PRACTICAL MACHINE LEARNING

OPTIMISATION AND REGRESSION

Queen Mary
University of London

THIS COURSE – 4 WORKSHOPS (LECTURE+EXERCISES)

- Lecture 1 (Monday) Introduction to machine learning with neural networks and linear regression
- Lecture 2 (Tuesday) Optimisation and non-linear regression with neural networks
- Lecture 3 (Wednesday) Classification and convolutional neural networks for image classification
- Lecture 4 (Thursday) Robustness and adversarial examples to image classification problems

WHO WE ARE

Dr Alex Booth [he/him]

Dr Linda Cremonesi [she/her]

Dr Abbey Waldron [she/her]

LECTURE PLAN

- Parameter optimisation
- Supervised learning
  - Loss functions
- Gradient descent
  - Adam optimiser
    - Multiple minima
- Over fitting
- Regression
- Deep Learning
- Summary

HYPERPARAMETER OPTIMISATION

- Models have parameters that are required to fix the response function.
- The set of parameters forms a parameter space.
- The purpose of optimisation is to select a point in parameter space 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.

PARAMETER OPTIMISATION

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

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

PARAMETER OPTIMISATION

- Neural networks have a lot of weights. Deep networks, especially CNNs can have *millions* of weights to optimise.
  - This requires appropriate computing resource.
  - It also requires appropriately efficient methods for parameter optimisation.
  - What is acceptable for an optimisation of 10 or 100 weights will not generally scale well to a problem with 10^3 - 10^6 weights.
- The more parameters to determine the more data is required to obtain a generalisable solution for the weights.
  - By generalisable we mean that the model defined using a set of parameters 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.

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 weights in order to minimise the loss function.
- The loss function is a function of:
  - labels (ground truth)
  - Weights
  - activation functions
  - architecture of the network (arrangement of perceptrons)

SUPERVISED LEARNING: LOSS FUNCTIONS

- There are a number of different types of loss function that are commonly used.
- We will use the mean squared error (MSE) loss function based on the sum over training examples of

$$\varepsilon_i = (y_i - t_i)^2$$

- normalised by the number of examples, N.

$\boldsymbol{\varepsilon_i}$ is the loss function contribution for the i^{th} example given the model or perceptron output $\mathbf{y_i}$ and the true target label value $\mathbf{t_i}$. See `tf.losses.mean_squared_error`.

Note: MSE/2 is called the L2 loss function; See `tf.nn.l2_loss`.

SUPERVISED LEARNING: MINI-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 mini-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

SUPERVISED LEARNING: 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 global minima.
- Often results in better optimisation performance than batch learning.

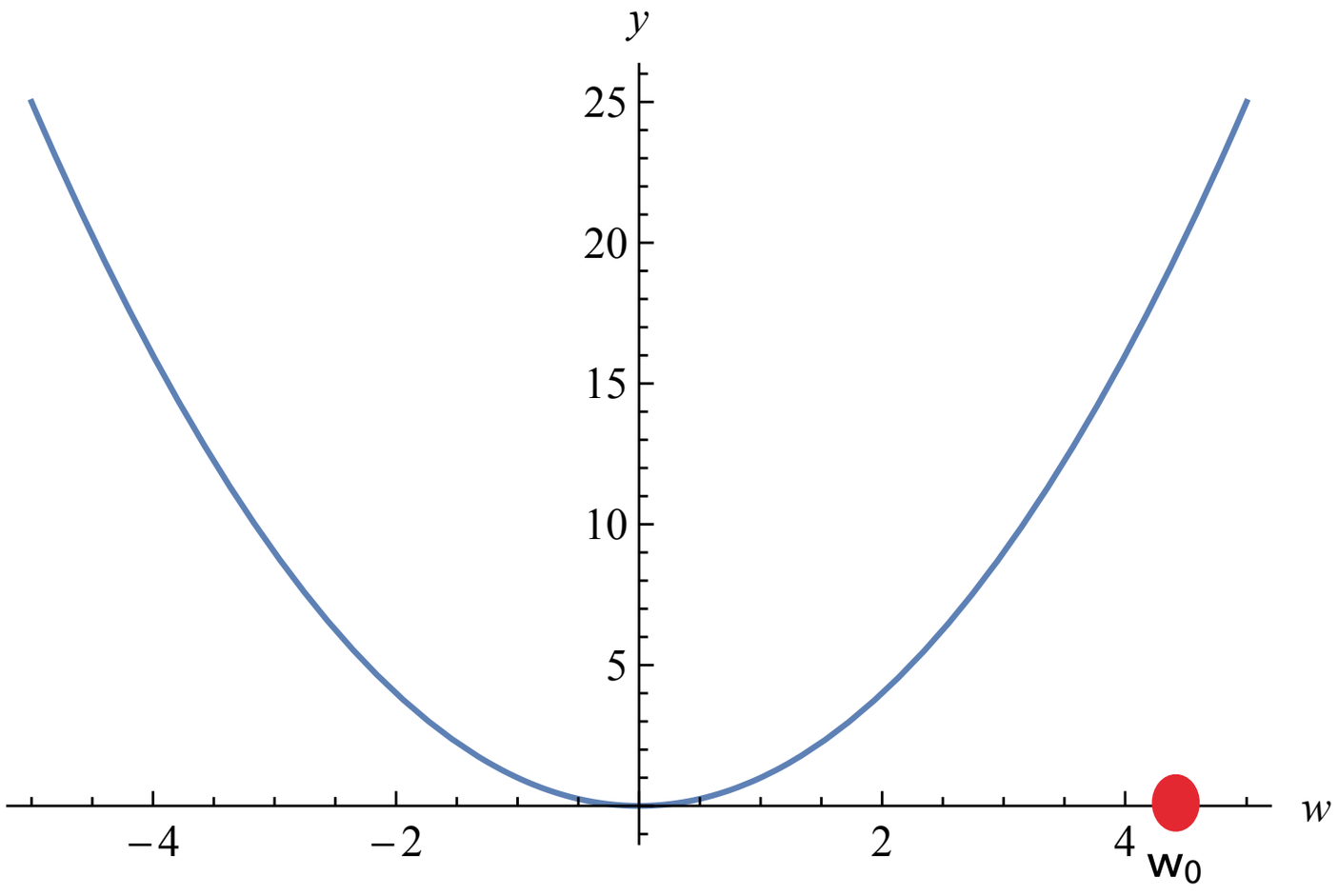
[1] LeCun et al., Efficient BackProp, Neural Networks Tricks of the Trade, Springer 1998

GRADIENT DESCENT

- Newtonian gradient descent
- Adam optimiser

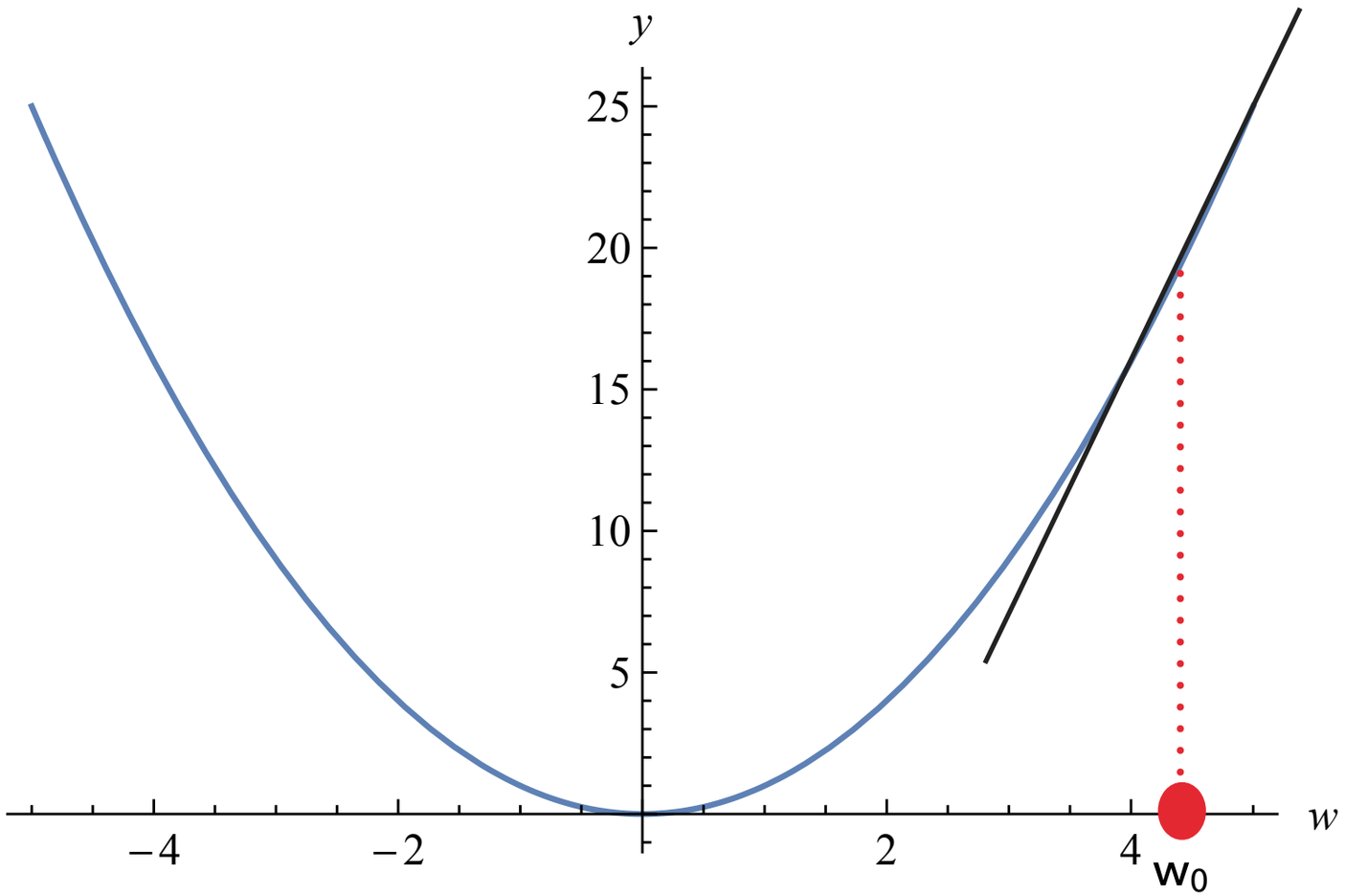
GRADIENT DESCENT

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



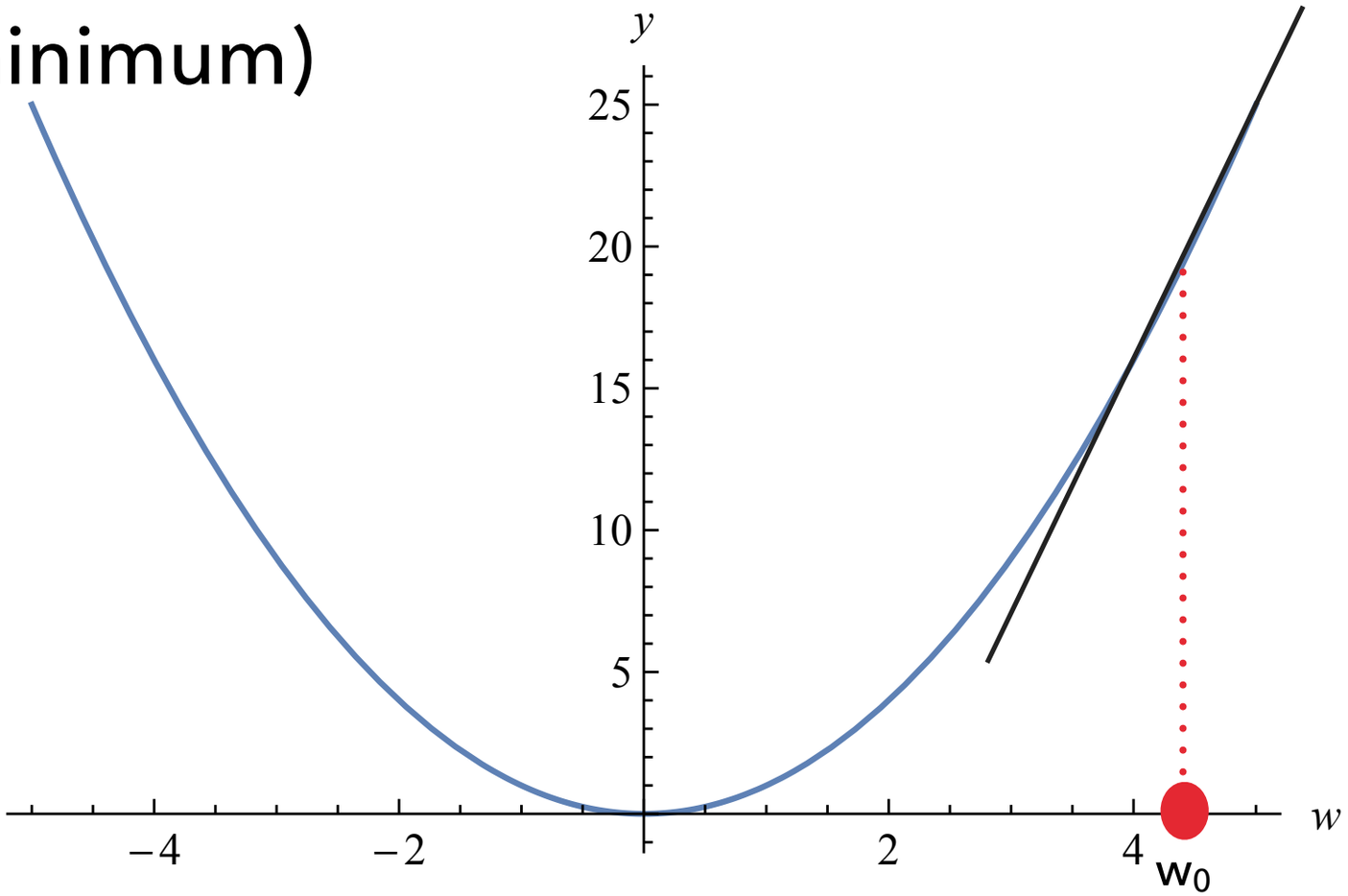
GRADIENT DESCENT

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



GRADIENT DESCENT

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



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

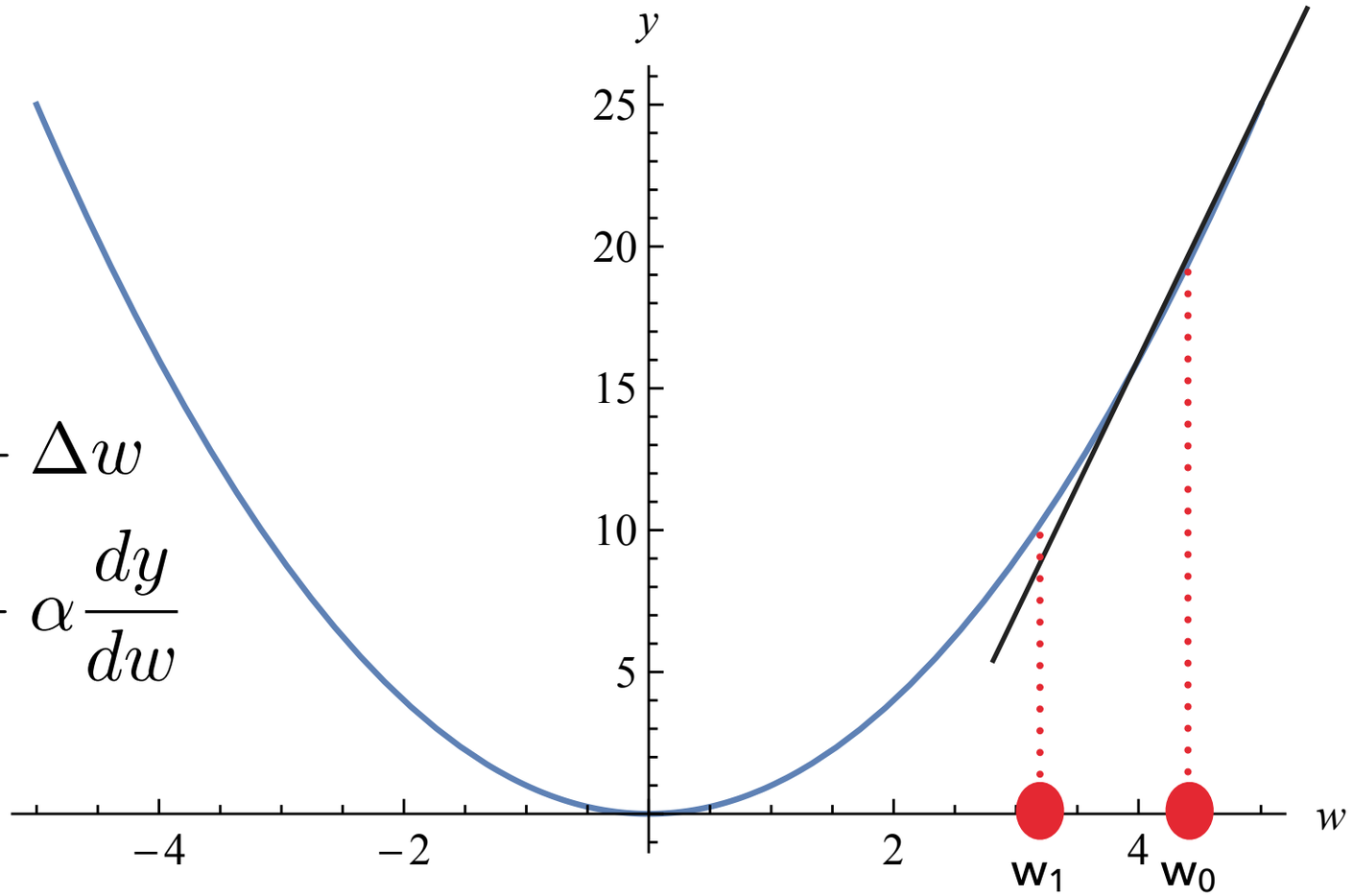
$$= -\alpha \left(\frac{dy}{dw}\right)^2$$

α is the learning rate: a small positive number

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

GRADIENT DESCENT

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



$$w_{i+1} = w_i + \Delta w$$
$$= w_i - \alpha \frac{dy}{dw}$$

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

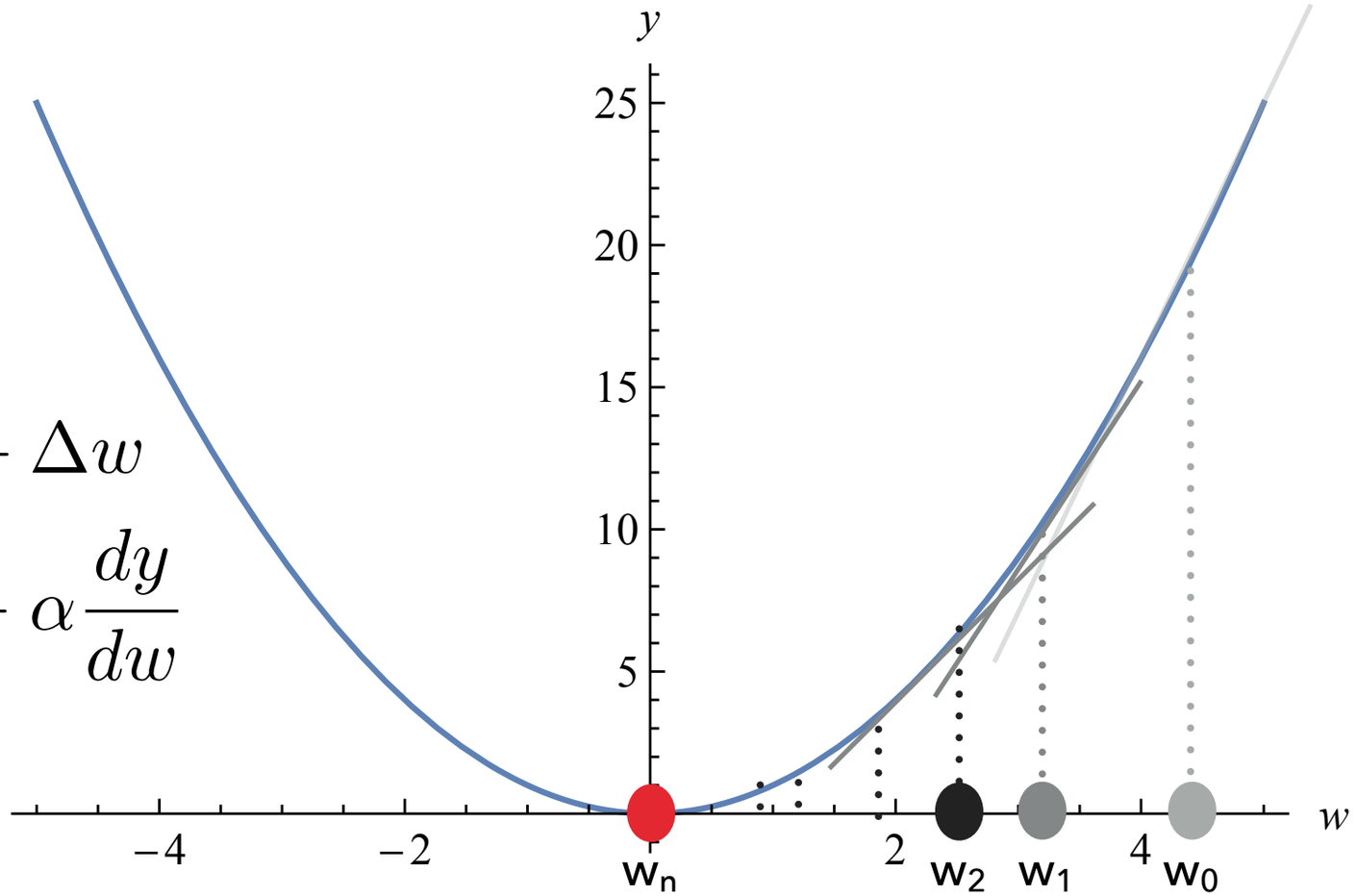
α is the learning rate: a small positive number

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

GRADIENT DESCENT

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

$$w_{i+1} = w_i + \Delta w$$
$$= w_i - \alpha \frac{dy}{dw}$$



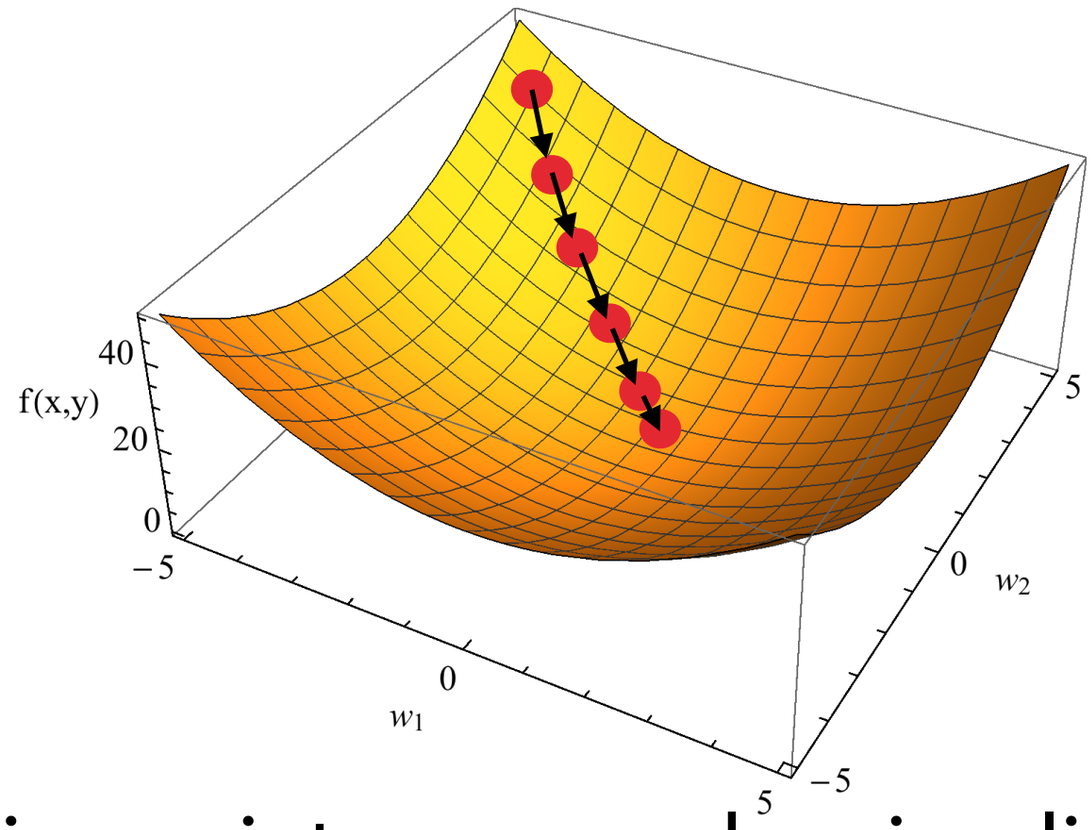
$$\Delta y = \Delta w \frac{dy}{dw}$$
$$= -\alpha \left(\frac{dy}{dw}\right)^2$$

α is the learning rate: a small positive number

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

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.

GRADIENT DESCENT

- In general for an n-dimensional space of 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

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

GRADIENT DESCENT: REFLECTION

- The examples shown illustrate problems with parabolic minima.
- With selection of 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 space is not parabolic, this becomes more complicated.

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 distribution as a standard target shape.

$$z = \frac{(x - \mu)}{\sigma} \quad \text{where} \quad \mu = \text{mean}, \quad \sigma = \text{RMS}$$

- The transformed data inputs will be scale invariant in the sense that parameters such as the learning rate will be come general, rather than problem (and therefore scale) dependent.
- Some input features can be difficult to renormalise to a unit Gaussian.
- 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.

GRADIENT DESCENT: ADAM OPTIMISER

- This is a stochastic gradient descent algorithm, combines momentum and RMS propagation algorithms
- Consider a model f(θ) that is differentiable with respect to the parameters 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 parameters is taken: θ_0, and the parameters 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

GRADIENT DESCENT: ADAM OPTIMISER

- ADAptive Moment estimation based on gradient descent.

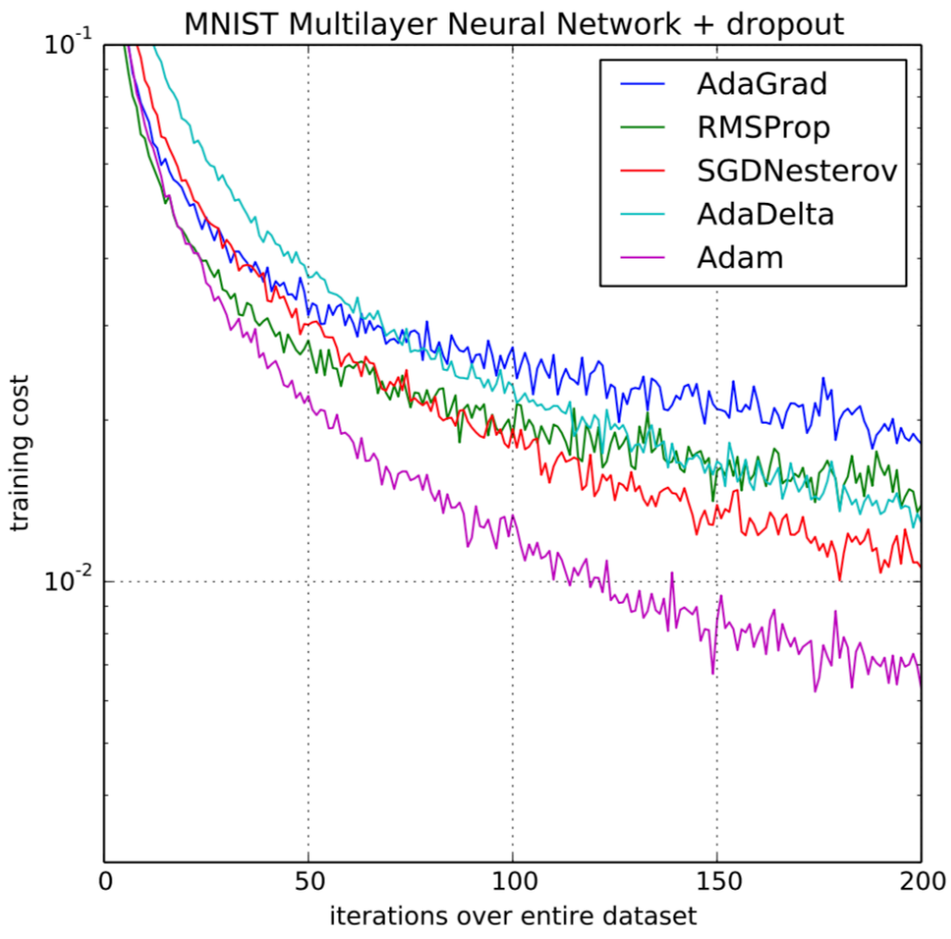
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha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

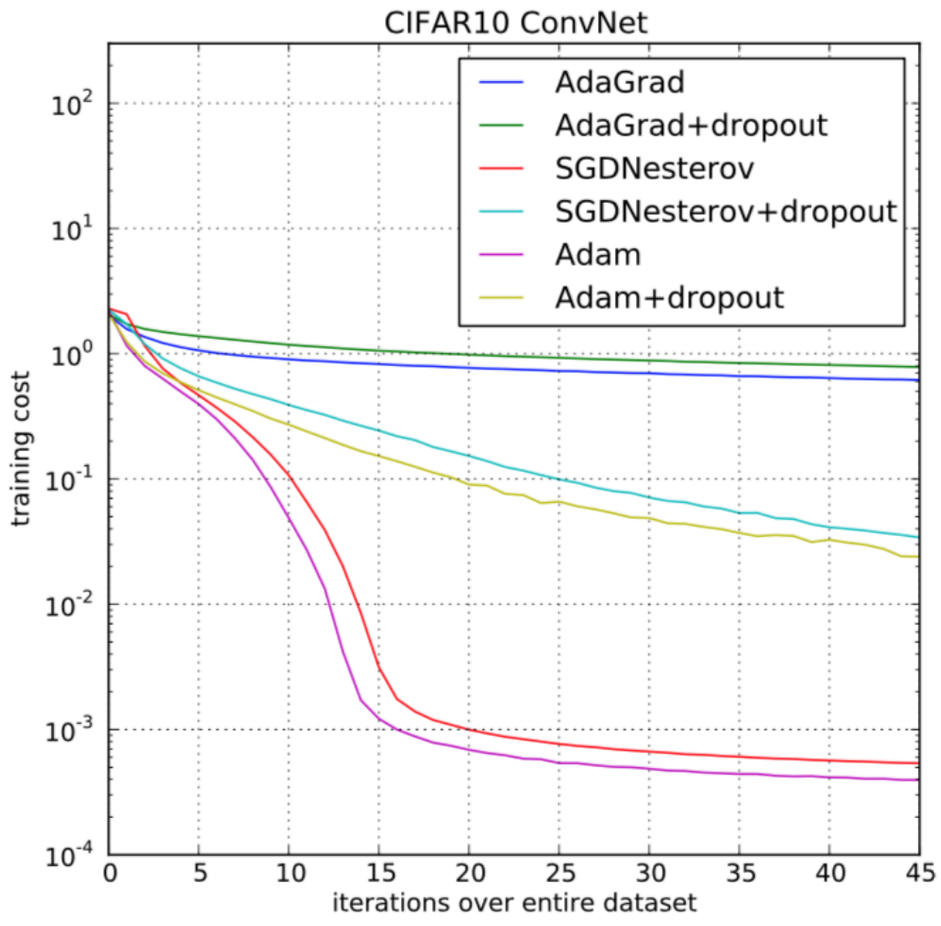
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd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

GRADIENT DESCENT: 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

GRADIENT DESCENT:MULTIPLE MINIMA

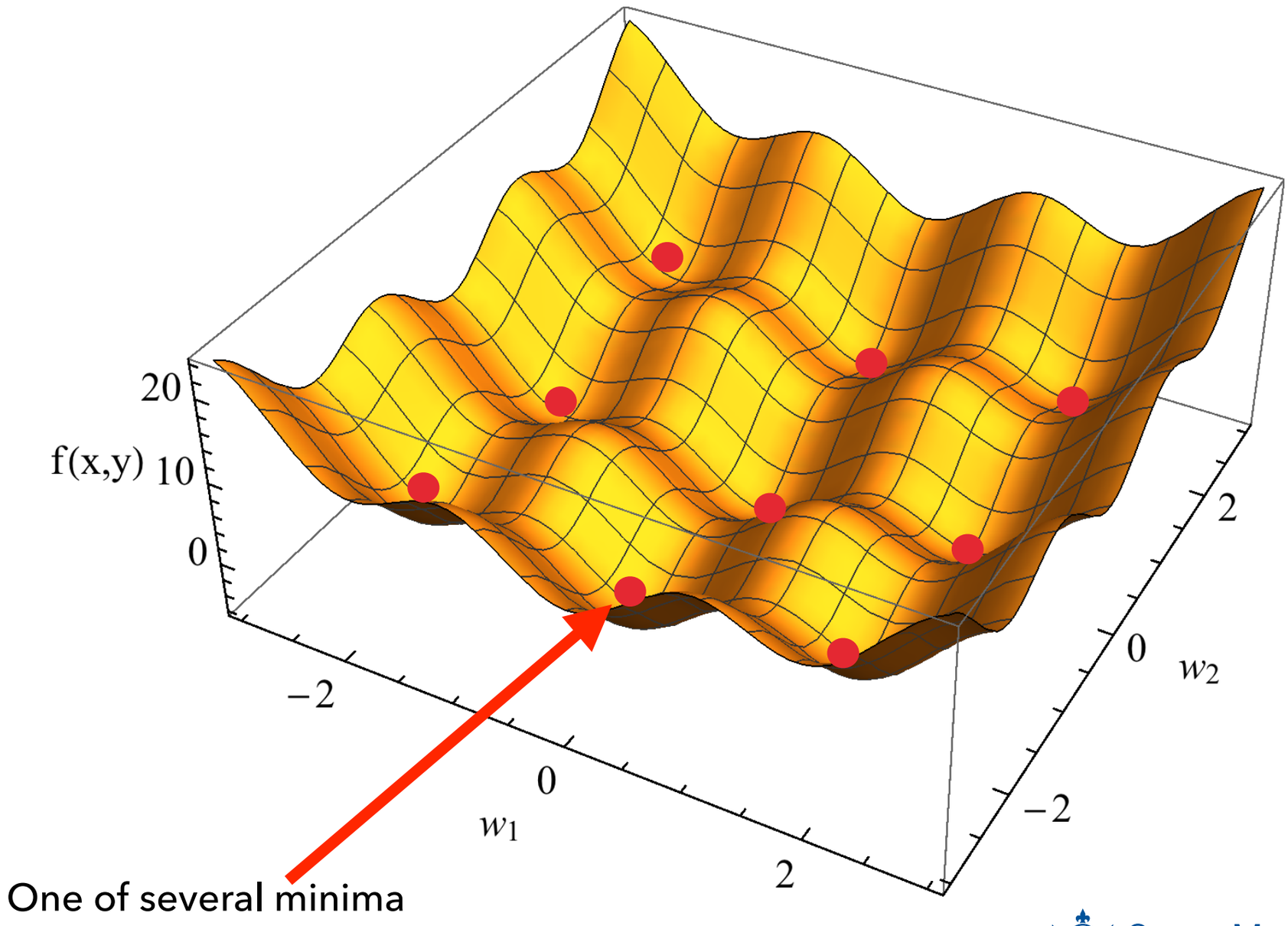
- Often more complication parameter space optimisation problems are encountered, where there are multiple minima.

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?



GRADIENT DESCENT:MULTIPLE MINIMA

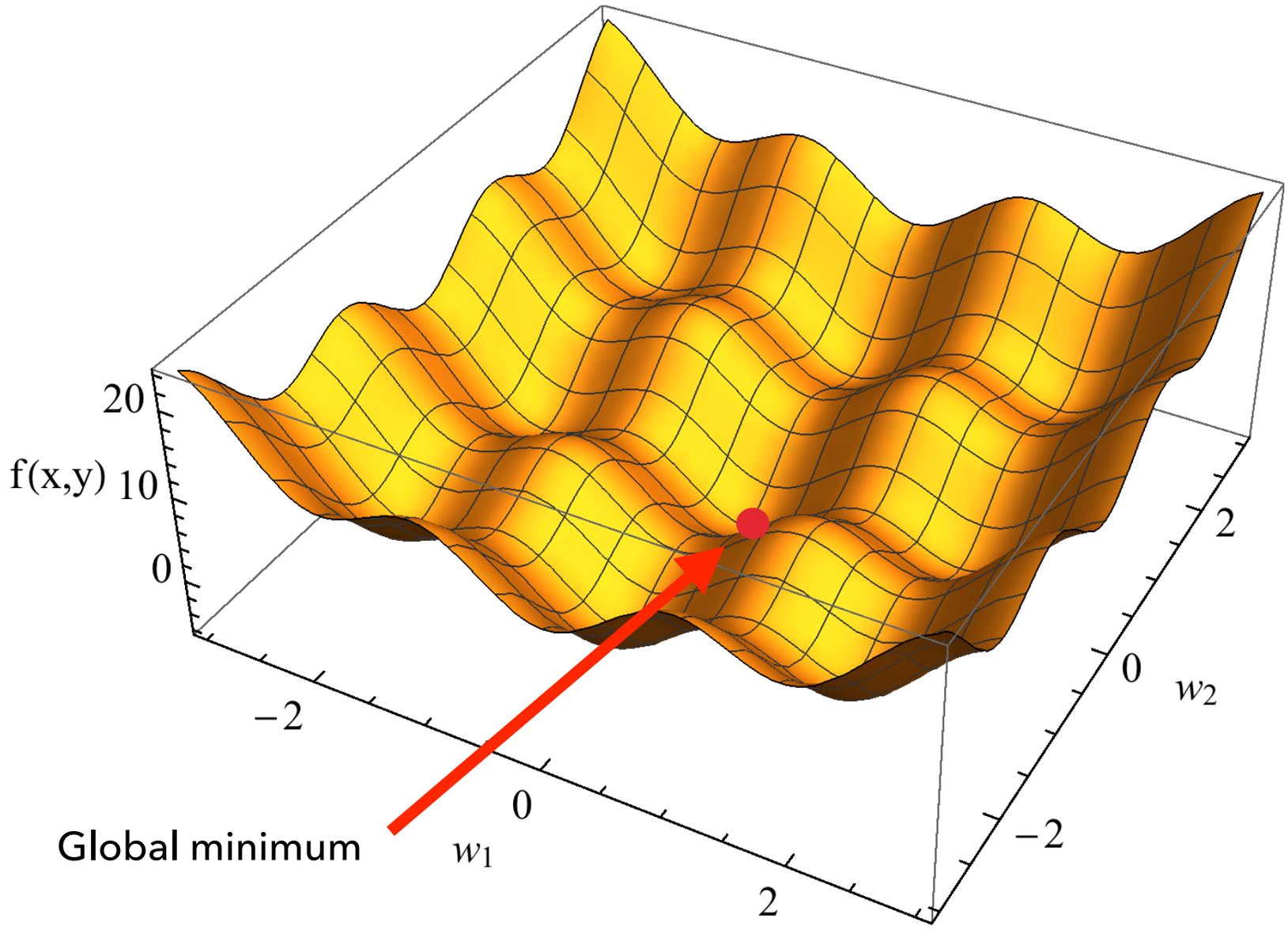
- Often more complication parameter space optimisation problems are encountered, where there are multiple minima.

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?

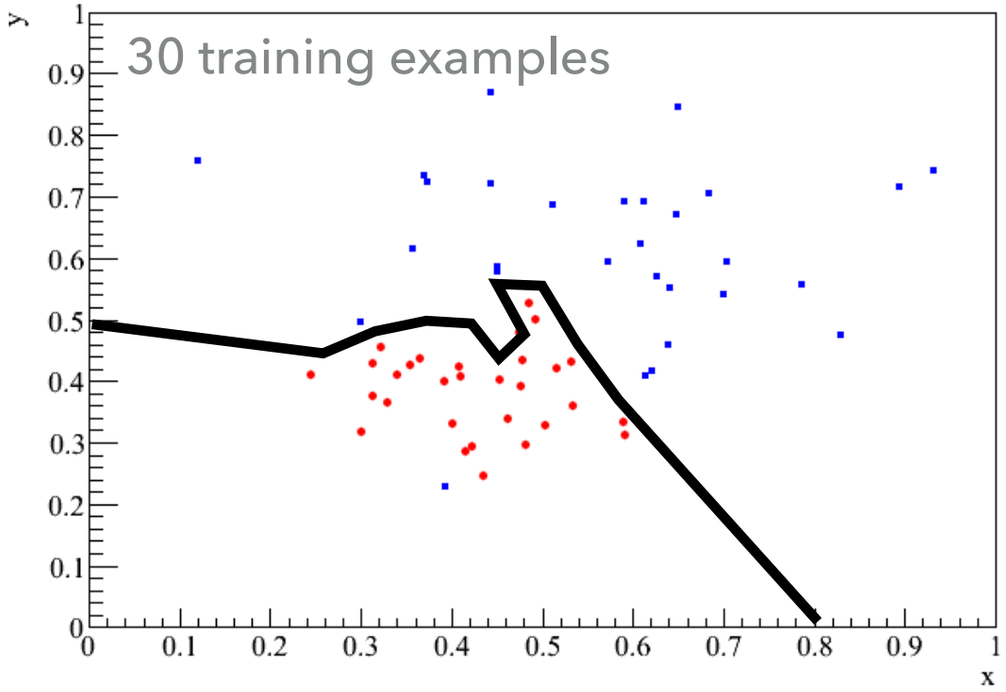


OVER FITTING (OVER TRAINING)

- Overfitting
- Training Validation
- Mitigation methods
  - Weight regularisation
  - Cross validation
  - Dropout

OVER FITTING

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



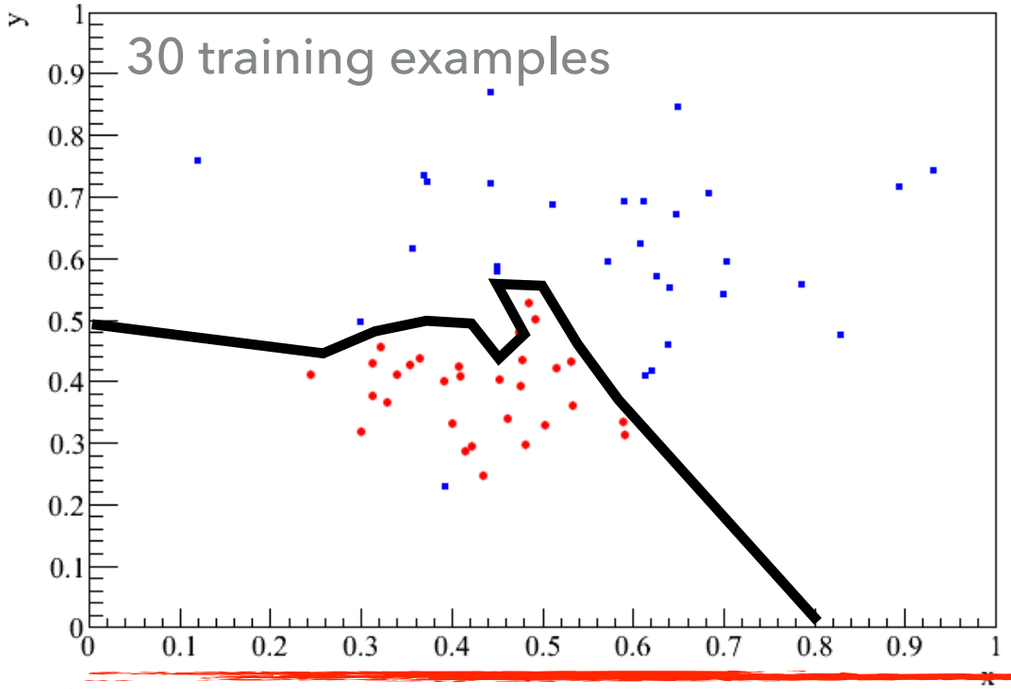
The decision boundary selected here does a good job of separating the red and blue dots.

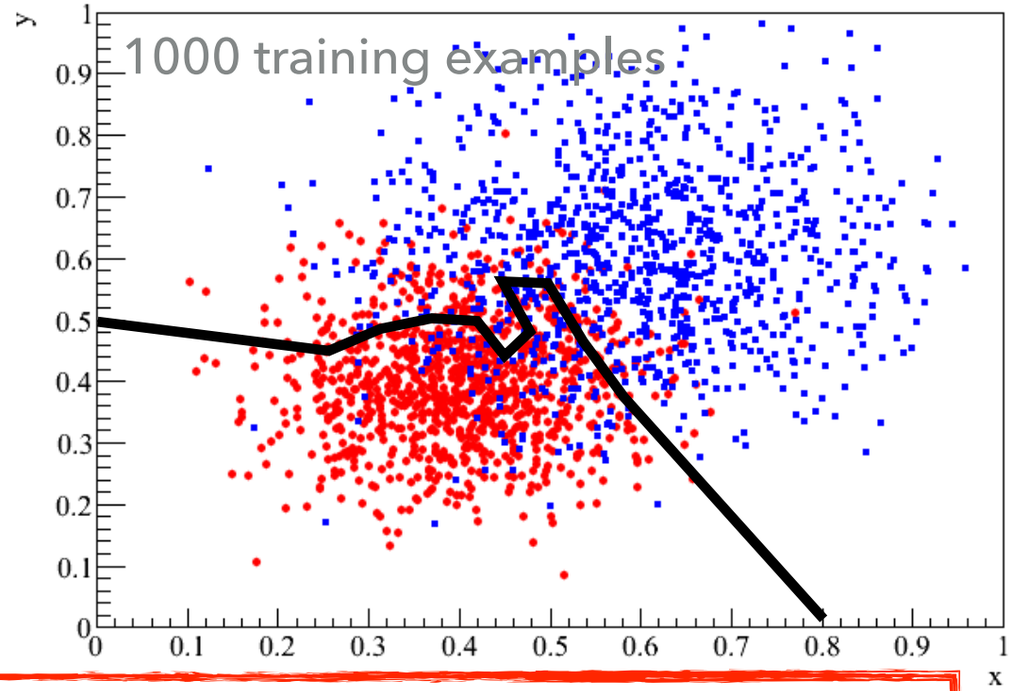
Boundaries like this can be obtained by training models on limited data samples. The accuracies can be impressive.

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

OVER FITTING

- A model is over fitted if the parameters 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 paramours of a model.

OVER FITTING: 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 parameters 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.

OVER FITTING: 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.

OVER FITTING: WEIGHT REGULARISATION

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

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

- 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

OVER FITTING: WEIGHT REGULARISATION

- 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:

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

- This is a simple modification to make to the NN training process.

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.

- The central limit theorem tells us that:

If one takes N random samples of a distribution of data that describes some variable x, where each sample is independent and has a mean value μ_i and variance σ_i^2, then the sum of the samples will have a mean value M and variance V where:

$$M = \sum_{i=1}^{N} \mu_i$$

$$V = \sum_{i=1}^{N} \sigma_i^2$$

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).

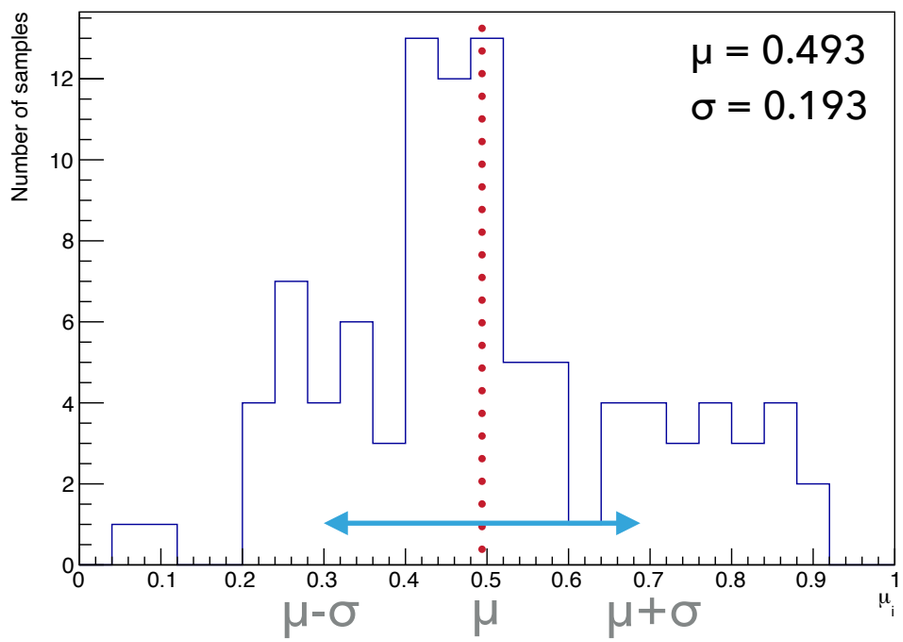
OVER FITTING: CROSS VALIDATION

- 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.

- The central limit theorem tells us that:

The average of the sample of samples will (on average) be more representative than any given training example.

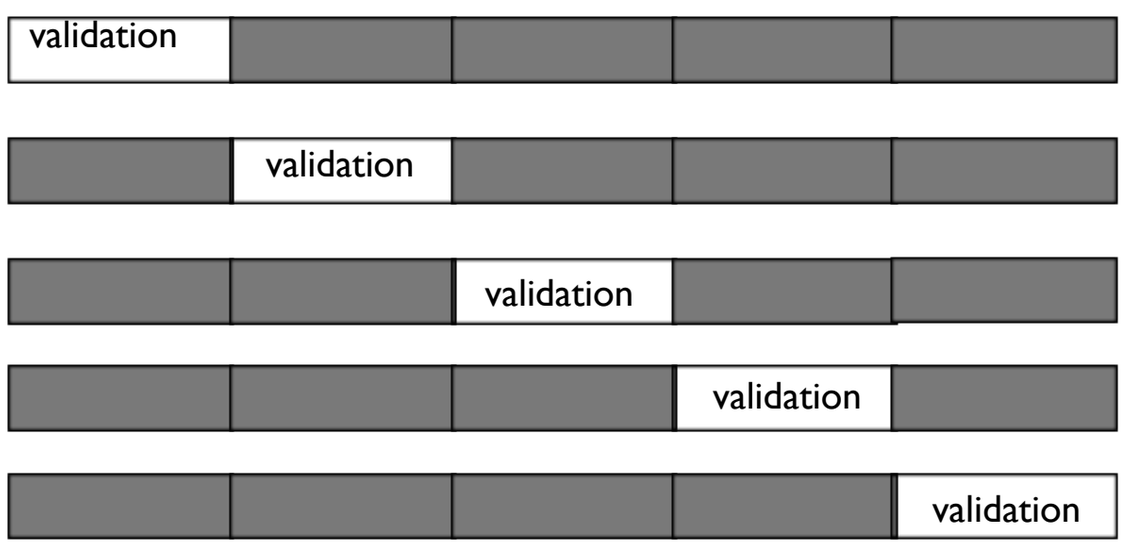
An extension of this concept is that if we train a model many times on smaller sub-samples, the average will be a more representative performance than an individual training.



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.
  - One can then train the data on each of the k training sets, validating the performance of the network on the corresponding validation set.

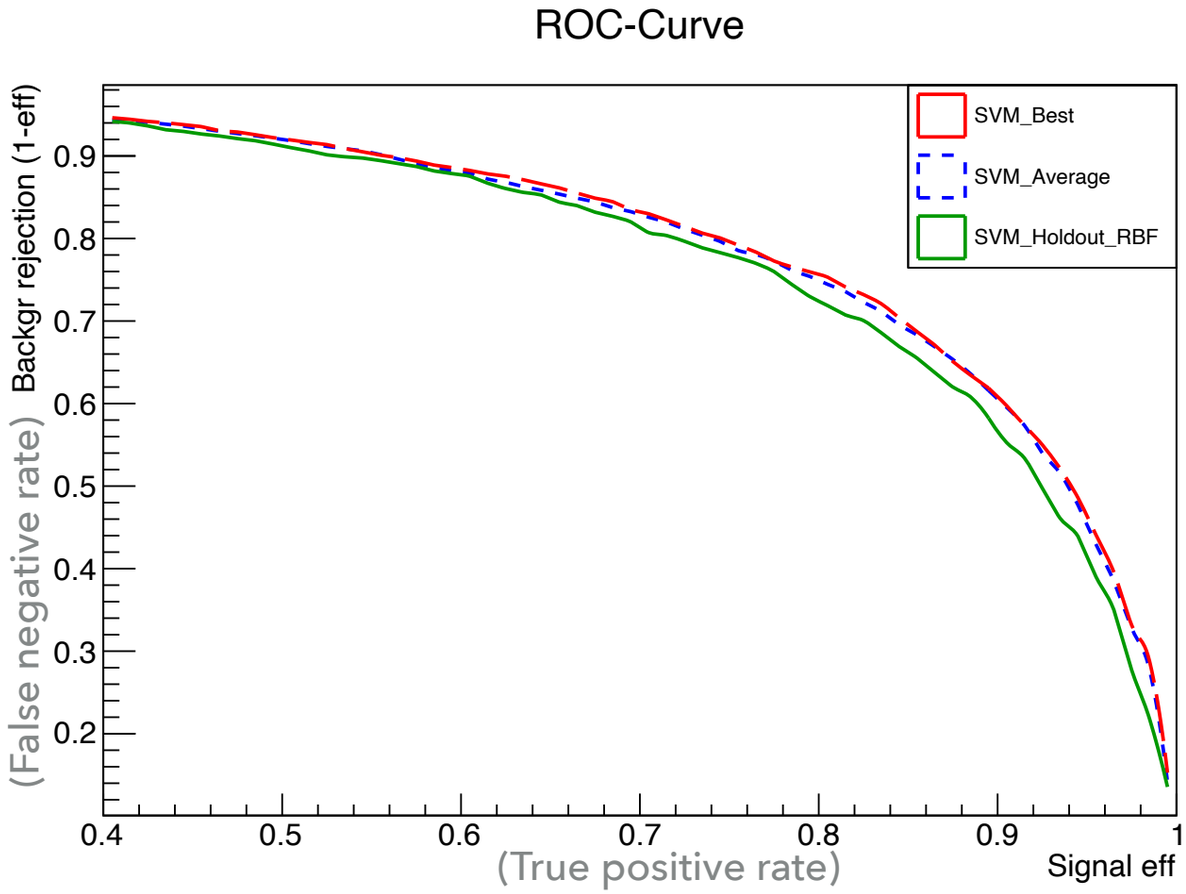


*Variants include the extremes of leave 1 out CV and Hold out CV as well as leave p-out CV. These involve reserving 1 example, 50% of examples and p examples for testing, and the remainder of data for training, respectively.

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, but the principle is the same.
  - It is counter-intuitive, but the robust response comes from the average, not the best performance using the ROC FOM.

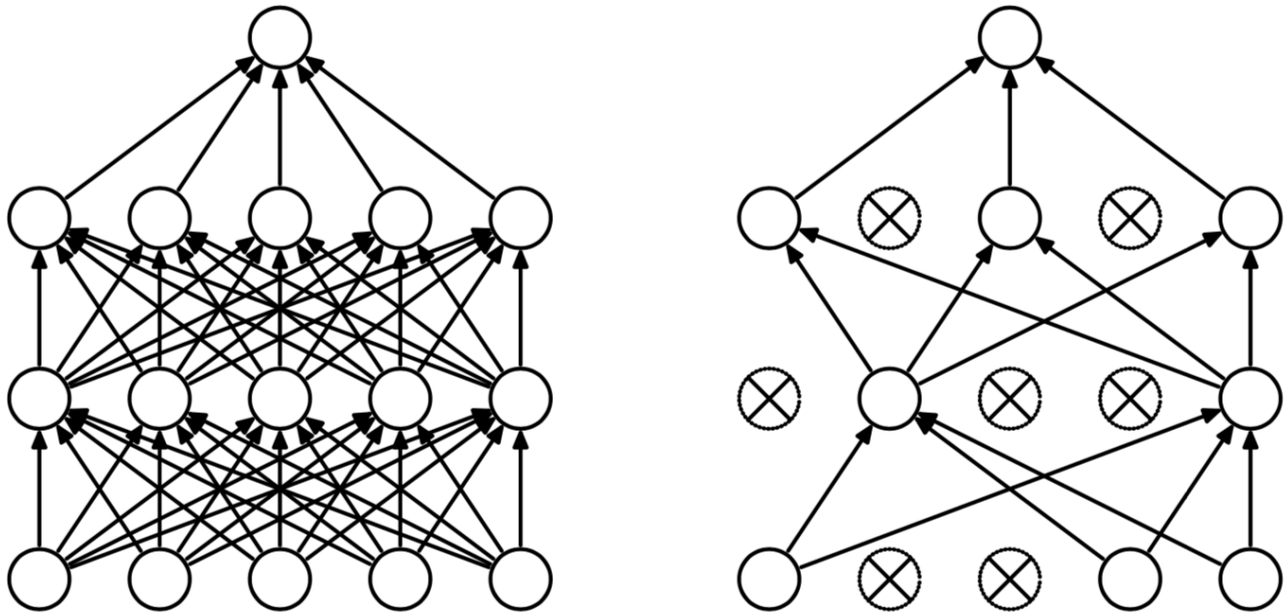


*Variants include the extremes of leave 1 out CV and Hold out CV as well as leave p-out CV. These involve reserving 1 example, 50% of examples and p examples for testing, and the remainder of data for training, respectively.

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: DROPOUT

- 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.

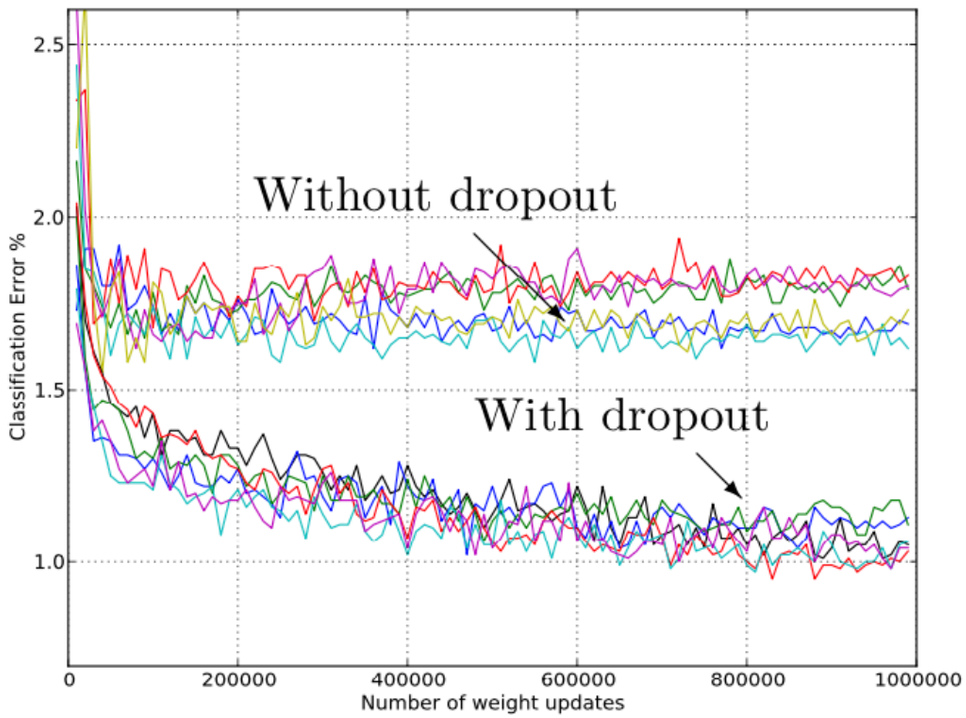
Dropout is used during training; when evaluating predictions with the validation or unseen data the full network of Fig. (a) is used.

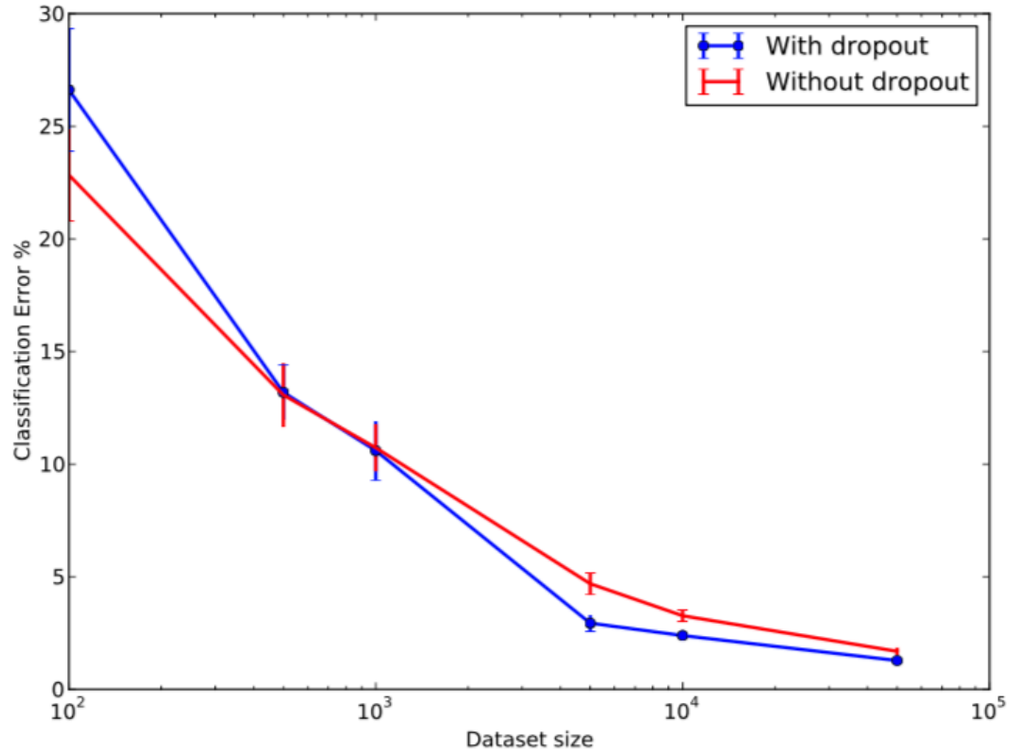
- That way the whole model will be effectively trained on a sub-sample of the data in the hope that the effect of statistical fluctuations will be limited.
- This does not remove the possibility that a model is overtrained, as with the previous discussion network generalisation is promoted by using this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

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

OVER FITTING: DROPOUT

- 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

PRACTICAL MACHINE LEARNING

REGRESSION

REGRESSION

- Using a model for regression means that we compute a quantitative output, called the *response*.
  - Essentially the process is the same as addressing a classification problem, however instead of a quantitative outcome (a yes/no decision on types associated with an example) we have a number associated with a data example.
- This number (the response) can be used in a probabilistic sense to avoid having to make an absolute choice between types.

REGRESSION

- Neural network models can be thought of as non-linear regression functions; a generalisation of a linear regression model.
- To explain what this means we can consider a Taylor series expansion of some function.

REGRESSION

- e.g. consider the function $y=f(x) = \sin(x)$
- We can model this with a polynomial; but which one?
  - $y=a_1x$
  - $y=a_1x+a_2x^2$
  - $y=a_1x+a_2x^2+a_3x^3$
  - $y=a_1x+a_2x^2+a_3x^3+a_4x^4$
  - $y = \sum_{i=0}^{\infty} a_i x^i$

Each model is an approximation of the function we are interested in.

For some region of x the approximation will be good (accurate prediction).

For some region in x the approximation will be bad (inaccurate prediction).

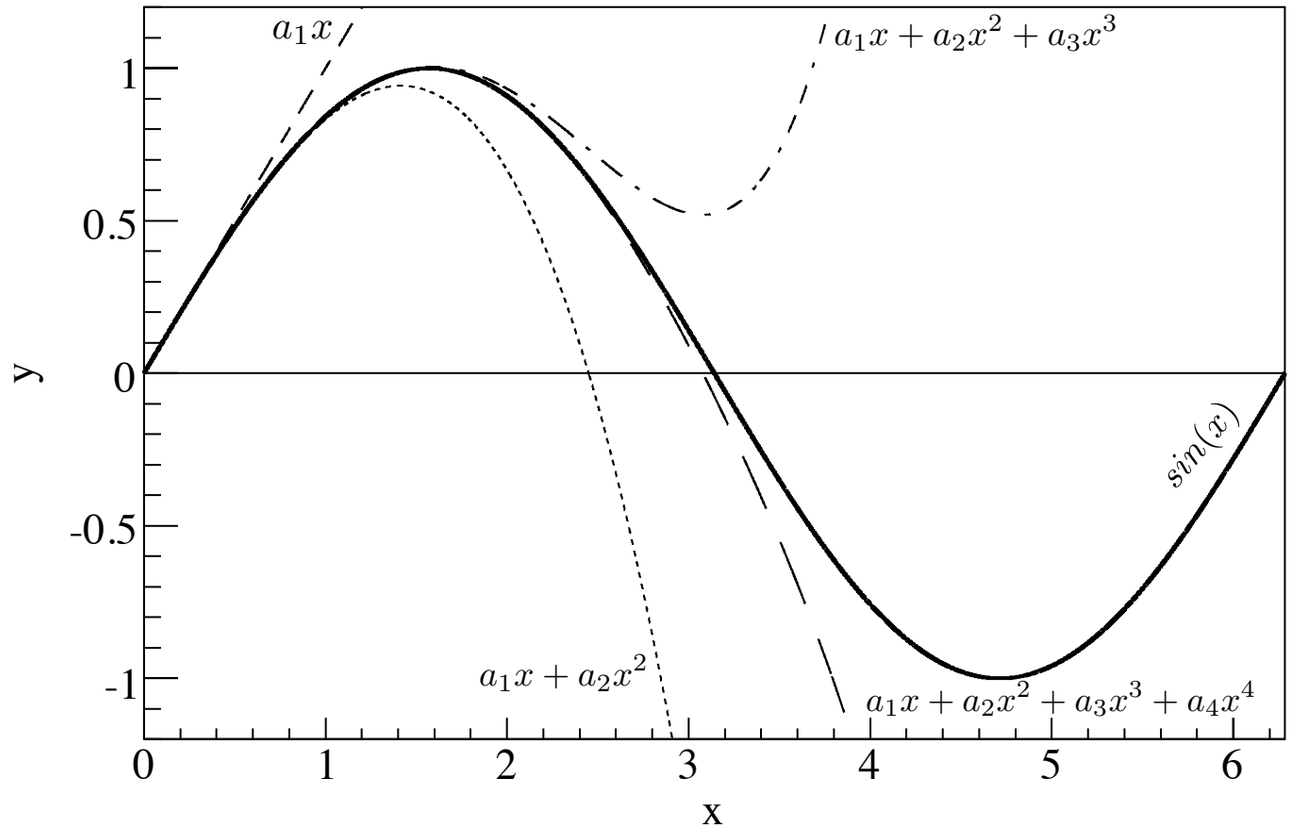
Like all approximations, there are conditions associated with the validity of a given model

We can neglect the a_0 term for this function approximation problem as sin(x) is an odd function (i.e. y=0 zero at x=0).

REGRESSION

- e.g. consider the function y=f(x) = sin(x)
- We can model this with a polynomial; but which one?
  - $y=a_1x$
  - $y=a_1x+a_2x^2$
  - $y=a_1x+a_2x^2+a_3x^3$
  - $y=a_1x+a_2x^2+a_3x^3+a_4x^4$
  - $$y = \sum_{i=0}^{\infty} a_i x^i$$

Range of validity increases with complexity of the model

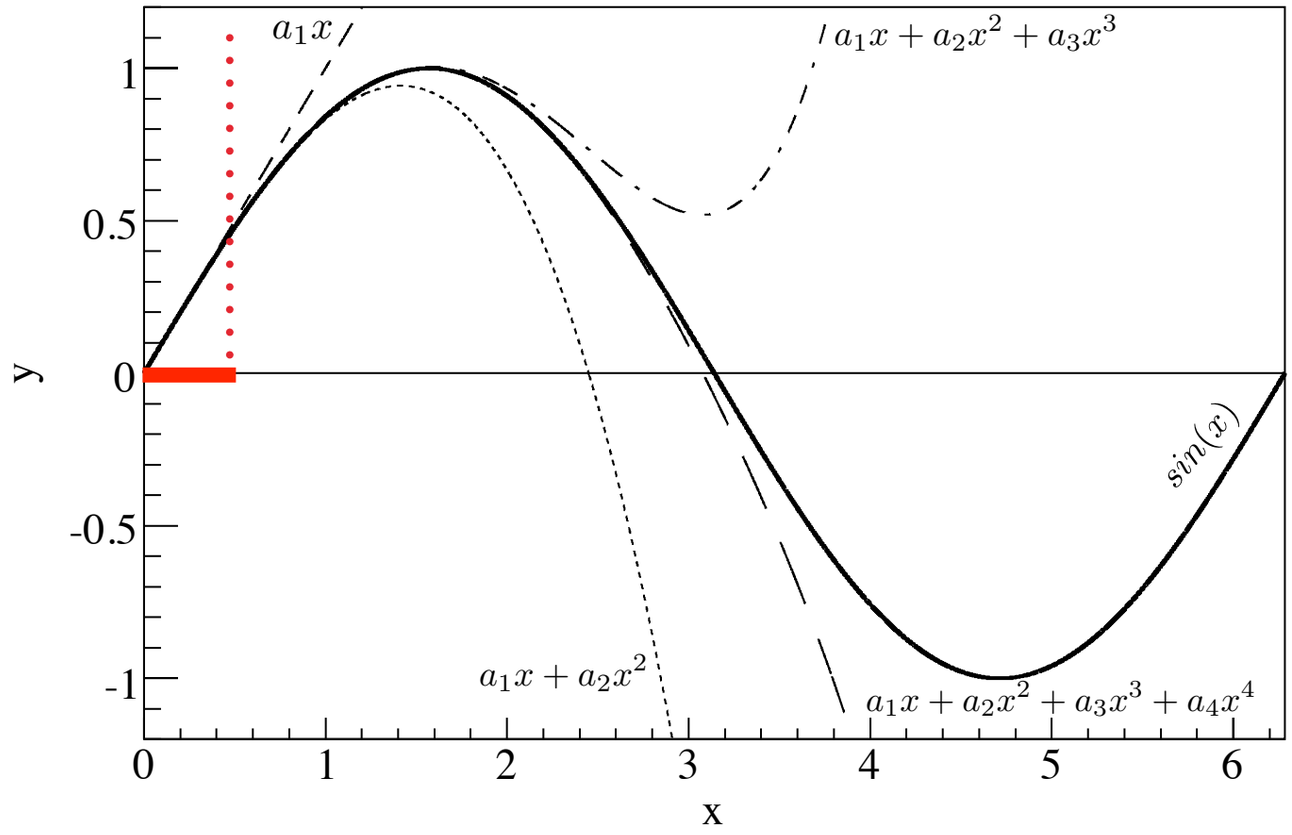


We can neglect the a_0 term for this function approximation problem as sin(x) is an odd function (i.e. y=0 zero at x=0).

REGRESSION

- e.g. consider the function y=f(x) = sin(x)
- We can model this with a polynomial; but which one?
  - $y=a_1x$
  - $y=a_1x+a_2x^2$
  - $y=a_1x+a_2x^2+a_3x^3$
  - $y=a_1x+a_2x^2+a_3x^3+a_4x^4$
  - $$y = \sum_{i=0}^{\infty} a_i x^i$$

Range of validity increases with complexity of the model

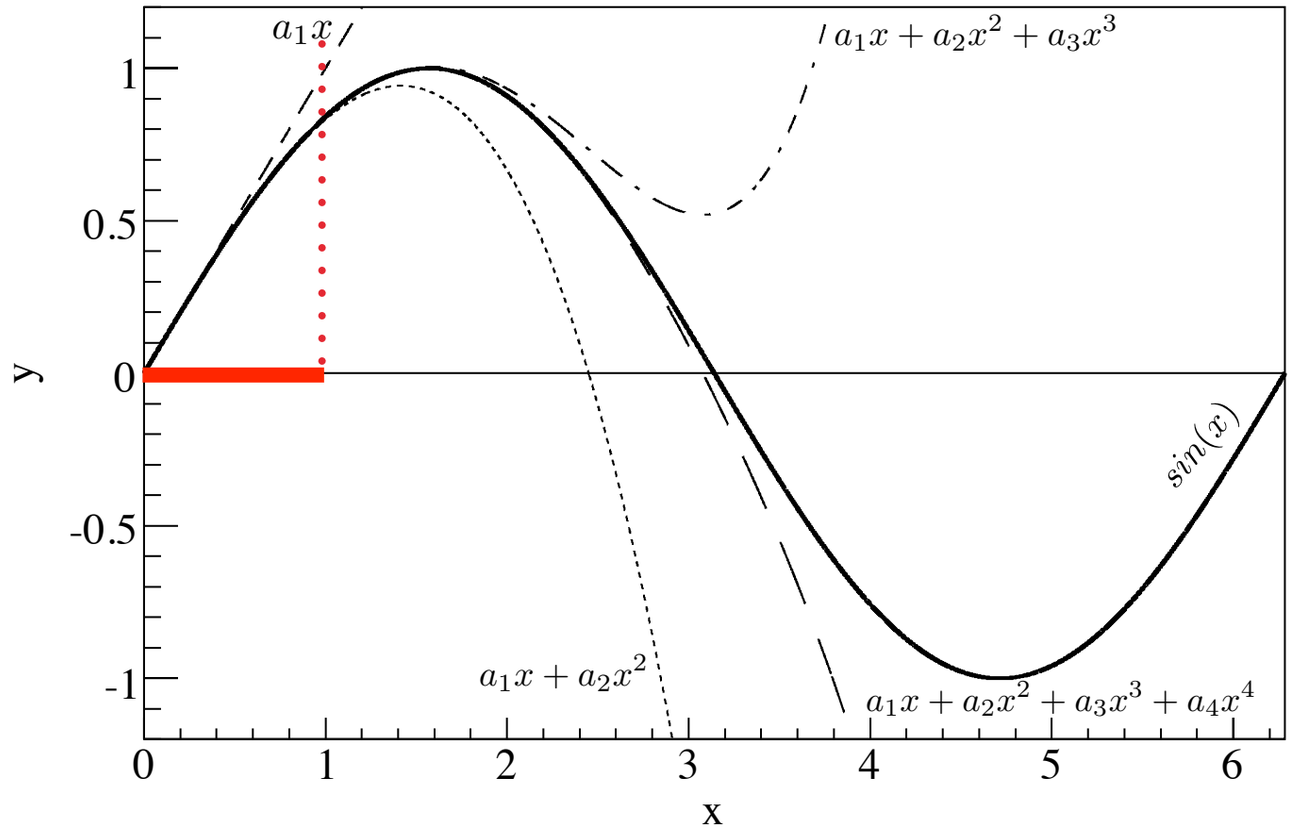


We can neglect the a_0 term for this function approximation problem as sin(x) is an odd function (i.e. y=0 zero at x=0).

REGRESSION

- e.g. consider the function y=f(x) = sin(x)
- We can model this with a polynomial; but which one?
  - $y=a_1x$
  - $y=a_1x+a_2x^2$
  - $y=a_1x+a_2x^2+a_3x^3$
  - $y=a_1x+a_2x^2+a_3x^3+a_4x^4$
  - $$y = \sum_{i=0}^{\infty} a_i x^i$$

Range of validity increases with complexity of the model

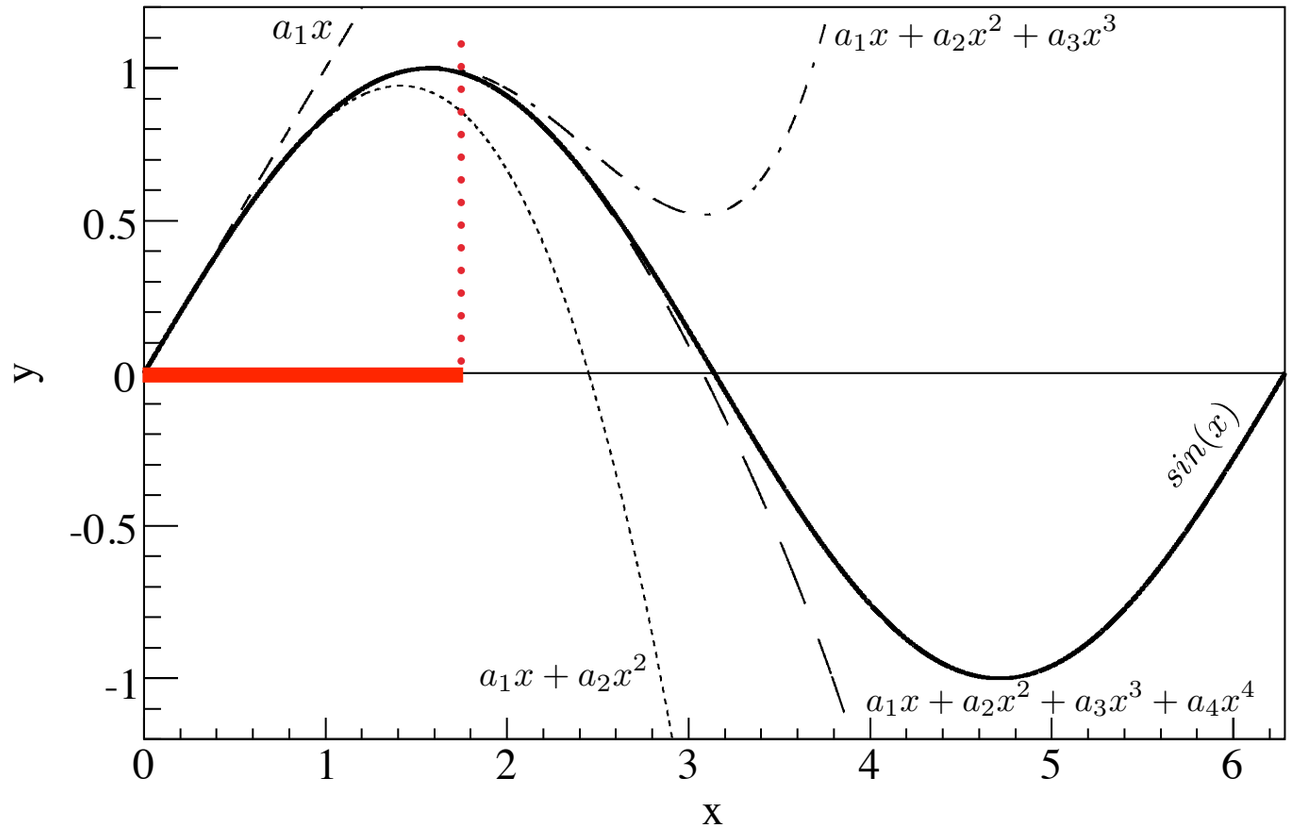


We can neglect the a_0 term for this function approximation problem as sin(x) is an odd function (i.e. y=0 zero at x=0).

REGRESSION

- e.g. consider the function y=f(x) = sin(x)
- We can model this with a polynomial; but which one?
  - $y=a_1x$
  - $y=a_1x+a_2x^2$
  - $y=a_1x+a_2x^2+a_3x^3$
  - $y=a_1x+a_2x^2+a_3x^3+a_4x^4$
  - $$y = \sum_{i=0}^{\infty} a_i x^i$$

Range of validity increases with complexity of the model



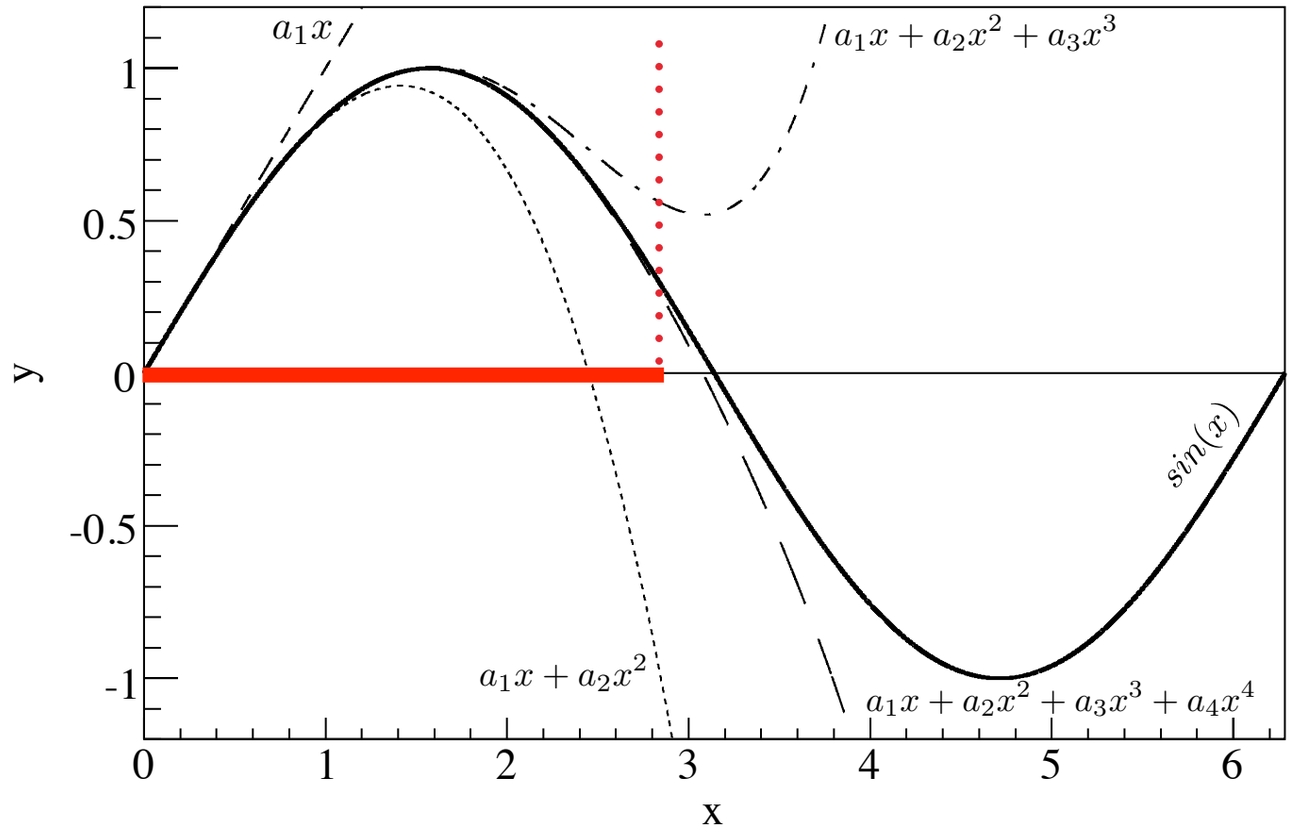
We can neglect the a_0 term for this function approximation problem as sin(x) is an odd function (i.e. y=0 zero at x=0).

Queen Mary University of London

REGRESSION

- e.g. consider the function y=f(x) = sin(x)
- We can model this with a polynomial; but which one?
  - $y=a_1x$
  - $y=a_1x+a_2x^2$
  - $y=a_1x+a_2x^2+a_3x^3$
  - $y=a_1x+a_2x^2+a_3x^3+a_4x^4$
  - $$y = \sum_{i=0}^{\infty} a_i x^i$$

Range of validity increases with complexity of the model



We can neglect the a_0 term for this function approximation problem as sin(x) is an odd function (i.e. y=0 zero at x=0).

REGRESSION

- e.g. consider the function y=f(x) = sin(x)
- We can model this with a polynomial; but which one?

Taylor series expansions are analytically determined.

The analogy that a more complicated model can provide a better approximation to a function can down with a supervised learning approach:

If there are not enough training examples to determine the model parameters, then a complicated neural network can provide a bad approximation of a function.

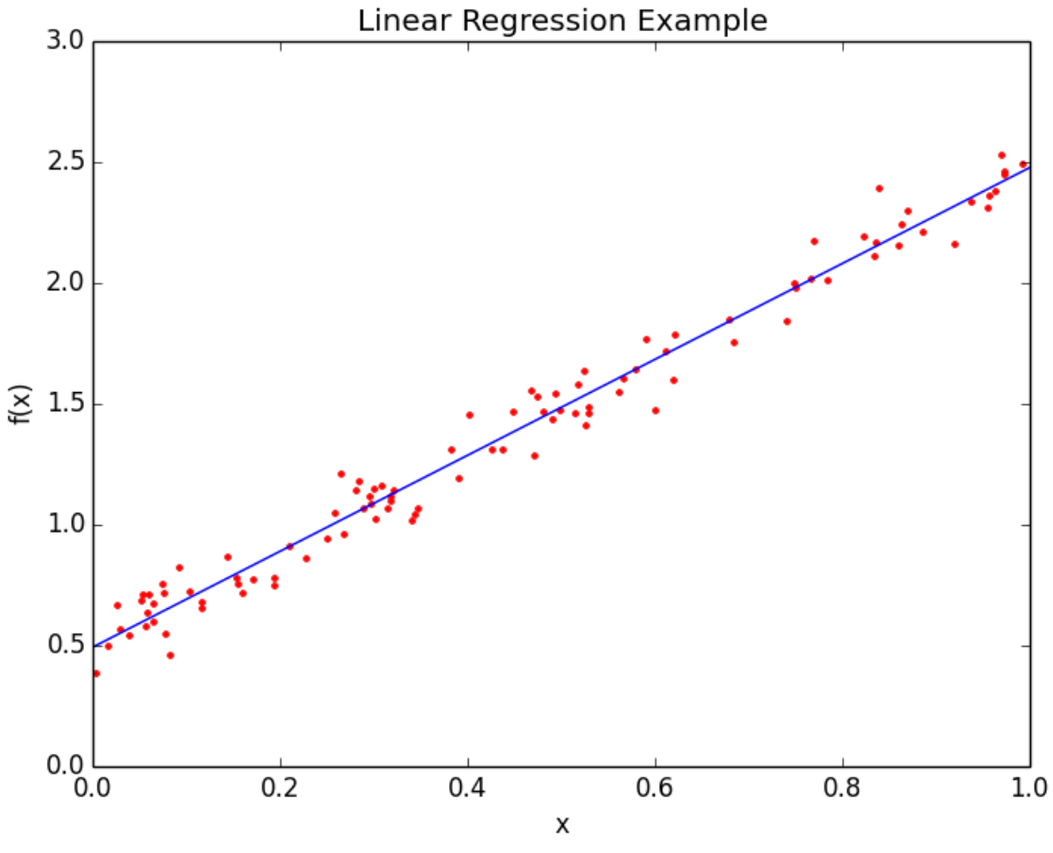
We can neglect the a_0 term for this function approximation problem as sin(x) is an odd function (i.e. y=0 zero at x=0).

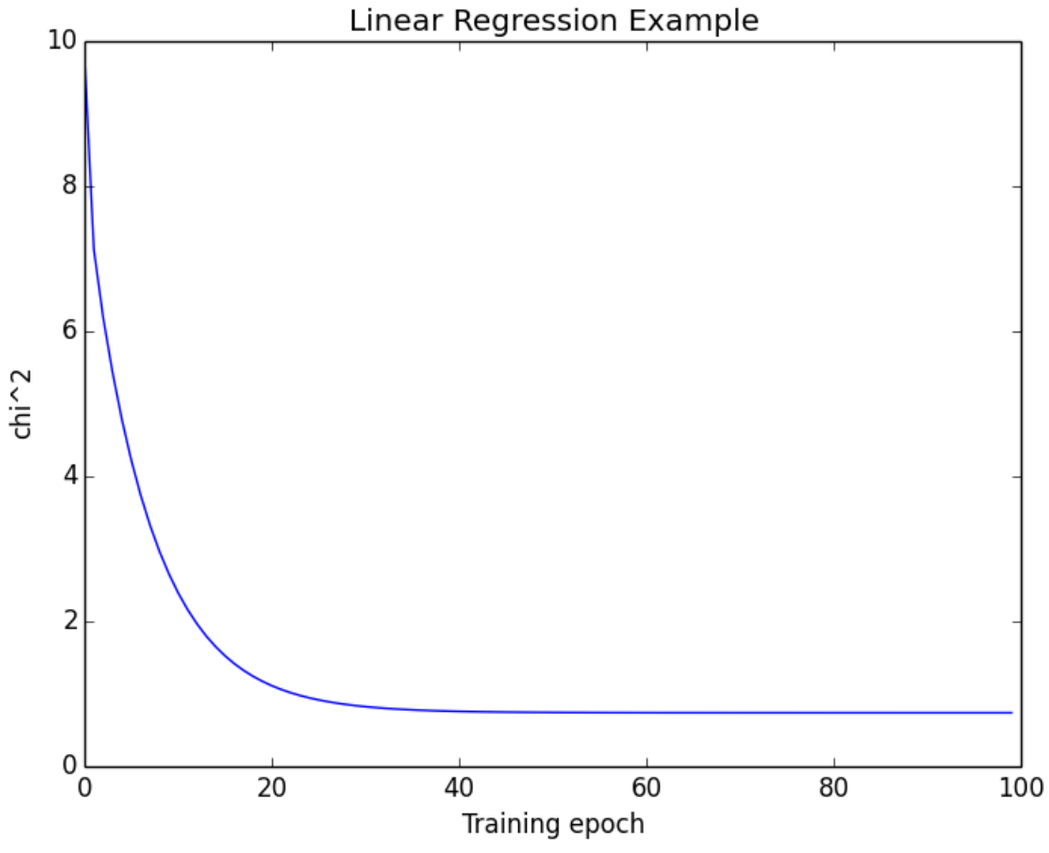
EXAMPLES: LINEAR REGRESSION

- We have already seen the linear regression problem; using optimisation to determine the model parameters for

$$y = f(x)$$

$$mx + c$$





The model y is a prediction of a continuous output; this is a regression model prediction.

REGRESSION: SUMMARY

- Regression analysis of data outputs a quantitative score for each example.
- Regression output of a model can be used in subsequent analysis.

SUGGESTED READING

- Discussion of regression models in text books
  - C. Bishop: *Neural Networks for Pattern Recognition*
    - Chapter: 1, 2, 6
  - C. Bishop: *Pattern Recognition and Machine Learning*
    - Chapter: 1, 2
  - T. Hastie, R. Tibshirani, J. Friedman, *Elements of statistical learning*
    - Chapter: 2, 3, 6
- A few examples of regression used in particle physics:
  - B Physics using an MLP: Aubert et al., Phys.Rev.D76:052007,2007
  - Exotic Particle Search using deep networks: Baldi, Sadowski, Whiteson, Nature Comm. 5308
  - Jet substructure: Baldi et al. Phys.Rev. D93 (2016) no.9, 094034
  - … many other examples out there to contextualise regression problems.

PRACTICAL MACHINE LEARNING

DEEP LEARNING

WHAT IS DEEP LEARNING?

- Deep learning is ill defined in that different people have different definitions.
- The working definition that we are going to use here is that a deep network has more than 2 hidden layers.
  - Deep networks with a few hidden layers are broad (hundreds of nodes per layer).
  - Deep networks with with many hidden layers generally are long and thin (i.e. not many nodes per layer)
- There are different types of algorithm that fall into this genre, e.g. deep MLPs (this lecture) and Convolutional Neural Networks (next week).

DEEP MLPS

- The structure of a deep MLP is define in the usual way:
- Input layer:
  - Input layer has the dimensionality of the input feature space.
- Hidden layers:
  - typically hundreds of nodes, leading to $O(10^3)$ - $O(10^6)$ parameters to determine.
- Output layer:
  - either a single node or for a multi classification problem N_{class} output nodes.

DEEP MLPS

- Consider the function approximation example - this starts off as a 1 hidden layer neural network and is extended to a 2 hidden layer variant.
- In general for a deep learning problem we want to be able to create a hidden layer with M inputs to N hidden nodes, each of which will provide an output.
- As the number of parameters requiring optimisation increases, the amount of computing resource required to determine those parameters also increases.
- Techniques for increasing convergence speed discussed earlier now become more important (e.g. dropout, normalisation of input parameters, de-correlation of input parameters) unless you have access to effectively infinite computing resource.

DROPOUT

- Dropout is trivial to implement in the training cycle of a neural network.
  - Users just need to specify a keep probability for nodes.
    - 0.0 - keep all nodes
    - 0.5 - drop half of the nodes
    - 1.0 - drop everything (not viable).

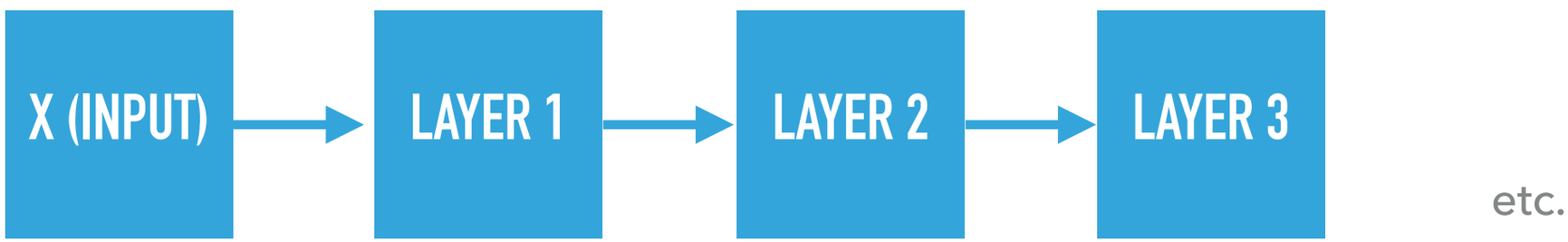
- Compare the performance of training convergence with and without dropout for one of your networks to help understand the benefits of this technique.

DROPOUT

- If I have a neural network, does it make sense to:
  - Use drop out on the input layer?
  - Use drop out on a hidden layer?
  - Use drop out on the output layer/node?

EXAMPLES

- One can implement an MLP with multiple layers (5 here) by repeating the pattern of the first layer and considering the flow of outputs from one hidden layer as the inputs to the next hidden layer.
- e.g. for 3 hidden layers one has



SUMMARY

- We have discussed how to extend a single layer perceptron into a multilayer perceptron that is potentially a deep network.
- The use of:
  - dropout
  - optimisation
  - feature space normalisation
  - mini-batch sample training
- All that remains is for you to explore the use of these techniques to consolidate your understanding of how they can benefit optimisation performance for deep networks.

NOTEBOOK FOR TODAY

https://github.com/abbeywaldron/cinvestav_ML_2024