Queen Mary
University of London

DR ADRIAN BEVAN

PRACTICAL MACHINE LEARNING

OPTIMISATION

LECTURE PLAN

- Hyperparameter optimisation
- Supervised learning
  - Loss functions
- Gradient descent
  - Adam optimiser
  - Multiple minima
  - Davidon-Fletcher-Powell (old algorithm, included for context)
- Over fitting
- Summary

QMUL Summer School: https://www.qmul.ac.uk/summer-school/
Practical Machine Learning QMplus Page: https://qmplus.qmul.ac.uk/course/view.php?id=10006

HYPERPARAMETER OPTIMISATION

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

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

HYPERPARAMETER OPTIMISATION

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

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

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

HYPERPARAMETER OPTIMISATION

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

HYPERPARAMETER OPTIMISATION

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

SUPERVISED LEARNING

- The type of machine learning we are using is referred to as supervised learning.
  - We present the algorithm with known (labeled) samples of data, and optimise the HPs in order to minimise the loss function.
- The loss function is a function of:
  - labels (ground truth)
  - hyper parameters
  - 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: LOSS FUNCTIONS

- Writing and using a custom loss function: *just use ops*
  - Define your loss function (lets implement our own MSE)

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

  - Assign the computation of the loss function to some variable called cost

```
cost = tf.reduce_mean(tf.square(tf.sub(yi,ti)))
```

Subtract y_i and t_i

Square the difference

Compute the mean sum

- Run the optimiser of choice with the cost specified as the argument.

```
tf.train.AdamOptimizer().minimize(cost)
```

SUPERVISED LEARNING: 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

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.

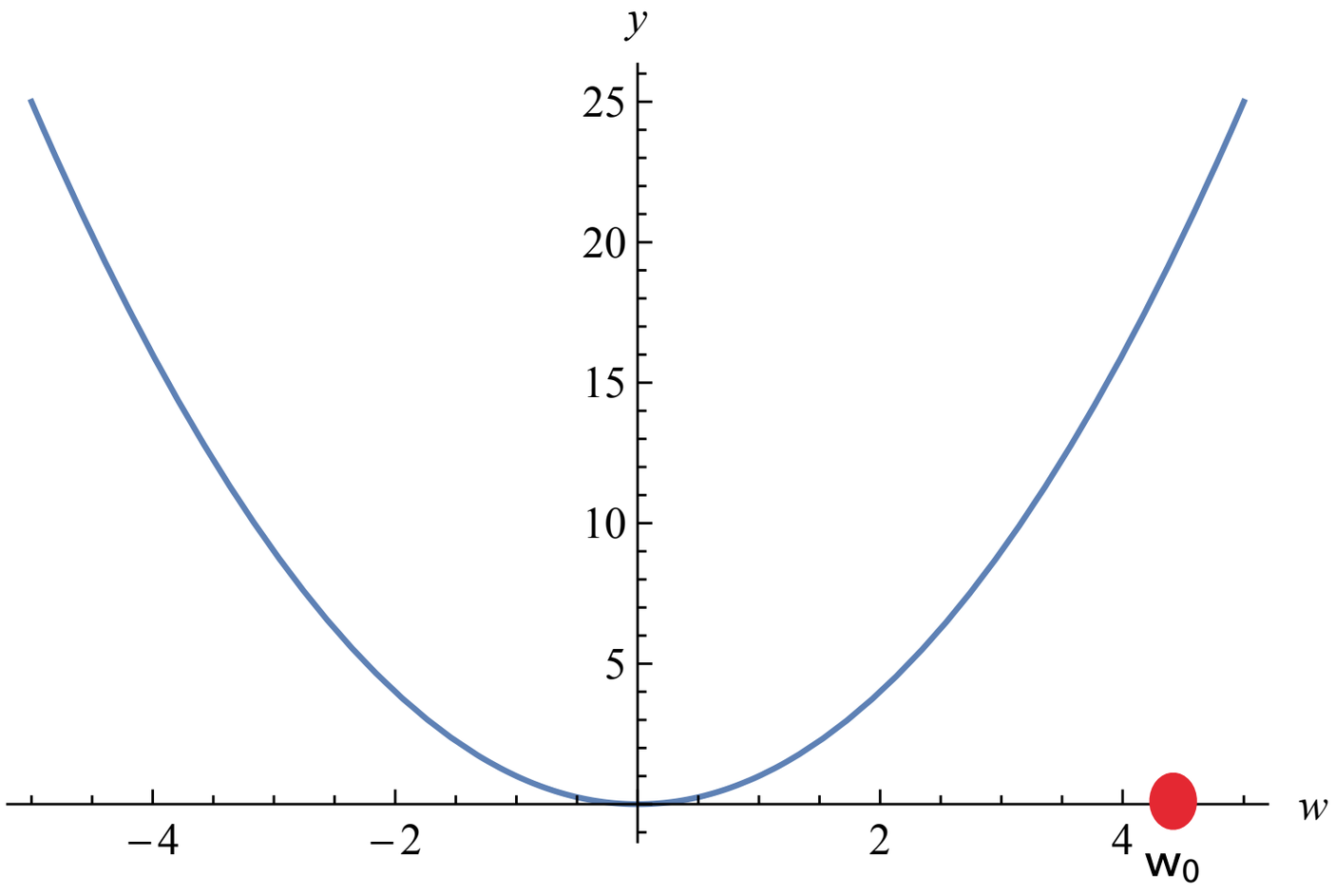
[1] LeCun et al., Efficient BackProp, Neural Networks Tricks of the Trade, Springer 1998

GRADIENT DESCENT

- Newtonian gradient descent
- Adam optimiser
- Other types of optimiser
- Davidon-Flectcher-Powell (DFP)

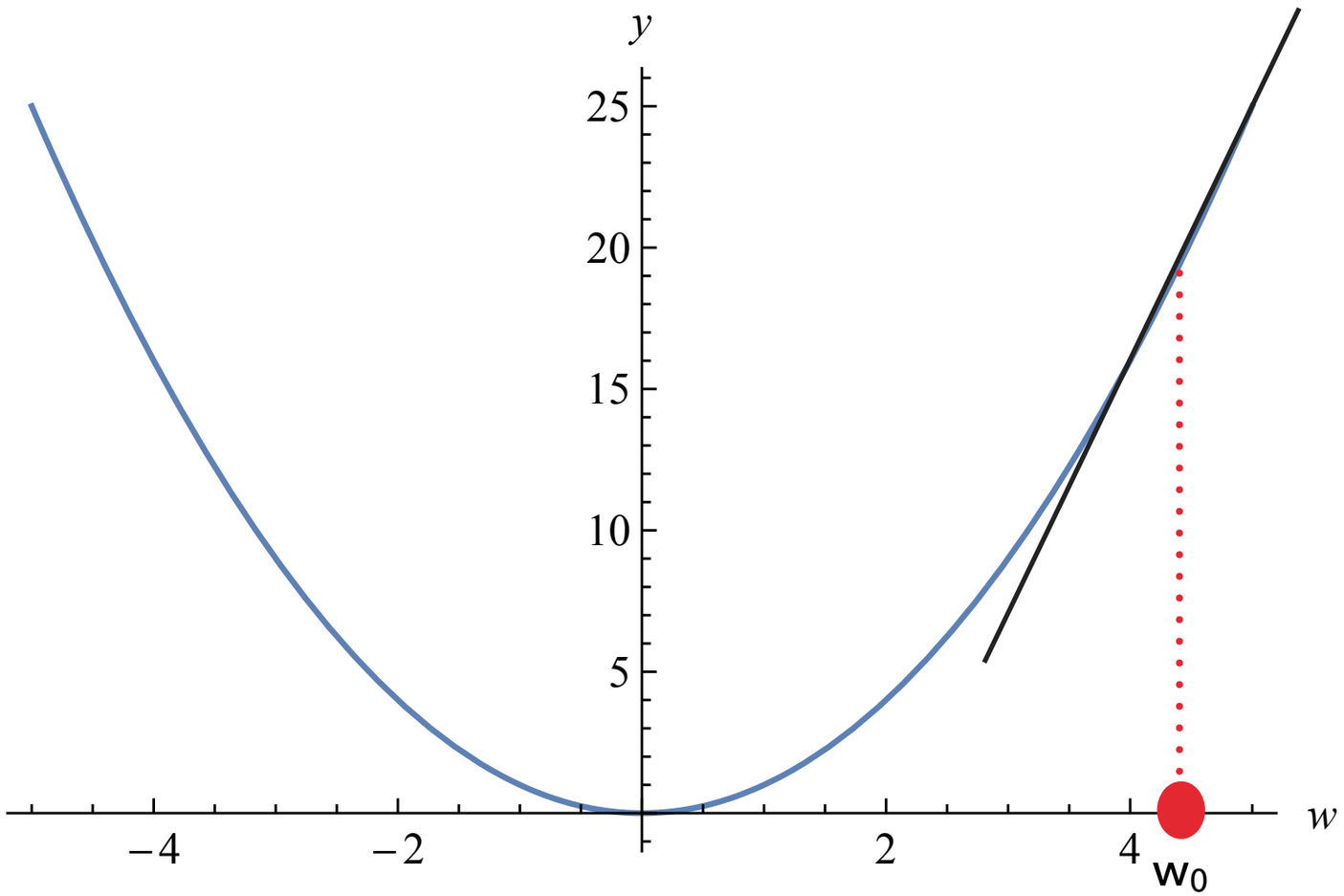
GRADIENT DESCENT

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



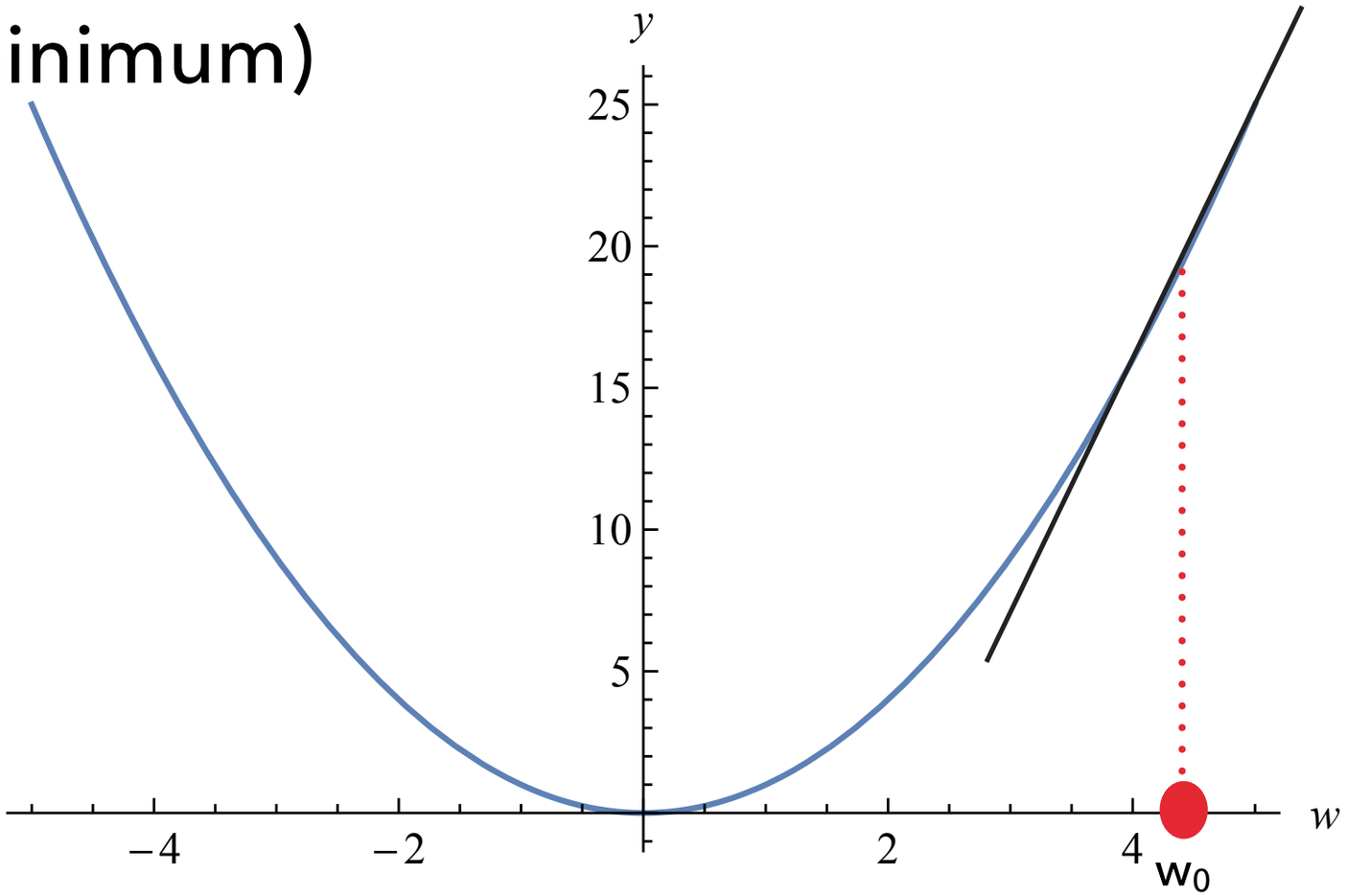
GRADIENT DESCENT

- 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)



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

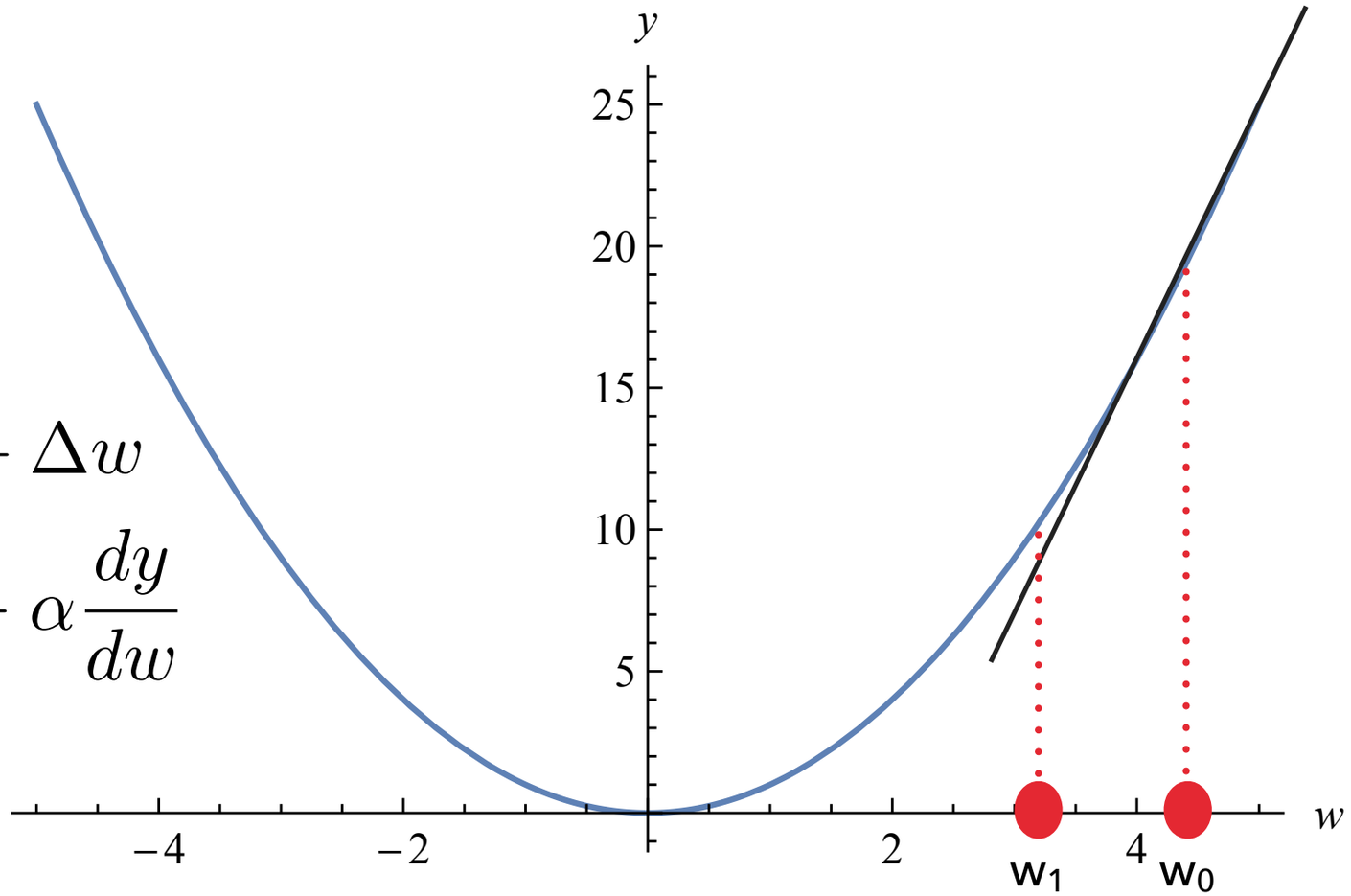
$$= -\alpha \left(\frac{dy}{dw} \right)^2$$

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: $w_1 = w_0 + \Delta w$



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

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

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

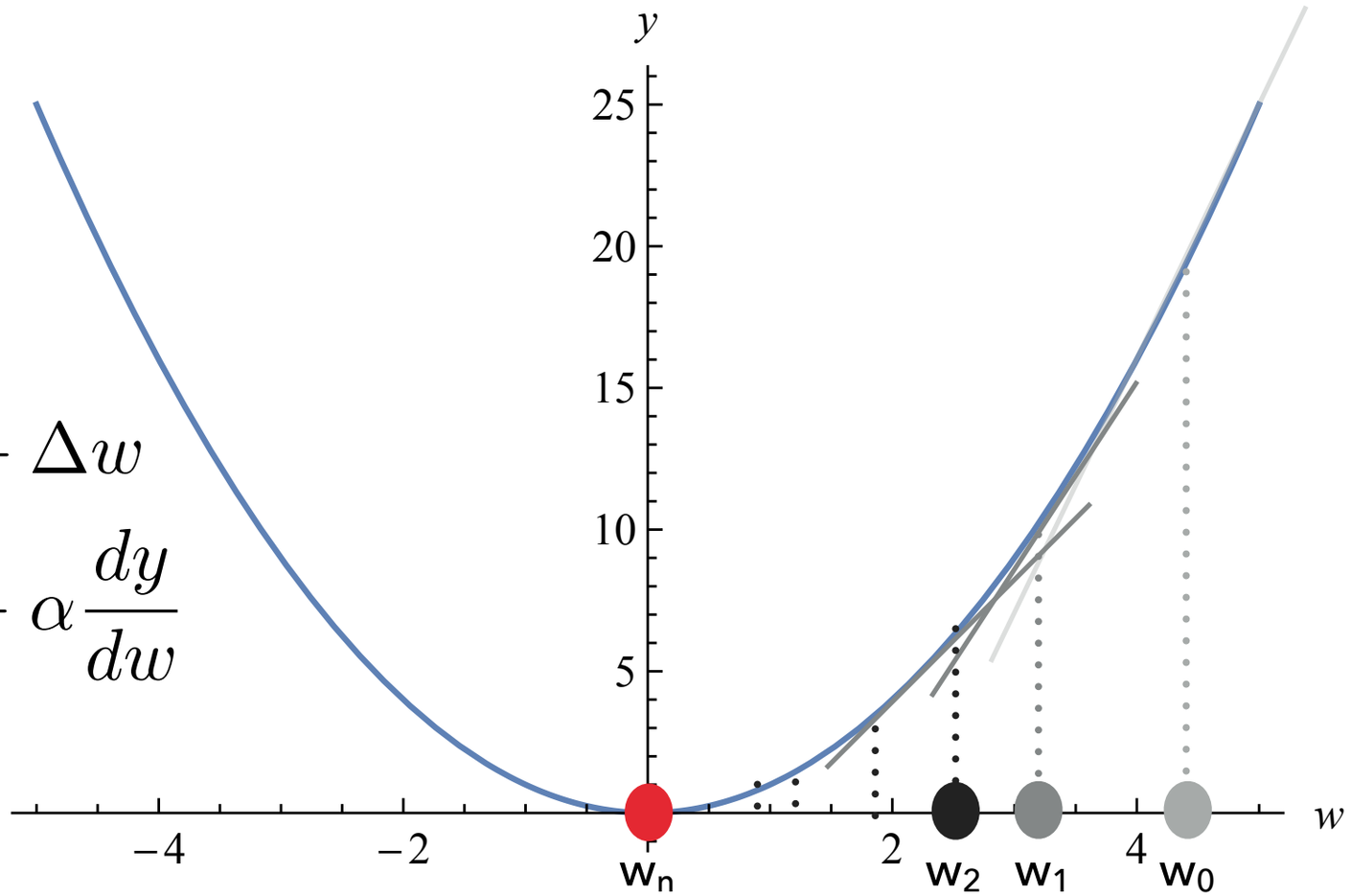
$$= w_i - \alpha \frac{dy}{dw}$$

α is the learning rate: a small positive number

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

GRADIENT DESCENT

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



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

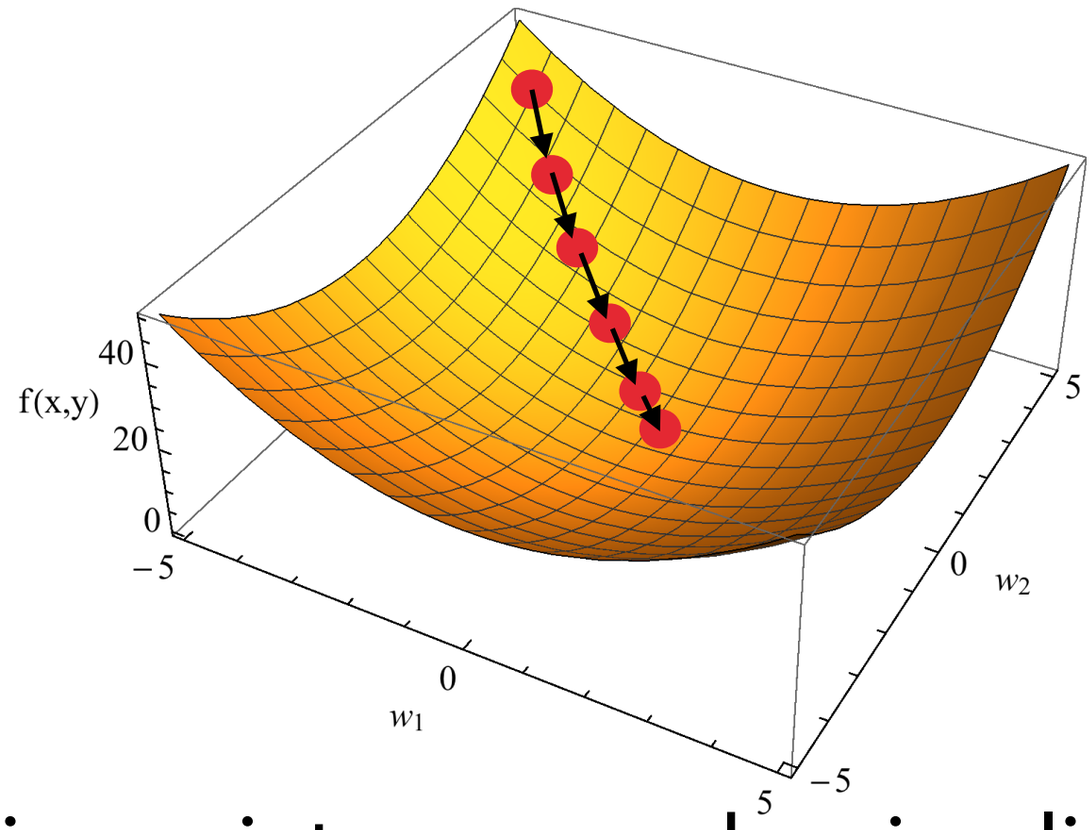
$$\Delta y = \Delta w \frac{dy}{dw}$$
$$= -\alpha \left(\frac{dy}{dw}\right)^2$$

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 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

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

$$= w_i - \alpha \nabla y$$

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

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 hyperspace 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 HPs 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.
- 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

GRADIENT DESCENT: ADAM OPTIMISER

- ADAptive Moment estimation based on gradient descent.

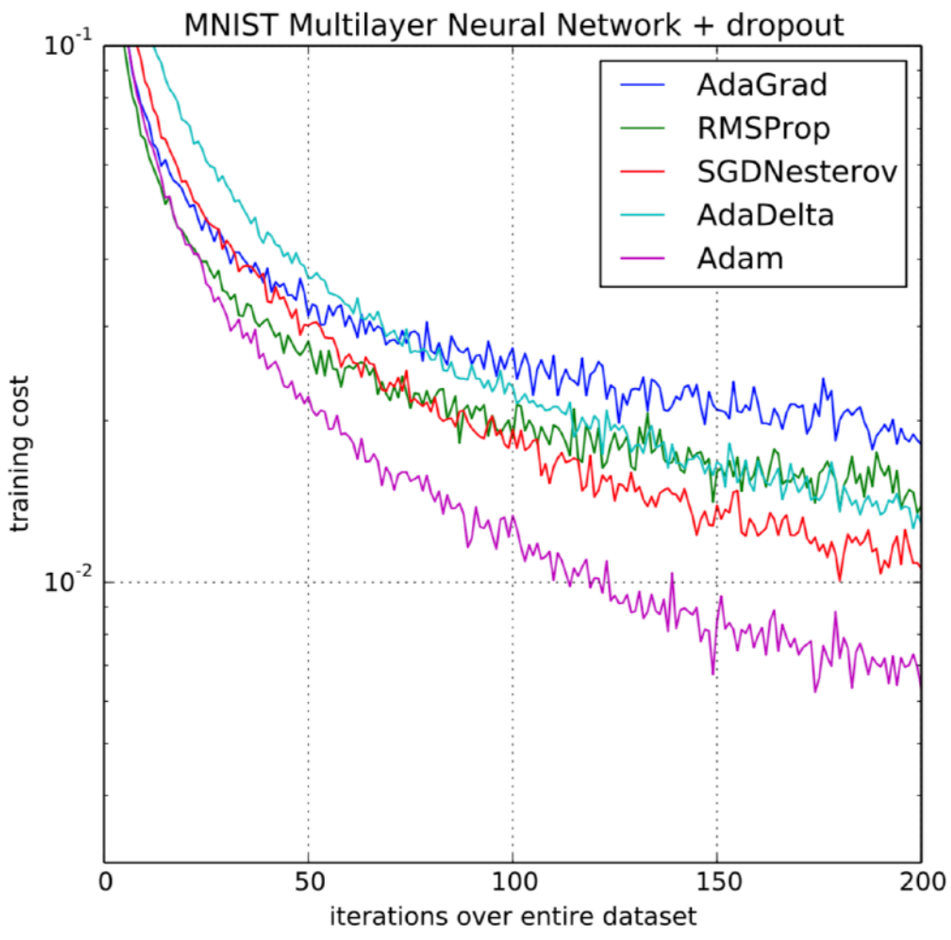
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha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

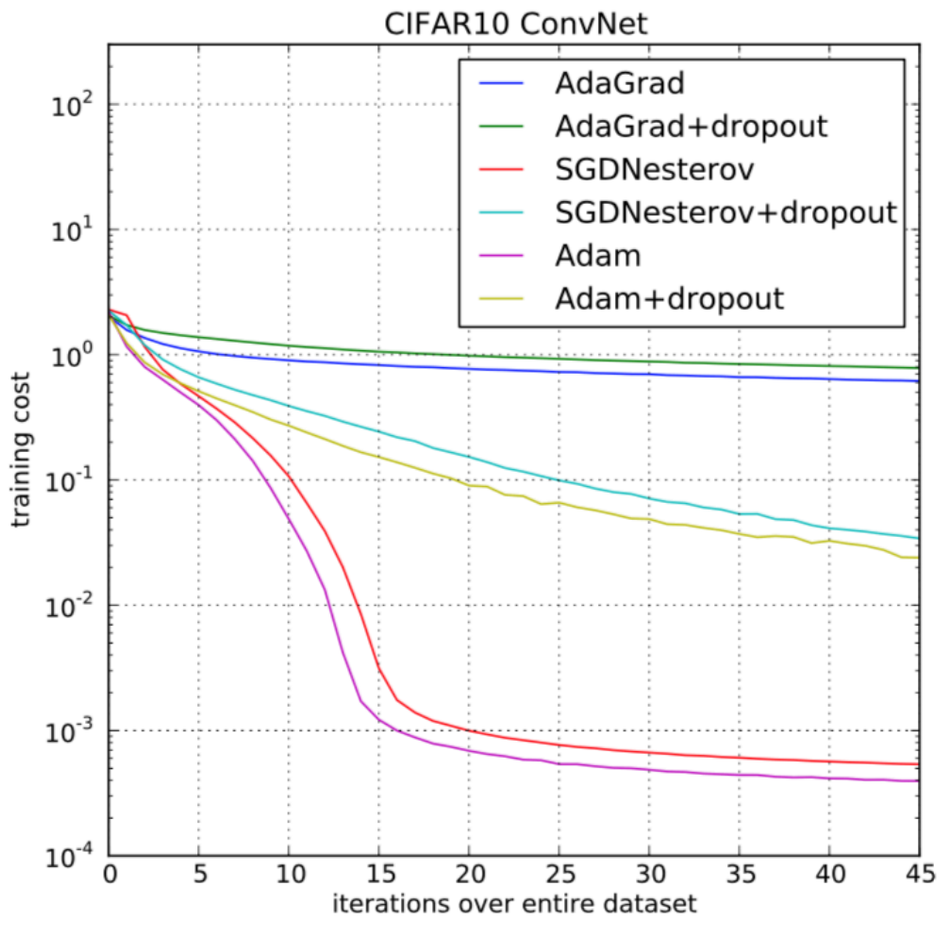
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd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

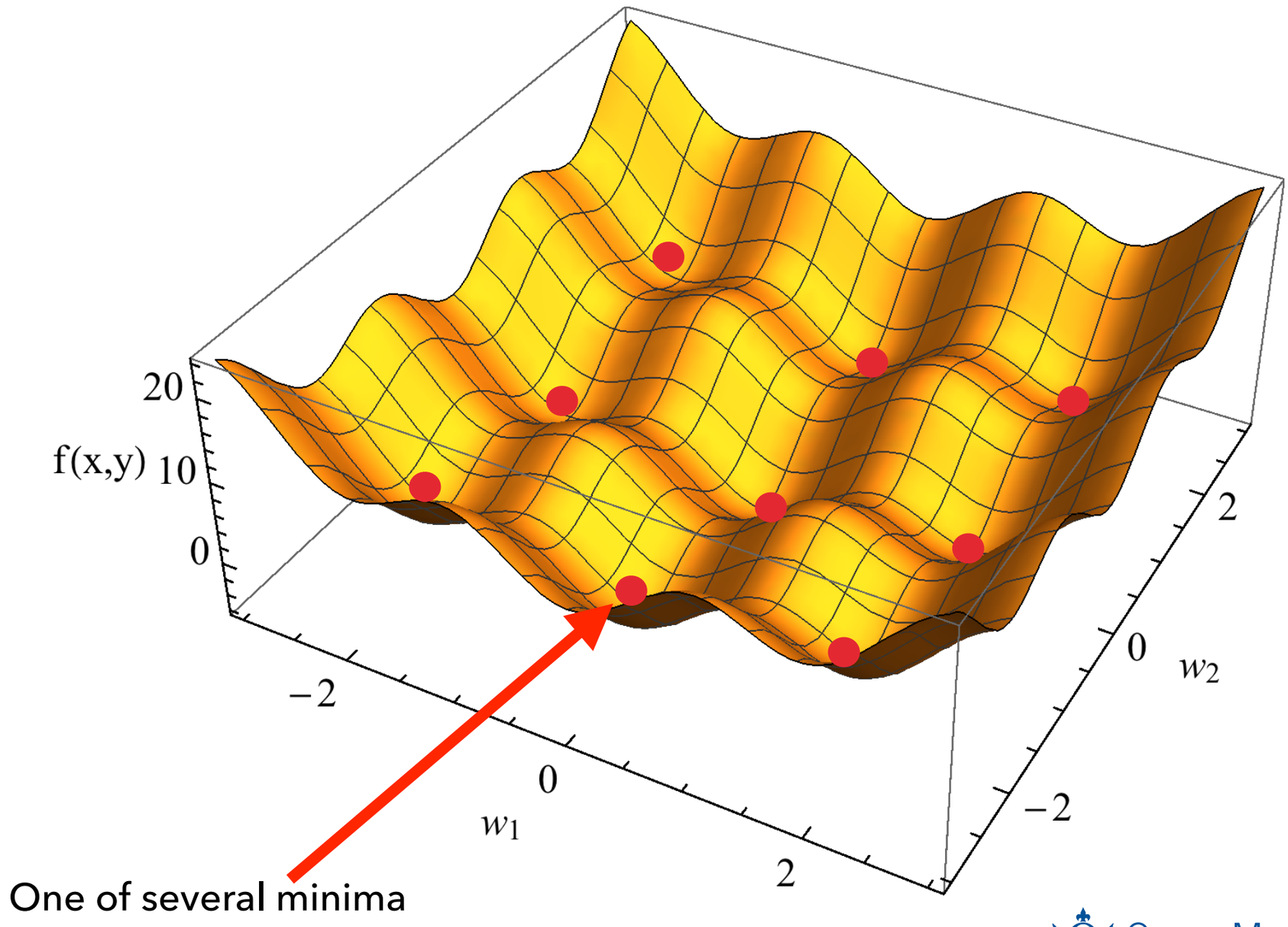
- Often more complication hyperspace optimisation problems are encountered, where there are multiple minima.

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?



GRADIENT DESCENT:MULTIPLE MINIMA

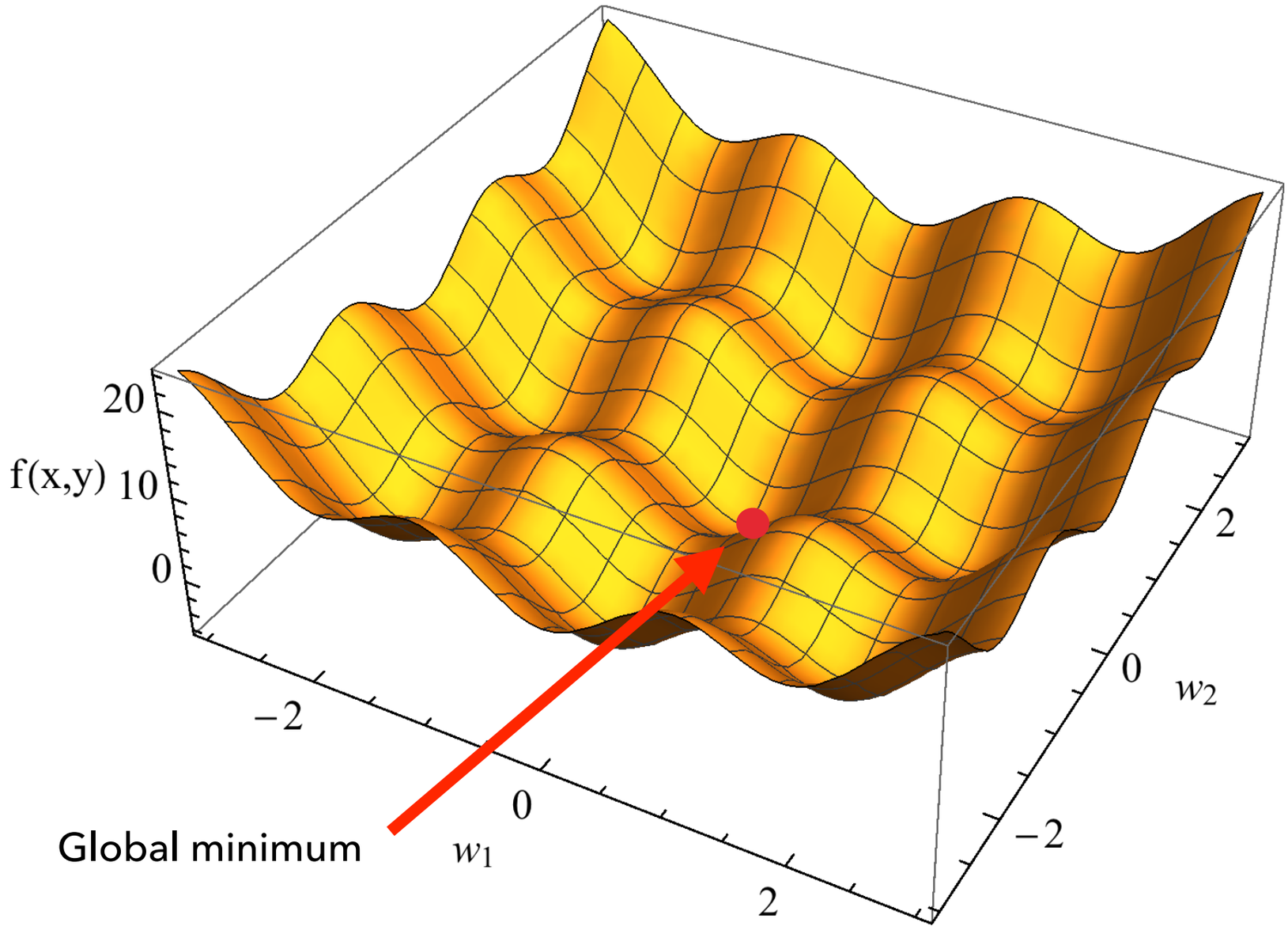
- Often more complication hyperspace optimisation problems are encountered, where there are multiple minima.

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?



GRADIENT DESCENT: DAVIDON–FLETCHER–POWELL

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

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

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

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

GRADIENT DESCENT: DAVIDON–FLETCHER–POWELL

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

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

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

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

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

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

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

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

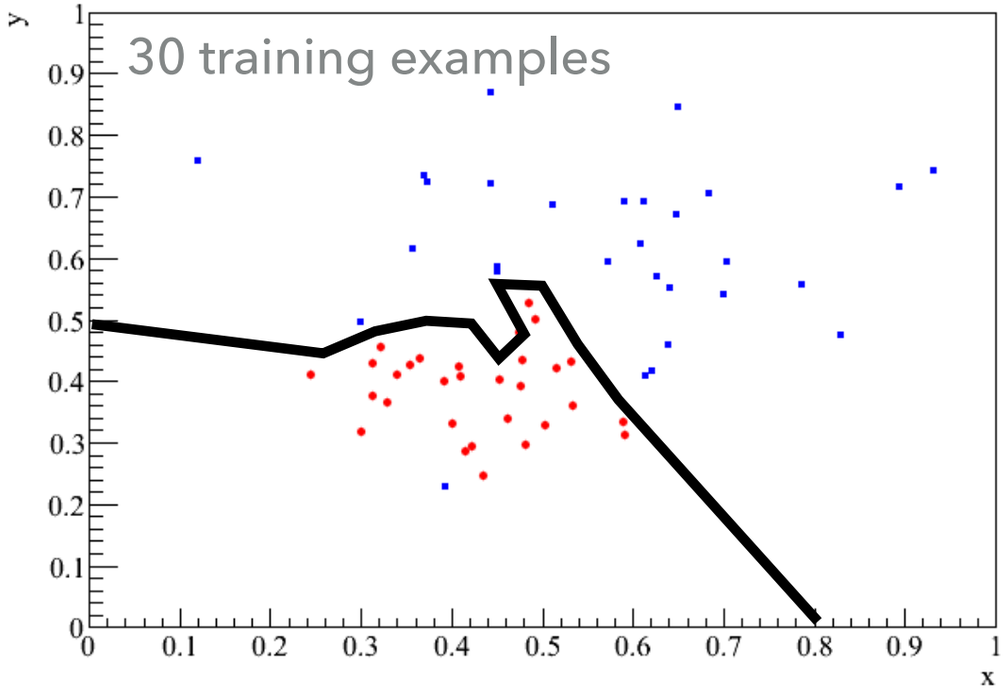
OVER FITTING (OVER TRAINING)

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

OVER FITTING

- 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:



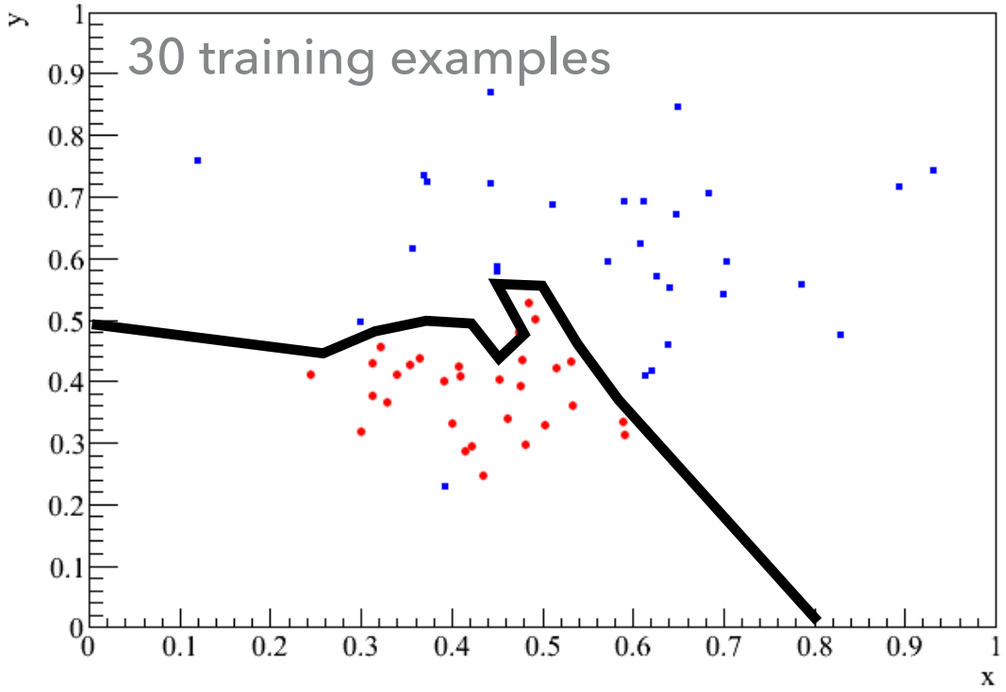
The decision boundary selected here does a good job of separating the red and blue dots.

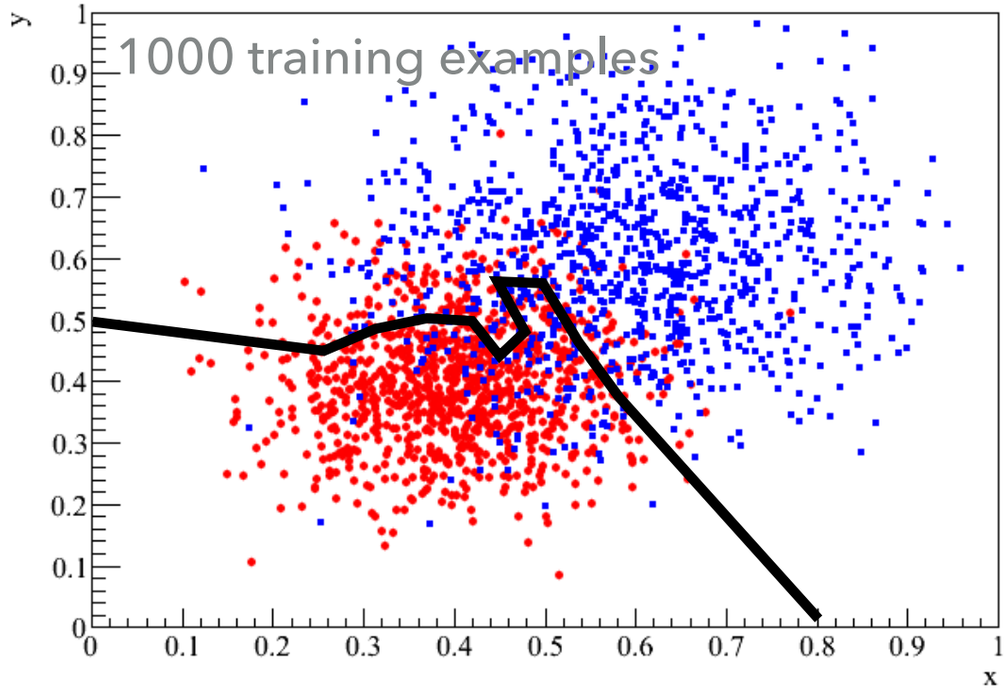
Boundaries like this can be obtained by training models on limited data samples. The accuracies can be impressive.

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

OVER FITTING

- 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.

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 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.

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 Δcost 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 HPs 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).

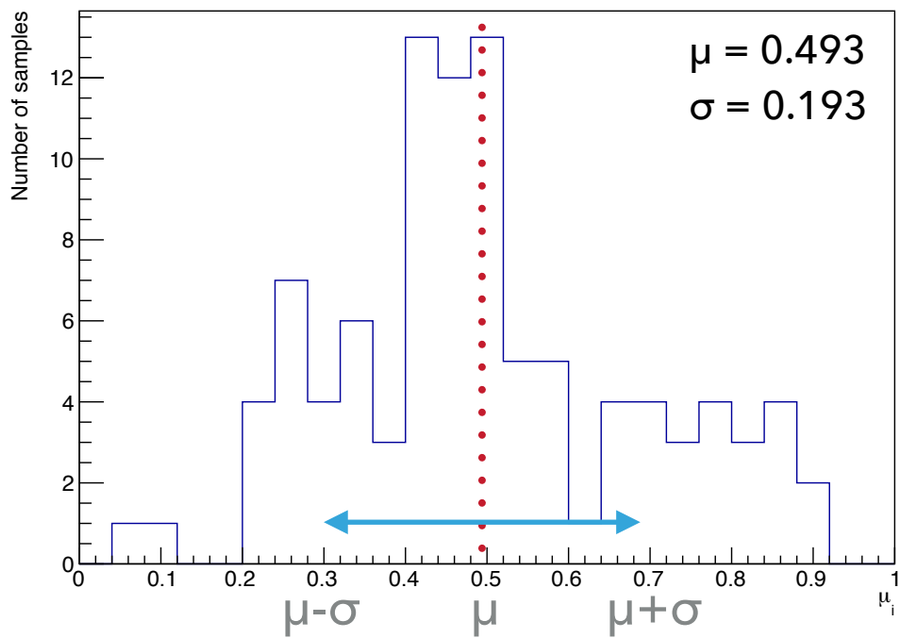
OVER FITTING: CROSS VALIDATION

- 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.

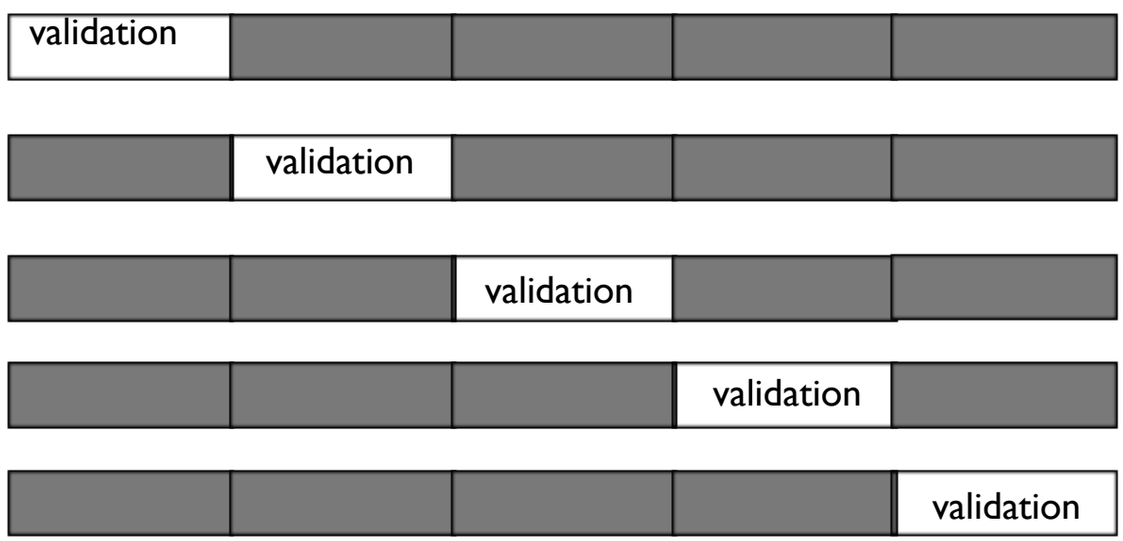
An extension of this concept is that if we train a model many times on smaller sub-samples, the average will be a more representative performance than an individual training.



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.

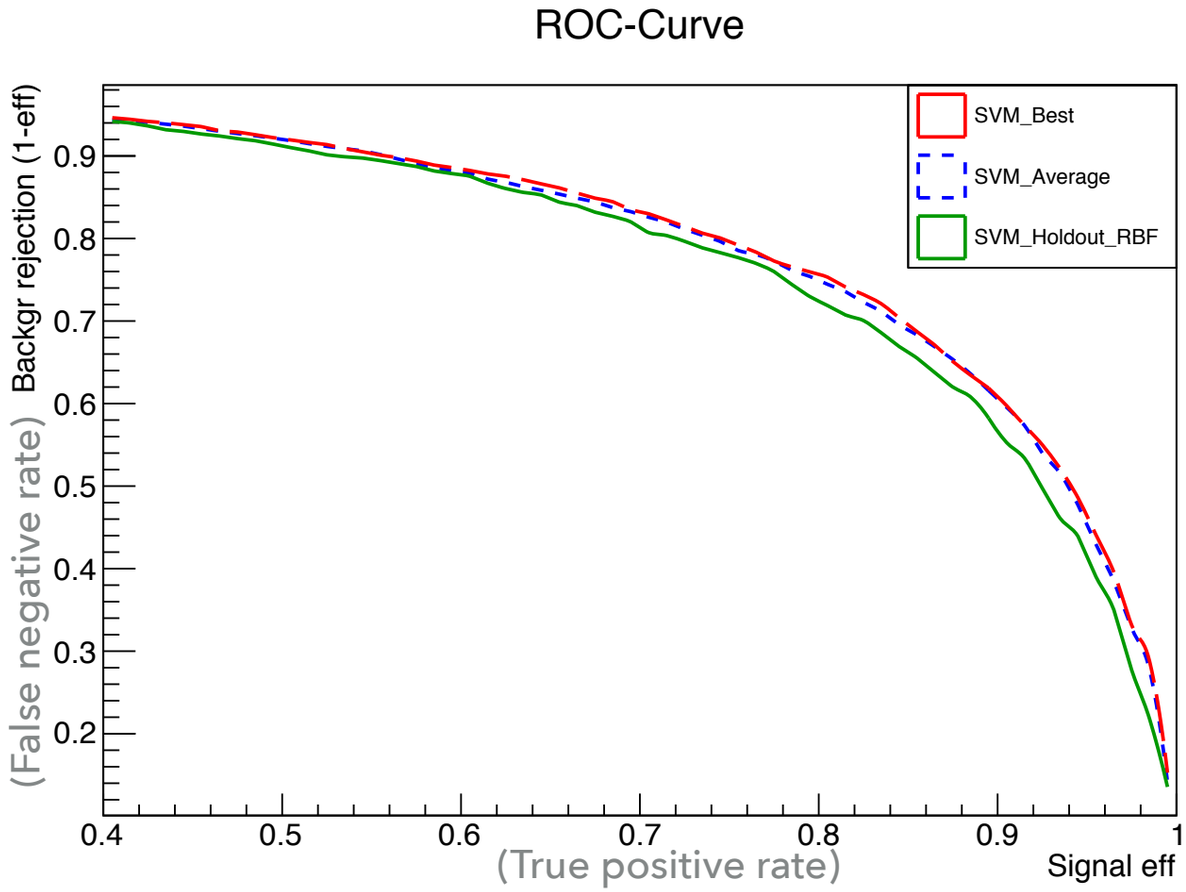


*Variants include the extremes of leave 1 out CV and Hold out CV as well as leave p-out CV. These involve reserving 1 example, 50% of examples and p examples for testing, and the remainder of data for training, respectively.

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.

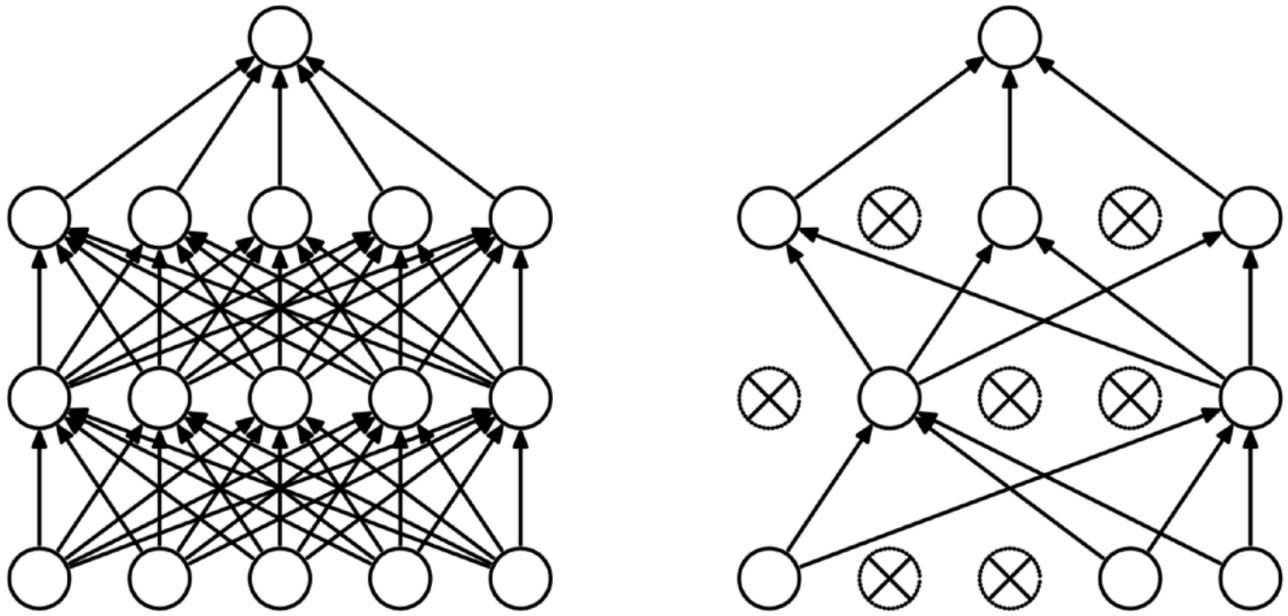


*Variants include the extremes of leave 1 out CV and Hold out CV as well as leave p-out CV. These involve reserving 1 example, 50% of examples and p examples for testing, and the remainder of data for training, respectively.

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.

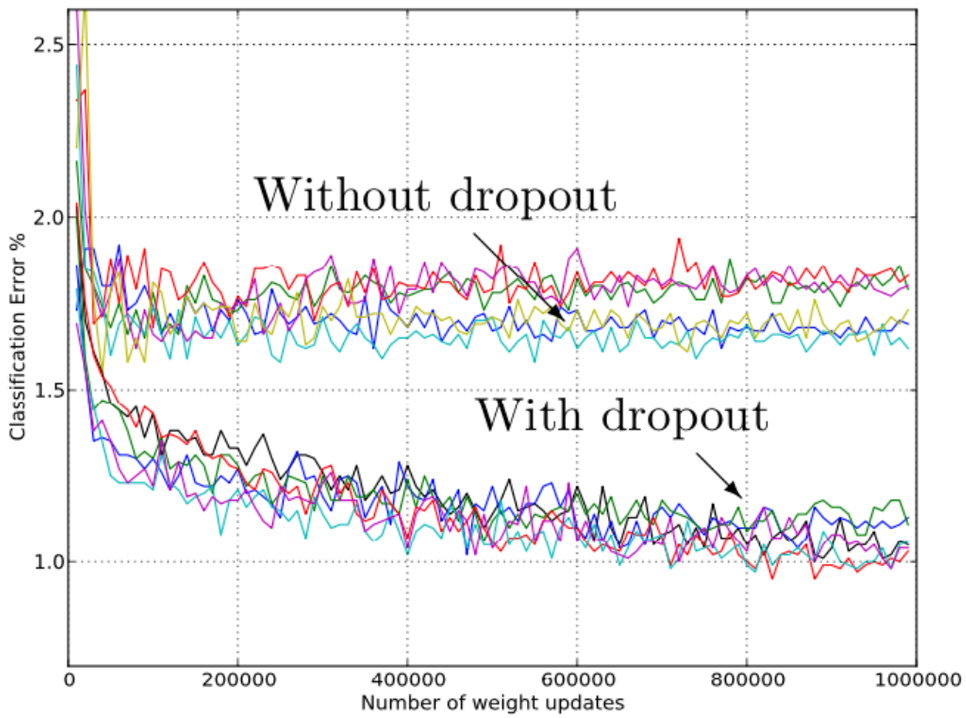
Dropout is used during training; when evaluating predictions with the validation or unseen data the full network of Fig. (a) is used.

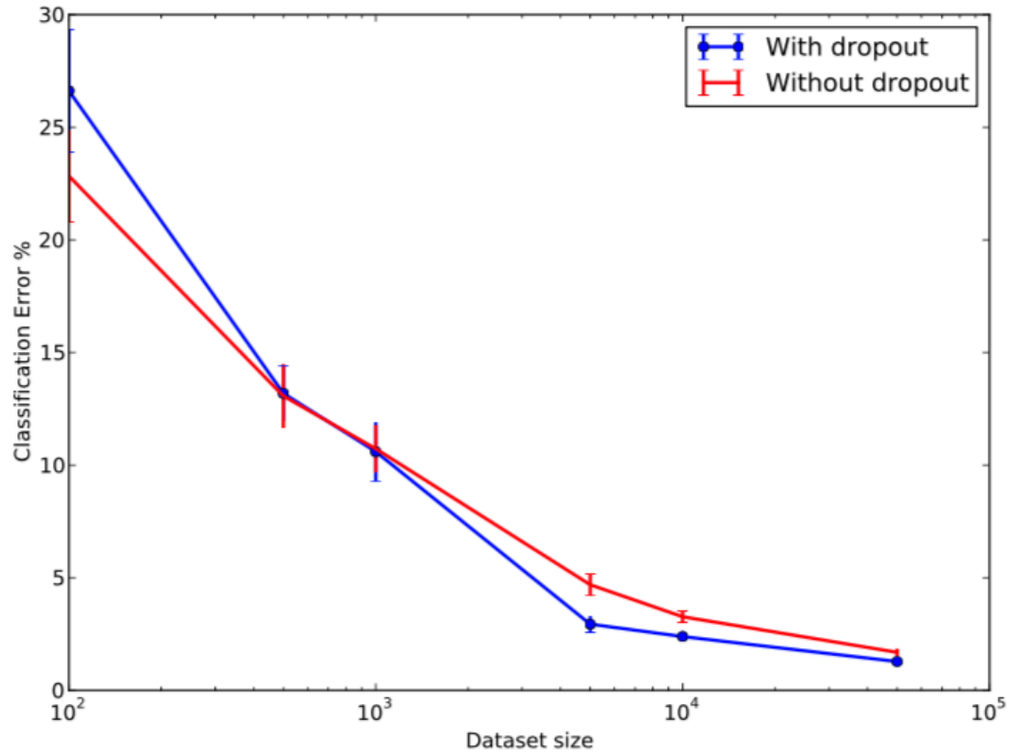
- That way the whole model will be effectively trained on a sub-sample of the data in the hope that the effect of statistical fluctuations will be limited.
- This does not remove the possibility that a model is overtrained, as with the previous discussion HP generalisation is promoted by using this method.

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

OVER FITTING: DROPOUT

- We can implement dropout trivially when training our networks: if you use dropout, training will require a large number of epochs.
  - Specify some placeholder: keep_prob:

```
keep_prob = tf.placeholder(tf.float32, name = "dropout_keep_prob")
```

- Use tf.nn.dropout on the model, and use the model with the dropout wrapper when computing the cost:

```
dmodel = tf.nn.dropout(model, keep_prob)
```

- Specify the drop out keep probability when training:

```
train_step.run(feed_dict={x: batch_img, y: batch_lbl, keep_prob: DropOutKeepProb})
```

SUMMARY

- Set the context for optimisation for supervised learning.
- Discussed several gradient descent based optimisers, including Adam;
- Over fitting discussed: the need to validate training to obtain robust predictive power for a model;
  - Weight regularisation;
  - Cross validation;
  - Dropout.
- The potential benefit of batch sample training to provide faster convergence has also been discussed.

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: 7
- C. Bishop: *Pattern Recognition and Machine Learning*
  - Chapters: 5
- T. Hastie, R. Tibshirani, J. Friedman, *Elements of statistical learning*
  - Chapter: 10 and 11

- In addition to the references given in the slides, you may also wish to read up on the topics of batch normalisation (a training acceleration process) and activation whitening (transforming activation inputs to look like a unit Gaussian and decorating the inputs to speed up training). For example see:
  - Ioffe and Szegedy, arXiv:1502.03167 and references therein
  - LeCun et al., Efficient BackProp, Neural Networks Tricks of the Trade, Springer 1998