x_3, \ldots, x_n\}$



Output layer of perceptrons; one for each output type. In this case the network has only one output.

Input layer of n perceptrons; one for each dimension of the input feature space

Hidden layer of some number of perceptrons, M; at least one for each dimension of the input feature space.

TRAINING

- Parameter tuning is referred to as training.
  - A perceptron of the form $f(w^Tx+\theta)$ has $n+1=\dim(x)+1$ hyper-parameters to be tuned.
  - The input layer of perceptrons in an MLP has $n(n+1)$ hyper parameters to be tuned.
  - ... and so on.
- We tune parameters based on some metric[1] called the loss function.
- We optimise the hyper-parameters of a network in order to minimise the loss function for an ensemble of data.

[1]Also called a figure of merit. The general term when applied to machine learning is the loss function.

TRAINING AN MLP

1. Define an algorithm to assign an error to a given set of weights.
2. Define the procedure for terminating training, based on the computed error, or other information.
3. Guess an initial set of weights to test the classification process.
4. Evaluate the error defined in step (1) for a given set of data containing (preferably) equal numbers of target types for signal and background.
5. Determine a new set of weights based on mis-classified events.
6. Iterate the last two steps until the convergence criteria defined in step (2) has been reached.
7. Validate the weights obtained via this procedure (see section 9.4.4).

TRAINING A MLP

- Consider a single perceptron with the L2 loss function:
  - For the i^{th} event:

true (label) value

error → $\epsilon_i = \frac{1}{2}(t_i - y_i)^2$

prediction

TRAINING A MLP: LOSS FUNCTIONS

- Consider a single perceptron with the L2 loss function:
  - For the N events the total loss function is the sum of losses for each individual event.

$$
\begin{aligned}
E &= \sum_{i=1}^{N} \epsilon_i \\
&= \frac{1}{2} \sum_{i=1}^{N} (t_i - y_i)^2 .
\end{aligned}
$$

HYPERPARAMETER OPTIMISATION: LOSS FUNCTIONS

- L2 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 i^{th} example

t_i = True target type for the i^{th} 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 i^{th} example

t_i = True target type for the i^{th} 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

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

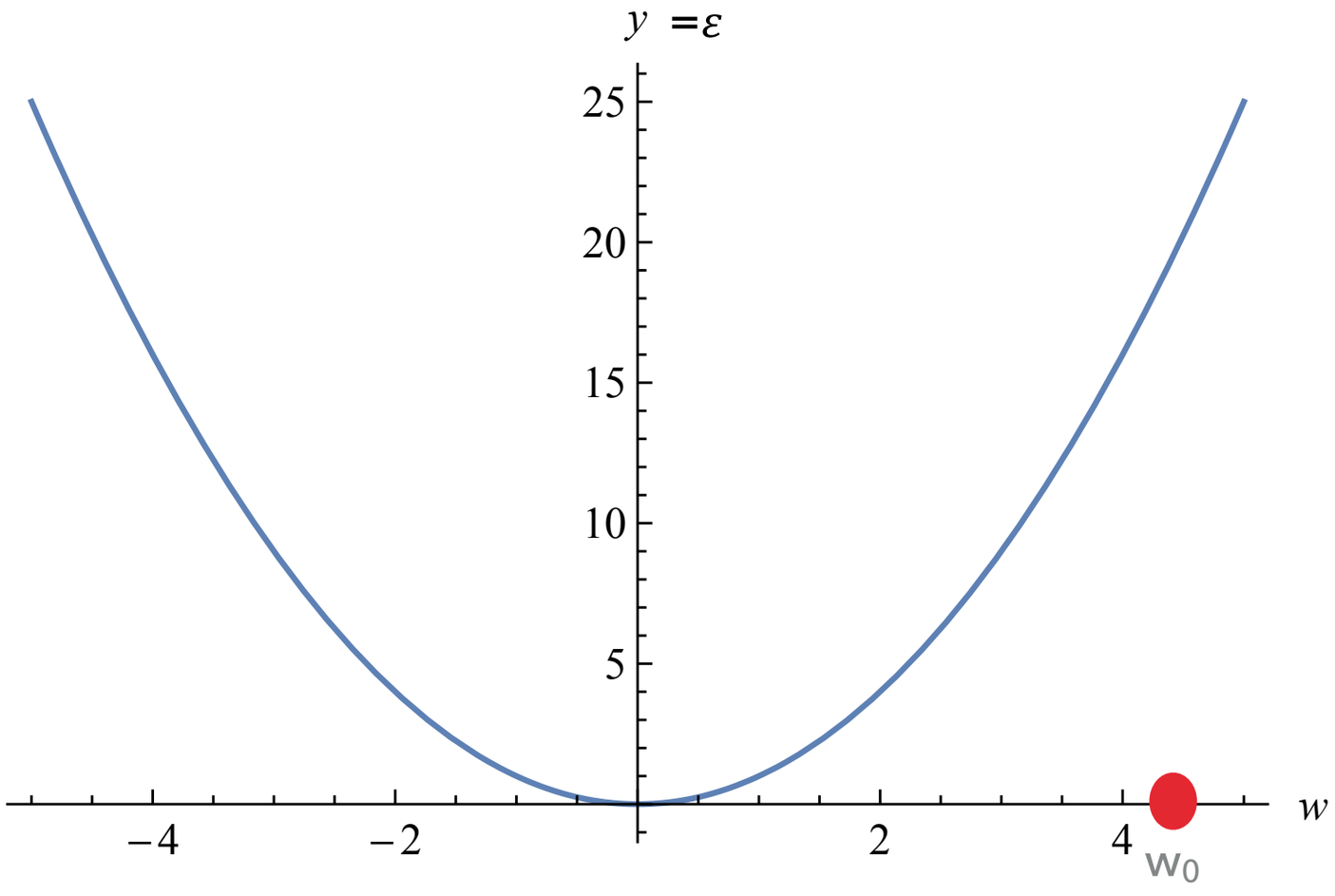
n = number of examples

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

y = output prediction of model

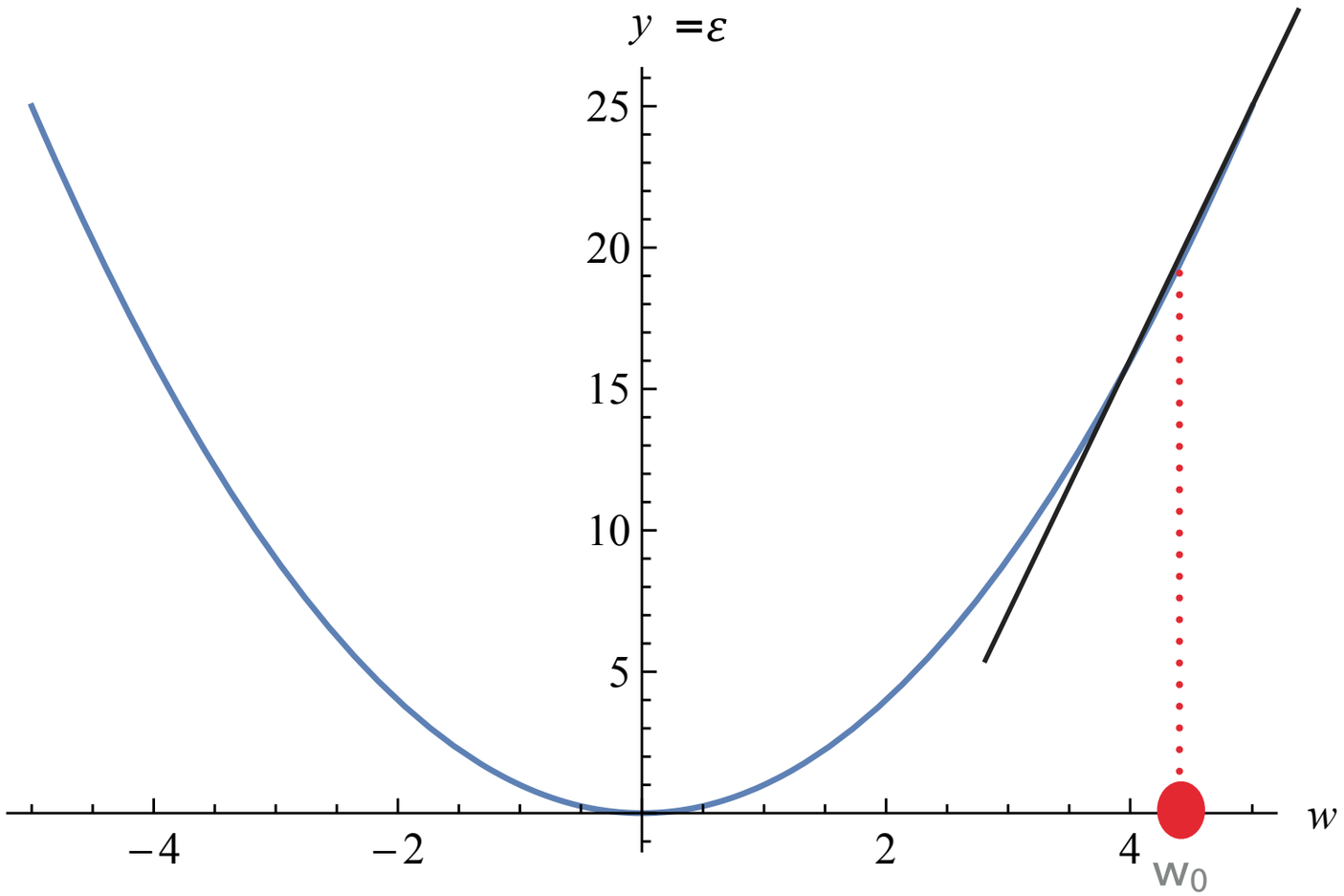
HYPERPARAMETER OPTIMISATION: GRADIENT DESCENT

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



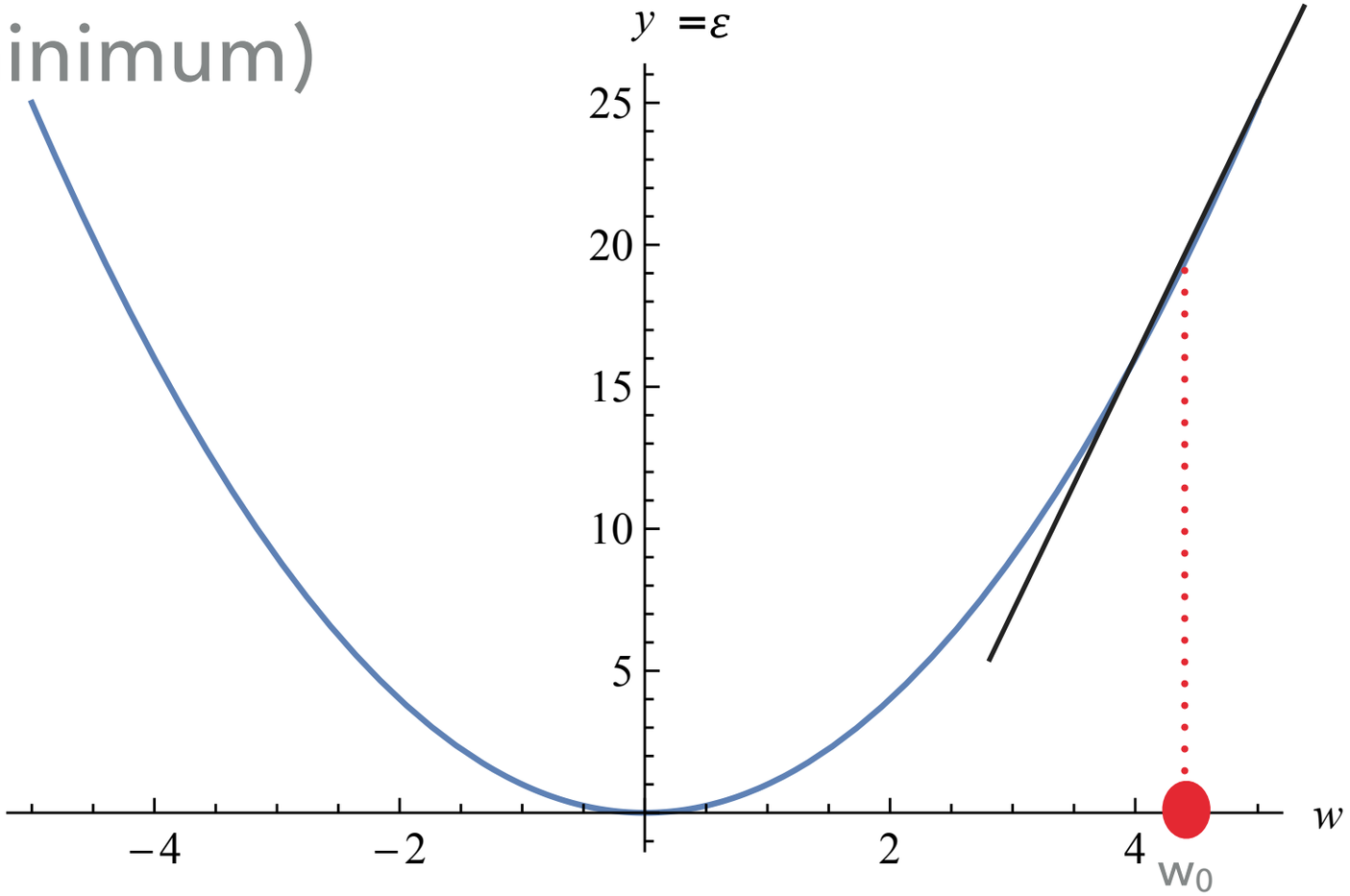
HYPERPARAMETER OPTIMISATION: GRADIENT DESCENT

- 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)



$$\Delta y = \Delta w \frac{dy}{dw}$$

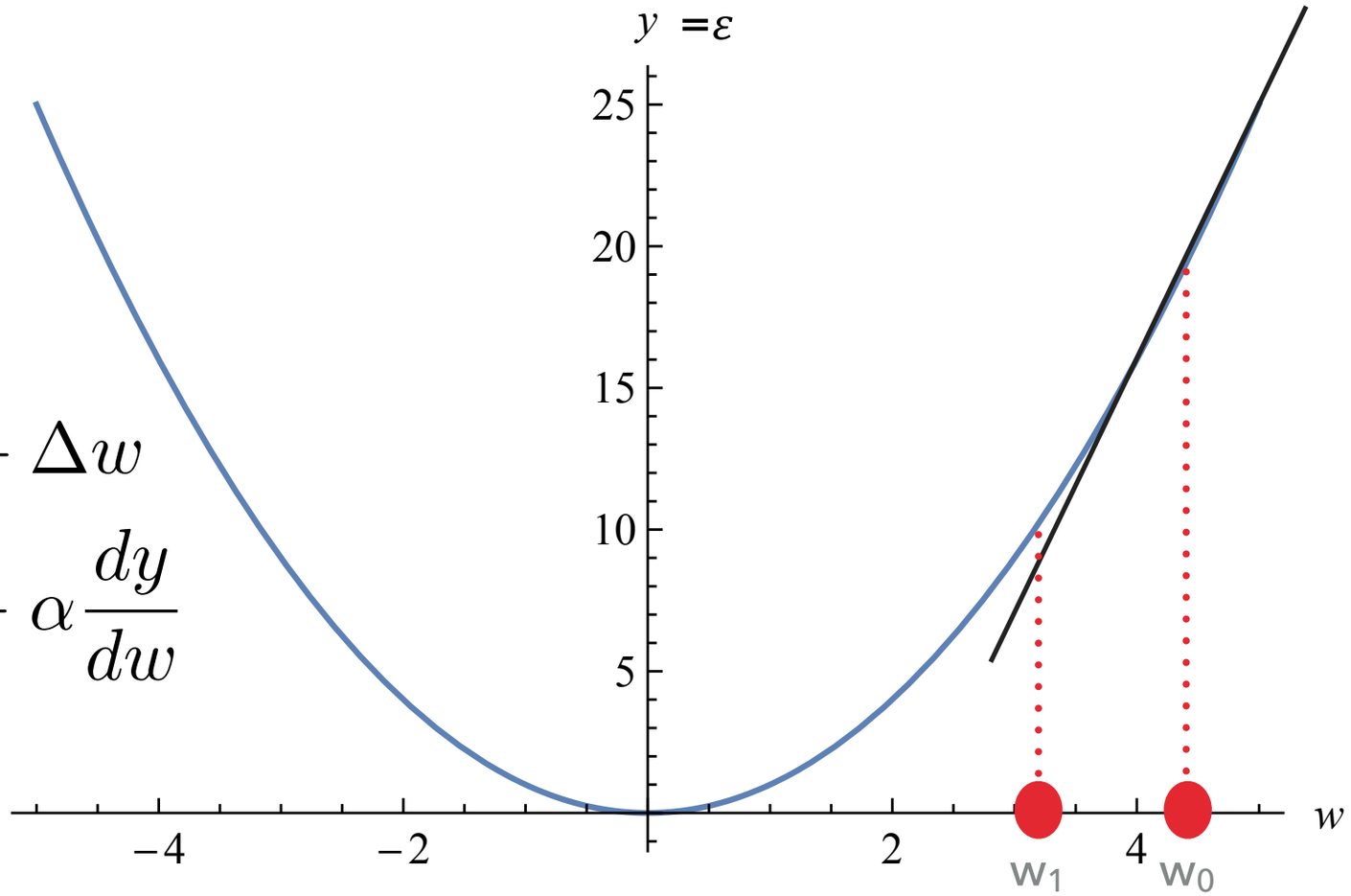
$$= -\alpha \left(\frac{dy}{dw}\right)^2$$

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: $w_1 = w_0 + \Delta w$



$$\Delta y = \Delta w \frac{dy}{dw}$$

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

$$w_{i+1} = w_i + \Delta w$$

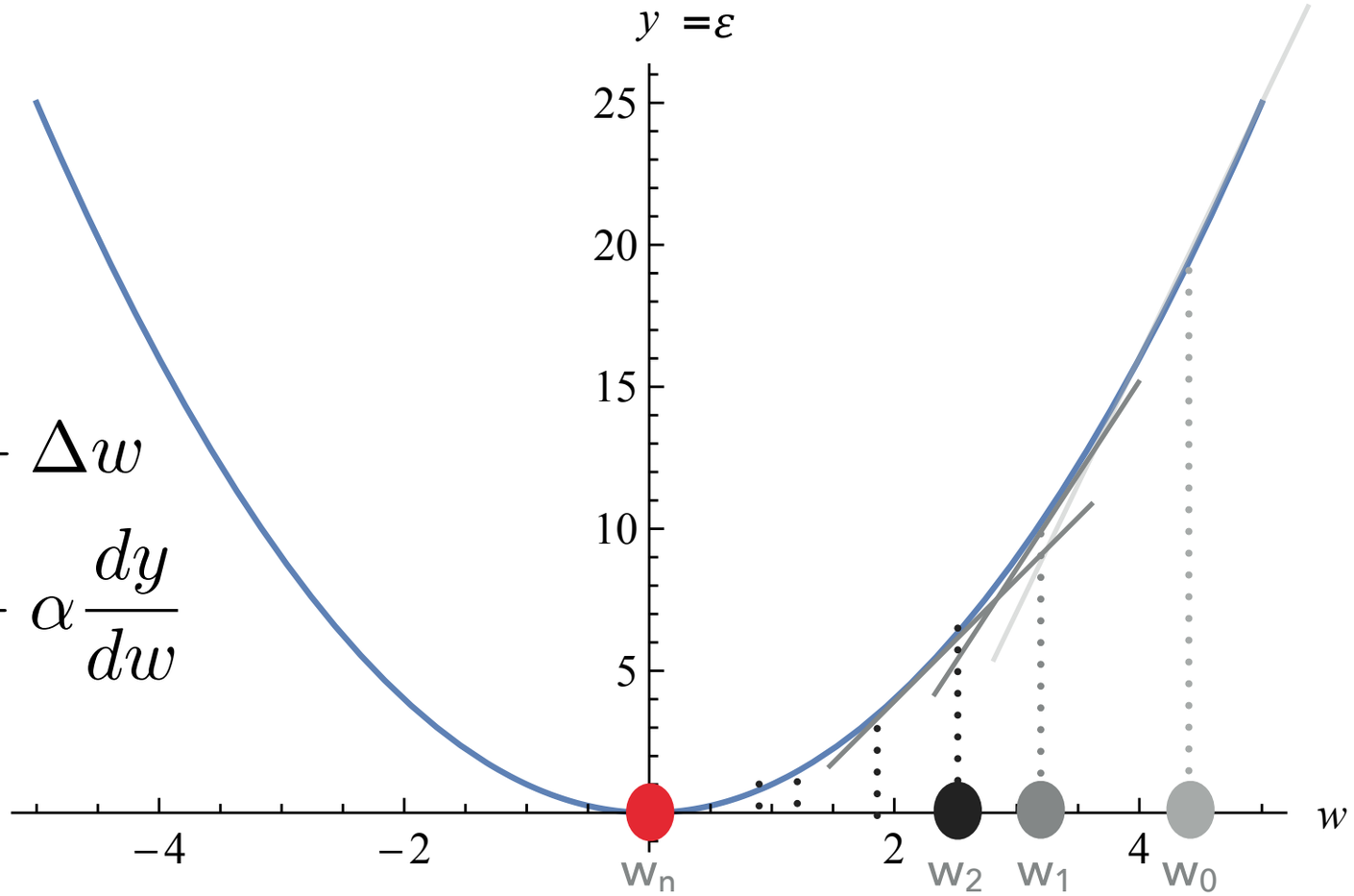
$$= w_i - \alpha \frac{dy}{dw}$$

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.



$$\Delta y = \Delta w \frac{dy}{dw}$$

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

$$w_{i+1} = w_i + \Delta w$$

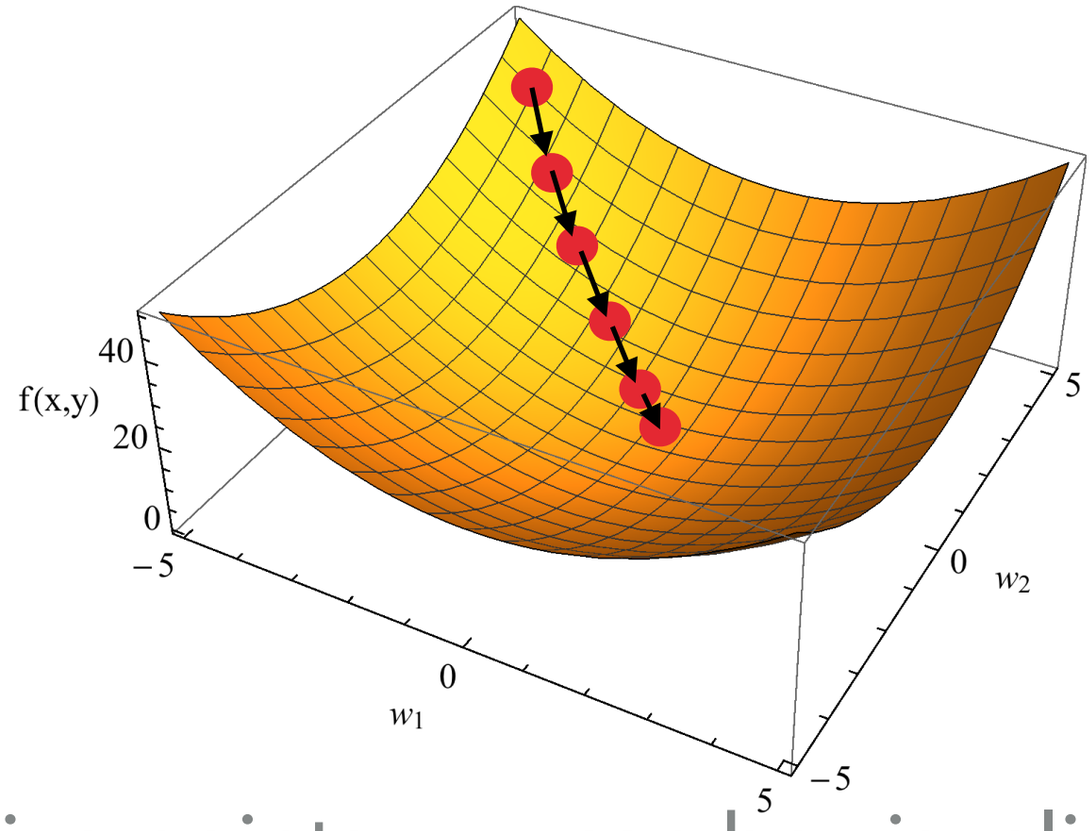
$$= w_i - \alpha \frac{dy}{dw}$$

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:

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

- where

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

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 i^{th} 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)

Queen Mary University of London
A. Bevan

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

HYPERPARAMETER OPTIMISATION: BACK PROPAGATION

- The parameters w and θ are updated using:

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

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

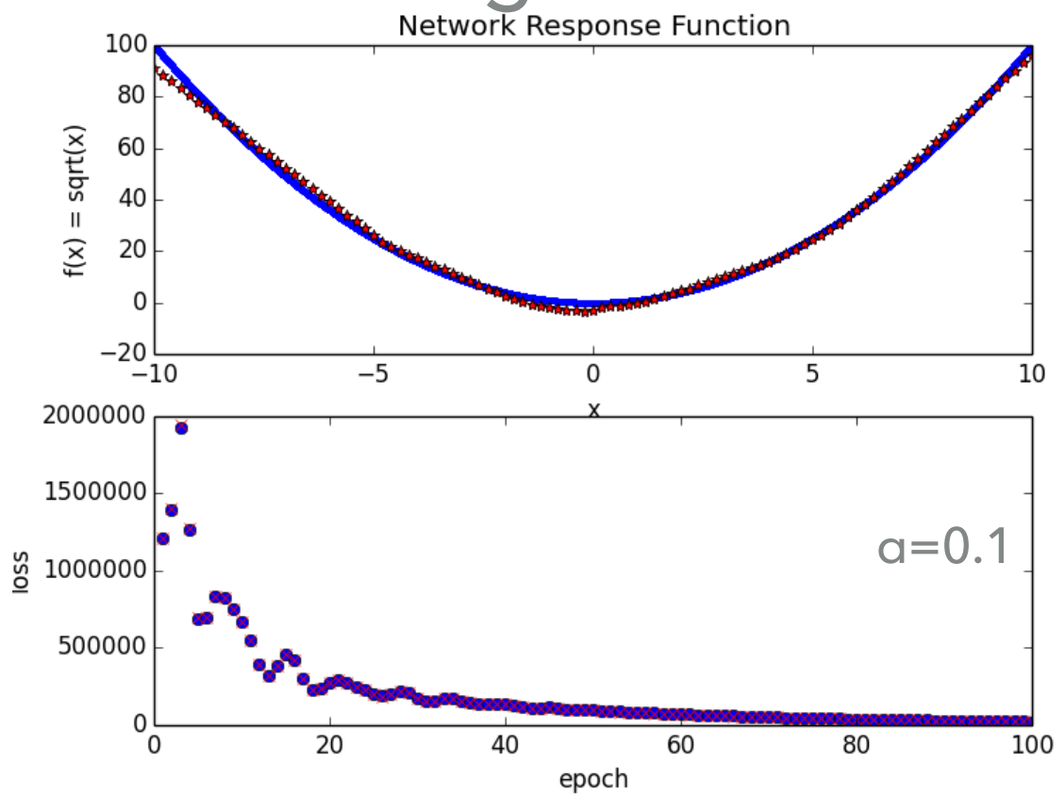
where α is the small positive learning rate

- 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.

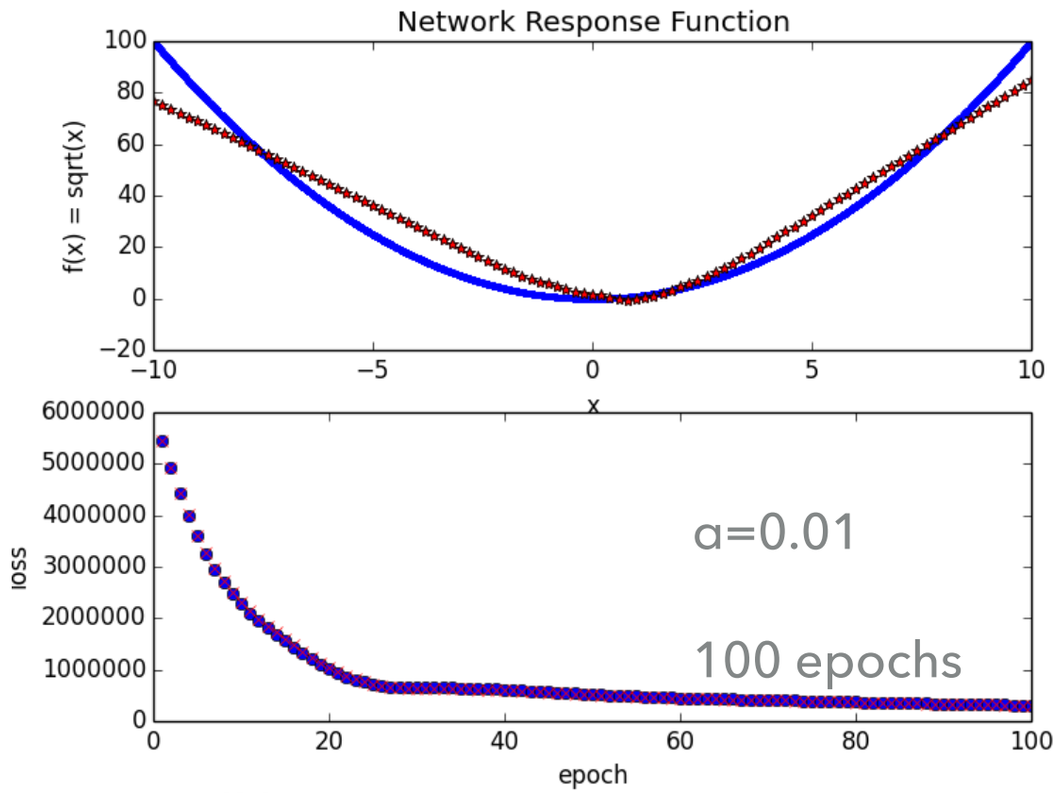
*Weights are randomly initialised for the whole network.

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.

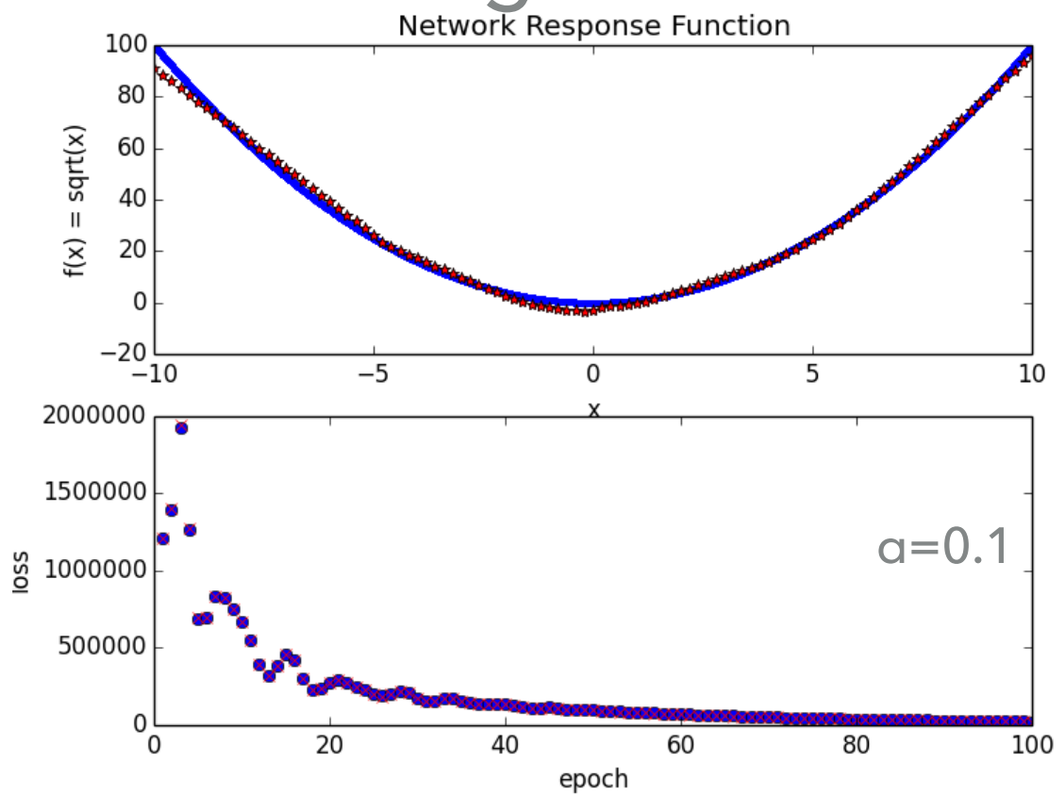


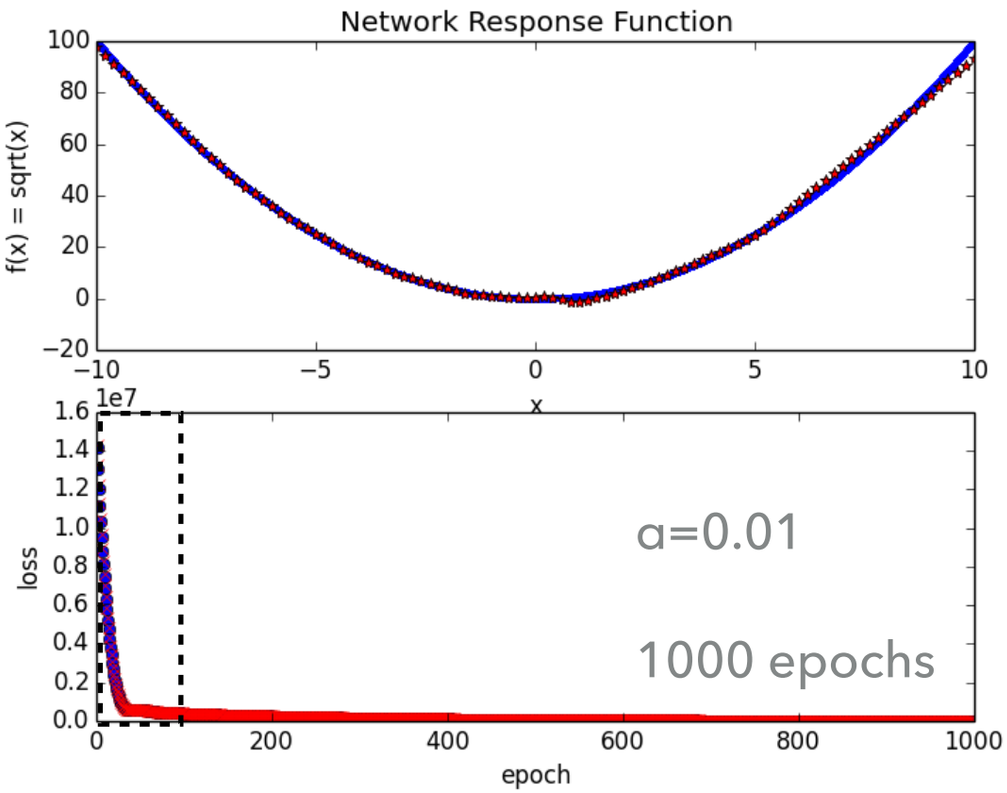
A small learning rate means the convergence takes much longer (a x10 reduction in the learning rate will require a x10 increase in optimisation steps)

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

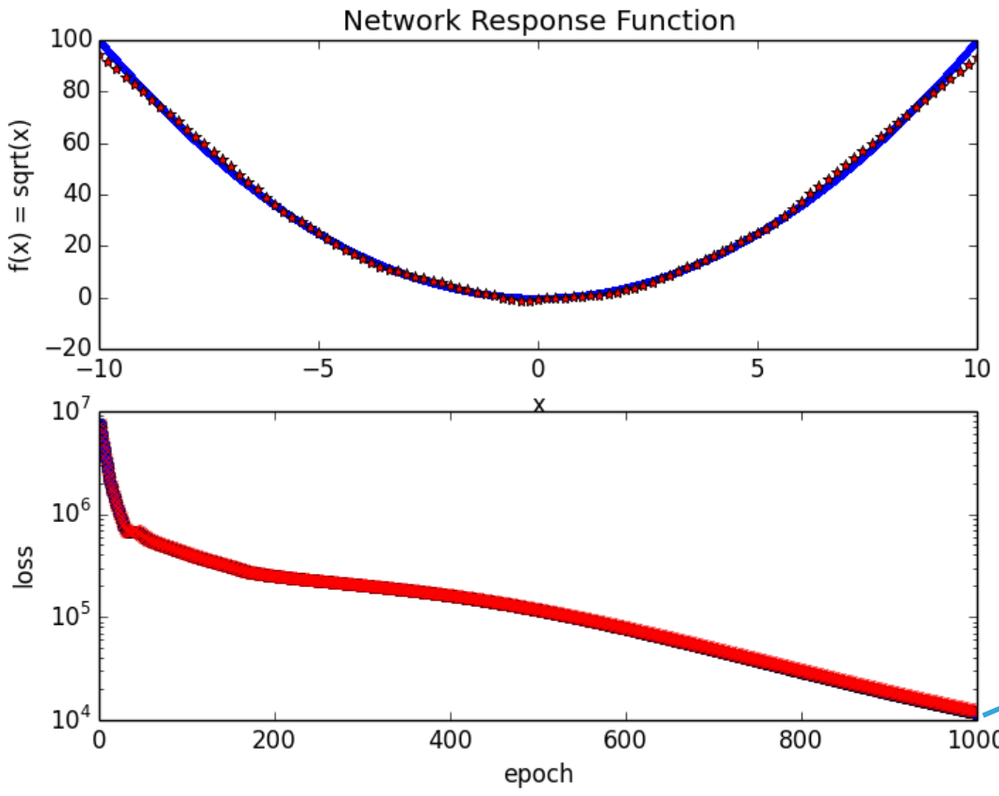
1. Conditions of convergence are well understood.
2. Many acceleration techniques (e.g. conjugate gradient) only operate in batch learning.
3. Theoretical analysis of the weight dynamics and convergence rates are simpler.

- 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.

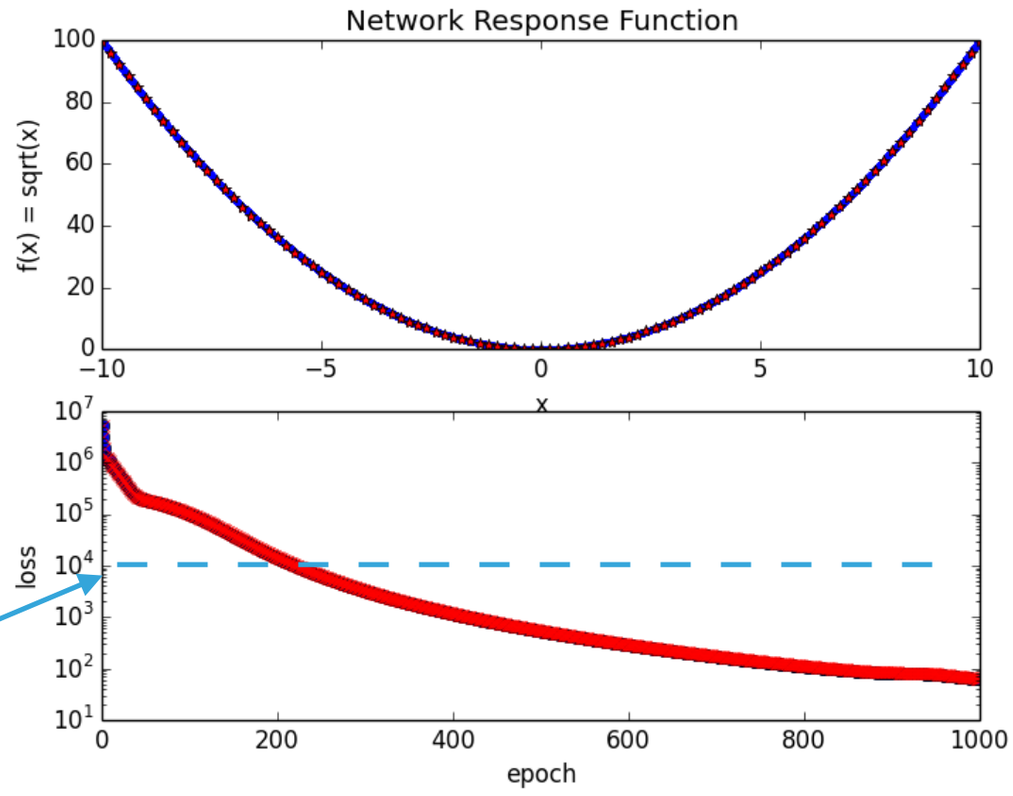
[1] LeCun et al., Efficient BackProp, Neural Networks Tricks of the Trade, Springer 1998

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-μ)/σ;
    - 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

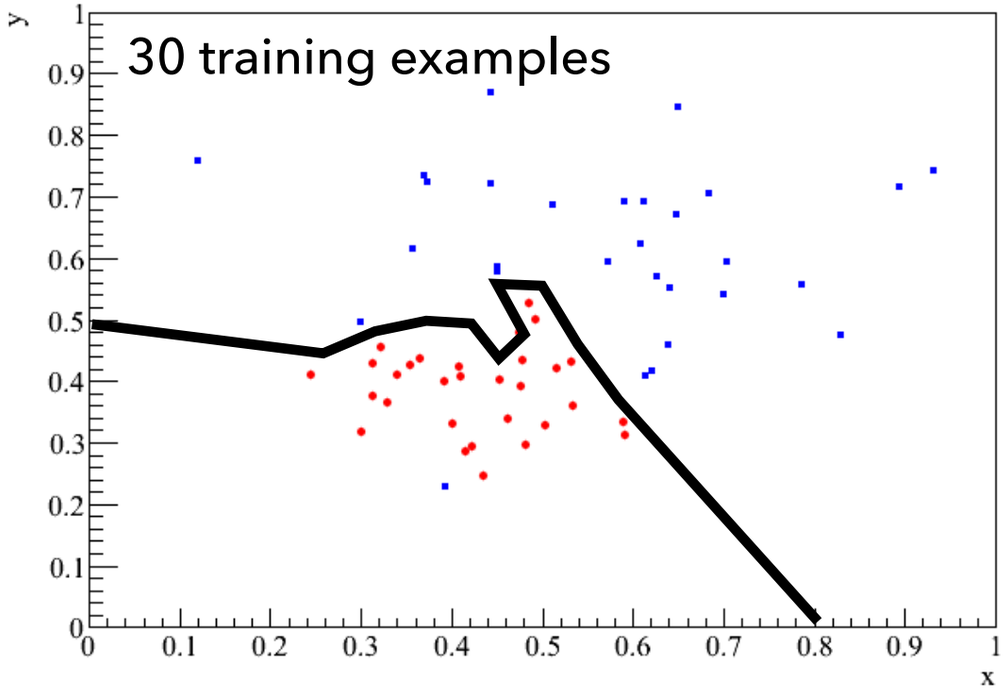
OVERTRAINING

- 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:



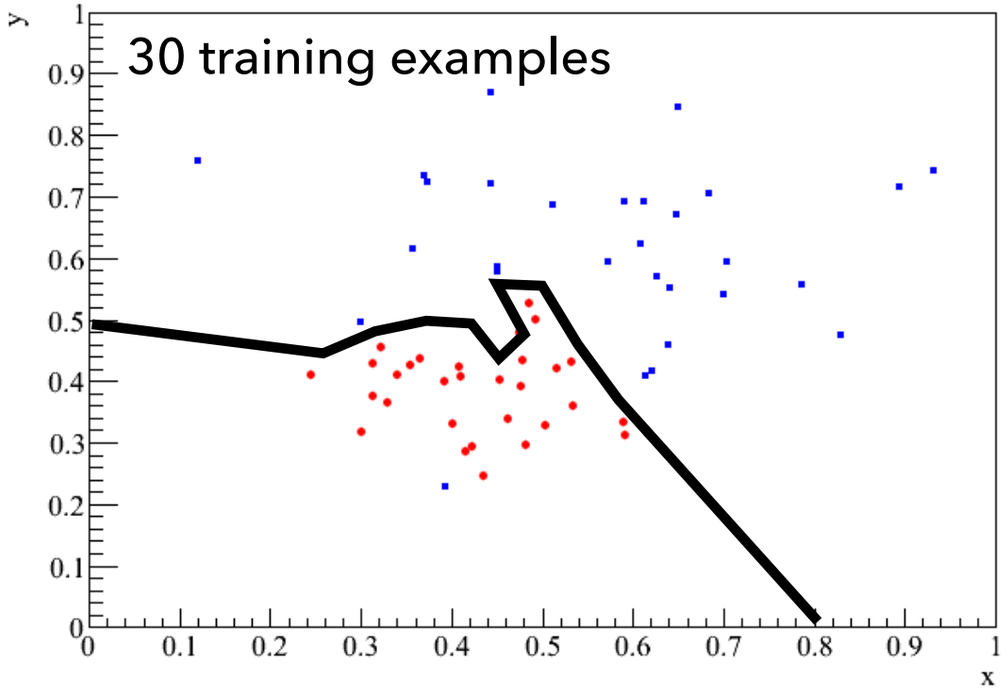
The decision boundary selected here does a good job of separating the red and blue dots.

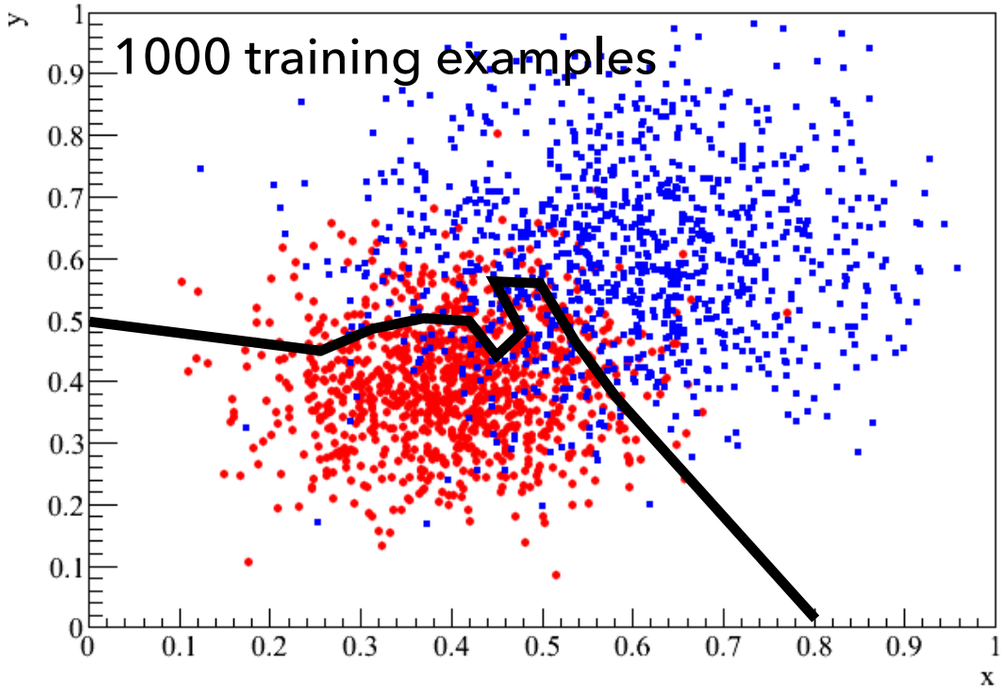
Boundaries like this can be obtained by training models on limited data samples. The accuracies can be impressive.

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

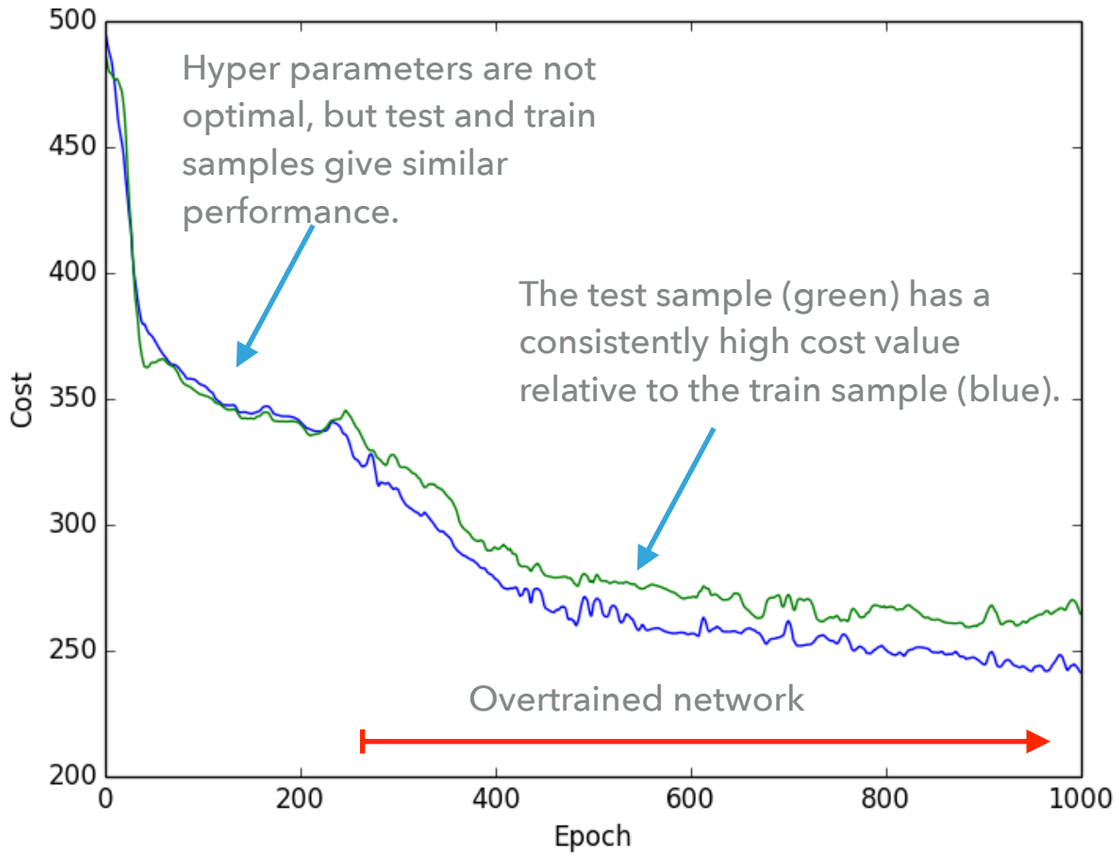
- One way to avoid tuning to statistical fluctuations in the data is to impose a training convergence criteria based on a data sample independent from the training set: a validation sample.
  - Use the cost evaluated for the training and validation samples to check to see if the HPs are over trained.
    - 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.

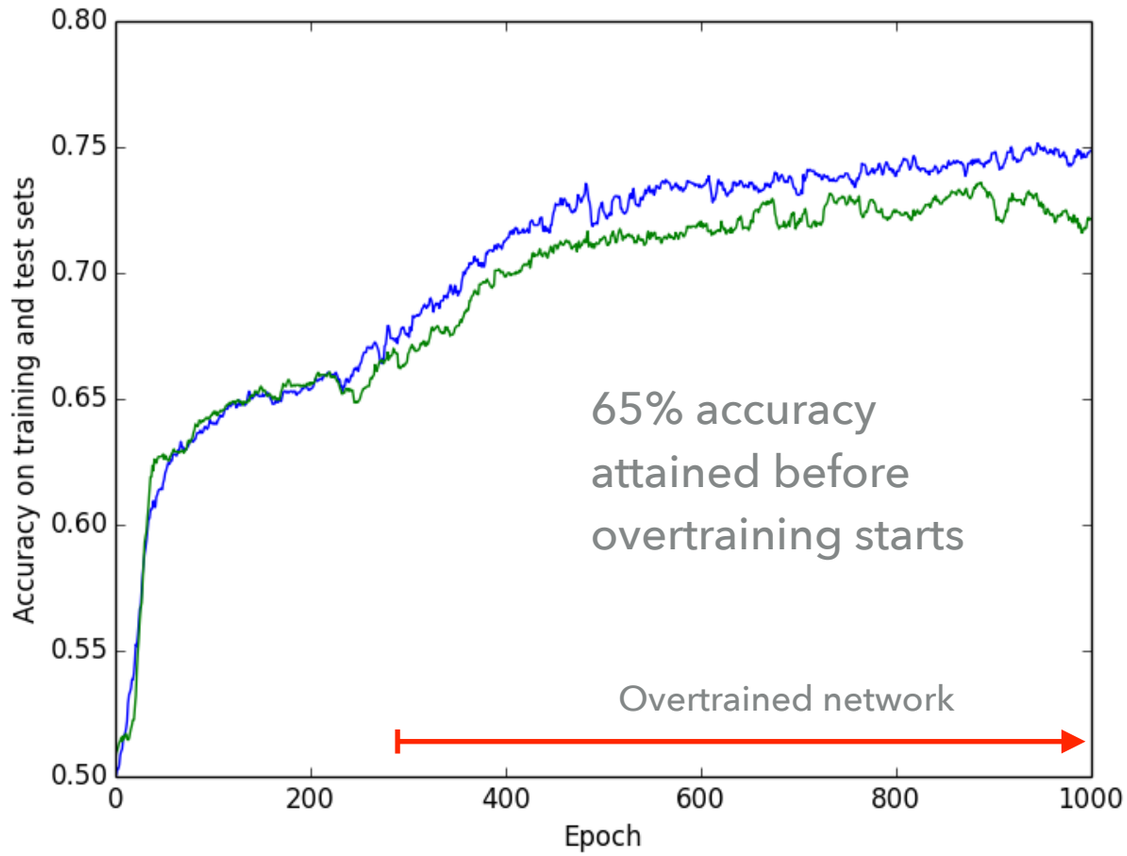
OVERTRAINING: TRAINING VALIDATION

- 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 $\Delta cost$ starts to increase for the validation sample.

OVERTRAINING: TRAINING VALIDATION: EXAMPLE HIGGS KAGGLE DATA

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





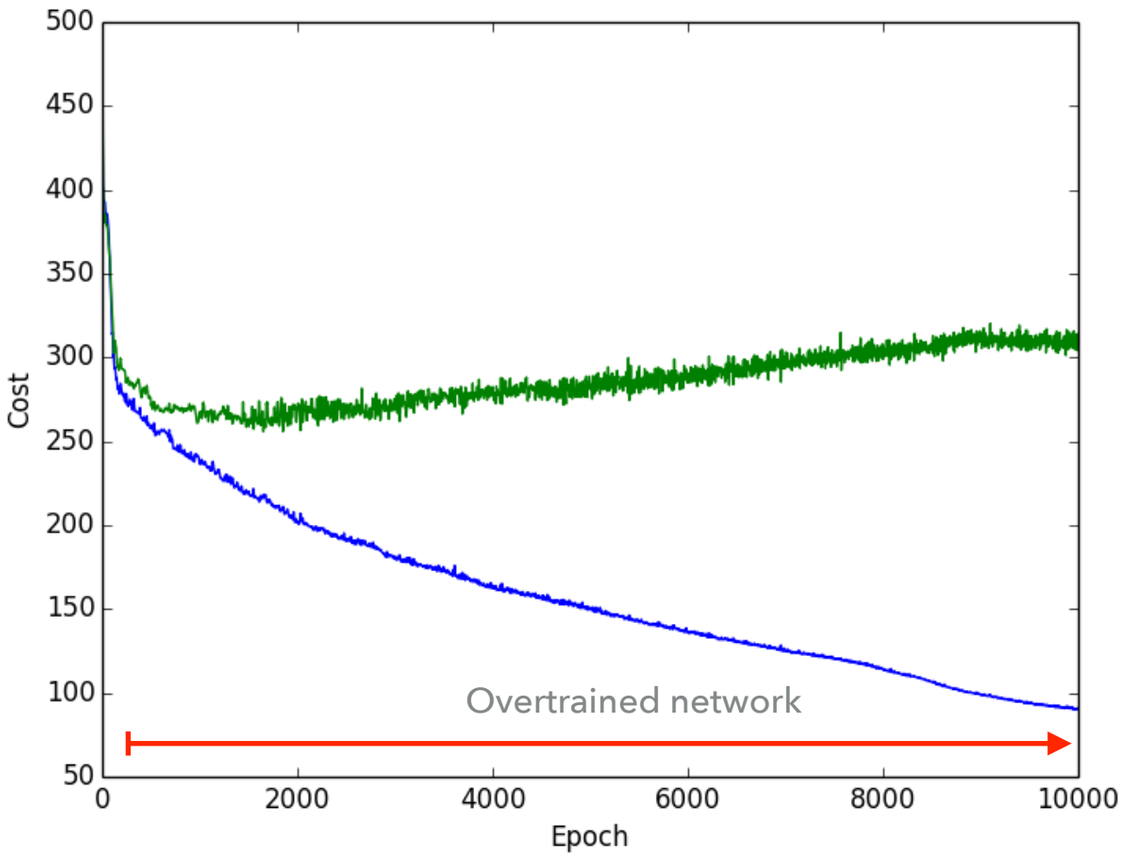
This example uses all features of the data set, but only 2000 test/train events with a learning rate of 0.001 and dropout is not being used. The network has a single layer with 256 nodes and a single output to classify if a given event is signal or background. There is no batch training used for this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

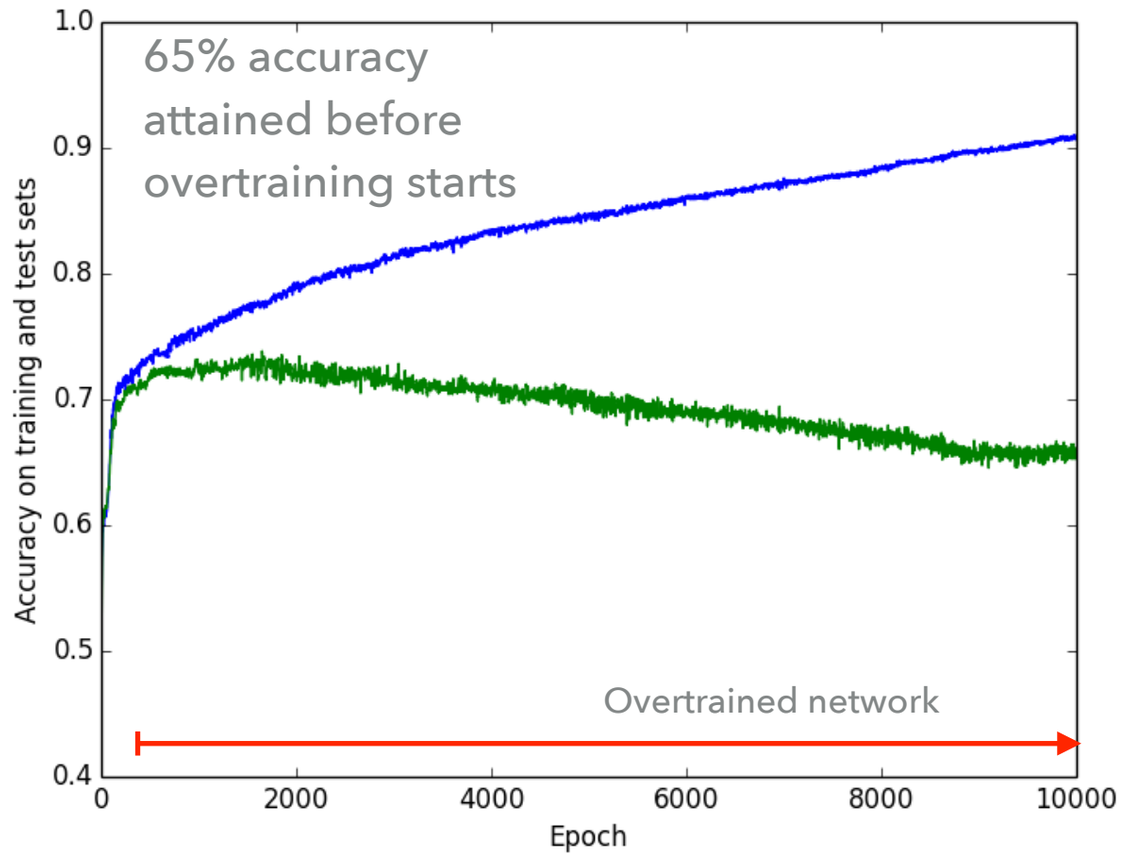
Queen Mary University of London
A. Bevan

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

OVERTRAINING: TRAINING VALIDATION: EXAMPLE HIGGS KAGGLE DATA

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





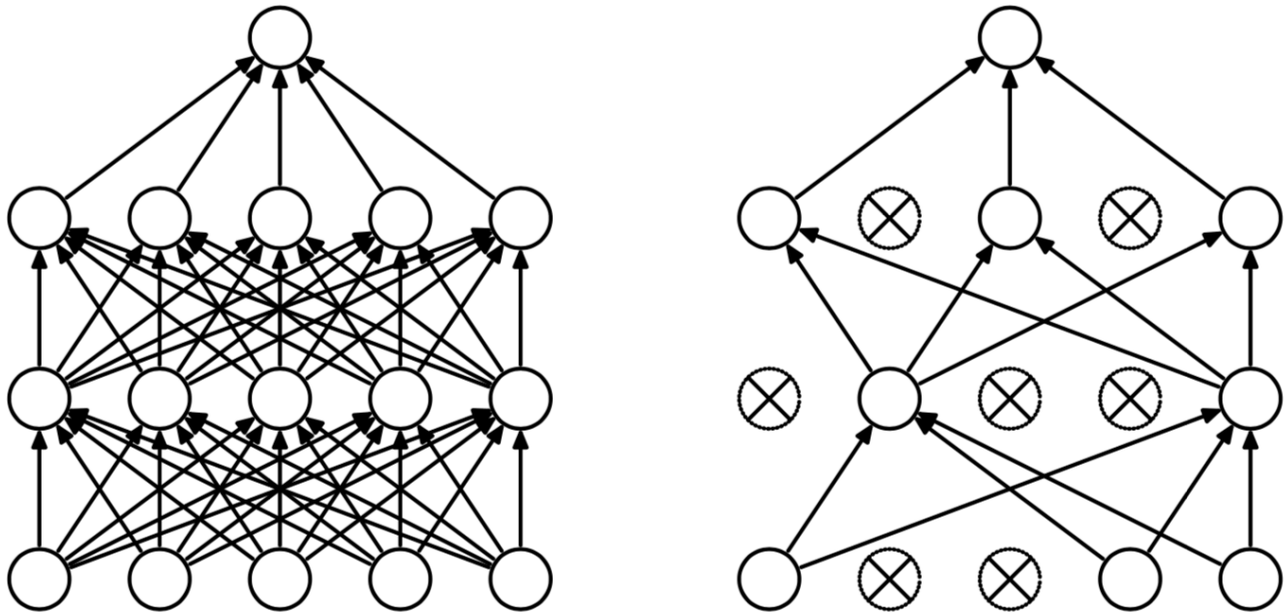
In this case the level of overtraining continues to increase and the difference in the performance of the train and test samples in terms of accuracy increases. After 10000 training cycles we have an accuracy of over 93% for the train sample, but less than 70% for the test sample. This is not a good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

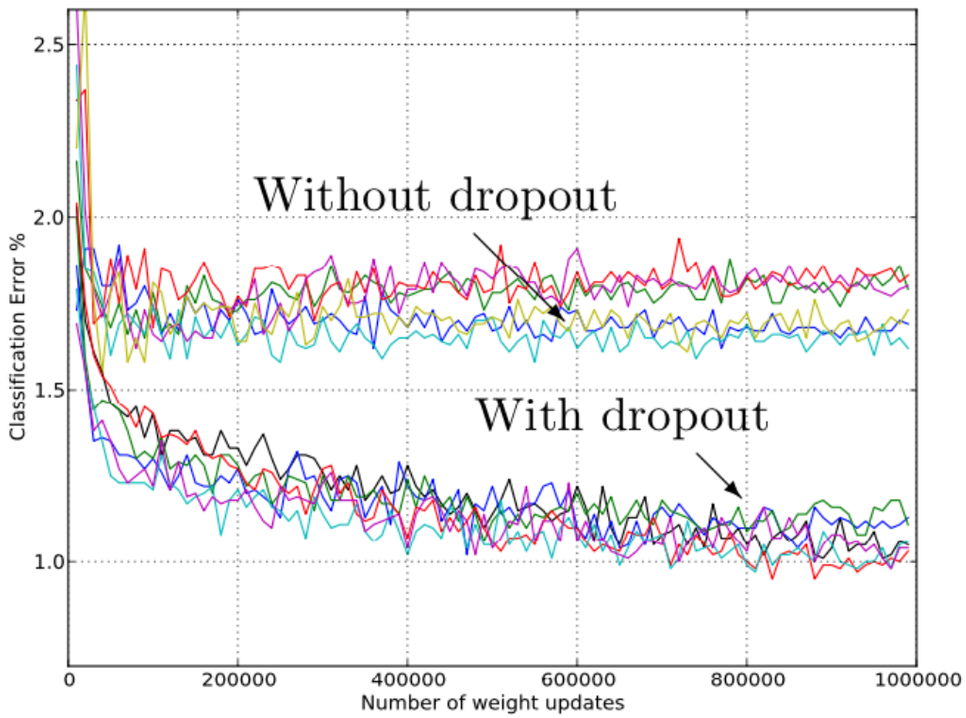
Dropout is used during training; when evaluating predictions with the validation or unseen data the full network of Fig. (a) is used.

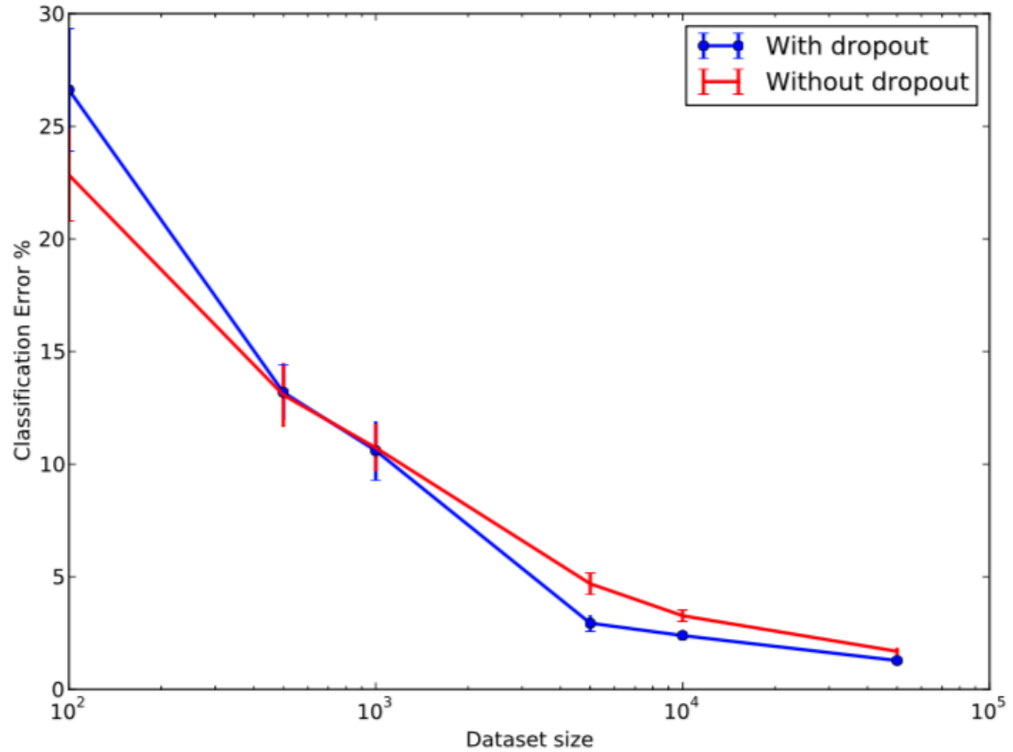
- That way the whole model will be effectively trained on a sub-sample of the data in the hope that the effect of statistical fluctuations will be limited.
- This does not remove the possibility that a model is overtrained, as with the previous discussion HP generalisation is promoted by using this method.

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).



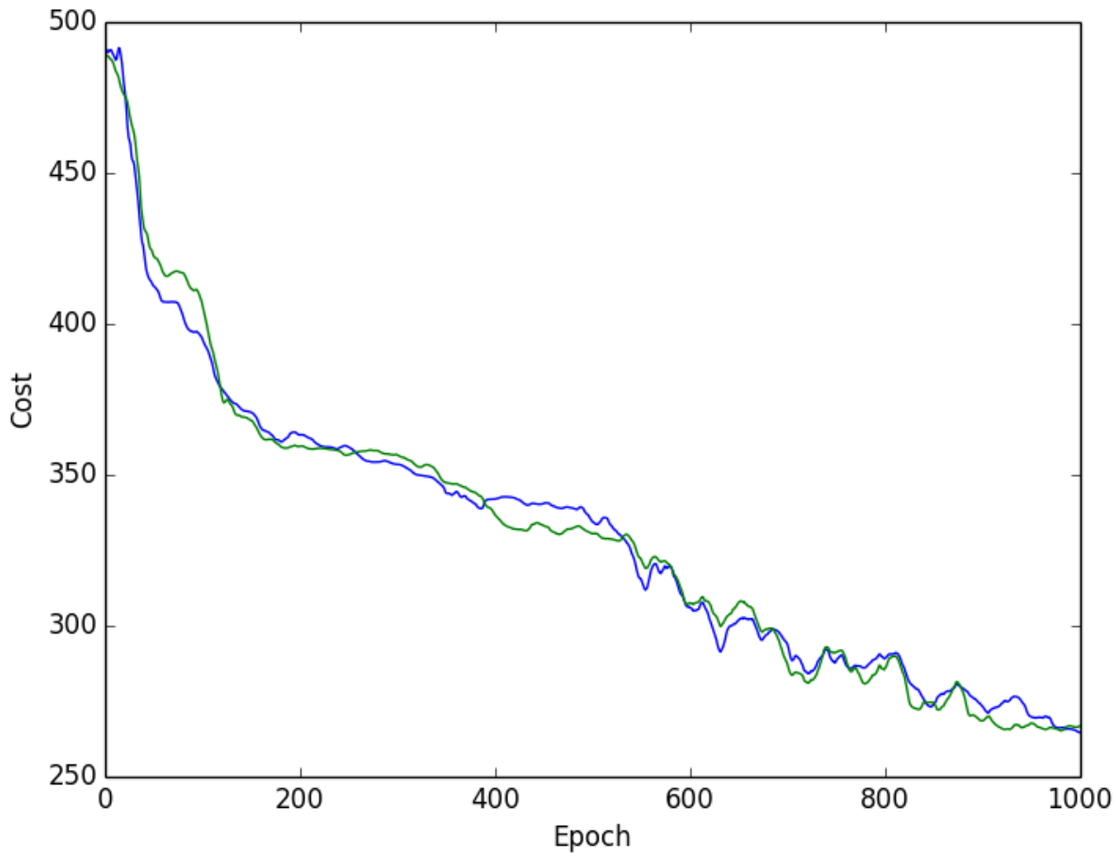


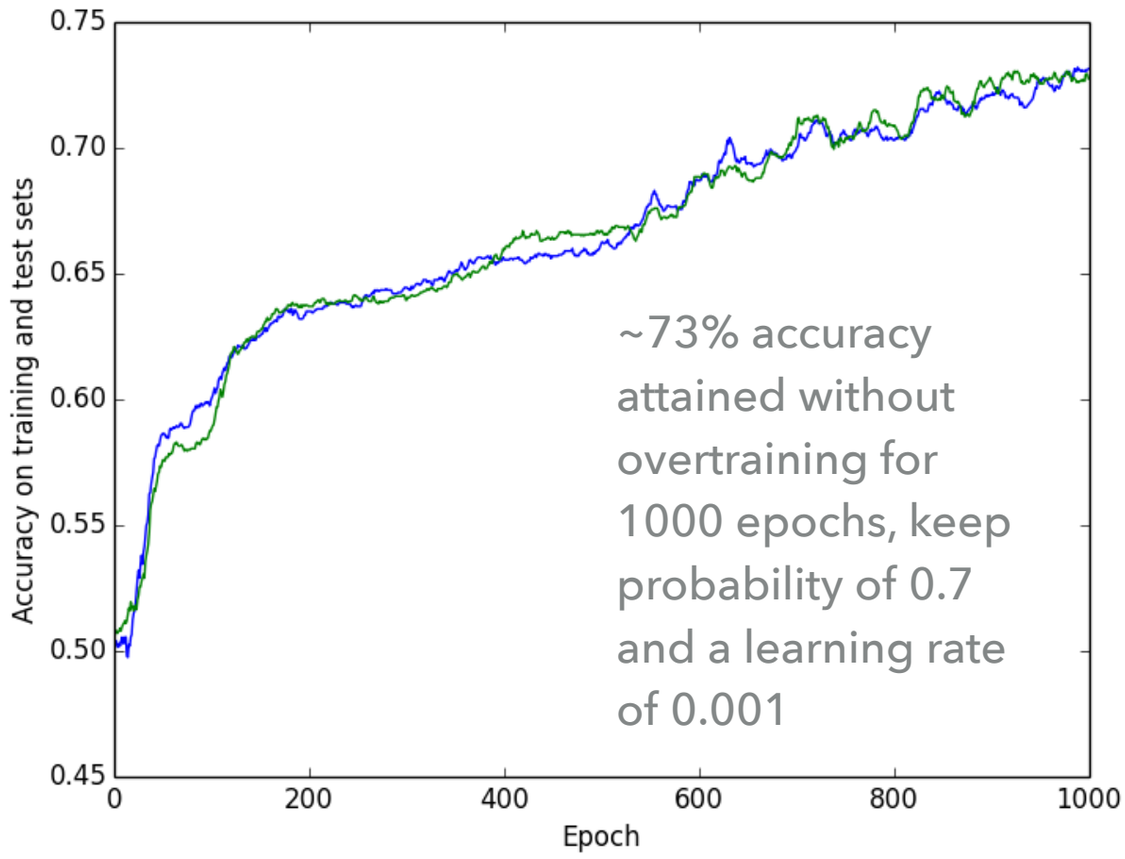
- Dropout can be detrimental for small training samples, however in general the results show that dropout is beneficial.
- For deep networks or typical training samples O(500) examples or more this technique is expected to be beneficial.

[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%.

Queen Mary University of London
A. Bevan

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

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.

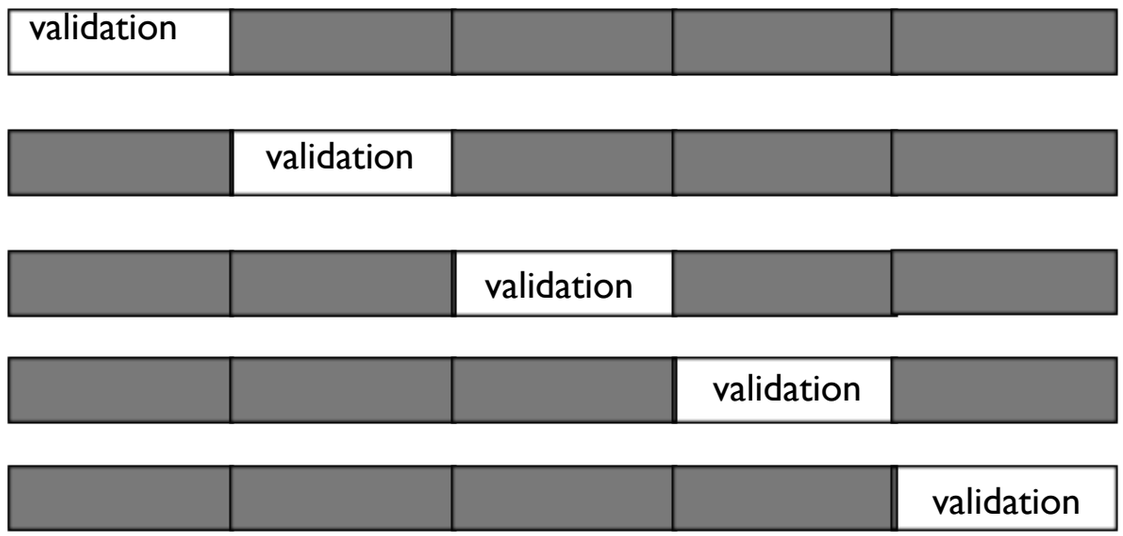
We can use cross validation to estimate the prediction error.

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.

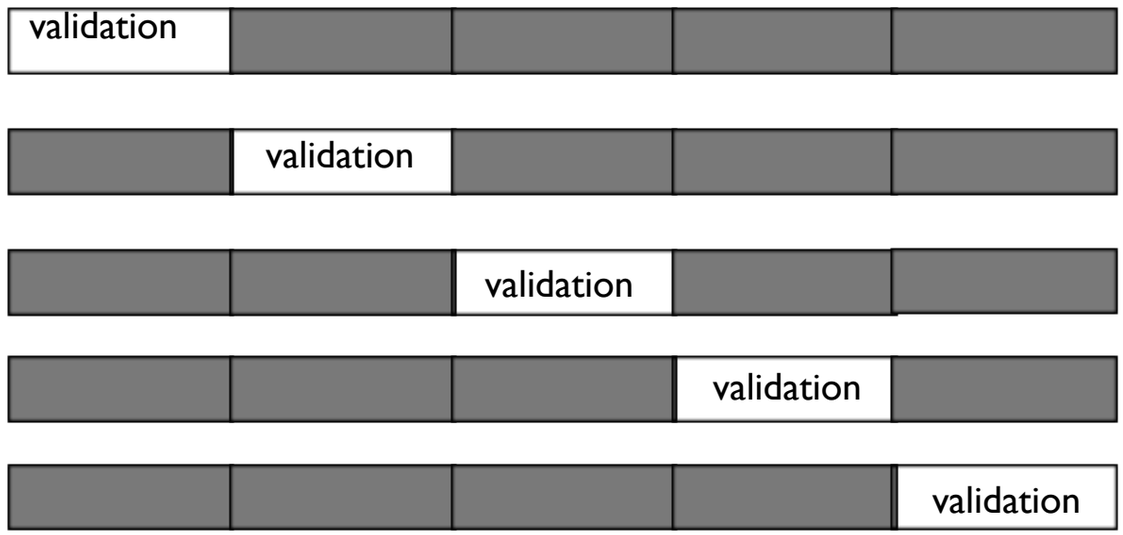


*Variants include the extremes of leave 1 out CV and Hold out CV as well as leave p-out CV. These involve reserving 1 example, 50% of examples and p examples for testing, and the remainder of data for training, respectively.

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.



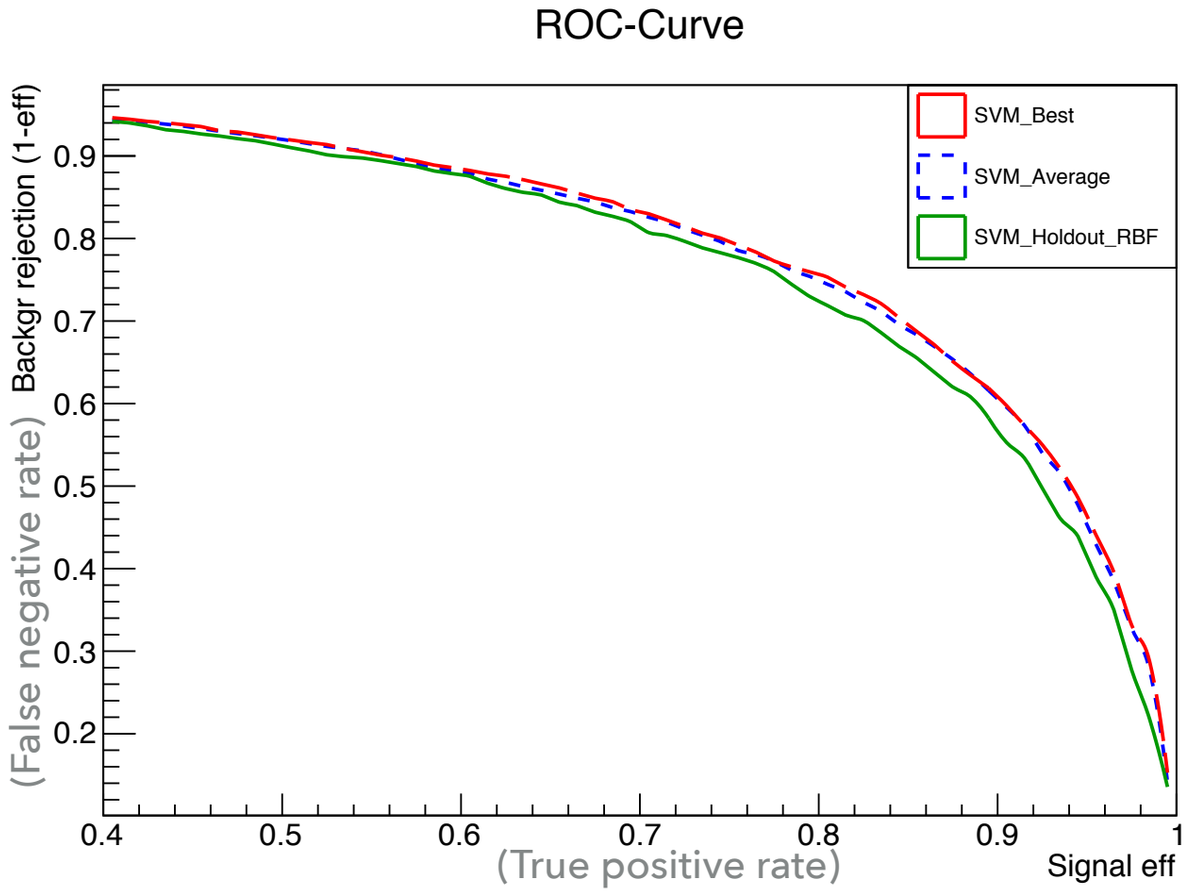
*Variants include the extremes of leave 1 out CV and Hold out CV as well as leave p-out CV. These involve reserving 1 example, 50% of examples and p examples for testing, and the remainder of data for training, respectively.

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).

MULTIPLE MINIMA

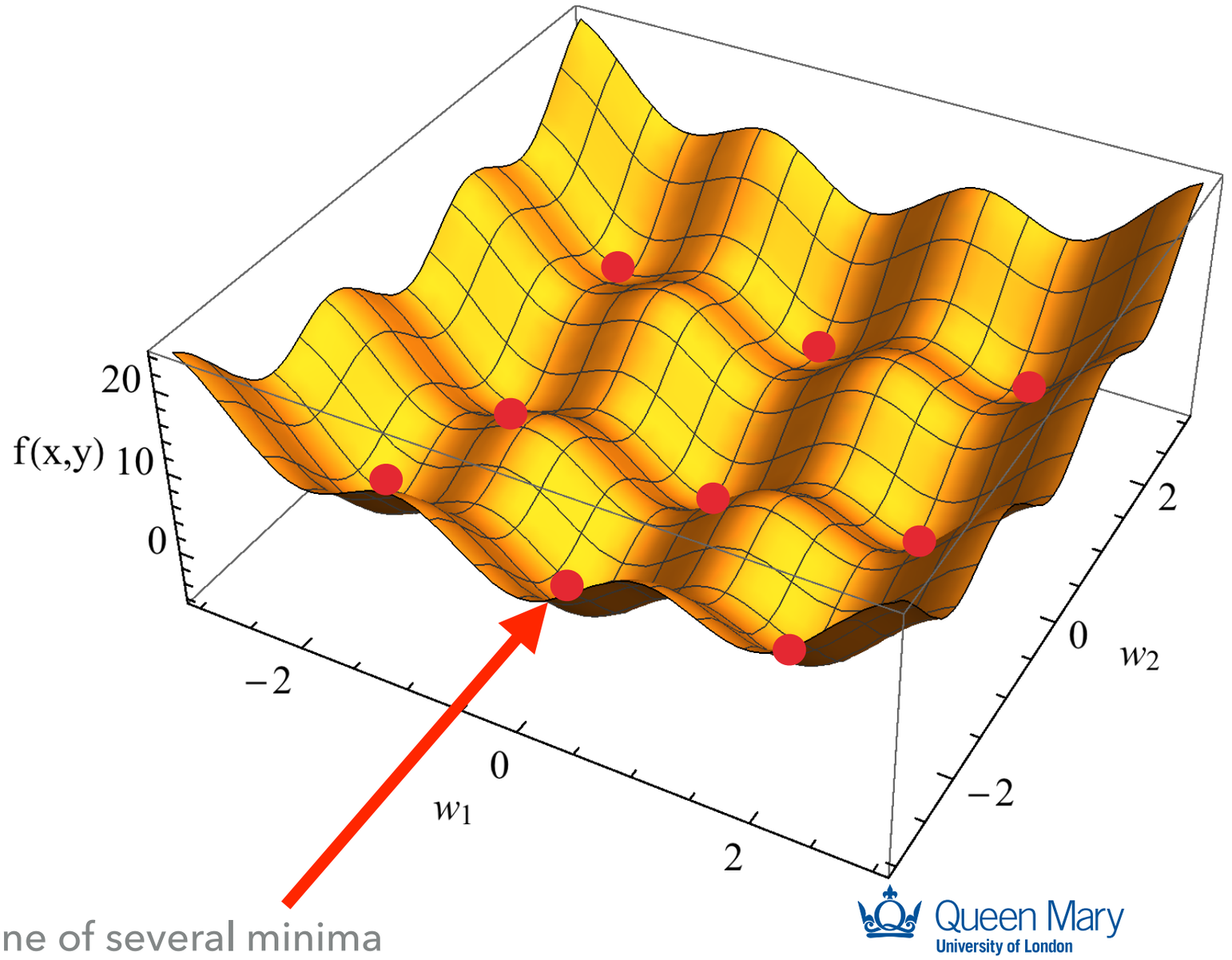
- Often more complication hyperspace optimisation problems are encountered, where there are multiple minima.

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?



MULTIPLE MINIMA

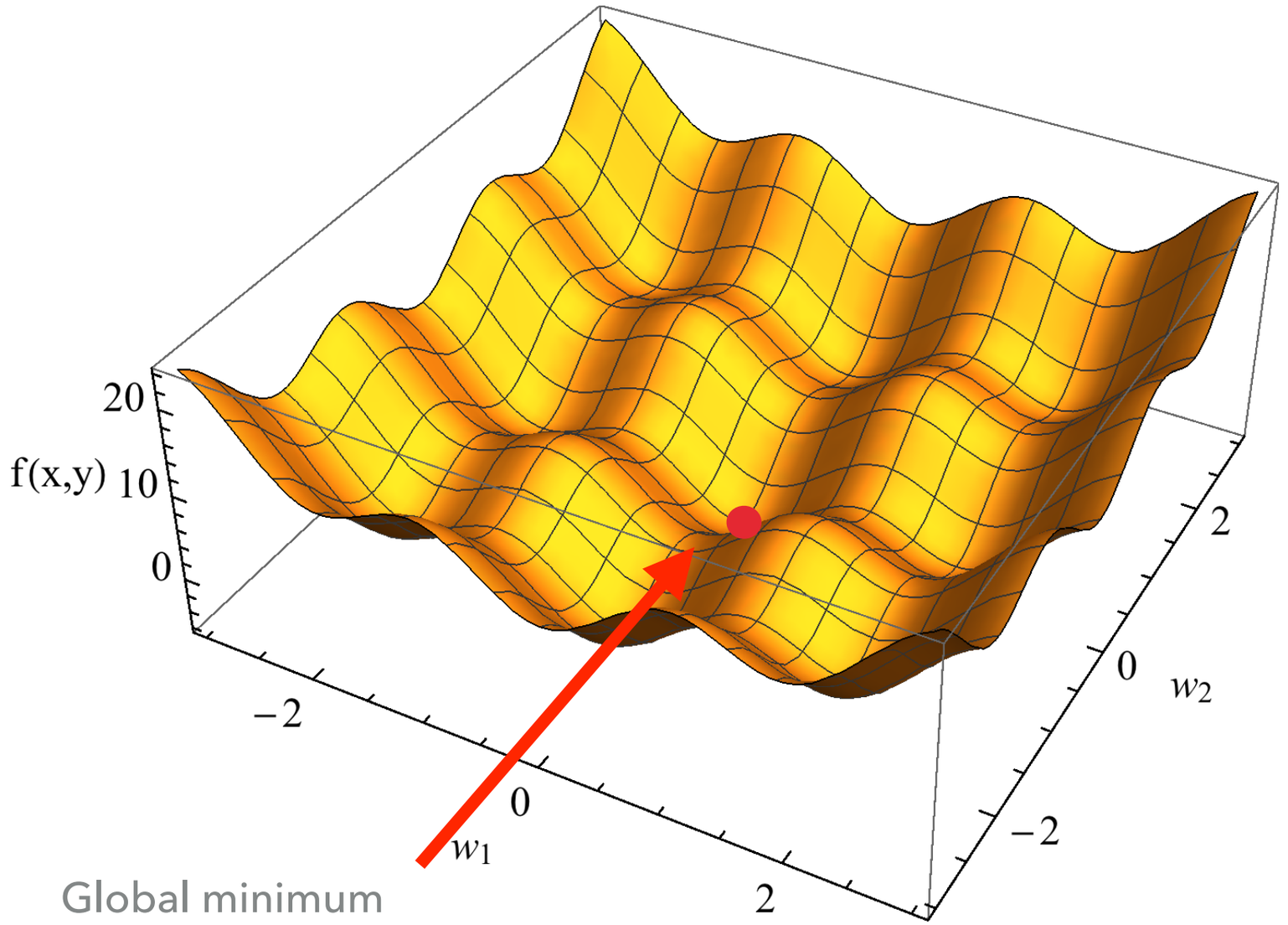
- Often more complication hyperspace optimisation problems are encountered, where there are multiple minima.

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.

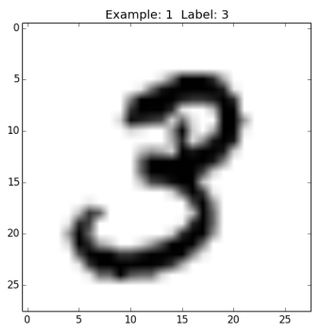
Sometimes the minima are degenerate, or near degenerate.

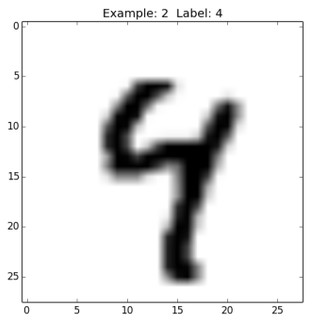
How do we know we have converged on the global minimum?

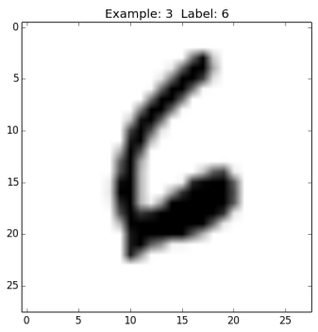


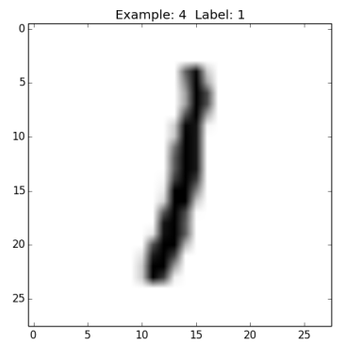
EXAMPLES: MNIST

- Consider the MNIST data.









- Each example is a 28x28 array of pixels (784 features).
- Each pixel is an integer between 0 and 255.
  - Rescale [0, 255] -> [0, 1]
- How can we classify these numbers?

[1] Neural Computation, Volume 22, Number 12, December 2010
http://yann.lecun.com/exdb/mnist/

MULTICLASS CLASSIFICATION

- Set the output layer to have multiple nodes; each node is tasked with making a single classificaiton of an example being of one type or not.
- The N_{type} = 10 perceptrons are used to make the following decisions:
  - The number 1 vs not the number 1
  - The number 2 vs not the number 2
  - The number 3 vs not the number 3
  - The number 4 vs not the number 4
  - The number 5 vs not the number 5
  - The number 6 vs not the number 6
  - The number 7 vs not the number 7
  - The number 8 vs not the number 8
  - The number 9 vs not the number 9
  - The number 0 vs not the number 0

For those with a statistical background, this is like a null hypothesis and an alternative hypothesis.

The null hypothesis provides a specific response/expectation.

The alternative hypothesis is the complement of the null.

In this context you classify an example as a specific type, or you provide a decision that it is not that type.

We will see more of the MNIST data when talking about convolutional neural networks.

http://yann.lecun.com/exdb/mnist/

MULTICLASS CLASSIFICATION

- An alternative representation is to use a softmax activation function to encode the 10 outputs in a single function.

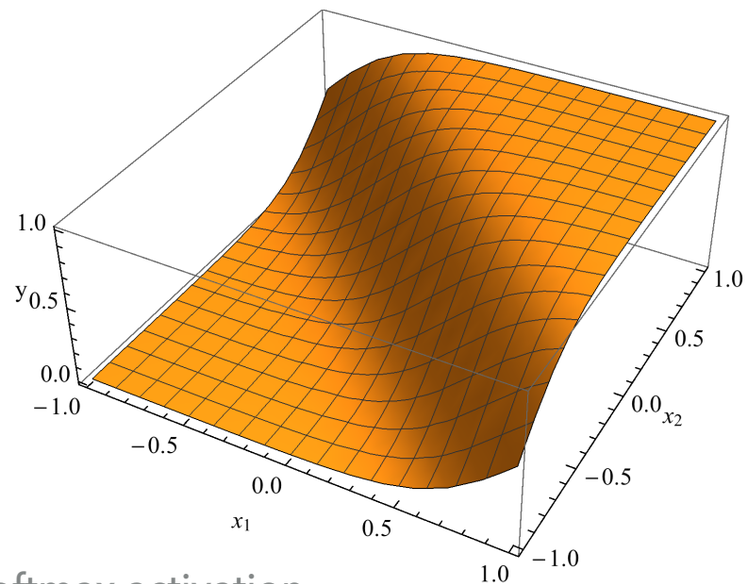
$$f_j(x) = \frac{e^{w_j^T x}}{\sum_{i=1}^{N} e^{w_i^T x}}$$

i is the index for the output classification type

The score for the i^{th} output is normalised by the sum of outputs.

$$f(x) = \frac{1}{\sum_{i=1}^{N} e^{w_i^T x}} \begin{bmatrix} e^{w_1^T x} \\ e^{w_2^T x} \\ \vdots \\ e^{w_N^T x} \end{bmatrix}$$

$f_i(x)$ is normalised to lie in the range [0, 1]



Example of the i^{th} output of a softmax activation function for a 2D input feature space.

- Can convert output to {0, 1}.

http://yann.lecun.com/exdb/mnist/

SUMMARY

- Neural networks are built on perceptrons:
  - Inspired by desire to understand the biological function of the eye and how we perceive based on visual input.
  - The output threshold of a perceptron can be all or nothing, or be continuous between those extremes.
- Artificial neural networks are constructed from perceptrons.
- Perceptron/network weights need to be determined via some optimisation process, called training.
- … This leads us on to issues related to training and toward deep neural networks.

SUGGESTED READING

- The suggestions made here are for some of the standard text books on the subject. These require a higher level of math than we use in this course, but may have less emphasis on the practical application of the methods we discuss here as a consequence.

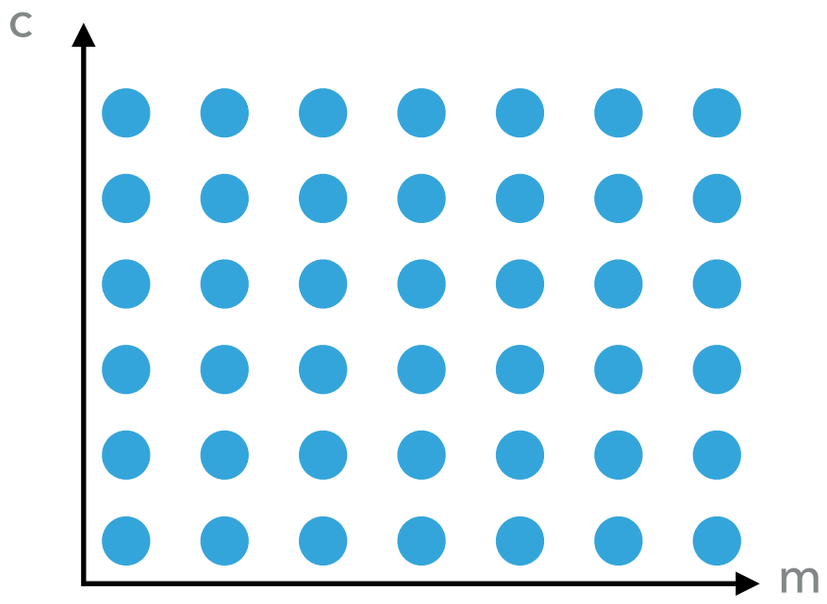
- MacKay: *Information theory, inference and learning algorithms*
  - Chapter: V
- C. Bishop: *Neural Networks for Pattern Recognition*
  - Chapters: 3 and 4
- C. Bishop: *Pattern Recognition and Machine Learning*
  - Chapter: 5
- T. Hastie, R. Tibshirani, J. Friedman, *Elements of statistical learning*
  - Chapter: 11

- In addition to books, you may find interesting articles posted on the preprint archive: https://arxiv.org. There are several useful categories as part of the Computing Research Repository (CoRR) related to this course including *Artificial Intelligence*. Note that these are research papers, so again they will generally have a strong mathematical content.

APPENDICES

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.

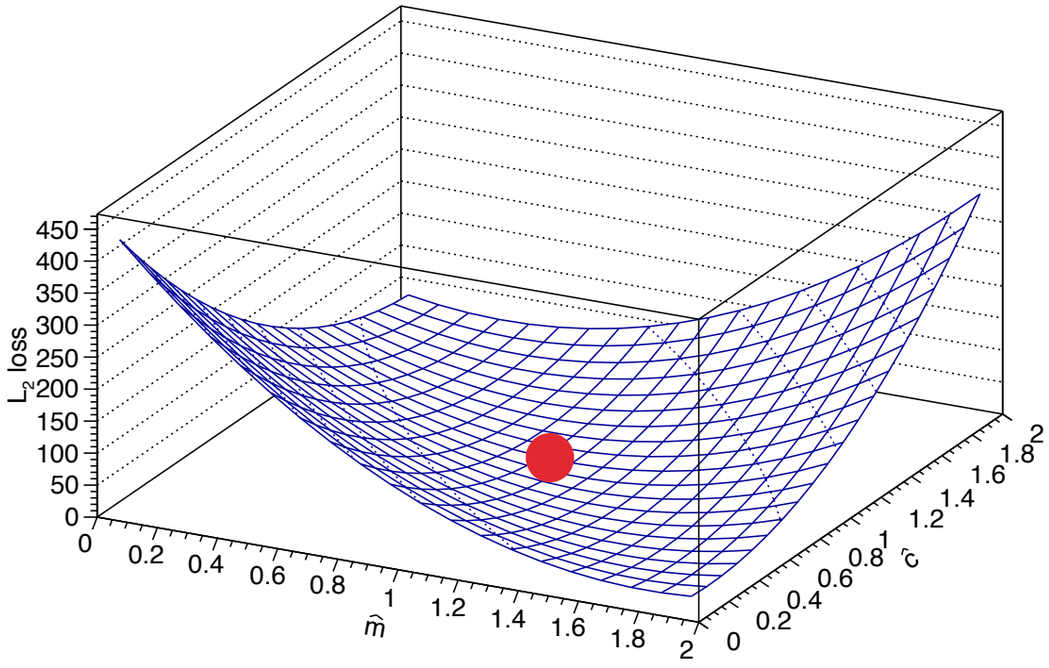


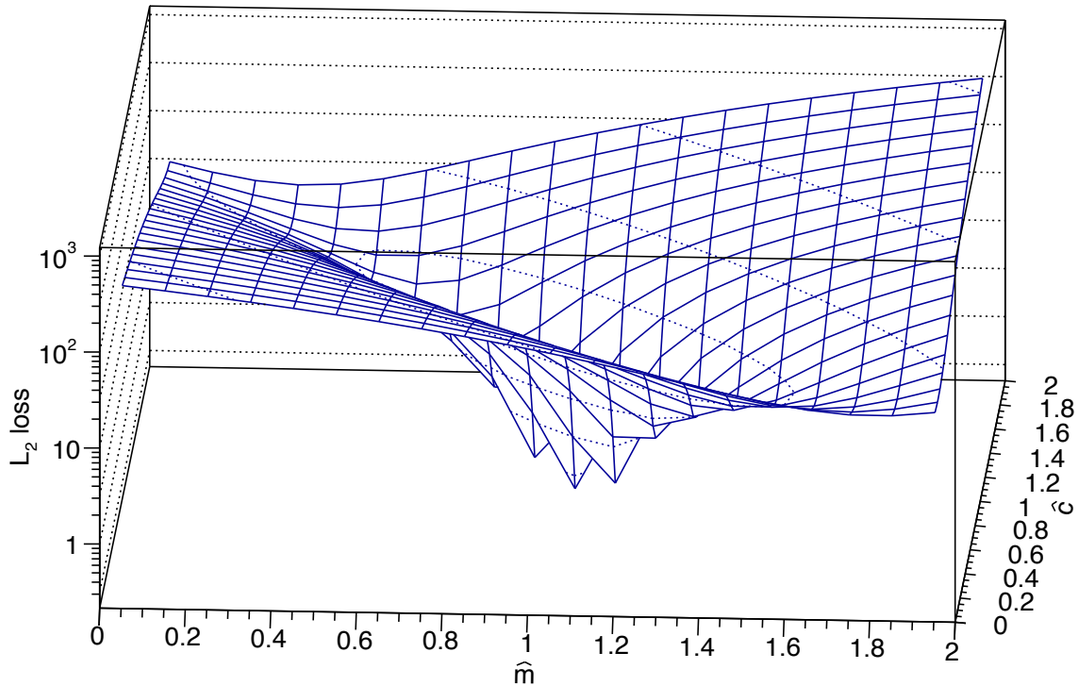
This type of parameter search is often used for Support Vector Machine HPs (kernel function parameters and cost). e.g. see libsvm.

The method does not scale to large numbers of parameters. It suffers from the curse of dimensionality.

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(θ) 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.

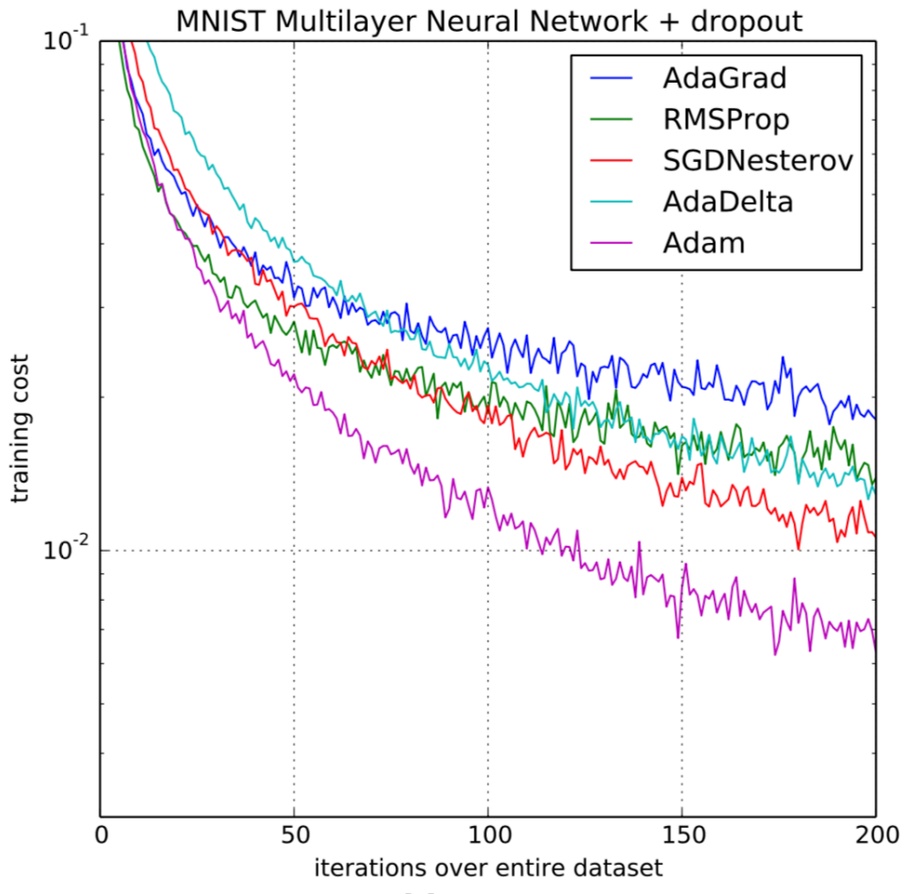
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ 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.

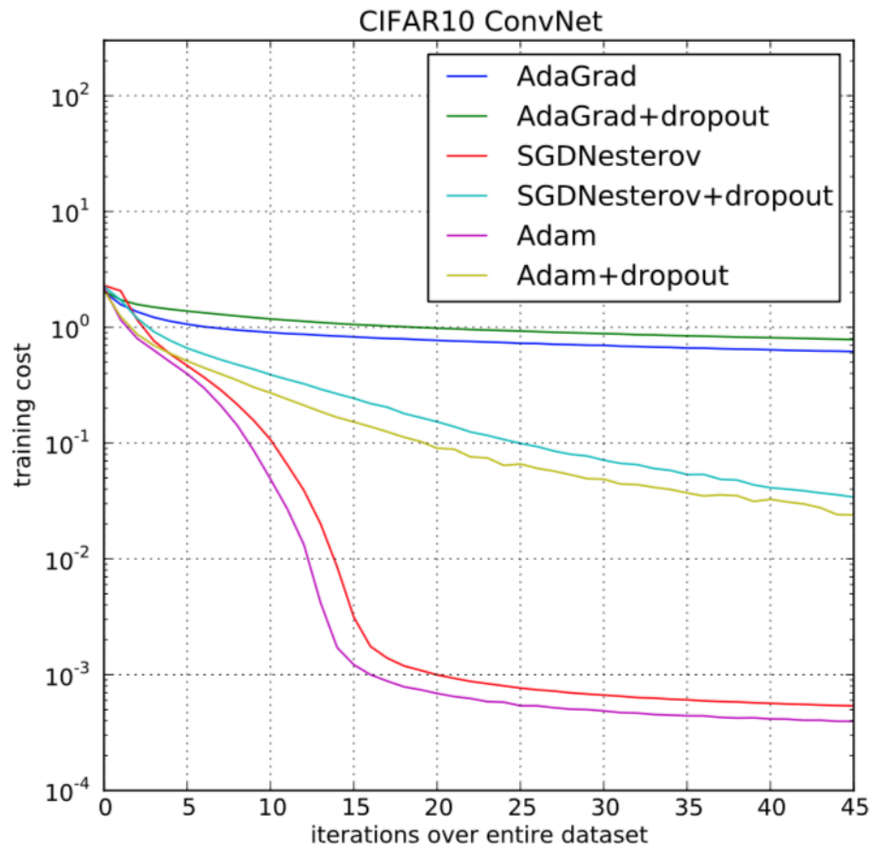
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd 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