ADRIAN BEVAN

# INTRODUCTION TO MACHINE LEARNING PART 1

LECTURES GIVEN IN THE PARTICLE PHYSICS DIVISION AT THE RUTHERFORD APPLETON LABORATORY, MAY/JUNE 2020

# OVERVIEW

- What is machine learning?
- Ethics
- Data wrangling
- Optimisation
- Decision Trees
- Neural Networks
    - Multilayer Perceptron (MLP)
    - Auto-encoders
    - Convolutional Neural Networks (CNN)
    - Generative Adversarial Networks
- Support Vector Machines
- KNN
- Explainability and Interpretability
- Appendix

Machine Learning is a huge field, and here I focus on a restricted set of topics, as an introduction into the subject.

Where algorithms or issues are explored in more depth this is generally done for pedagogical reasons, or the algorithm is widely used, or because the issue is general to the field.

# OVERVIEW

- Examples are provided for the tutorials.
- These use ROOT or Python and have been tested using the following versions:
  - ROOT: 6.06/02
  - Python 3.7 (via an Anaconda install) with the following modules
    - TensorFlow 2.2 [Note: examples are not taking advantage of eager execution and are using the V1 backward compatibility mode]
    - matplotlib
    - numpy
    - sklearn

# WHAT IS MACHINE LEARNING?

# WHAT IS MACHINE LEARNING?

- Oxford English Dictionary:
  - *“a type of artificial intelligence in which computers use huge amounts of data to learn how to do tasks rather than being programmed to do them”*
- *Collins Dictonary:*
  - *“a branch of artificial intelligence in which a computer generates rules underlying or based on raw data that has been fed into it”*
- Google Developers glossary:
  - *“A program or system that builds (trains) a predictive model from input data. The system uses the learned model to make useful predictions from new (never-before-seen) data drawn from the same distribution as the one used to train the model. Machine learning also refers to the field of study concerned with these programs or systems.”*
- There is no single definition agreed of machine learning, so I will use a working definition.
  - *“The process of using an algorithm to approximate data using some underlying optimisation heuristic”*

# WHAT IS MACHINE LEARNING?

  - *"The process of using an algorithm to approximate data using some underlying optimisation heuristic"*

- We are doing function approximation using some algorithm fit to some reference data using some heuristic.

- The fitting in this context is referred to as training or learning.

  - There are different learning paradigms, we will focus on supervised and unsupervised learning.

- The trained function is used as a model (i.e. for prediction).

- Dimensionality and parameters are implied in these slides when referring to general situations, i.e. $f(\underline{x}, \underline{\theta}) \rightarrow f(x, \theta) \rightarrow f(x)$.

# ETHICS

# ETHICS

- How does ethics fit into a lecture on machine learning?

**Ethics** *plural in form but singular or plural in construction* **:** the discipline dealing with what is good and bad and with moral duty and obligation.

**Morals** describes one's particular values concerning what is right and what is wrong.

Using definitions from the Miriam Webster Dictionary

# ETHICS

- How does ethics fit into a lecture on machine learning?
  - Is it ethical to develop an algorithm to identify the political leaning of an individual with the intent of targeting advertising, to polarise the viewpoint of voters with positive reinforcement messages or fake news, to undermine or influence the outcome of an election?
  - Is it ethical to develop a pandemic model (e.g. using an SIR approach[1]) using an AI, that could influence Government policy, without fully testing the robustness of predictions?

[1] e.g. see this article on wikipedia Compartmental models in epidemiology. A good illustration of a variant of this model can be found at https://upload.wikimedia.org/wikipedia/commons/3/31/SIRD_model_anim.gif

# ETHICS

- But that is the “real world” why should I care about ethics?
    - Is it ethical to use an algorithm for science without checking that the result makes sense?

# ETHICS

- But that is the "real world" why should I care about ethics?
  - Is it ethical to use an algorithm for science without checking that the result makes sense?
  - Is it ethical to use an energy intensive algorithm when a computationally cheap alternative that performs just as well exists?

# ETHICS

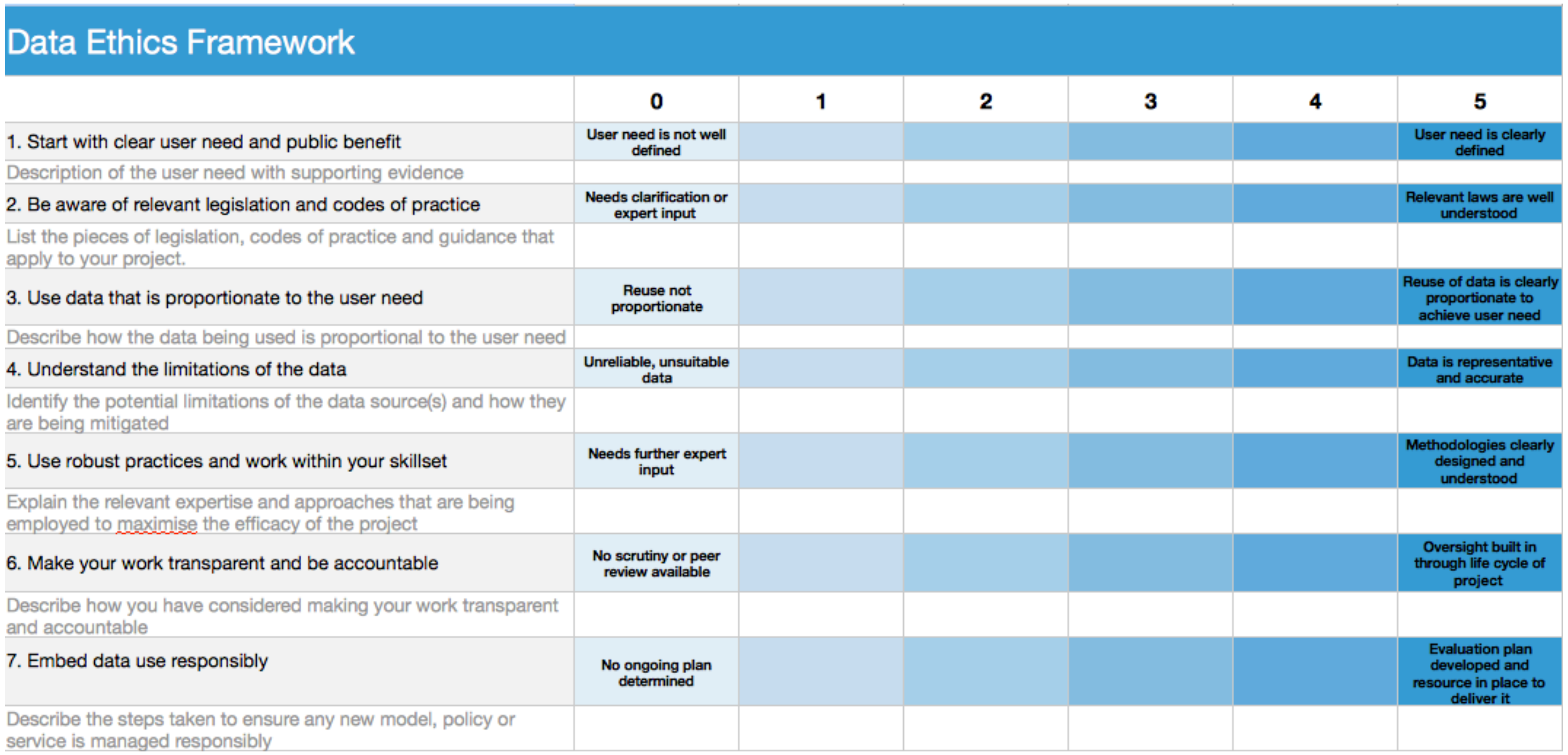
- But that is the “real world” why should I care about ethics?
  - Is it ethical to use an algorithm for science without checking that the result makes sense?
  - Is it ethical to use an energy intensive algorithm when a computationally cheap alternative that performs just as well exists?
- What we think of as good scientific practice, is also ethical behaviour. It leads to robust results.
- Ethical behaviour in the wider world can have deeper ramifications than getting a robust scientific result or not.

# ETHICS

- But that is the "*real world*" why should I care about ethics?
    - In reality most people who do a PhD in an STFC (or EPSRC) area of science will not end up in academia, and ethics will play a role beyond scientific correctness for those scientists.
    - For those of us who do stay in academia, then we have an obligation to help people understand the ramifications of algorithms and our rationale for using them (or not).
    - For all of us, we have transferable skills that can be applied to real world problems.

# ETHICS

- The UK Government developed a data ethics framework[1]:
    - *"Public sector organisations should use the Data Ethics Framework to guide the appropriate use of data to inform policy and service design"*



Data Ethics Framework

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 1. Start with clear user need and public benefit | User need is not well defined | | | | | User need is clearly defined |
| Description of the user need with supporting evidence | | | | | | |
| 2. Be aware of relevant legislation and codes of practice | Needs clarification or expert input | | | | | Relevant laws are well understood |
| List the pieces of legislation, codes of practice and guidance that apply to your project. | | | | | | |
| 3. Use data that is proportionate to the user need | Reuse not proportionate | | | | | Reuse of data is clearly proportionate to achieve user need |
| Describe how the data being used is proportional to the user need | | | | | | |
| 4. Understand the limitations of the data | Unreliable, unsuitable data | | | | | Data is representative and accurate |
| Identify the potential limitations of the data source(s) and how they are being mitigated | | | | | | |
| 5. Use robust practices and work within your skillset | Needs further expert input | | | | | Methodologies clearly designed and understood |
| Explain the relevant expertise and approaches that are being employed to maximise the efficacy of the project | | | | | | |
| 6. Make your work transparent and be accountable | No scrutiny or peer review available | | | | | Oversight built in through life cycle of project |
| Describe how you have considered making your work transparent and accountable | | | | | | |
| 7. Embed data use responsibly | No ongoing plan determined | | | | | Evaluation plan developed and resource in place to deliver it |
| Describe the steps taken to ensure any new model, policy or service is managed responsibly | | | | | | |

- Other organisations such as the UN and IEEE (as well as the EU) are also concerned about ethical use of data, and ethical algorithms, from regulatory and the perspective of rights.

[1] See https://www.gov.uk/government/publications/data-ethics-framework

# DATA WRANGLING

# DATA WRANGLING (AKA FEATURE ENGINEERING)

- Need to understand the data being analysed.
  - Domain context will provide most insight into getting information to highlight the solution to your problem.
  - Identify the features of interest in the data.
  - Allow you to reduce the dimensionality of the problem into as few a set of inputs as possible.
  - Most of the analysis can be done with this domain context background:
    - Deep Learning (DL) can replace the hard work of feature engineering for a resource cost and a lack of explainability and interpretability. [1]
    - “Smart Learning” (SL) is an alternative way of approaching the problem. [2]

[1] e.g. See work by Pierre Baldi on Higgs analysis at the LHC: DOI: 10.1038/ncomms5308. Also see appendix.

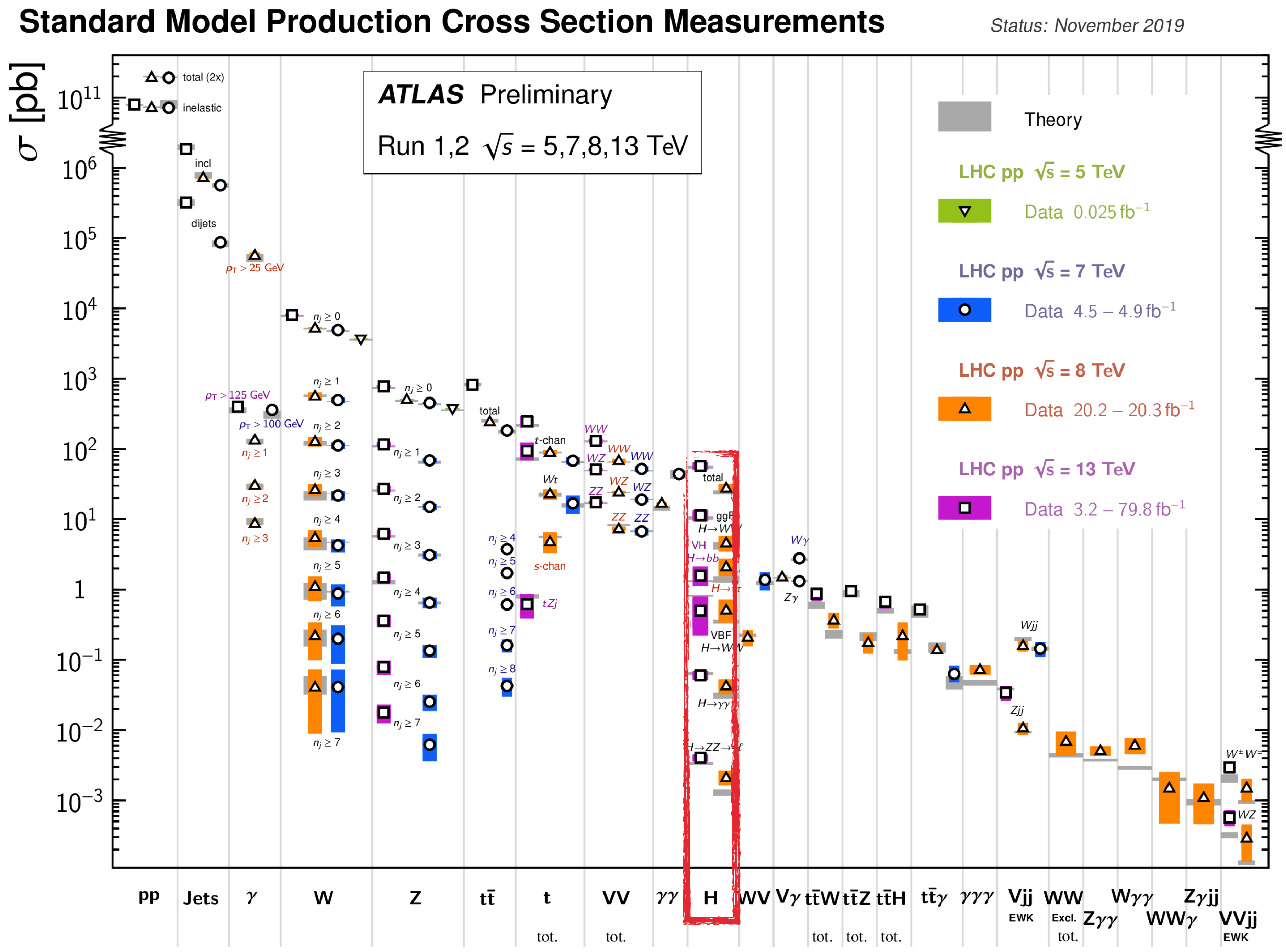
[2] I attribute the term Smart Learning to the use of domain knowledge and understandable AI, such as Bayesian Networks. A term that I first heard about from Norman Fenton who has a good book on the topic.

# DATA WRANGLING

- In HEP we use cut based selection to:
  - Remove pathological data (poorly calibrated or partially complete data, bad beam conditions, etc).
  - Remove obvious background examples from the data (well known SM processes that are just not interesting for study, or use as a calibration or control sample).
  - Prepare data for the "statistical analysis" that will include the use of multivariate analysis techniques and fitting.

# DATA WRANGLING

- e.g. Higgs physics: Throw away ~$10^9$ recorded events for each interesting one.



# DATA WRANGLING

- Knowing what the task is allows one to identify the features of interest. This is the domain context knowledge.
  - If your signal is the process pp→HH+X, then:
    - Laws of physics provide insights for you to refine your list of features to train on.
    - e.g. the Higgs mass or $p_T$ will play a role in identifying the signal, along with decay product properties (e.g. b-tag quality), etc.

# DATA WRANGLING

- Reduce the number of dimensions of interest:
    - Trial and error with different inputs to identify what improves performance of the algorithm and what does not.
    - Linearly correlated features can be combined to reduce the number of features providing information for the algorithm to learn from.
        - Principal Component Analysis (PCA) to address this problem.
    - Some neural network configurations do that automatically for you (e.g. auto-encoders).
    - Let the algorithm learn what is important and what is not.
- Some algorithms (e.g. support vector machines) implicitly increase the dimensionality of the problem.

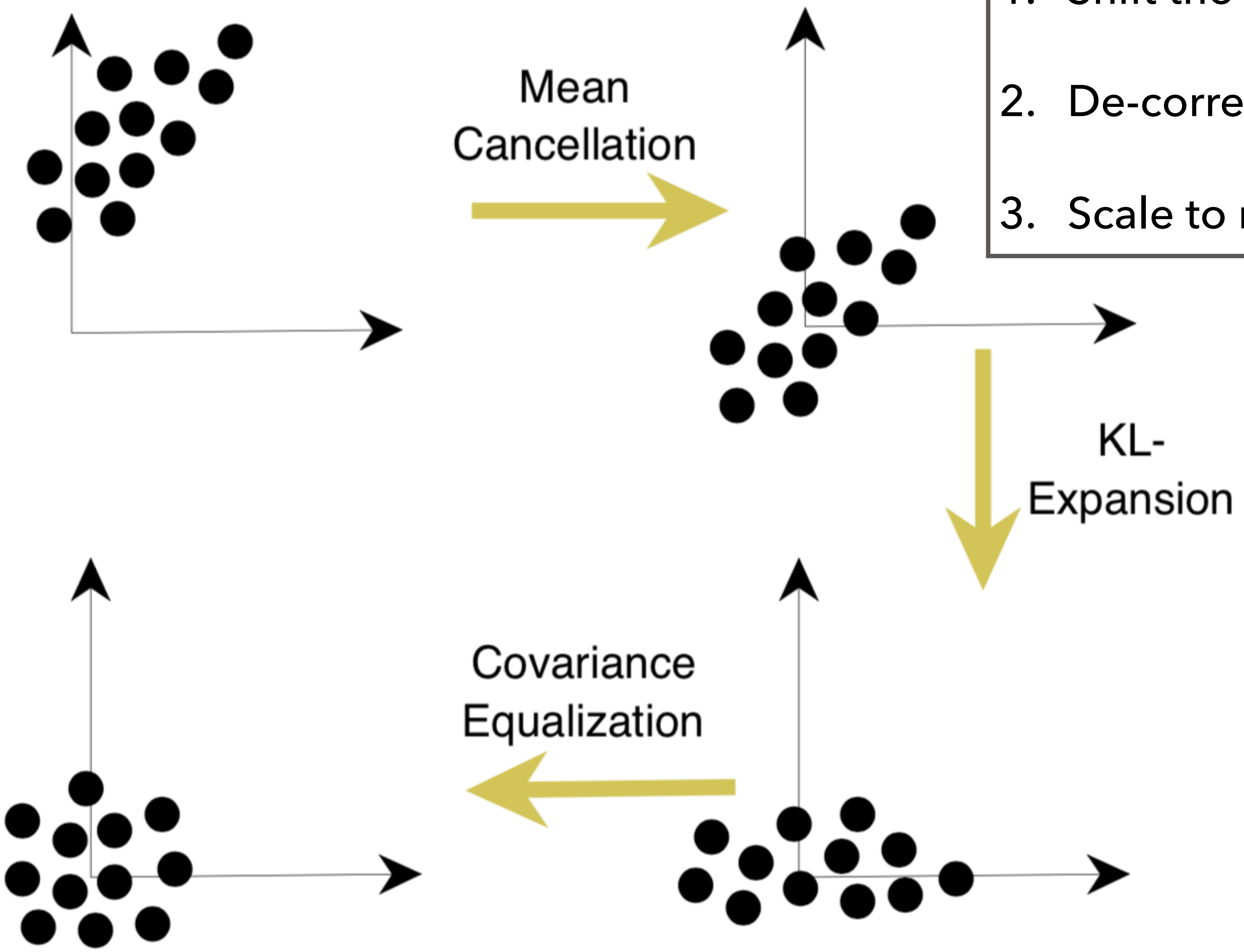
# ACTIVATION FUNCTIONS: DATA PREPARATION

- Input features are arbitrary; whereas (for example) activation functions in neural networks work best for 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 $\longmapsto$ TeV would translate $10^{12}$ eV $\longmapsto$ 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.

# DATA WRANGLING

- Variable transformations are also useful (can be vital).

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



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

# DATA WRANGLING

- Data wrangling is an important part of ensuring you get the most out of applying machine learning; yet it is often not celebrated as the requirements can be very problem specific.

- If you are interested in this topic you may wish to review the data wrangling presentation and tutorials given by Dr Nick Barlow from the Alan Turing Institute at the GradNet meeting on Machine Learning and AI in January 2020:
  - Dr Barlow's talk and lecture can be found at: https://indico.ph.qmul.ac.uk/indico/conferenceOtherViews.py?view=standard&confId=543

TYPES OF LEARNING
GRID SEARCH
GRADIENT DESCENT
ADAM OPTIMISER
MISCELLANEOUS

# OPTMISATION

A number of optimisation methods exist, and I selectively focus on three. For example see C. Bishop, Neural Networks for Pattern Recognition or I. Goodfellow et. al, Deep Learning for more information on optimisation algorithms.

# OPTIMISATION

## TYPES OF LEARNING

# TYPES OF LEARNING

- Consider a model $y$ that depends on a parameter set $\theta$, we can select a point in the parameter hyperspace $\hat{\theta}$ that has a corresponding estimator of the function $\hat{y}$.

- The $\theta$ are hyper-parameters (HPs) of the model.

- **Unsupervised learning:**
  - Infer the probability distribution $P(\widehat{y})$ from the data without using labels.
  - Widely used in many fields of research.

- **Supervised learning:**
  - Use training examples from labeled data sets with a known type or value, $t$, for each example.
  - Some loss function is used to compare $t$ against model predictions $\widehat{y}$.
  - Can be thought of as computing a conditional probability $P(t\,|\,\widehat{y})$.
  - This is the most commonly used approach in particle physics today.

# TYPES OF LEARNING: SUPERVISED LEARNING

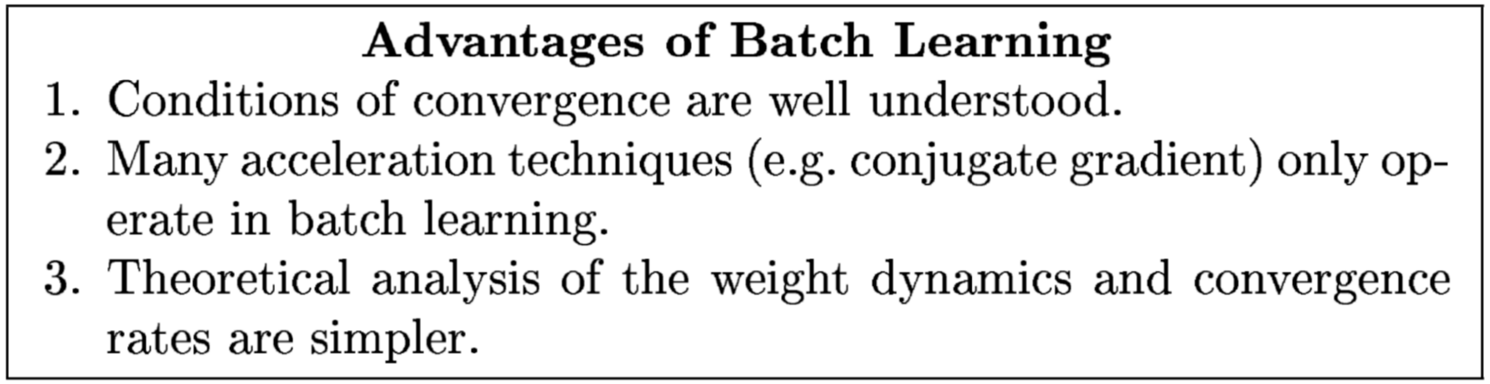
- Define a heuristic based on some figure of merit (FOM) designed to improve the value of that metric through some iterative process.
- The FOM is called the objective function or loss function or cost function in machine learning.
- e.g. the $L_2$ norm loss function: this is like a $\chi^2$, but without the error term:

$$L_2 = \sum_{i=1}^{N_{examples}} \left[ t_i - \hat{y}(\hat{\theta}) \right]^2$$

- The $\theta$ are called HPs as they form a hyperspace; other variables in the optimisation process are often included in the set of HPs (e.g. learning rate for a neural network, cost for a support vector machine, tree depth for a decision tree etc.).

# TYPES OF LEARNING

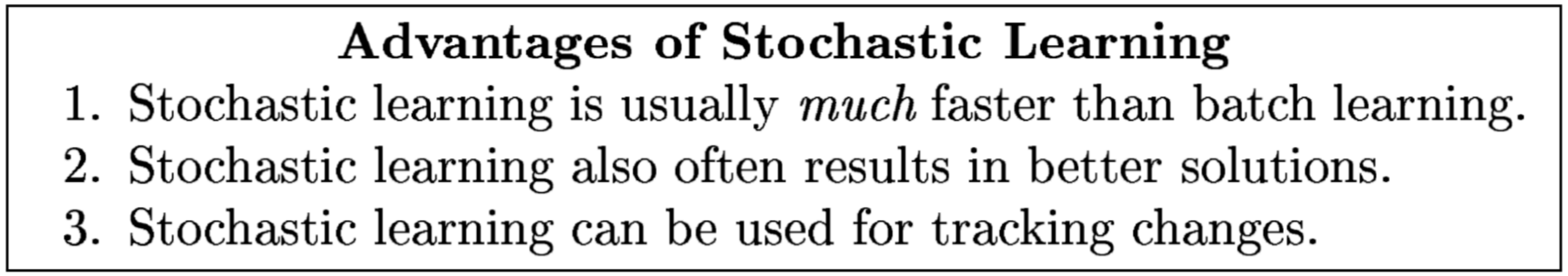
- **Batch learning**
  - Use a sample (batch) of training data to evaluate an estimate of the error and update weights in the optimisation.



**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.

- **Stochastic learning**
  - Use individual training examples to evaluate an estimate of the error and update weights in the optimisation.



**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.

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

- THIS IS A SIMPLE OPTIMISATION ALGORITHM THAT CAN BE USED WITH NO PROGRAMMING EXPERIENCE.
- ONE CAN IMPLEMENT A 1 OR 2 DIMENSIONAL GRID SEARCH USING AN EXCEL SPREADSHEET, AND FROM INSPECTION OF THE TABULATED RESULTS YOU CAN OPTIMISE SIMPLE PROBLEMS.
- THIS IS A BRUTE FORCE OPTIMISATION APPROACH, AND IT IS USED WIDELY IN CERTAIN APPLICATIONS ACROSS A NUMBER OF RESEARCH FIELDS.

# OPTIMISATION

## GRID SEARCH

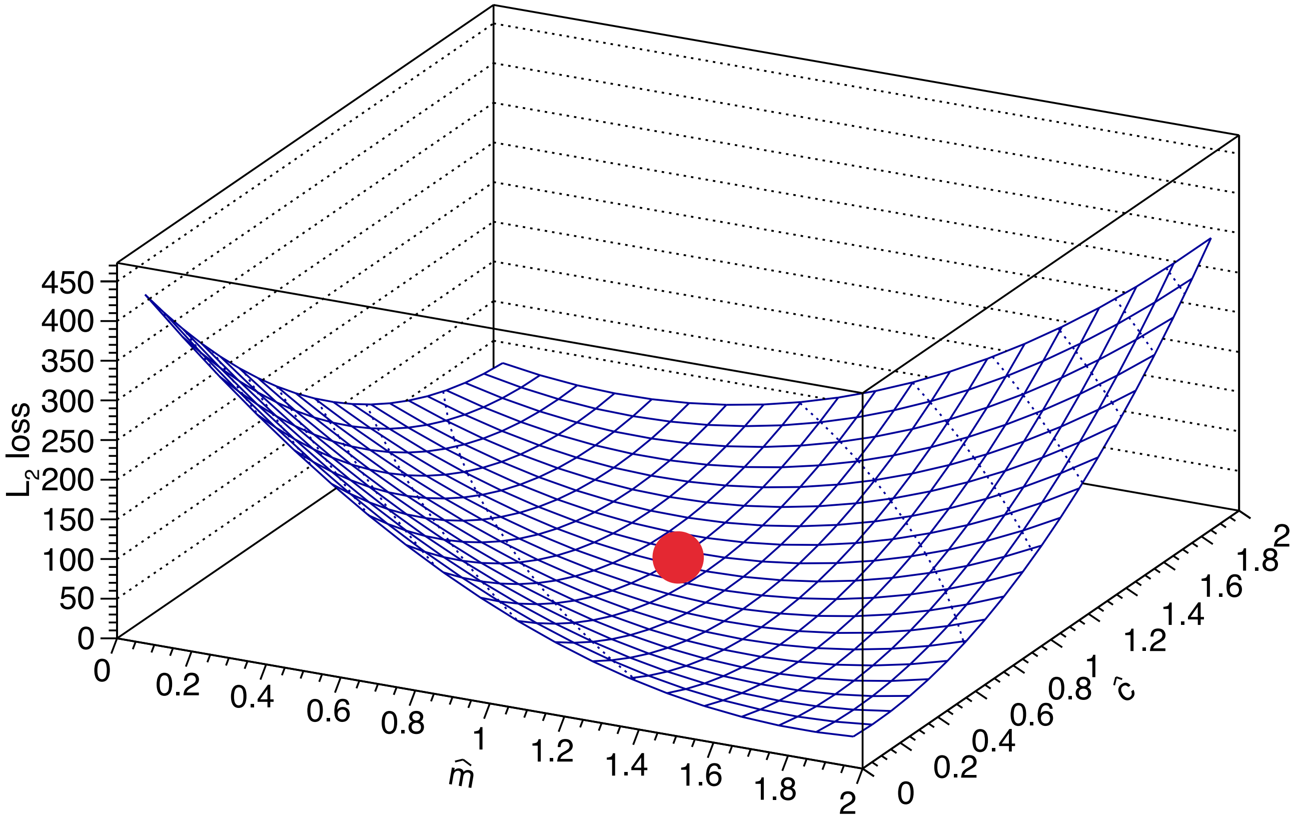
# GRID SEARCH

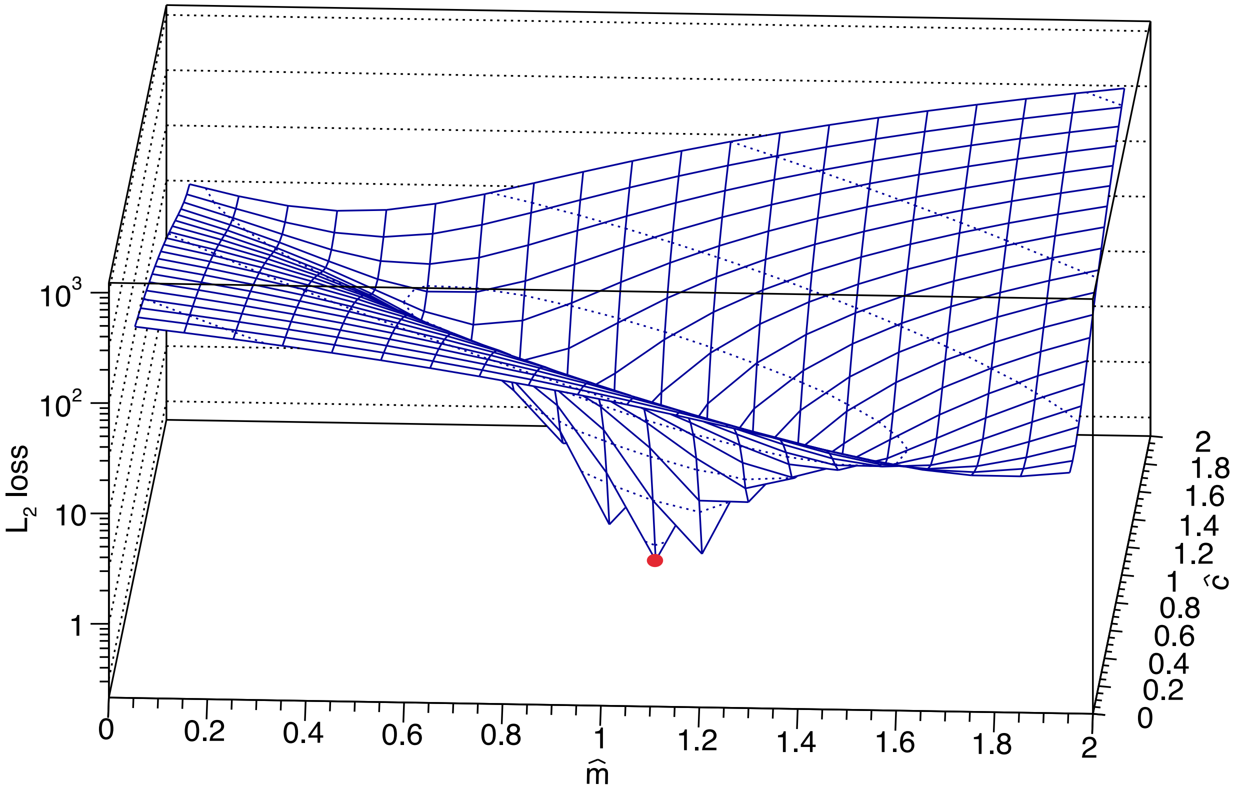
- Scanning through the hyperspace for each $\hat{\theta}_i$ allows us to compute $L_2$, and the minimum value obtained is the "best fit" or optimal estimate of the parameters $\theta$.
  - Simple heuristic to implement - you can do this in Excel for 1 or 2D problems. ✓
  - Easy to understand. ✓
  - Expensive to compute: scanning $n$ points in a dimension requires $n^M$ computations for $M = dim(\theta)$. ✗
- Heuristic suffers from the curse of dimensionality.

macros/gridSearch.cc

# GRID SEARCH

- e.g. Consider an L2 loss function optimisation of the HPs for $y = mx + c$, i.e. optimise $m$ and $c$. Here $m = 1.0, c = 1.0$





- The contours of the loss function show a minimum.
- The optimal value is selected from a discrete grid of points, only get an exact result if the grid maps onto the problem perfectly (as in this example).

# GRID SEARCH

- e.g. The libsvm[1] package uses a HP grid search for optimisation for the Support Vector Machine algorithm; where the cost C and $\Gamma$ hyper-parameters need to be optimised.
    - The grid search is done efficiently by adapting to whatever step is sensible, in this case the algorithm has $\Gamma$ as the parameter of an exponential (the radial basis function); and so a linear search in C and a logarithmic search in $\Gamma$ is appropriate to sample the parameter hyperspace effectively.

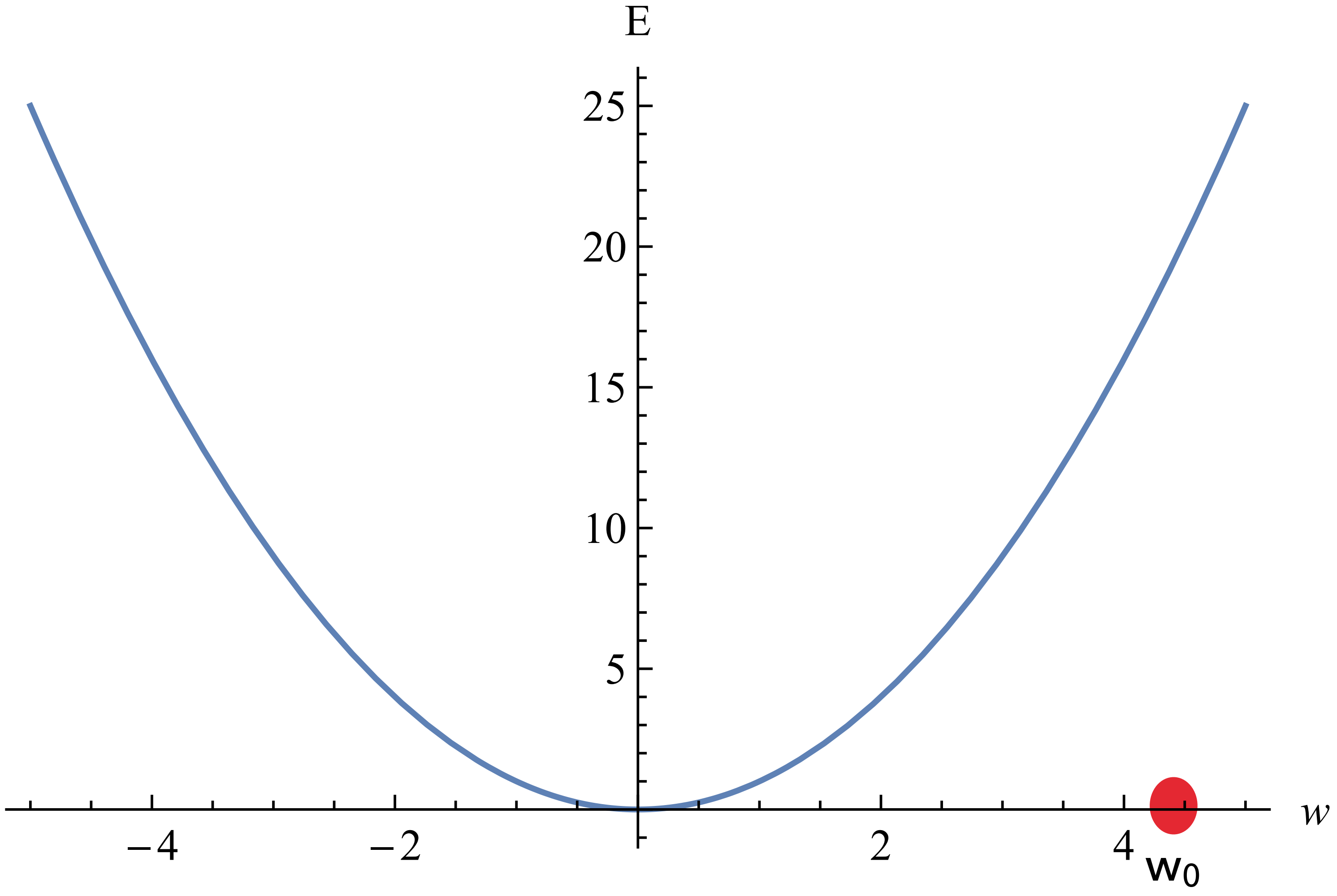
[1] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, **2**:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

# OPTIMISATION

## GRADIENT DESCENT

# GRADIENT DESCENT

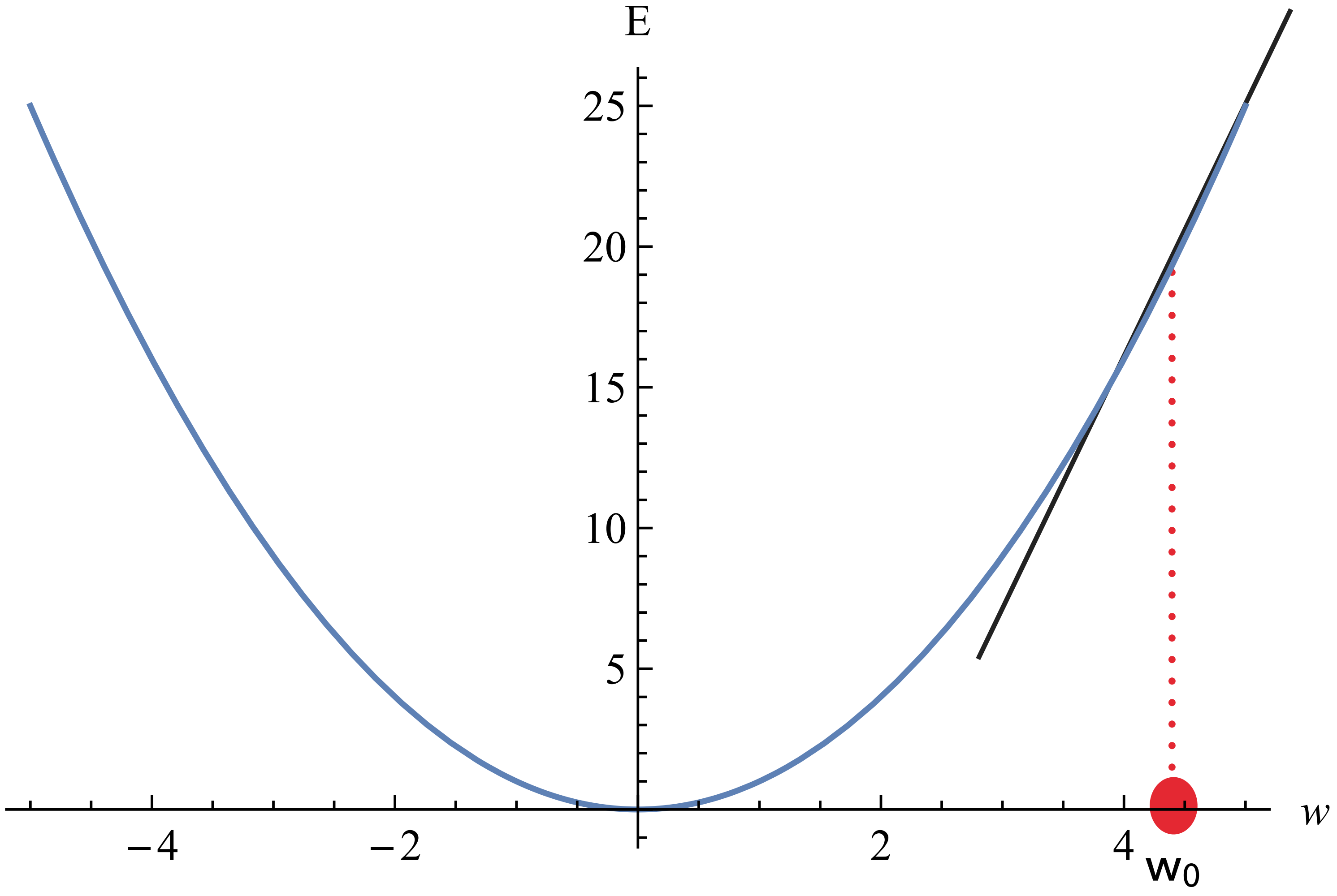
- Guess an initial value for the weight parameter: $w_0$.



e.g. see Bishop, Neural Networks for Pattern Recognition (2013) Oxford University Press

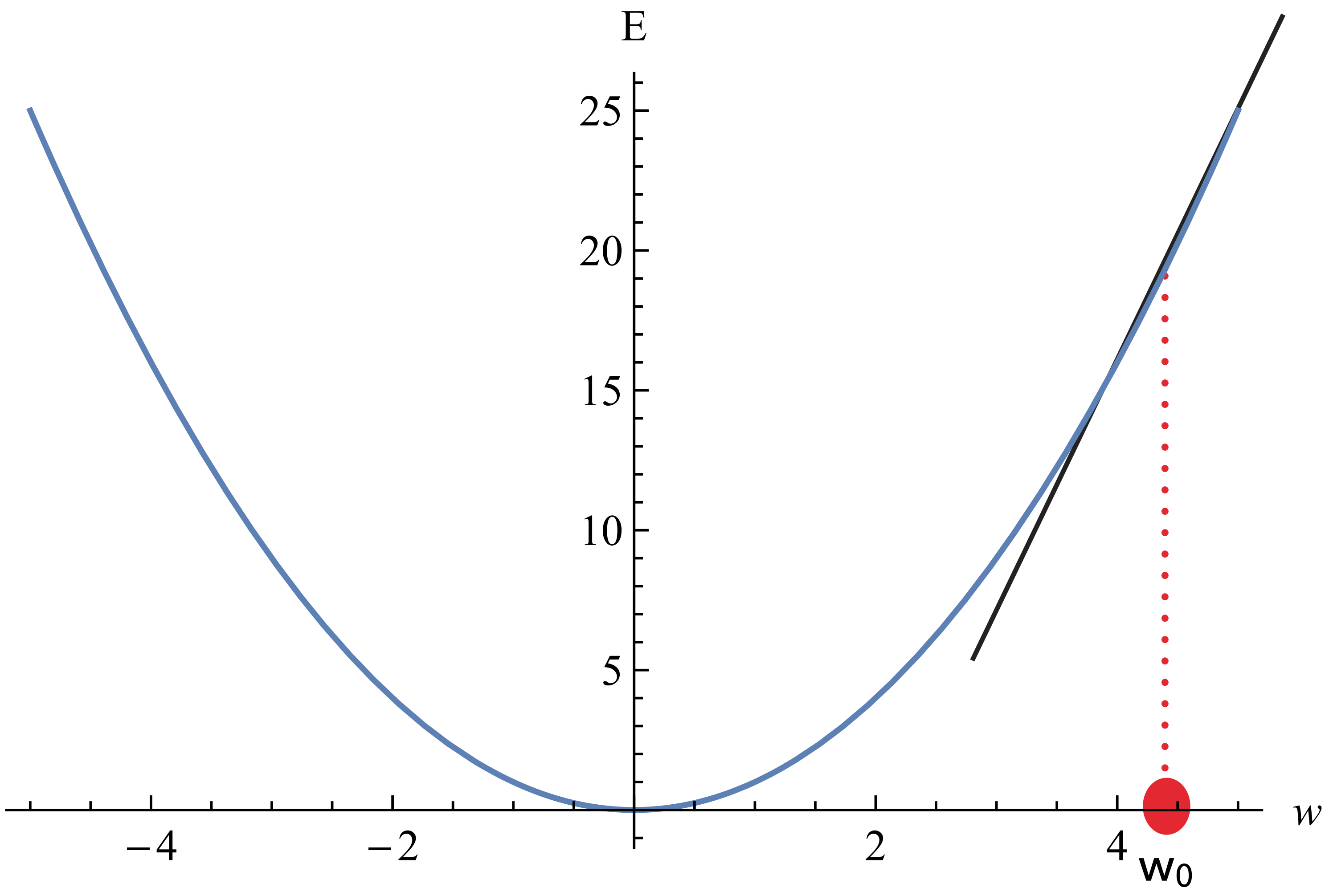
# GRADIENT DESCENT

- Estimate the gradient at that point (tangent to the curve)



# GRADIENT DESCENT

- Compute Δw such that ΔE is negative (to move toward the minimum)



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

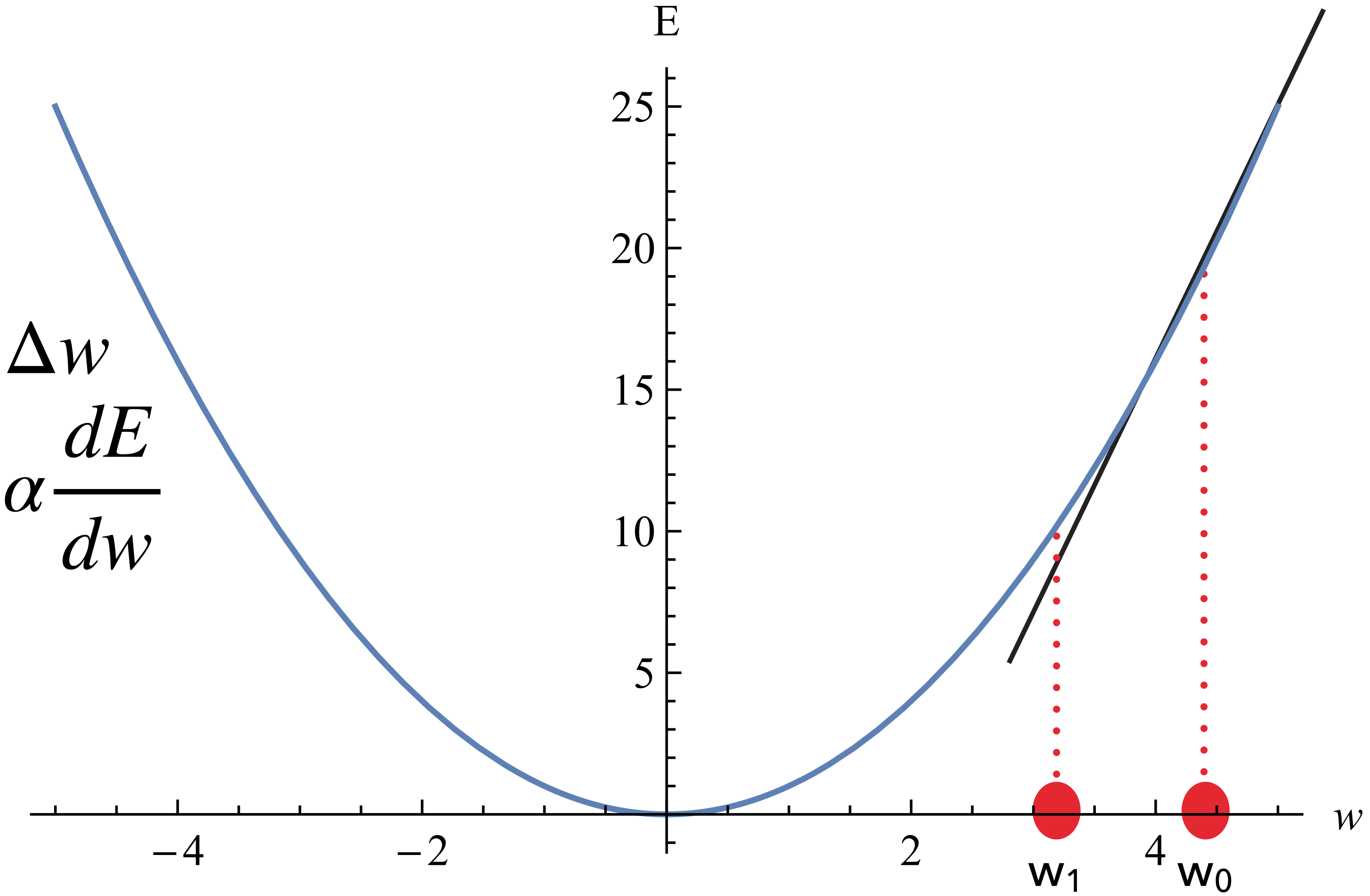
$\alpha$ is the learning rate: a small positive number

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

# GRADIENT DESCENT

- Compute a new weight value: $w_1 = w_0 + \Delta w$

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



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

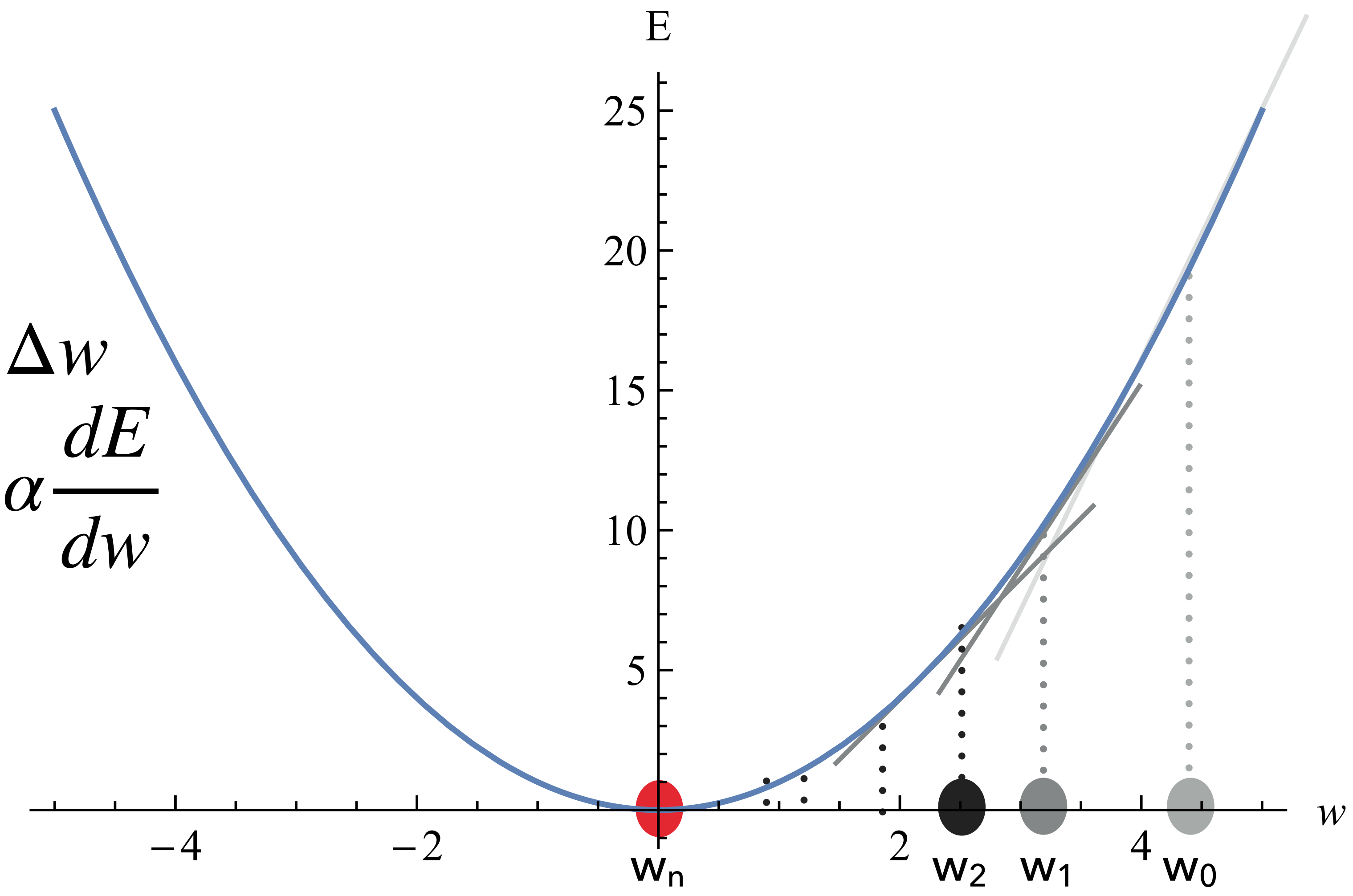
$\alpha$ is the learning rate: a small positive number

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

# GRADIENT DESCENT

- Repeat until some convergence criteria is satisfied.

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



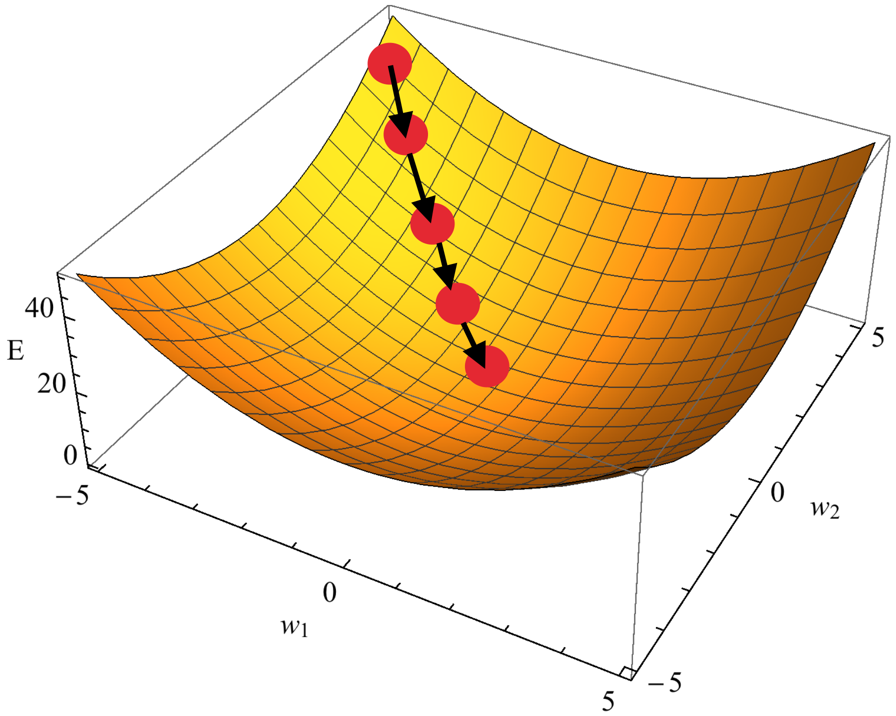
$$\Delta E = \Delta w \frac{dE}{dw}$$
$$= -\alpha \left(\frac{dE}{dw}\right)^2$$

$\alpha$ is the learning rate: a small positive number

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

# GRADIENT DESCENT

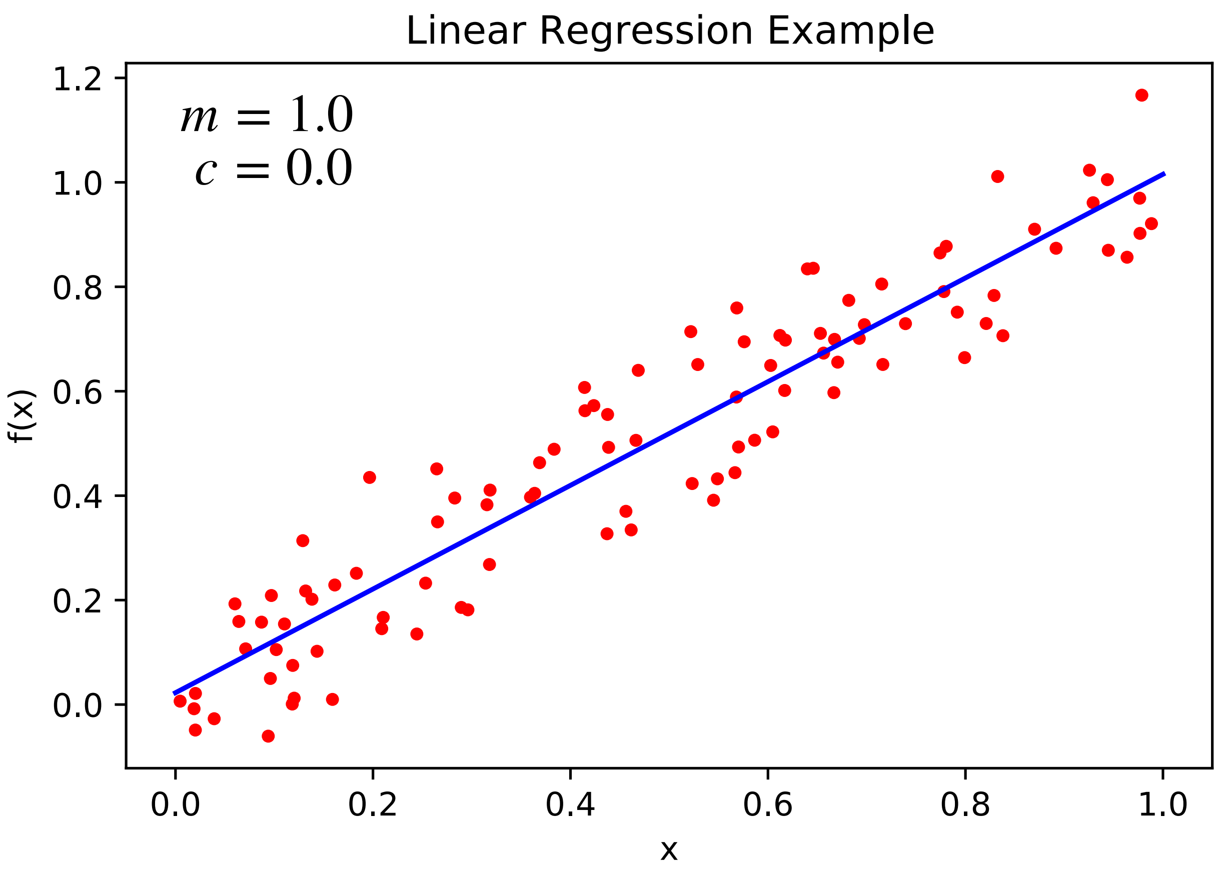
- We can extend this from a one parameter optimisation to a 2 parameter one, and follow the same principles, now in 2D.

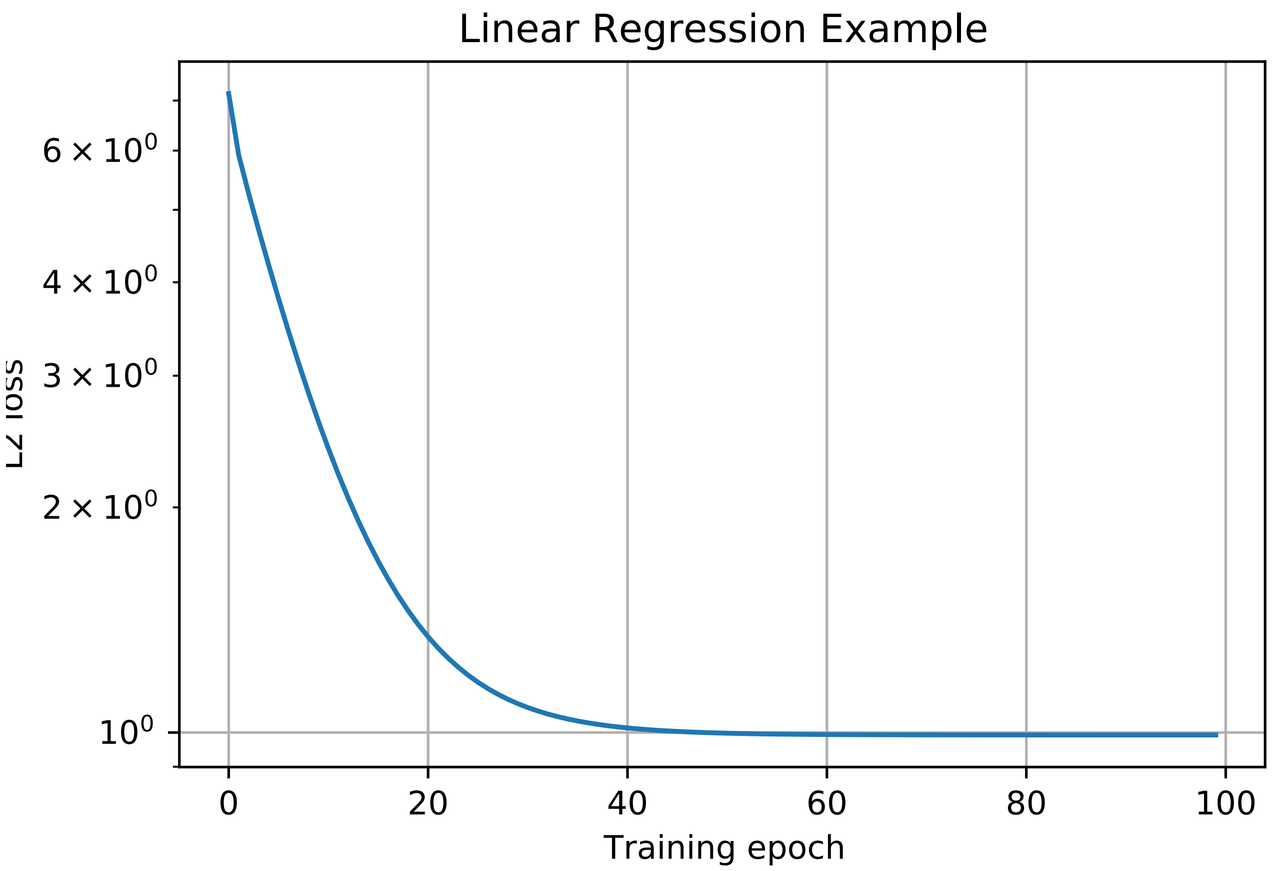


- The successive points $w_i$+1 can be visualised a bit like a ball rolling down a concave hill into the region of the minimum.
- In general update weights such that $\Delta E = \Delta w \nabla E = -\alpha \nabla^2 E$
- and $w_{i+1} = w_i - \alpha \nabla E.$

# GRADIENT DESCENT: EXAMPLE

- Returning to the $y = mx + c$ example, we can optimise this using the Gradient Descent algorithm.





Use N=100 and $\alpha = 0.005$ with the TensorFlow GradientDescent optimiser to obtain:

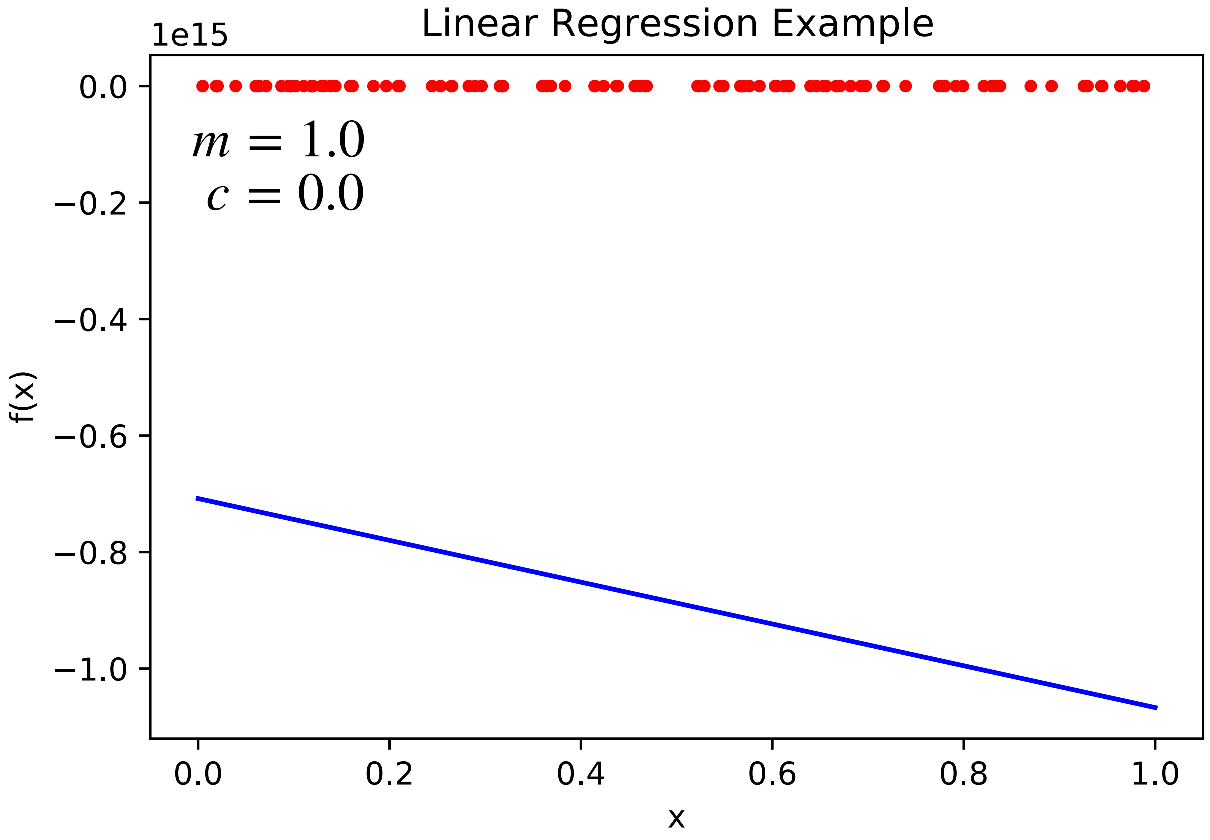
$$\widehat{m} = 0.9928...$$

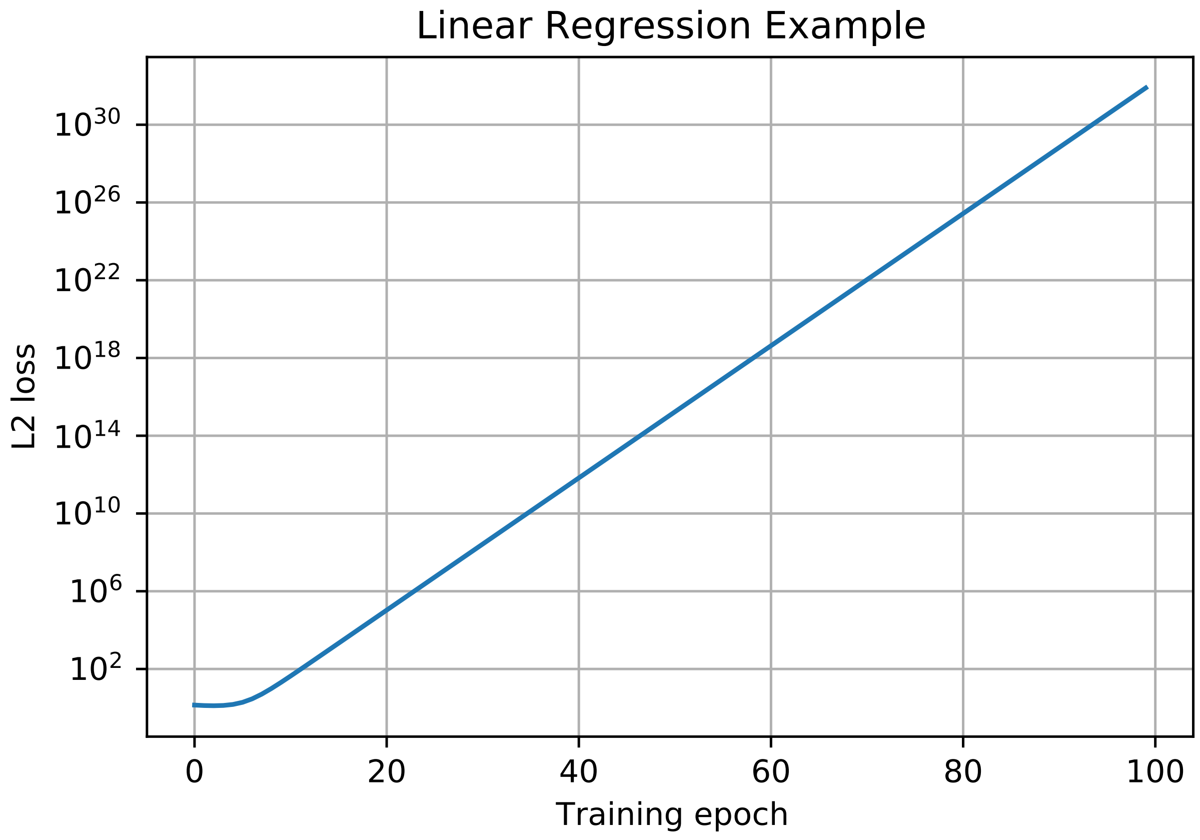
$$\hat{c} = 0.0226...$$

Implementation: TensorFlow 2.2 (using V1 compatibility)
Jupyter Notebook

# GRADIENT DESCENT: EXAMPLE

- Returning to the $y = mx + c$ example, we can optimise this using the Gradient Descent algorithm.





Use N=100 and $\alpha = 0.01$ with the TensorFlow GradientDescent optimiser to obtain:

$$\widehat{m} = -3.5 \times 10^{14}$$

$$\hat{c} = -7.08 \times 10^{14}$$

The minimiser fails; too large a learning rate is being used.

Implementation: TensorFlow 2.2 (using V1 compatibility)
Jupyter Notebook

# GRADIENT DESCENT: REFLECTION

- The examples shown illustrate problems with parabolic minima.
- With selection of an appropriate learning rate, $\alpha$, to fix the step size, we can guarantee convergence to a sensible minimum in some number of steps.
- Translating the distribution to a fixed scale, then we can predict how many steps it will take to converge to the minimum from some distance away from it for a given $\alpha$.
- If the problem hyperspace is not parabolic, this becomes more complicated, and there is no guarantee that we converge to the minimum.
- Modern machine learning algorithms use more refined variants on this method.

# OPTIMISATION

## ADAM OPTIMISER

# ADAM OPTIMISER

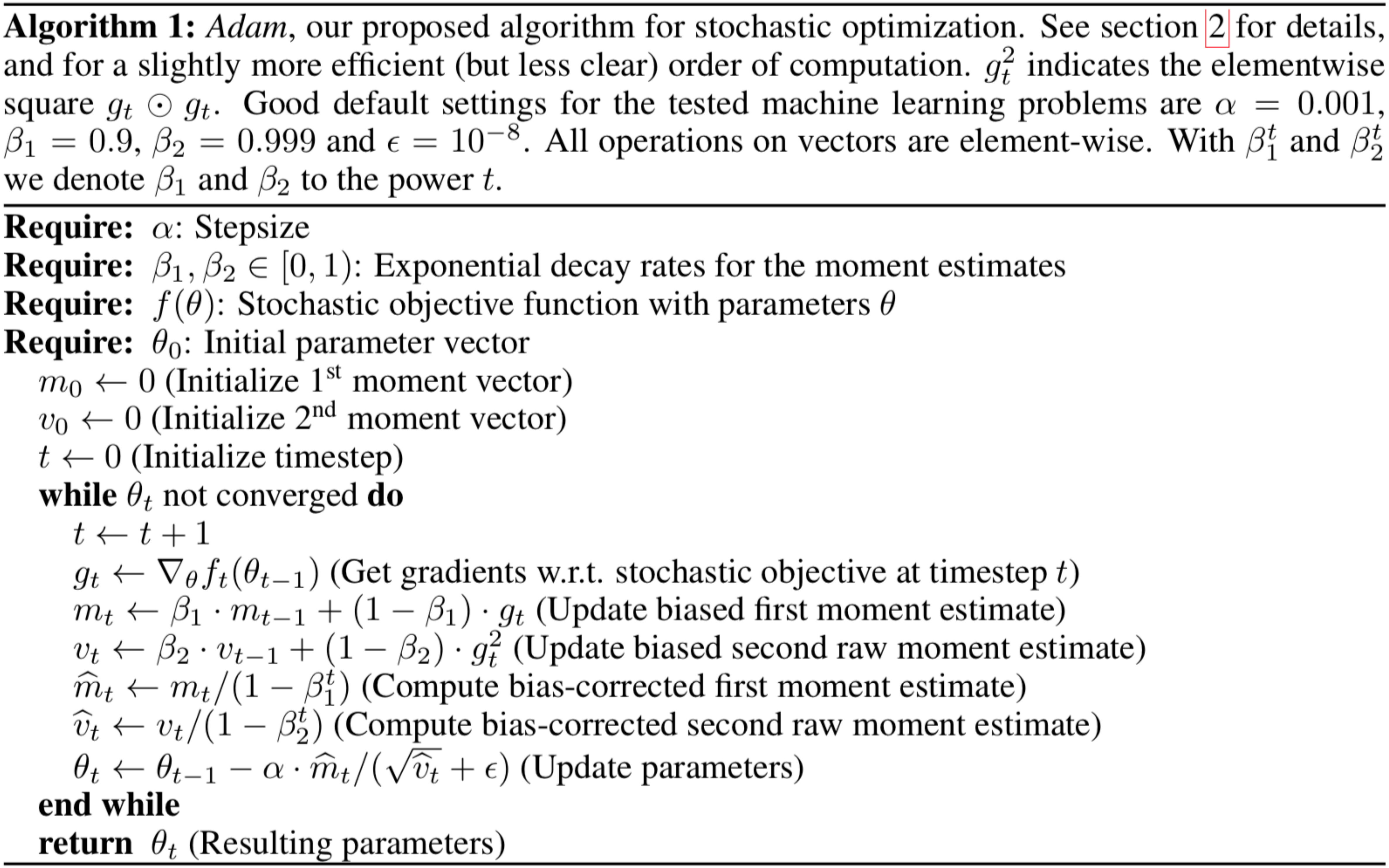
(ADAM: ADAptive Moment estimation based on gradient descent)

- This is a stochastic gradient descent algorithm.
- Consider a model $f(\theta)$ that is differentiable with respect to the HPs $\theta$ so that:
  - $t$ is the training epoch.
  - the gradient $g_t = \nabla f_t(\theta_{t-1})$ can be computed.
  - $m_t$ $(v_t)$ are biased values of the first (second) moment.
  - $\widehat{m}_t$ $(\hat{v}_t)$ are bias corrected estimator of the moments.
  - Some initial guess for the HP is taken: $\theta_0$, and the HPs for a given epoch are denoted by $\theta_t$.
  - $\alpha$ is the step size (i.e. learning rate).
  - $\beta_1$and $\beta_2$ are exponential decay rates of moments.

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

# ADAM OPTIMISER

(ADAM: 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 $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

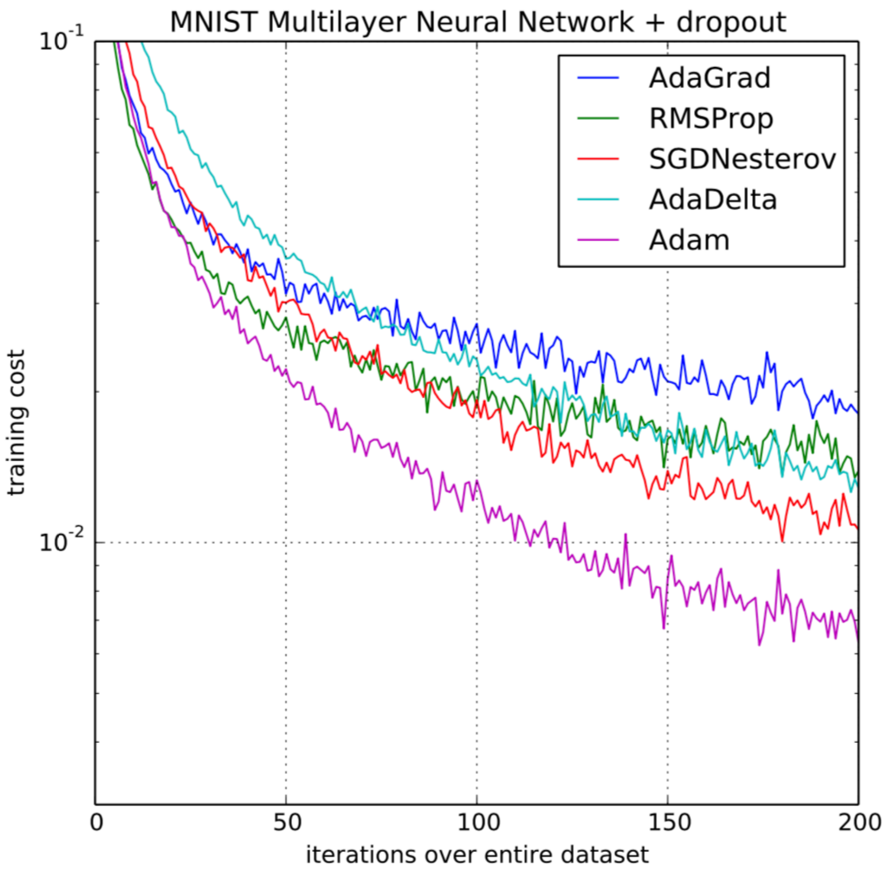
```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ 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 $\theta_t$ (Resulting parameters)
```

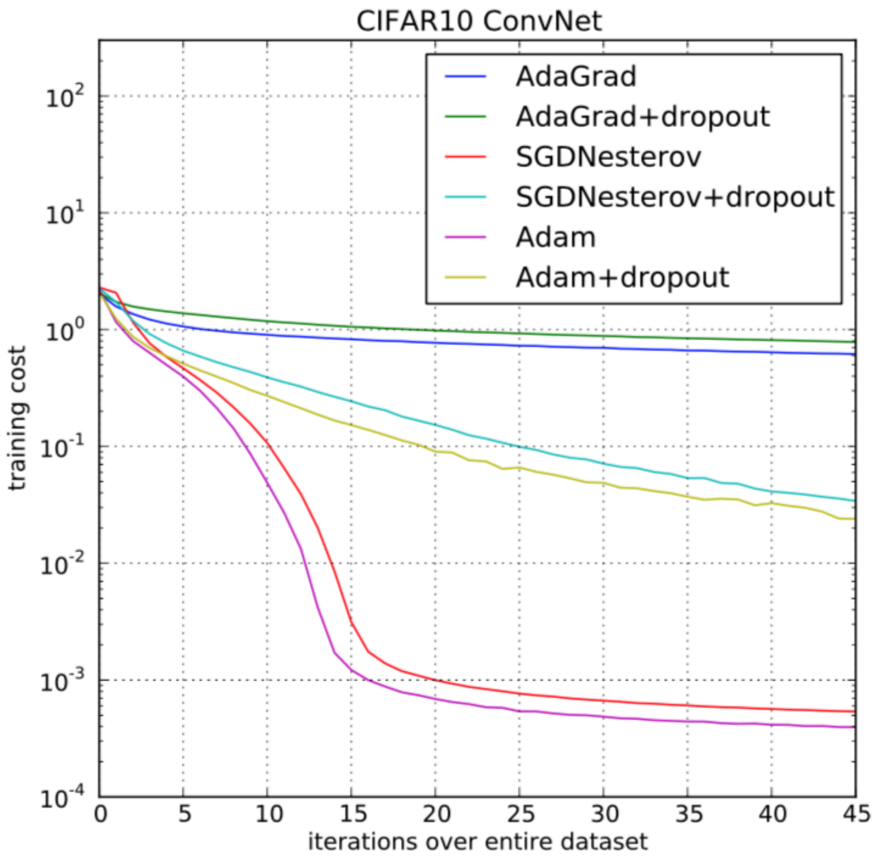
Kingma and Ba, Proc. 3rd Int Conf. Learning Representations, San Diego, 2015

# ADAM OPTIMISER

(ADAM: ADAptive Moment estimation based on gradient descent)

- 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 loss, and lower overall loss obtained vs other algorithms benchmarked at the time.

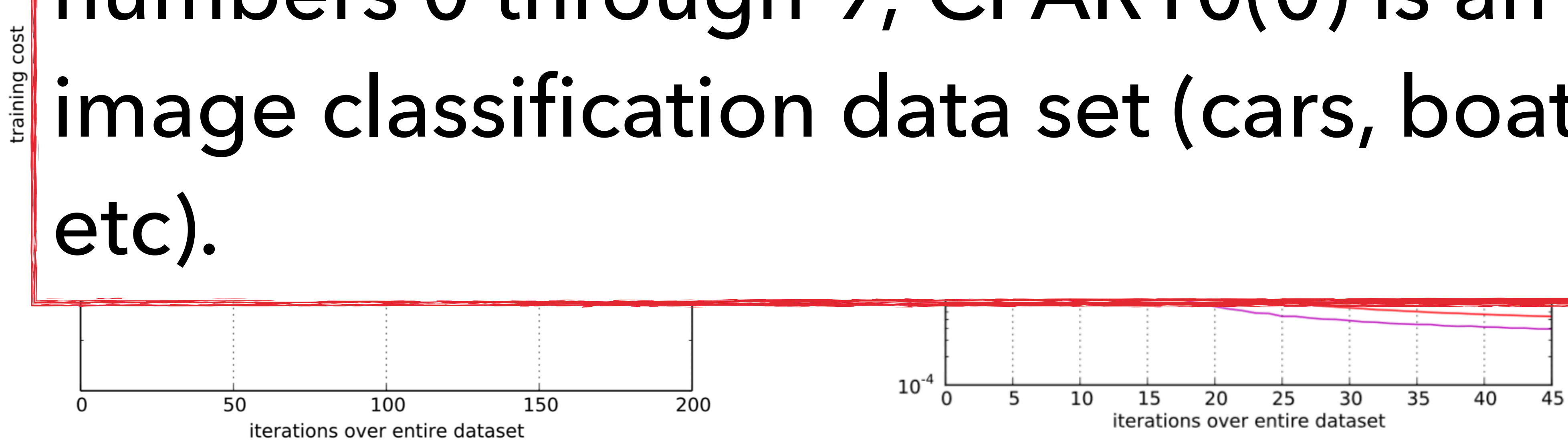
Kingma and Ba, Proc. 3rd Int Conf. Learning Representations, San Diego, 2015

# ADAM OPTIMISER

(ADAM: ADAptive Moment estimation based on gradient descent)

- Benchmarking performance using MNIST and CFAR10 data indicates that ith othe

MNIST and CFAR10 are standard benchmark data sets in CS (see appendix). MNIST is a set of handwritten numbers 0 through 9; CFAR10(0) is an image classification data set (cars, boats etc).



- Faster drop off in loss, and lower overall loss obtained vs other algorithms benchmarked at the time.

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

MULTIPLE MINIMA

MORE ON LOSS FUNCTIONS

# OPTIMISATION

## MISCELLANEOUS

# GRADIENT DESCENT:MULTIPLE MINIMA

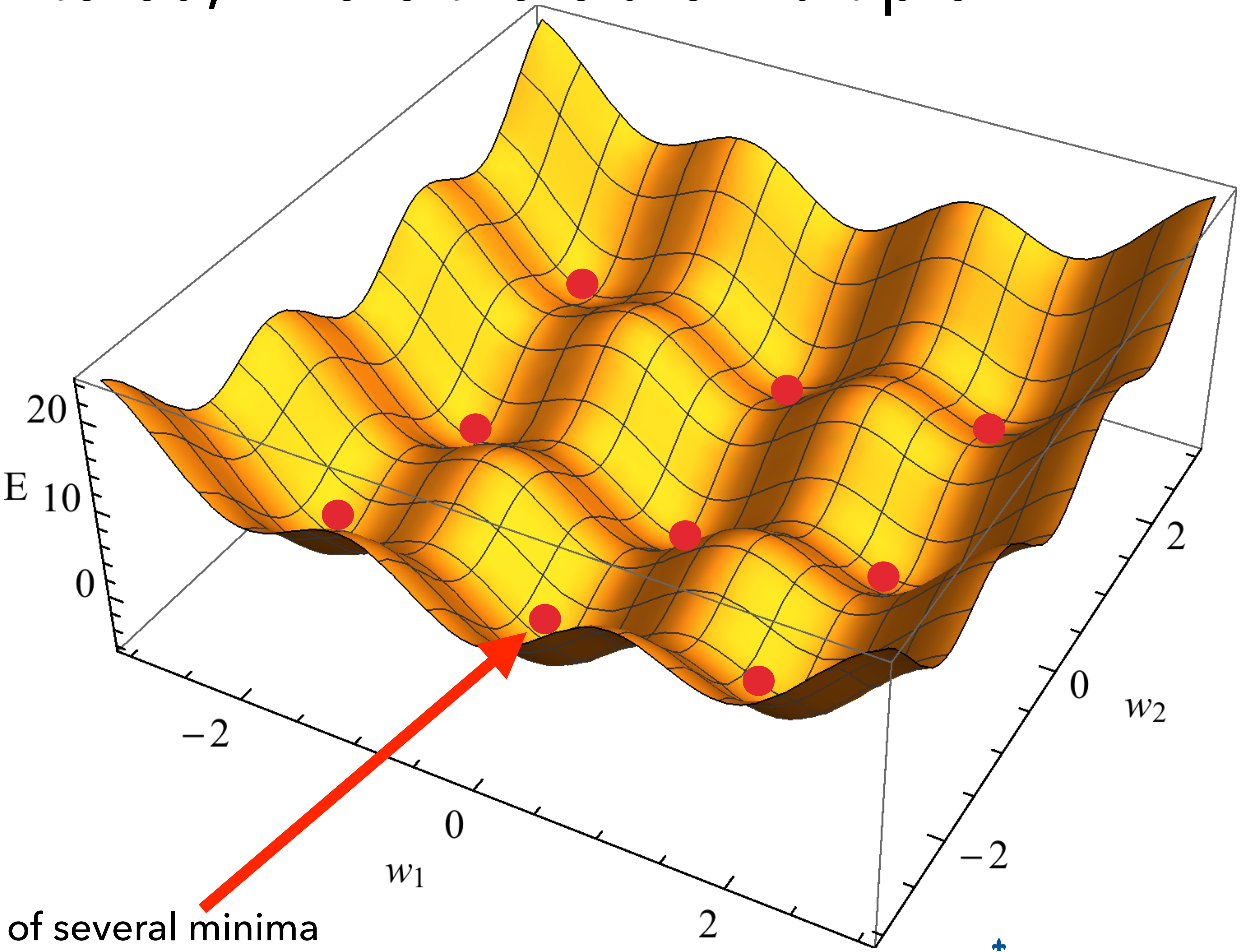
- Often more complication hyperspace optimisation problems are encountered, where there are multiple minima.

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?



One of several minima

# GRADIENT DESCENT:MULTIPLE MINIMA

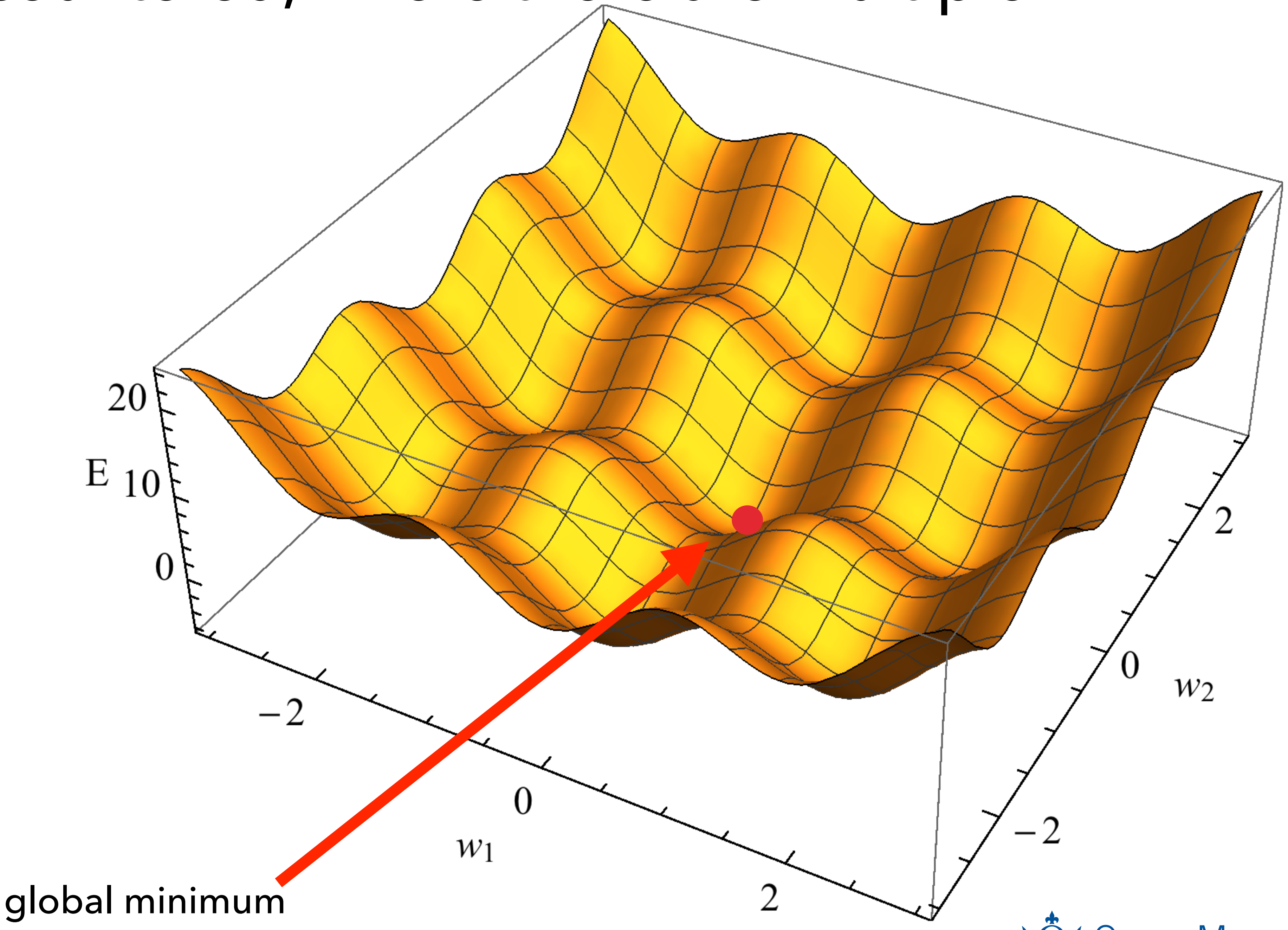
- Often more complication hyperspace optimisation problems are encountered, where there are multiple minima.

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?



# OPTIMISATION: LOSS FUNCTIONS

- There are many types of loss function other than the $L_2$ loss
  - L1 norm loss:

$$L_1 \sum_{i=1}^{N_{examples}} \left| t_i - \hat{y}(\hat{\theta}) \right|$$

- The mean square error (MSE) loss function:

$$L = \frac{1}{N} L_2 = \frac{1}{N} \sum_{i=1}^{N_{examples}} \left[ t_i - \hat{y}(\hat{\theta}) \right]^2$$

- Cross entropy

$$L = - \sum_{i=1}^{N_{examples}} \widehat{y}(x_i) \ln t_i$$

The cross entropy can be thought of as the negative log likelihood function for the data $t_i$ under the model $\widehat{y}$

SUPERVISED LEARNING

OVERTRAINING

WEIGHT REGULARISATION

CROSS VALIDATION

DROPOUT

# TRAINING

[1] G. Hinton et al. arXiv:1207.0580

# TRAINING

## SUPERVISED LEARNING

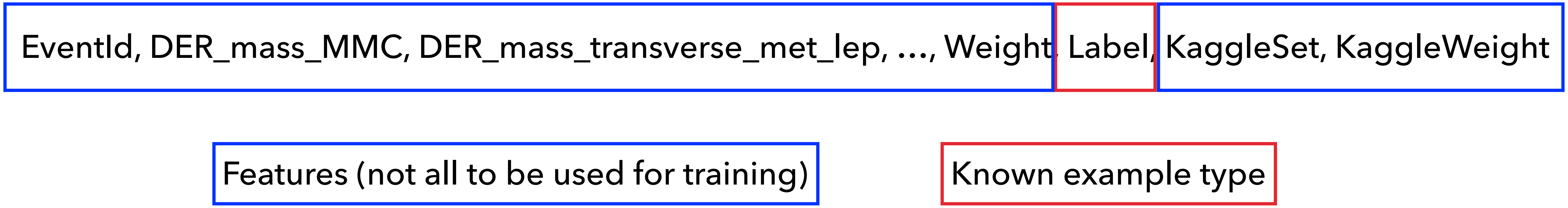
# SUPERVISED LEARNING

- For supervised learning the loss function depends on both model parameters and a known target value $t_i$ for a given example.
  - e.g. the L$_2$ loss: $L_2 = \sum_{i=1}^{N_{examples}} \left[t_i - \hat{y}(\hat{\theta})\right]^2.$
- This requires (at a minimum) a sample of data with the input feature space of interest, and for each example in the data, the known target output value.
- The loss function is optimised using the data, to obtain a model.
- Unlike fitting however we don't care about the uncertainty on the model parameter estimates, only on the nominal values obtained, $\widehat{\theta}$.
- Due to the complexity of models, it is generally not possible to understand if the optimisation converged to a sensible solution by inspecting marginalised projections of the model on the data.

[1] G. Hinton et al. arXiv:1207.0580

# SUPERVISED LEARNING

- e.g. the Kaggle Higgs data sample:



EventId, DER_mass_MMC, DER_mass_transverse_met_lep, …, Weight, Label, KaggleSet, KaggleWeight

Features (not all to be used for training)

Known example type

- The KaggleSet column in the CVS file for this data allows the user to understand if this is to be used for training or testing, or some other means (e.g. Kaggle Score Board evaluation).
- The Label column, is the known label; this defines the $t_i$ used in the loss function.
- I use this challenge data as an assignment for undergraduate lectures (pdf, zip). N.B. the zip file is 189Mb.

[1] G. Hinton et al. arXiv:1207.0580

WEIGHT REGULARISATION

CROSS VALIDATION

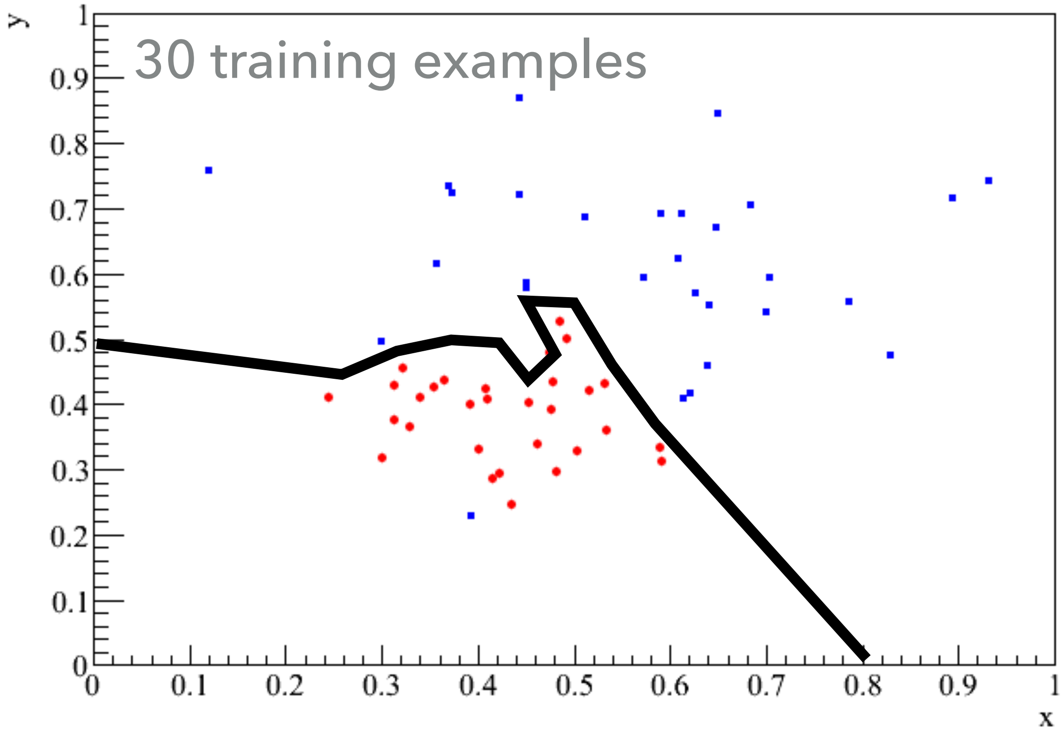
# TRAINING

# OVERTRAINING

# OVERTRAINING

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

- Simple illustration of the problem:



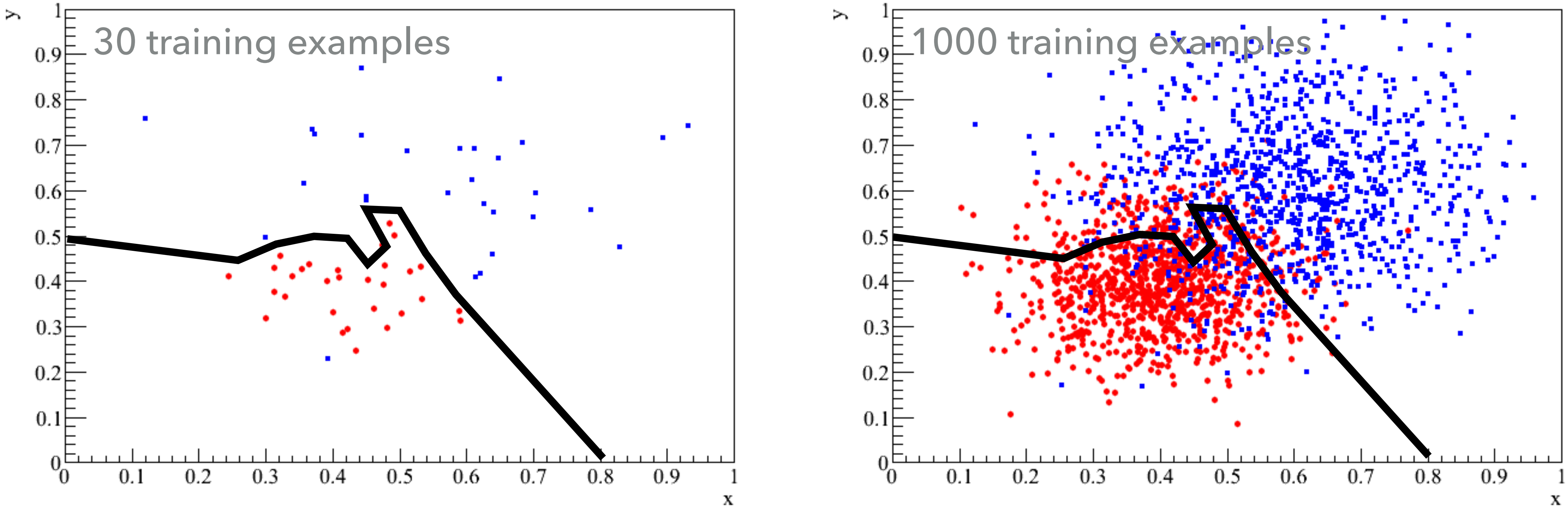
The decision boundary selected here does a good job of separating the red and blue dots.

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

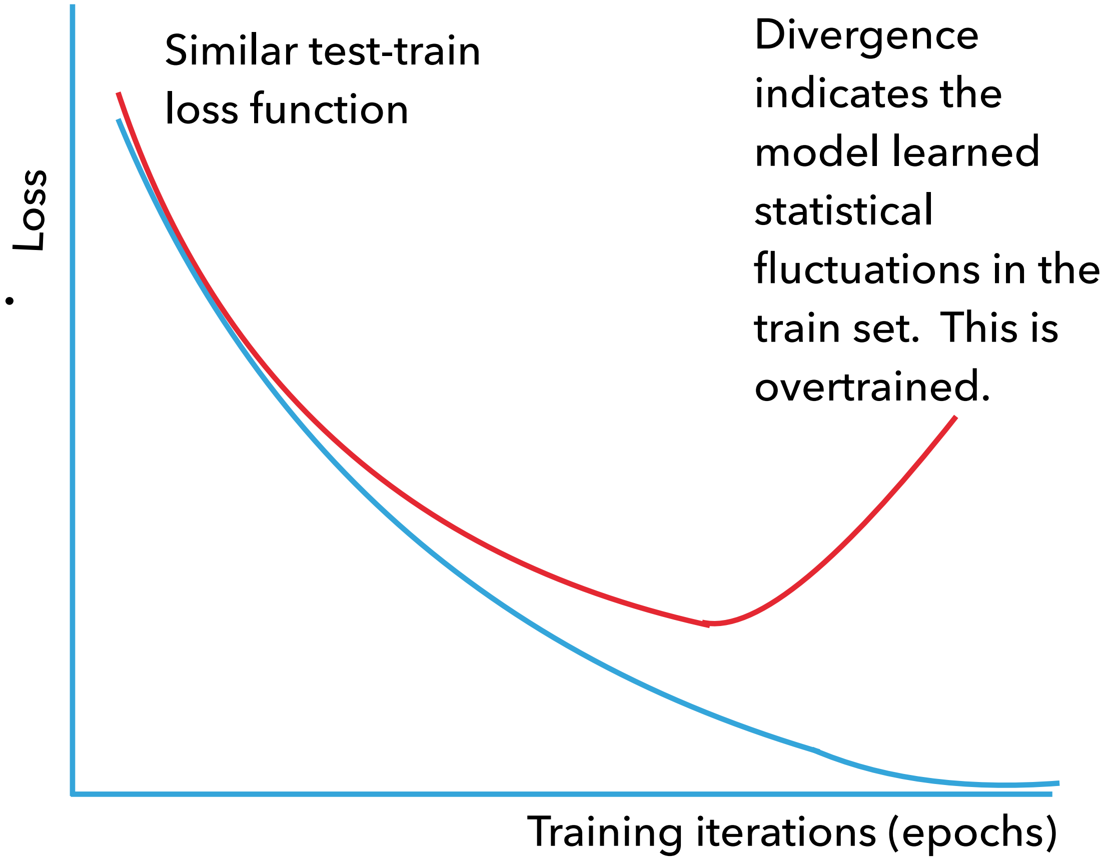
- A model is over trained if the HPs that have been determined are tuned to the statistical fluctuations in the data set.
- Simple illustration of the problem:



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

- 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 test sample.
  - Use the loss evaluated for the training and test samples to check to see if the HPs are over trained.
    - If both samples have similar loss then the model response function is similar on two statistically independent samples.
  - Note: 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.



Similar test-train loss function

Divergence indicates the model learned statistical fluctuations in the train set. This is overtrained.

# OVERTRAINING

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

# OVERTRAINING: 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

# OVERTRAINING: WEIGHT REGULARISATION

- For example we can consider extending an MSE loss function to allow for weight regularisation. The MSE loss 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

# OVERTRAINING: CROSS VALIDATION

- A well trained model will provide robust predictions, irrespective of the examples presented to it.
  - The variance of model predictions will be small.
  - The model predictions may be systematically biased independently of this.
- One can divide the training set up into k-folds, and then perform k trainings; each one leaving a single fold out.
- The amount of data used in a fold will depend (generally the more data the better.
- The ensemble of predictions indicates the variance on the model output.

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).

See the appendix for more on cross validation.

# OVERTRAINING: CROSS VALIDATION

- The use of cross validation to determine the spread of model predictions, and any systematic bias on prediction can be useful.
- e.g. in a physics context: Consider a new particle search at the Large Hadron Collider.
  - Using an overtrained model to suppress background and enhance signal will result in a lack of experimenter understanding as to how the expected and observed limits on the new particle relate to each other.
  - This could result in a false discovery of new physics.
  - It could result in missing out on a discovery of new physics.
  - Some people say it is not wrong to use an over trained model - I argue strongly that it is not scientifically correct to use an over trained model, unless the variance on that model, and hence the implications are well understood.

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).

# TRAINING

## DROPOUT

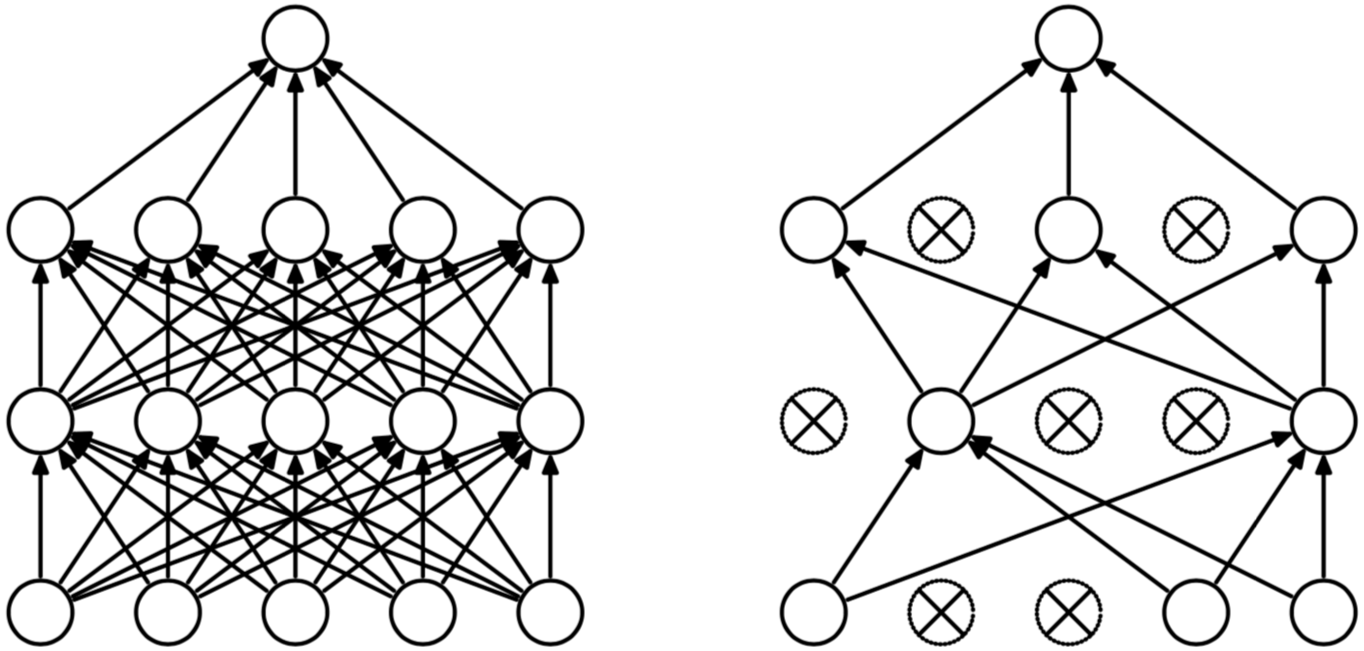
# DROPOUT

- A problem with deep networks is the number of hyper-parameters that need to be determined.
- This leads to a requirement for very large data sets to avoid overfitting (or fine-tuning).
- Hyperparameters can also learn to "co-adapt" in the training process.
  - co-adapt means that as one parameter is changed, another in the network can be modified in a correlated way to offset that change.
  - This kind of behaviour intentionally exists in some algorithms (Sequential Minimal Optimisation for Support Vector machines, where pairs of HPs are changed to conserve an overall zero sum); but is generally unwelcome behaviour.

G. Hinton et al. arXiv:1207.0580, Srivastava et al., J. Machine Learning Research 15 (2014) 1929-1958

# OVER FITTING: DROPOUT

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



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

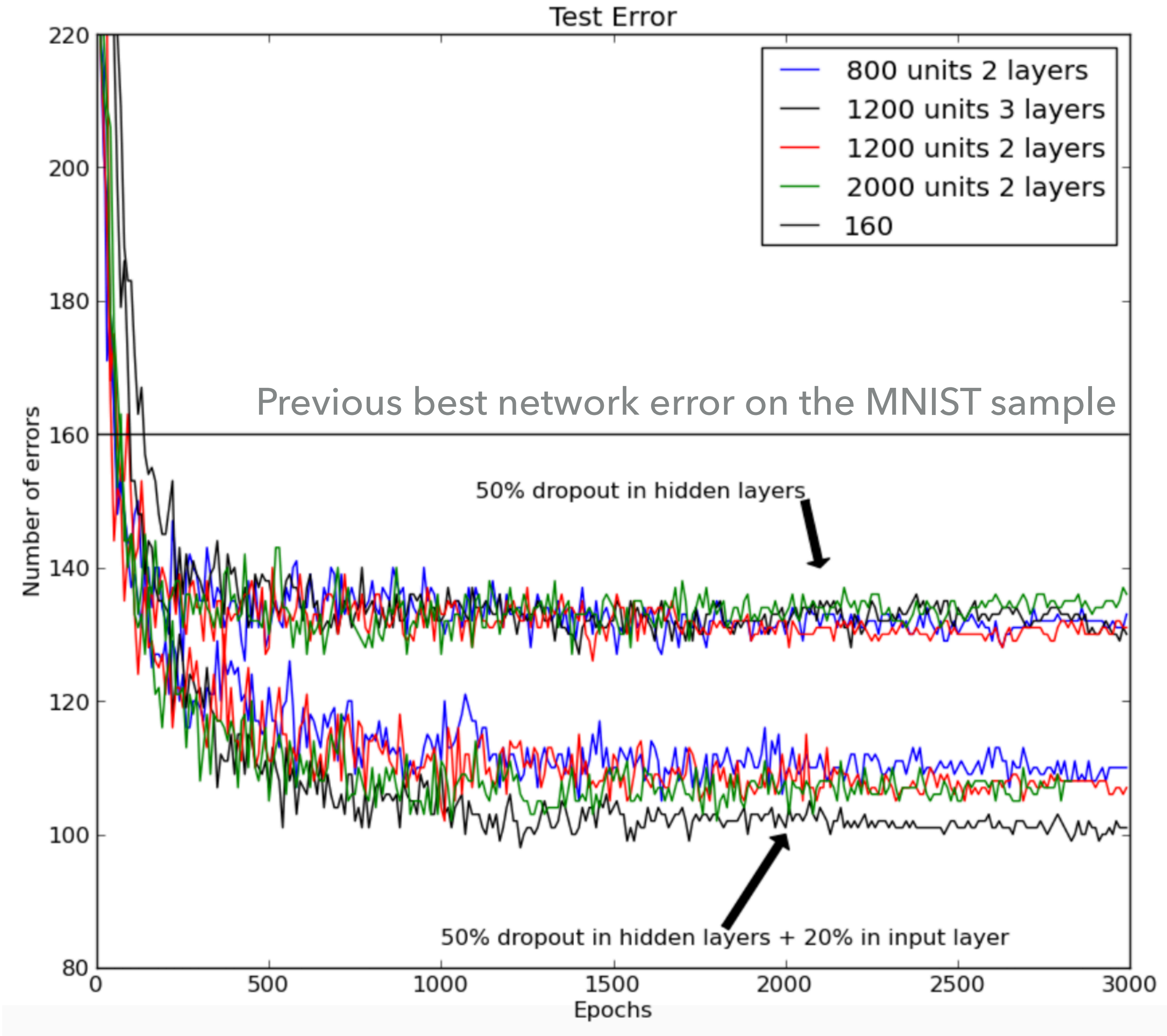
Dropout is used only during training.

The full model is used for making predictions.

- That way the whole model will be effectively trained on a sub-sample of the data with the aim of limiting the ability to learn statistical fluctuations in the data, and mitigating the co-adaption issue.
- This does not remove the possibility that a model is overtrained, as with the previous discussion HP generalisation is promoted by using this method.

G. Hinton et al. arXiv:1207.0580, Srivastava et al., J. Machine Learning Research 15 (2014) 1929-1958

# DROPOUT

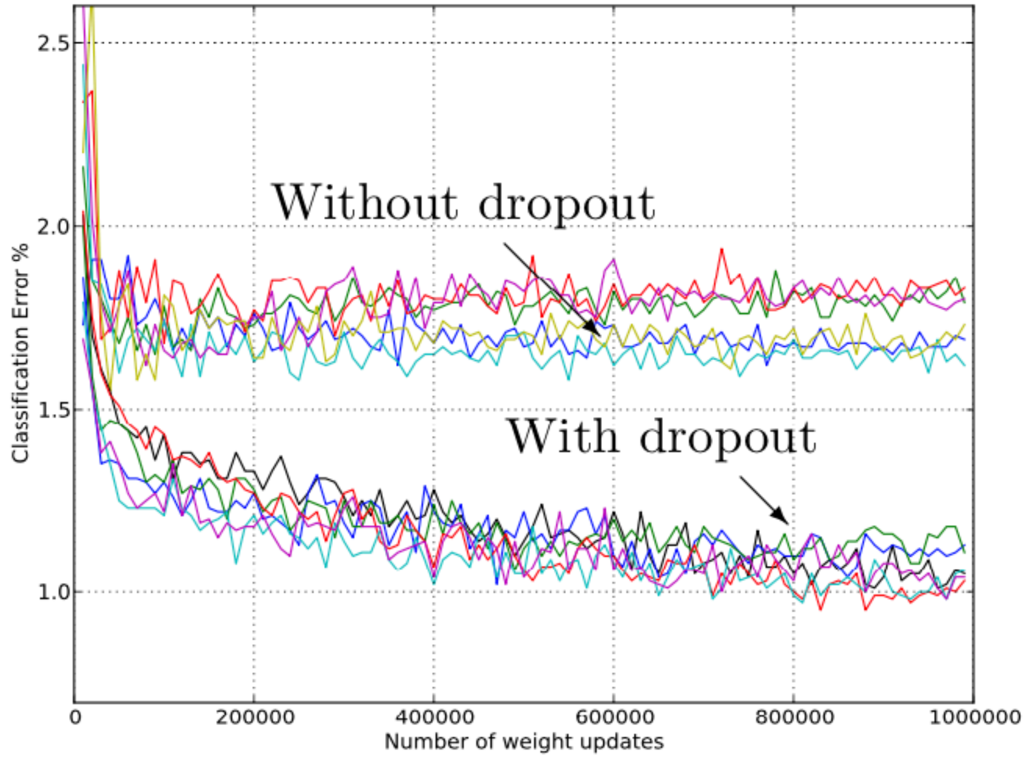


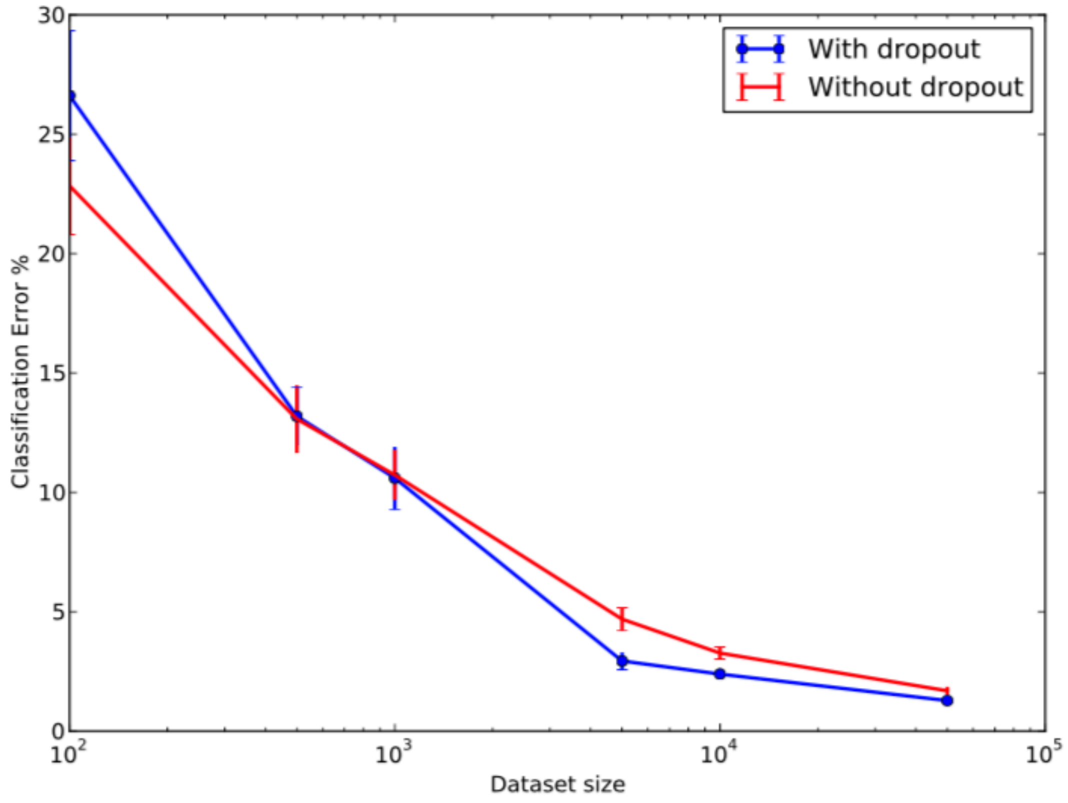
- Example from Hinton et al. for an MNIST sample.
- Using 50% drop out on the hidden layers gives an improvement over the previous best network architecture.
- Adding 20% drop out in the input layer provides further improvement in error.

G. Hinton et al. arXiv:1207.0580, 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 for this technique,

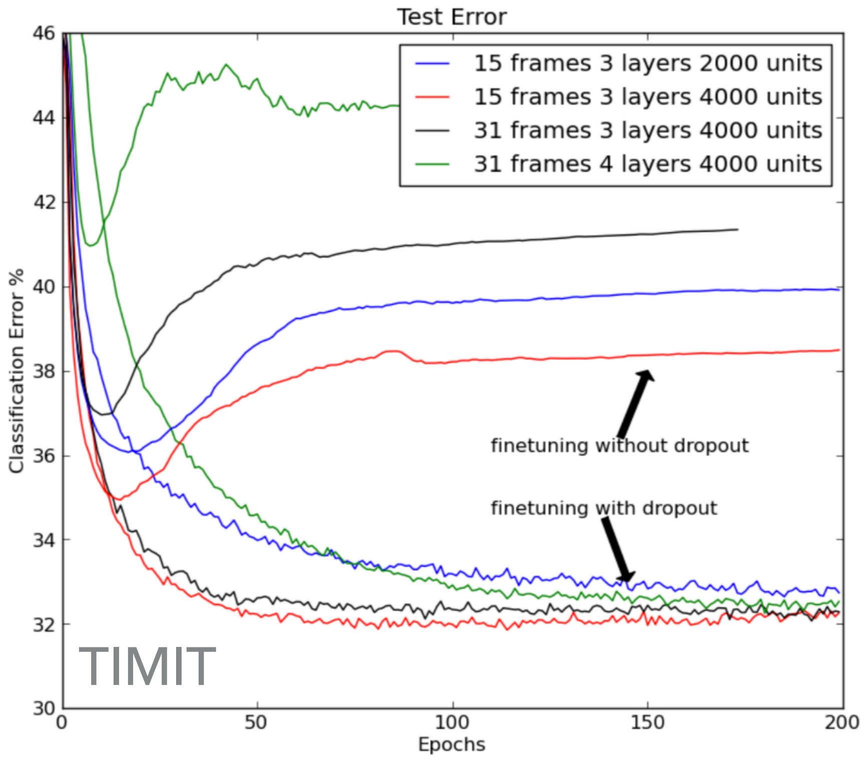


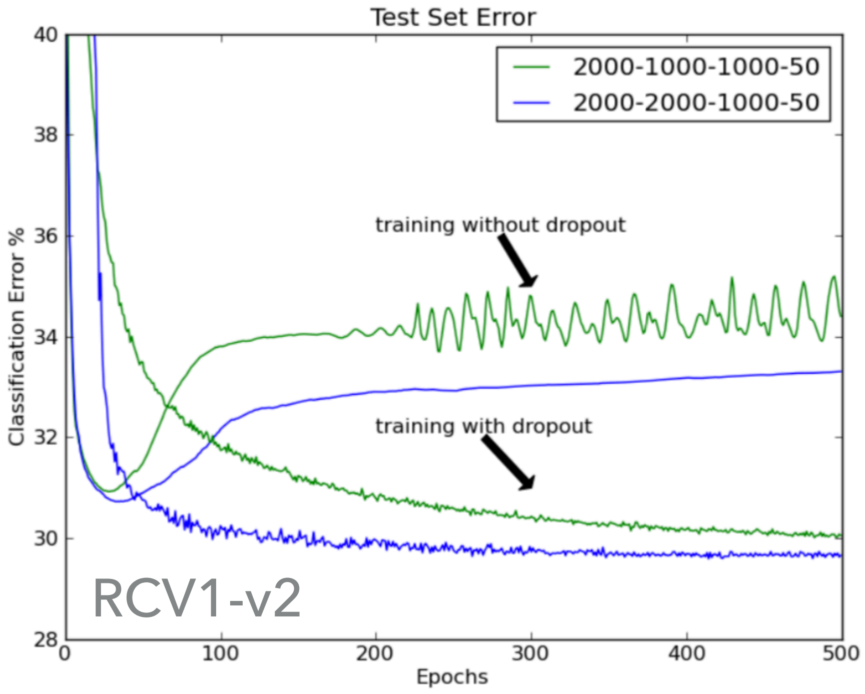


- Dropout can be detrimental for small training samples, however in general the results show that dropout is beneficial.
- For deep networks or typical training samples O(500) examples or more this technique is expected to be beneficial.
- For algorithms with very large training data sets, the benefits are less clear.

G. Hinton et al. arXiv:1207.0580, Srivastava et al., J. Machine Learning Research 15 (2014) 1929-1958

# DROPOUT





- Example from Hinton et al. for a voice recognition sample of data (TIMIT, speech recognition data).
- Illustrates the classification error as a function of epoch with and without dropout.
- Similar results were obtained by Hinton and his team with the Reuters newsfeed data set (RCV1-v2).

G. Hinton et al. arXiv:1207.0580, Srivastava et al., J. Machine Learning Research 15 (2014) 1929-1958

DECISION TREES

BOOSTING

ADABOOST.M1

RANDOM FORESTS

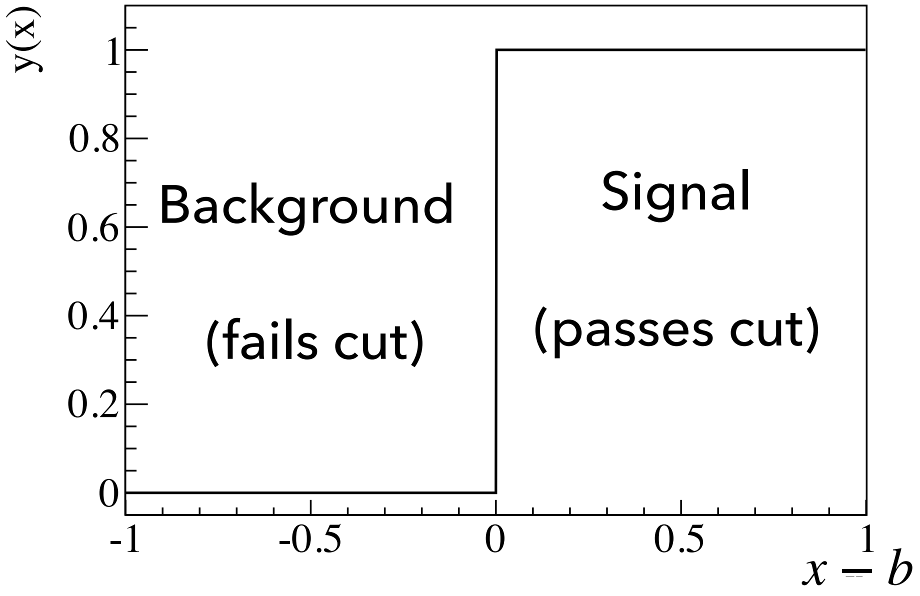
# DECISION TREES

# DECISION TREES

- The cut based and linear discriminant analysis methods are useful but limited [See Data Wrangling]
- The underlying concepts of applying Heaviside function constraints on data selection and on the use of a decision boundary definition (a plane in hyperspace) of the form of the dot product $\alpha^T x + \beta$ can be applied in more complicated algorithms.
- Here we consider extension to the concept of rectangular cuts to decision trees as a machine learning algorithm.
  - We will have to introduce the concepts of classification and regression; and methods to mitigate mis-classification of data.
  - The issue of overtraining is something we will come back to when discussing optimisation.

# DECISION TREES

- Consider a data sample with an N dimensional input feature space X.
- X can be populated by examples from two or more different species of event (also called classes, categories or types).
- Consider the two types and call them signal and background, respectively*.
- We can use a Heaviside function to divide the data into two parts:
  - We can use this to distinguish between regions populated signal and background:



For an arbitrary cut position in x we can modify the Heaviside function

$$H(x) = \frac{1}{2}(1 + sign(x - b))$$

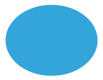
where b is the offset (bias) from zero.

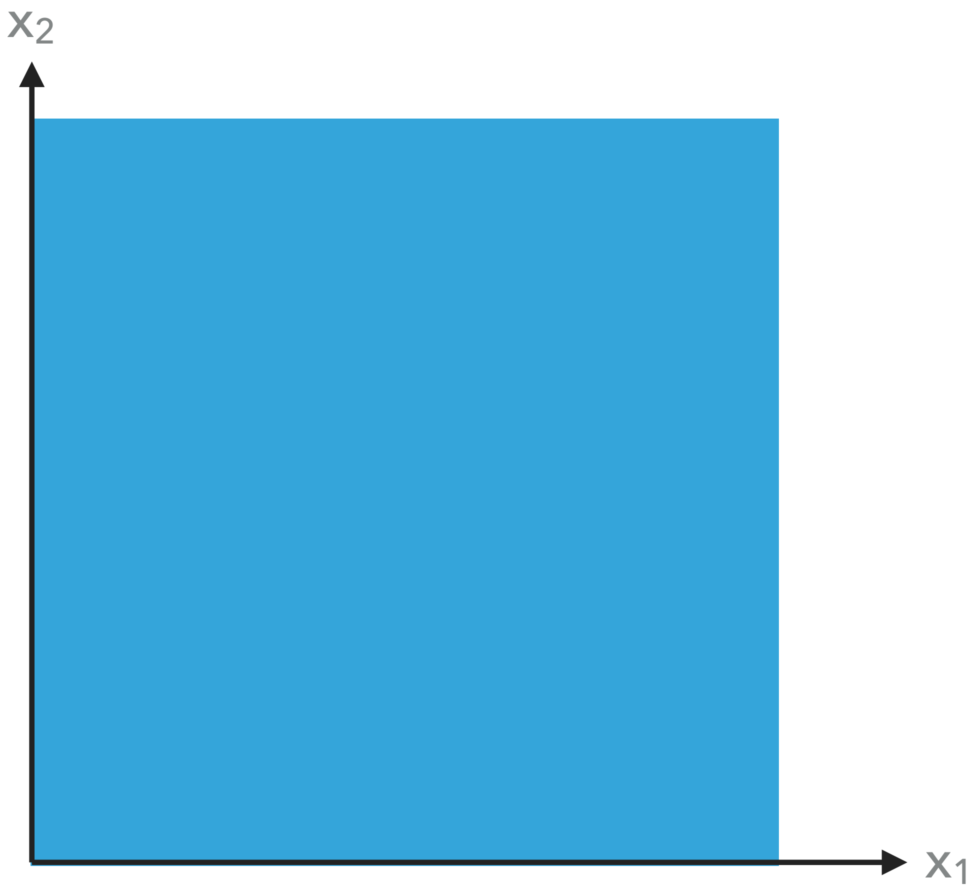
*Can generalise the problem to an arbitrary number of types.

# DECISION TREES

- Decision trees divide the data feature space into a set of hypercubes that are classified as signal (+1) or background (-1) like.
    - Each region can be fitted with a constant to represent the data in that region.
    - We can recursively continue to sub-divide the data until some minimum number of examples are left in each sub-division.

Can describe the data as the root node.



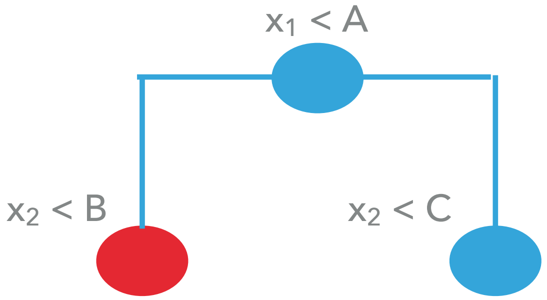


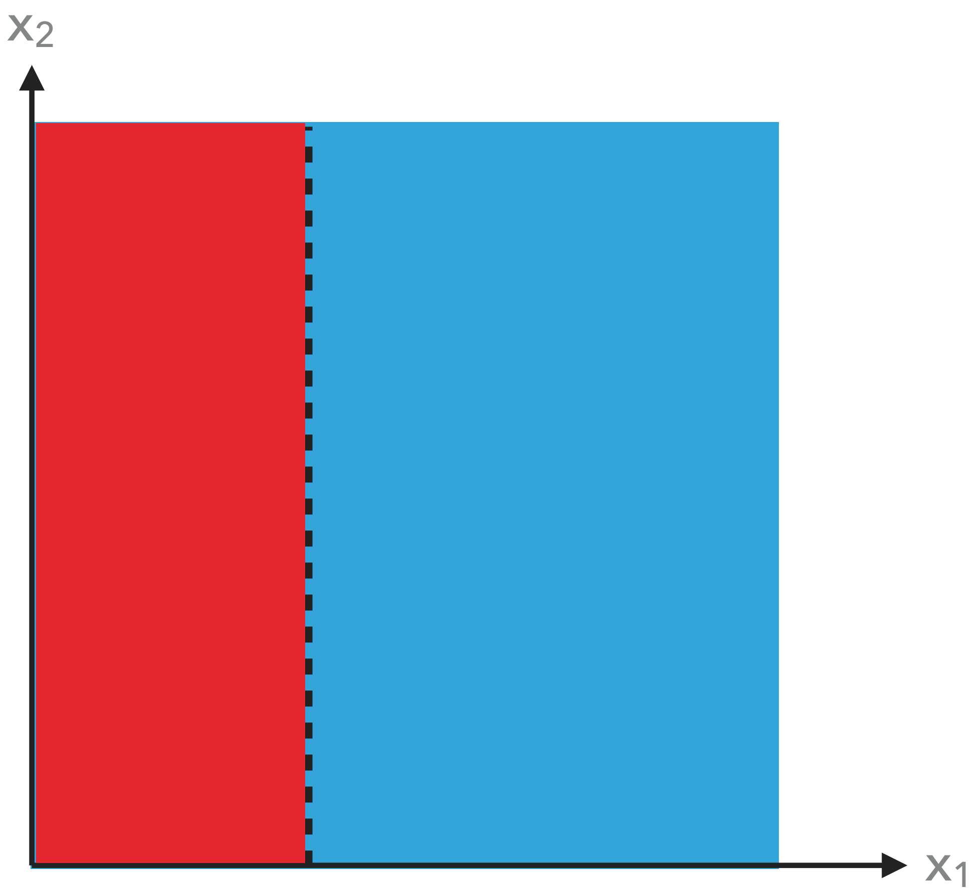
Example feature space described by $X=\{x_1, x_2\}$

# DECISION TREES

- Decision trees divide the data feature space into a set of hypercubes that are classified as signal (+1) or background (-1) like.
  - Each region can be fitted with a constant to represent the data in that region.
  - We can recursively continue to sub-divide the data until some minimum number of examples are left in each sub-division.

The data get divided into two partitions.



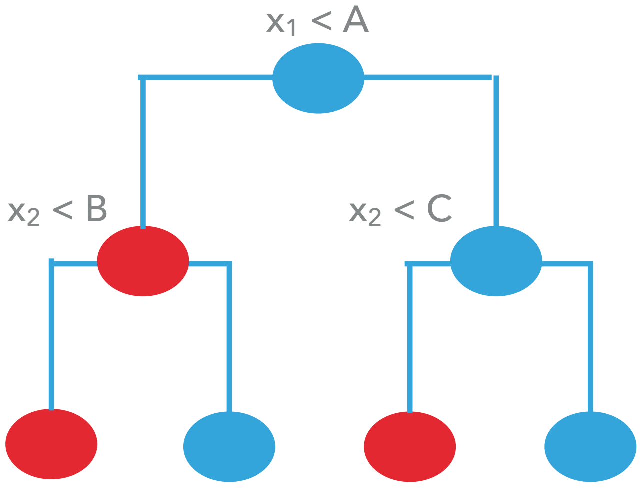


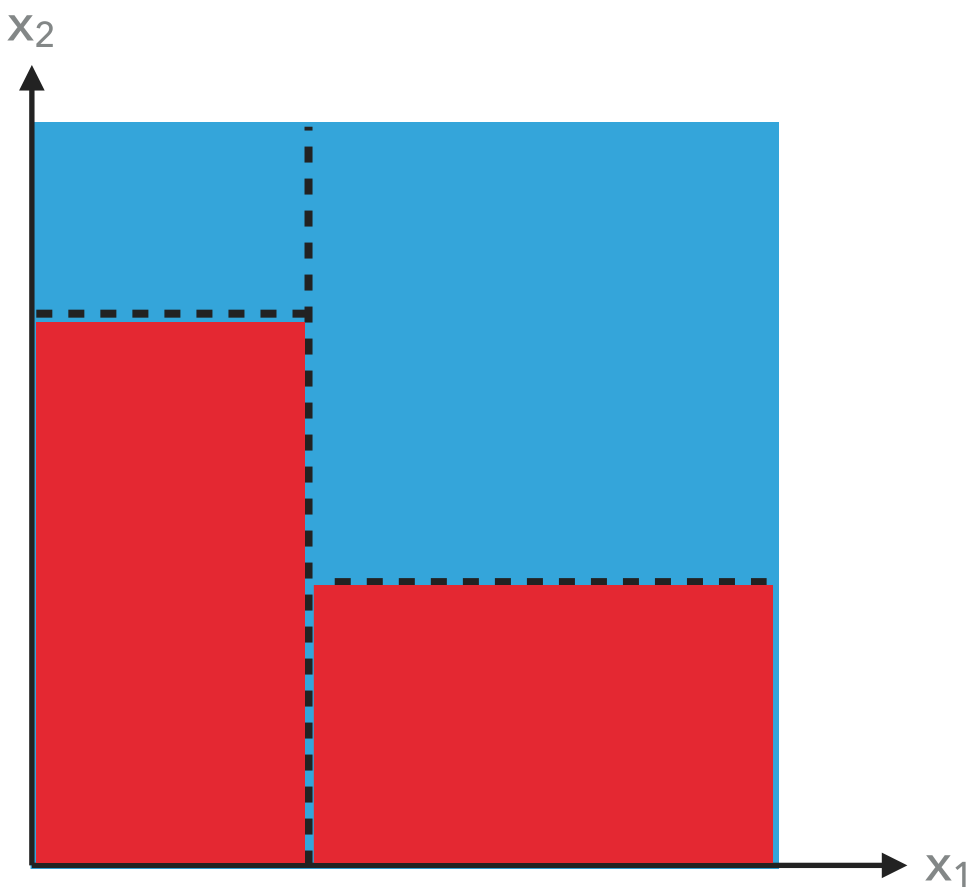
Cut on the feature space to separate the data into two different regions.

# DECISION TREES

- Decision trees divide the data feature space into a set of hypercubes that are classified as signal (+1) or background (-1) like.
  - Each region can be fitted with a constant to represent the data in that region.
  - We can recursively continue to sub-divide the data until some minimum number of examples are left in each sub-division.

Divide the data again



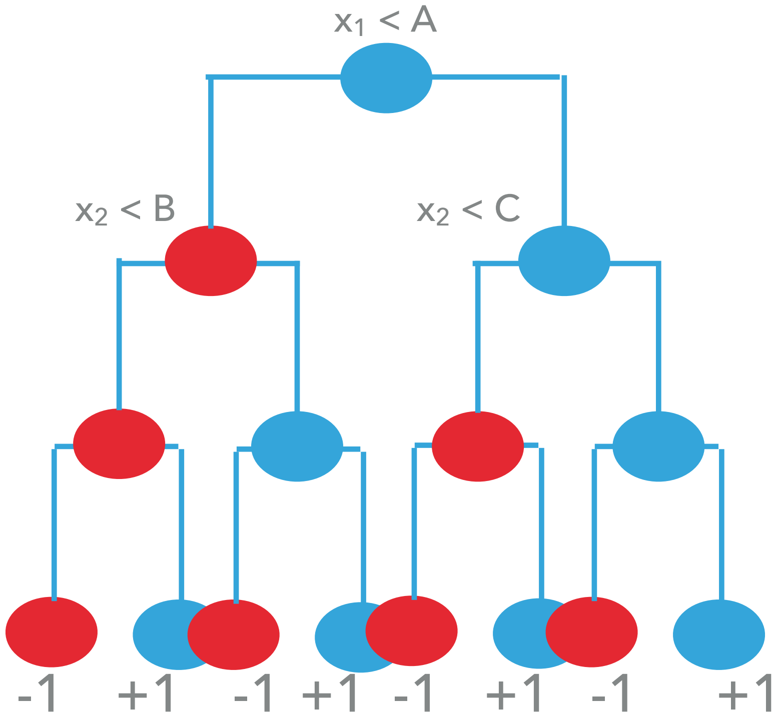


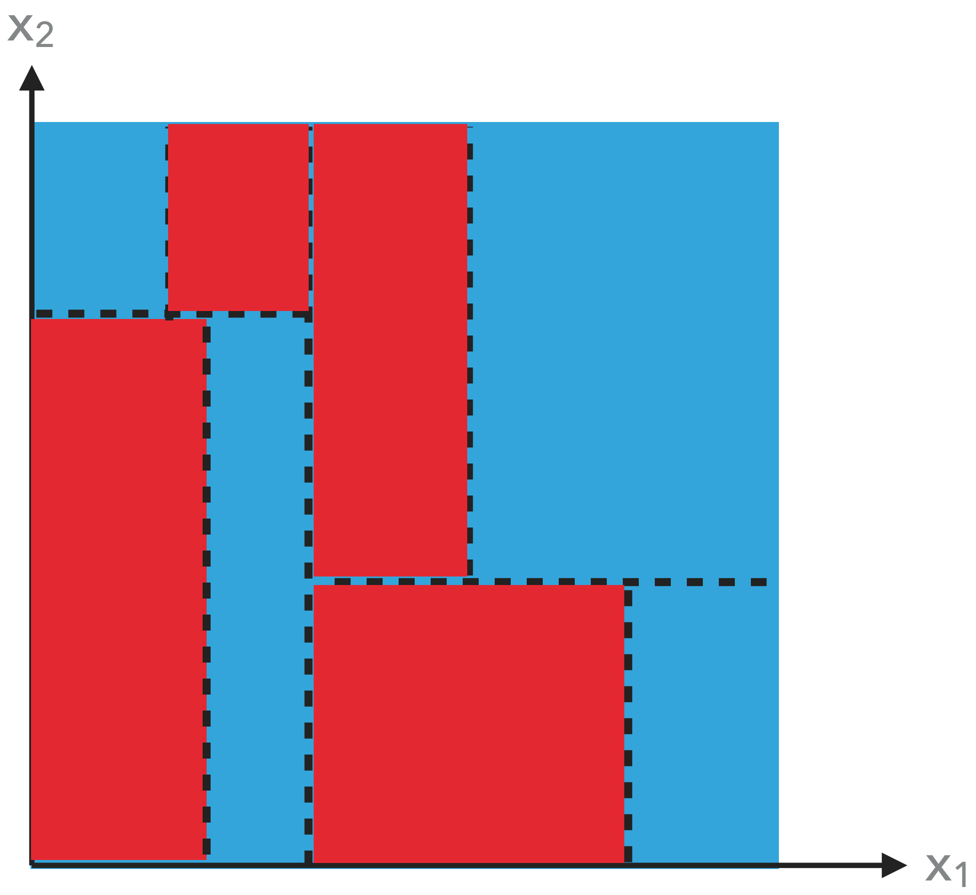
The feature space gets further sub-divided.

# DECISION TREES

- Decision trees divide the data feature space into a set of hypercubes that are classified as signal (+1) or background (-1) like.
    - Each region can be fitted with a constant to represent the data in that region.
    - We can recursively continue to sub-divide the data until some minimum number of examples are left in each sub-division.

Divide the data again

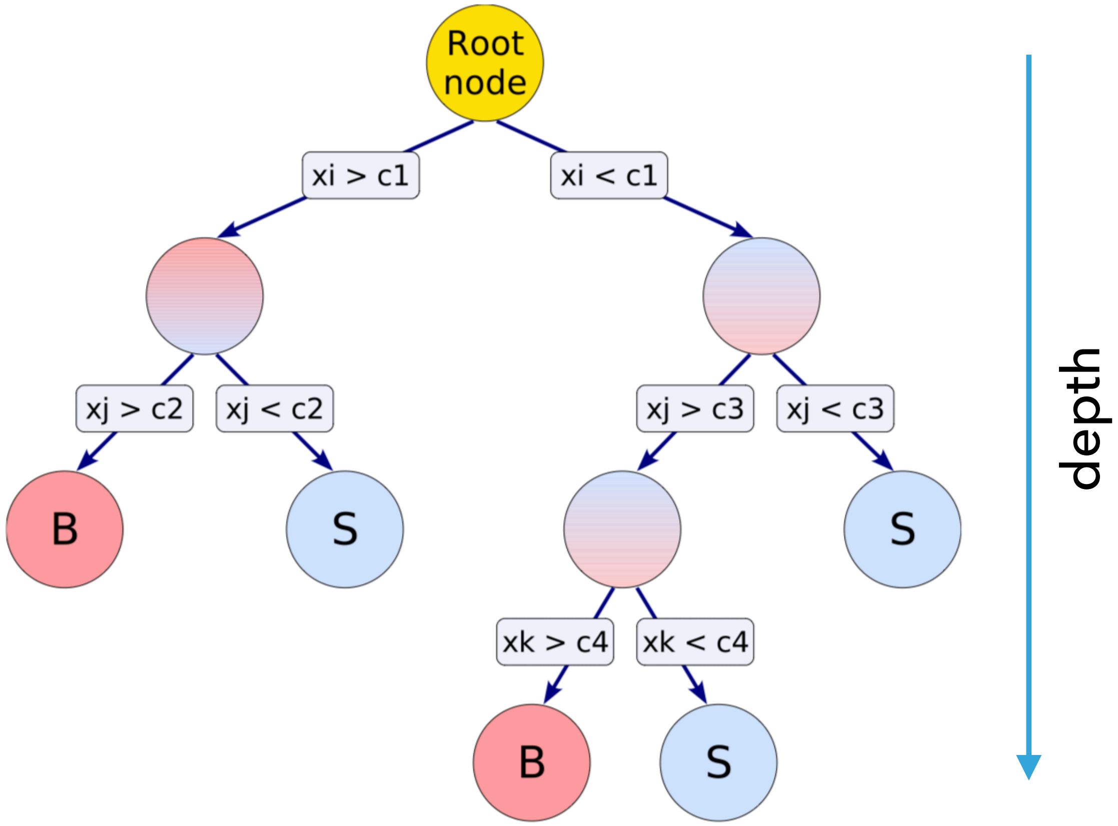




The feature space gets further sub-divided (again).

# DECISION TREES

- The set of rectangular cuts applied to the data allow us to build a tree from the root note.
- We can impose limits on:
  - Tree depth (how many divisions are performed).
  - Node size (how many examples per partition).
- Trees can be extended to more than 2 categories.
- They lend themselves to classifying examples or adapted to make a quantitative prediction (regression)



The decision tree output for a classification problem is

$$G(x) = +1 \text{ or } -1$$

# DECISION TREES

- Decision trees are “weak learners”, as they can take input features that only weakly separate types of example and combine those features to increase the separation.

- A single tree is susceptible to overtraining, and there are various methods of reducing this; including limiting the complexity of a tree, or the limiting the minimum number of examples in each node.

- The decision tree can be extended to an oblique decision tree, in which a linear combination of the features (instead of a single feature) is used to classify examples; so the Heaviside function cut becomes a hyperplane cut.

# BOOSTING

- If a training example has been mis-classified in a training epoch, then the weight of that event can be increased for the next training epoch; so that the cost of mis-classification increases.
- The underlying aim is to take a weak learner and try and boost this into becoming a strong learner.
  - This example re-weighting technique is called boosting.
  - There are several re-weighting methods commonly used; here we discuss:
    - AdaBoost.M1 (popular variant of the Adaptive boosting method)
- Boosted Decision Trees are known as BDTs

Freund and Schapire J. Jap. Soc. AI **14** (1999) 771-780

# BOOSTING: AdaBoost.M1

- i is the ith example out of a data set with N examples.
- m is the mth training out of an ensemble of M learners to be trained.

- Step 1:
  - Assign event weights of $w_i$ = 1/N to all of the N examples.

Freund and Schapire J. Jap. Soc. AI **14** (1999) 771-780

# BOOSTING: AdaBoost.M1

- i is the ith example out of a data set with N examples.
- m is the mth training out of an ensemble of M learners to be trained.

- Step 1:
  - Assign event weights of $w_i$ = 1/N to all of the N examples.

- Step 2: for m=1 through M
  - Train the weak learner (in our case this is a BDT): $G_m(x)$.
  - Compute the error rate $\varepsilon_m$.
  - Calculate the boost factor $\beta_m = \dfrac{\varepsilon_m}{1-\varepsilon_m}$.
  - Update weights for misclassified examples $w_i \mapsto w_i e^{\ln(1/\beta_m)}$.

Freund and Schapire J. Jap. Soc. AI **14** (1999) 771-780

# BOOSTING: AdaBoost.M1

- i is the ith example out of a data set with N examples.
- m is the mth training out of an ensemble of M learners to be trained.

- Step 1:
  - Assign event weights of $w_i$ = 1/N to all of the N examples.

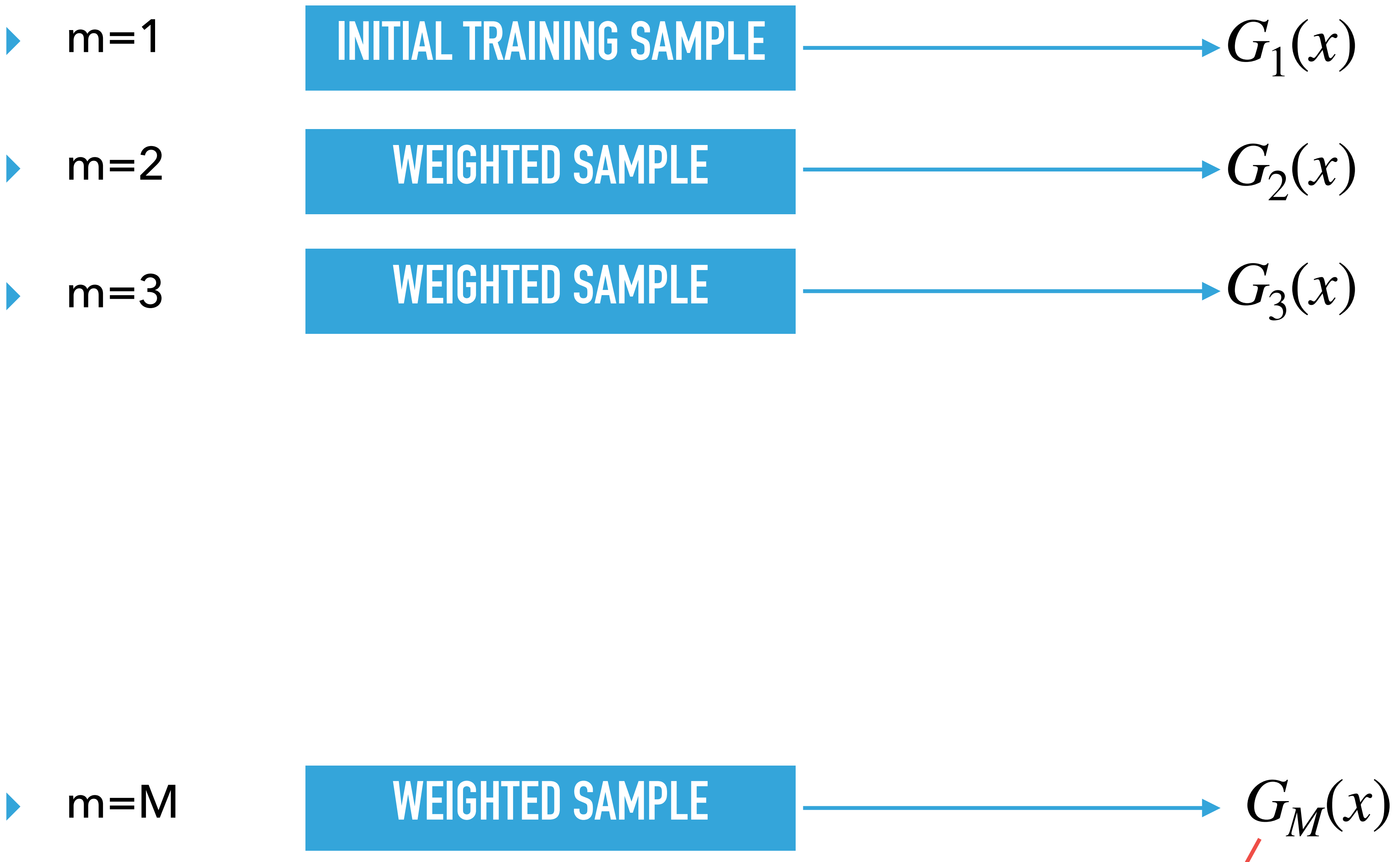
- Step 2: for m=1 through M
  - Train the weak learner (in our case this is a BDT): $G_m(x)$.
  - Compute the error rate $\varepsilon_m$.
  - Calculate the boost factor $\beta_m = \dfrac{\varepsilon_m}{1 - \varepsilon_m}$.
  - Update weights for misclassified examples $w_i \mapsto w_i e^{\ln(1/\beta_m)}$.

- Step 3:
  - Return the weighted committee: a combination of the M trees that have been learned from the data:

$$G(x) = sign\left(\sum_{m=1}^{M} ln\left[\frac{1-\varepsilon_m}{\varepsilon_m}\right] G_m(x)\right)$$

Freund and Schapire J. Jap. Soc. AI **14** (1999) 771-780

# BOOSTING: AdaBoost.M1



- m=1 INITIAL TRAINING SAMPLE → $G_1(x)$
- m=2 WEIGHTED SAMPLE → $G_2(x)$
- m=3 WEIGHTED SAMPLE → $G_3(x)$
- m=M WEIGHTED SAMPLE → $G_M(x)$

The $G_m(x)$ are individual weak learners; each is derived from a training using the data.

The m=1 training uses the original data; all subsequent trainings use reweighted data.

The final classifier output is formed from a committee that is a weighted majority vote algorithm.

$$G(x) = sign\left(\sum_{m=1}^{M} ln\left[\frac{1-\varepsilon_m}{\varepsilon_m}\right] G_m(x)\right)$$

Freund and Schapire J. Jap. Soc. AI **14** (1999) 771-780

# RANDOM FORESTS

- Random Forests are constructed from an ensemble of individual trees.
    - Each tree in the ensemble uses a randomly selected subset of the feature space, and the minimum node size is usually set to 1, so the classifier prediction is almost always accurate.
- The mode (classification) or mean (regression) of the ensemble is the output of the Random Forest.
- The probability that an example $x_i$ is assigned to a given class $c$, is given by

$$P(c\,|\,x_i) = \frac{P(c\,|\,x_i)}{\sum_{l=1}^{n} P(c_l\,|\,x_i)}$$

- and the output score $g_c(x_i)$ is given by the aggregate over $t$ trees in the forest:

$$g_c(x_i) = \frac{1}{t}\sum_{j=1}^{t} P_j(c\,|\,x_i)$$

- The classification of $x_i$ is simply the class $c$ that maximises $g_c(x_i)$.

Ho, Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.

# OVERVIEW

- Examples are provided for the tutorials.
- These use ROOT or Python and have been tested using the following versions:
  - ROOT: 6.06/02
  - Python 3.7 (via an Anaconda install) with the following modules
    - TensorFlow 2.2 [Note: examples are not taking advantage of eager execution and are using the V1 backward compatibility mode]
    - matplotlib
    - numpy
    - sklearn

# WHAT IS MACHINE LEARNING?

# WHAT IS MACHINE LEARNING?

- Oxford English Dictionary:
  - *"a type of artificial intelligence in which computers use huge amounts of data to learn how to do tasks rather than being programmed to do them"*
- *Collins Dictonary:*
  - *"a branch of artificial intelligence in which a computer generates rules underlying or based on raw data that has been fed into it"*
- Google Developers glossary:
  - *"A program or system that builds (trains) a predictive model from input data. The system uses the learned model to make useful predictions from new (never-before-seen) data drawn from the same distribution as the one used to train the model. Machine learning also refers to the field of study concerned with these programs or systems."*
- There is no single definition agreed of machine learning, so I will use a working definition.
  - *"The process of using an algorithm to approximate data using some underlying optimisation heuristic"*

# WHAT IS MACHINE LEARNING?

   - *“The process of using an algorithm to approximate data using some underlying optimisation heuristic”*

- We are doing function approximation using some algorithm fit to some reference data using some heuristic.

- The fitting in this context is referred to as training or learning.
   - There are different learning paradigms, we will focus on supervised and unsupervised learning.

- The trained function is used as a model (i.e. for prediction).

- Dimensionality and parameters are implied in these slides when referring to general situations, i.e. $f(\underline{x}, \underline{\theta}) \rightarrow f(x, \theta) \rightarrow f(x)$.

# ETHICS

# ETHICS

- How does ethics fit into a lecture on machine learning?

**Ethics** *plural in form but singular or plural in construction* **:** the discipline dealing with what is good and bad and with moral duty and obligation.

**Morals** describes one's particular values concerning what is right and what is wrong.

Using definitions from the Miriam Webster Dictionary

# ETHICS

- How does ethics fit into a lecture on machine learning?
    - Is it ethical to develop an algorithm to identify the political leaning of an individual with the intent of targeting advertising, to polarise the viewpoint of voters with positive reinforcement messages or fake news, to undermine or influence the outcome of an election?
    - Is it ethical to develop a pandemic model (e.g. using an SIR approach[1]) using an AI, that could influence Government policy, without fully testing the robustness of predictions?

[1] e.g. see this article on wikipedia Compartmental models in epidemiology. A good illustration of a variant of this model can be found at https://upload.wikimedia.org/wikipedia/commons/3/31/SIRD_model_anim.gif

# ETHICS

- But that is the “real world” why should I care about ethics?
  - Is it ethical to use an algorithm for science without checking that the result makes sense?

# ETHICS

- But that is the “real world” why should I care about ethics?
  - Is it ethical to use an algorithm for science without checking that the result makes sense?
  - Is it ethical to use an energy intensive algorithm when a computationally cheap alternative that performs just as well exists?

# ETHICS

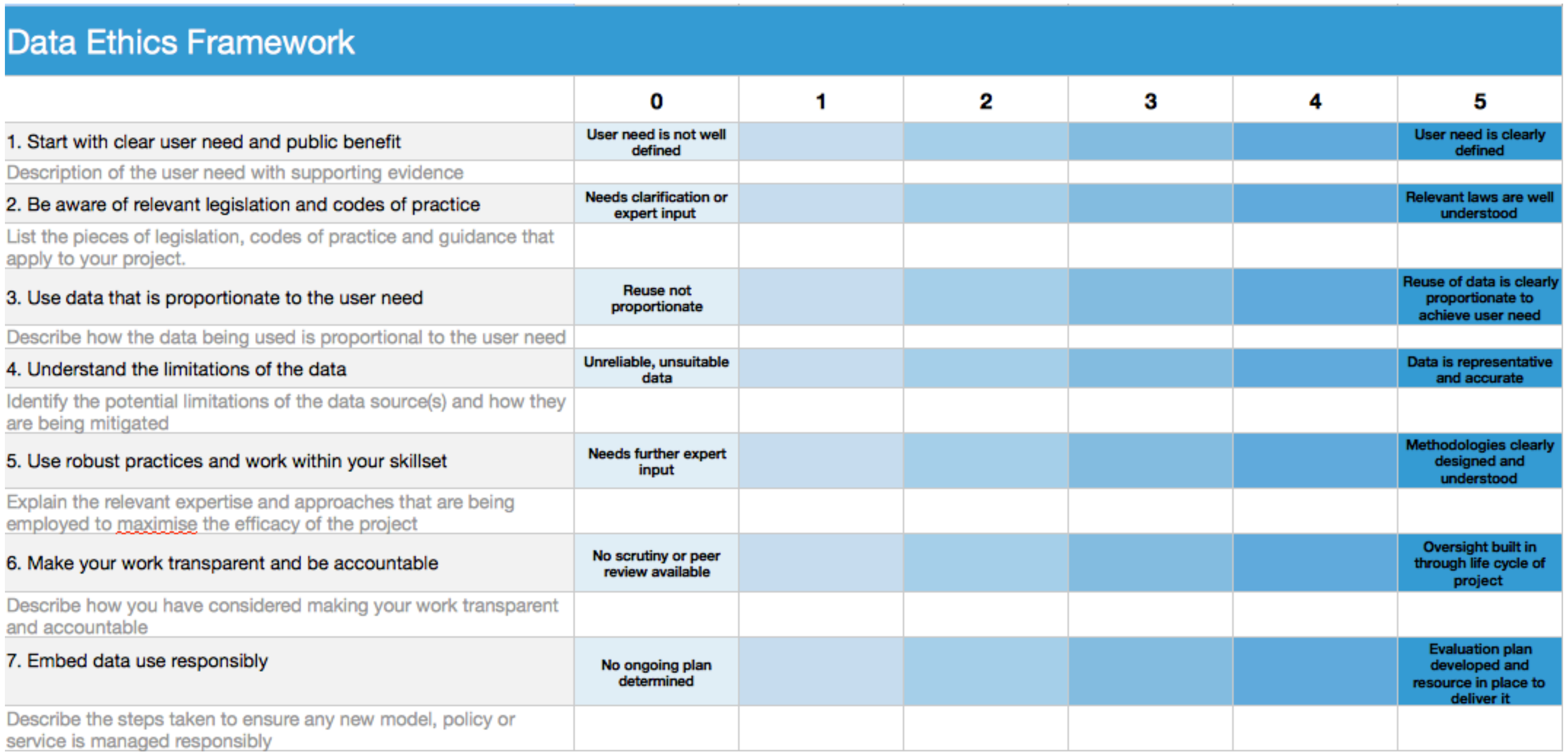
- But that is the "real world" why should I care about ethics?
  - Is it ethical to use an algorithm for science without checking that the result makes sense?
  - Is it ethical to use an energy intensive algorithm when a computationally cheap alternative that performs just as well exists?
- What we think of as good scientific practice, is also ethical behaviour. It leads to robust results.
- Ethical behaviour in the wider world can have deeper ramifications than getting a robust scientific result or not.

# ETHICS

- But that is the "*real world*" why should I care about ethics?
  - In reality most people who do a PhD in an STFC (or EPSRC) area of science will not end up in academia, and ethics will play a role beyond scientific correctness for those scientists.
  - For those of us who do stay in academia, then we have an obligation to help people understand the ramifications of algorithms and our rationale for using them (or not).
  - For all of us, we have transferable skills that can be applied to real world problems.

# ETHICS

- The UK Government developed a data ethics framework[1]:
  - *“Public sector organisations should use the Data Ethics Framework to guide the appropriate use of data to inform policy and service design”*



Data Ethics Framework

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 1. Start with clear user need and public benefit | User need is not well defined | | | | | User need is clearly defined |
| Description of the user need with supporting evidence | | | | | | |
| 2. Be aware of relevant legislation and codes of practice | Needs clarification or expert input | | | | | Relevant laws are well understood |
| List the pieces of legislation, codes of practice and guidance that apply to your project. | | | | | | |
| 3. Use data that is proportionate to the user need | Reuse not proportionate | | | | | Reuse of data is clearly proportionate to achieve user need |
| Describe how the data being used is proportional to the user need | | | | | | |
| 4. Understand the limitations of the data | Unreliable, unsuitable data | | | | | Data is representative and accurate |
| Identify the potential limitations of the data source(s) and how they are being mitigated | | | | | | |
| 5. Use robust practices and work within your skillset | Needs further expert input | | | | | Methodologies clearly designed and understood |
| Explain the relevant expertise and approaches that are being employed to maximise the efficacy of the project | | | | | | |
| 6. Make your work transparent and be accountable | No scrutiny or peer review available | | | | | Oversight built in through life cycle of project |
| Describe how you have considered making your work transparent and accountable | | | | | | |
| 7. Embed data use responsibly | No ongoing plan determined | | | | | Evaluation plan developed and resource in place to deliver it |
| Describe the steps taken to ensure any new model, policy or service is managed responsibly | | | | | | |

- Other organisations such as the UN and IEEE (as well as the EU) are also concerned about ethical use of data, and ethical algorithms, from regulatory and the perspective of rights.

[1] See https://www.gov.uk/government/publications/data-ethics-framework

# DATA WRANGLING

# DATA WRANGLING (AKA FEATURE ENGINEERING)

- Need to understand the data being analysed.
  - Domain context will provide most insight into getting information to highlight the solution to your problem.
  - Identify the features of interest in the data.
  - Allow you to reduce the dimensionality of the problem into as few a set of inputs as possible.
  - Most of the analysis can be done with this domain context background:
    - Deep Learning (DL) can replace the hard work of feature engineering for a resource cost and a lack of explainability and interpretability. [1]
    - "Smart Learning" (SL) is an alternative way of approaching the problem. [2]

[1] e.g. See work by Pierre Baldi on Higgs analysis at the LHC: DOI: 10.1038/ncomms5308. Also see appendix.

[2] I attribute the term Smart Learning to the use of domain knowledge and understandable AI, such as Bayesian Networks. A term that I first heard about from Norman Fenton who has a good book on the topic.

# DATA WRANGLING

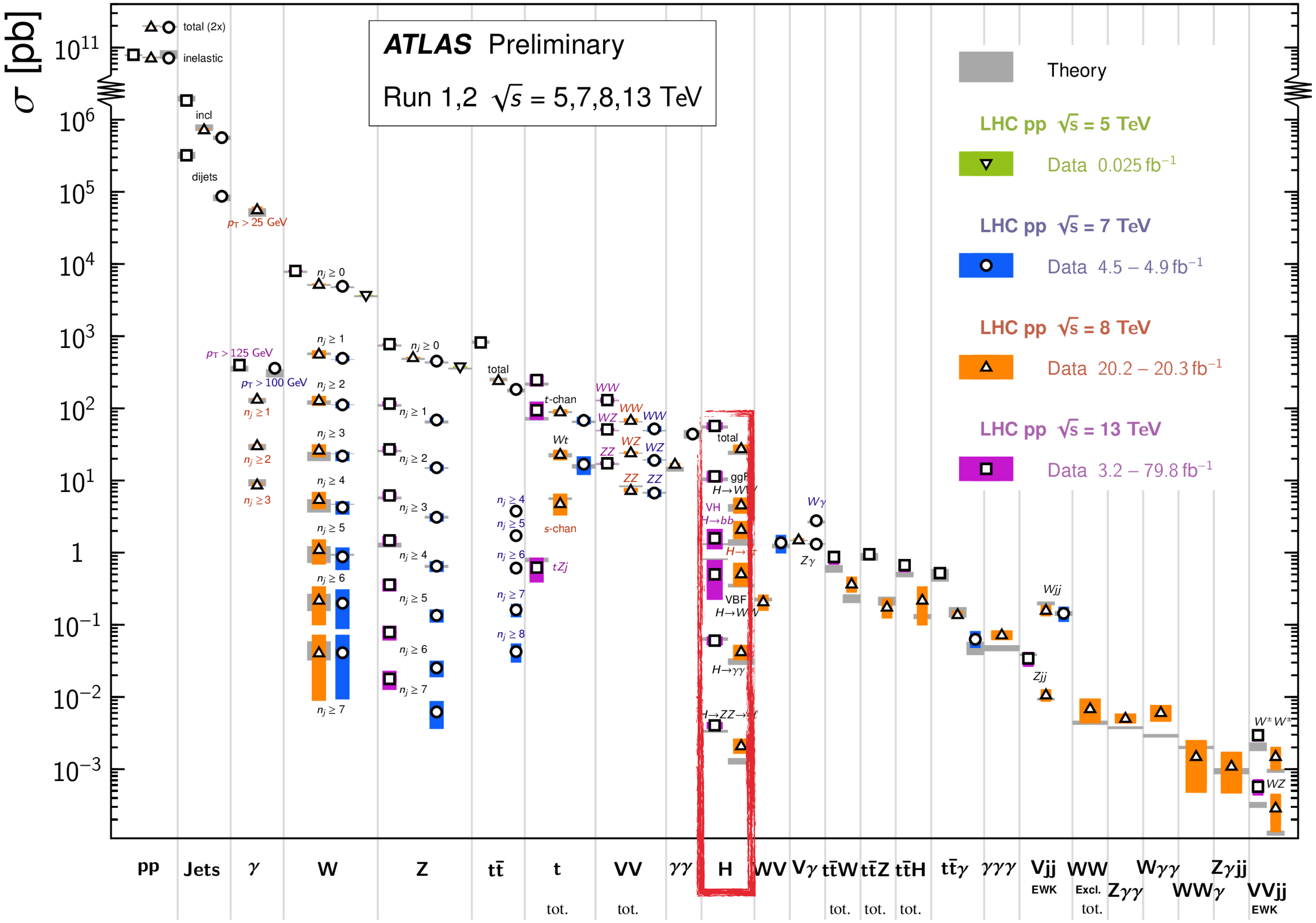
- In HEP we use cut based selection to:
  - Remove pathological data (poorly calibrated or partially complete data, bad beam conditions, etc).
  - Remove obvious background examples from the data (well known SM processes that are just not interesting for study, or use as a calibration or control sample).
  - Prepare data for the "statistical analysis" that will include the use of multivariate analysis techniques and fitting.

# DATA WRANGLING

- e.g. Higgs physics: Throw away ~$10^9$ recorded events for each interesting one.

Standard Model Production Cross Section Measurements

Status: November 2019



# DATA WRANGLING

- Knowing what the task is allows one to identify the features of interest. This is the domain context knowledge.
  - If your signal is the process pp→HH+X, then:
    - Laws of physics provide insights for you to refine your list of features to train on.
    - e.g. the Higgs mass or $p_T$ will play a role in identifying the signal, along with decay product properties (e.g. b-tag quality), etc.

# DATA WRANGLING

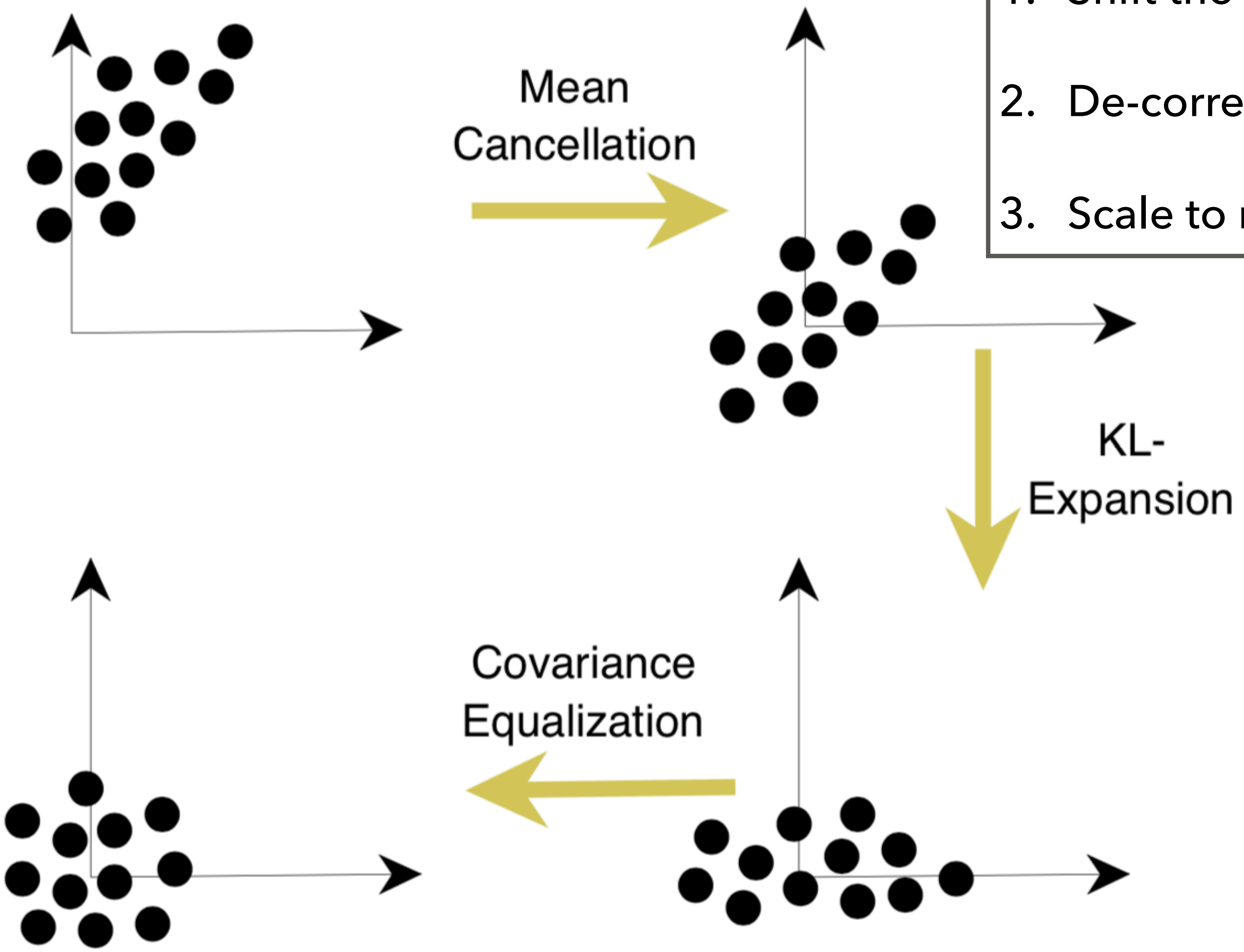
- Reduce the number of dimensions of interest:
  - Trial and error with different inputs to identify what improves performance of the algorithm and what does not.
  - Linearly correlated features can be combined to reduce the number of features providing information for the algorithm to learn from.
    - Principal Component Analysis (PCA) to address this problem.
  - Some neural network configurations do that automatically for you (e.g. auto-encoders).
  - Let the algorithm learn what is important and what is not.
- Some algorithms (e.g. support vector machines) implicitly increase the dimensionality of the problem.

# ACTIVATION FUNCTIONS: DATA PREPARATION

- Input features are arbitrary; whereas (for example) activation functions in neural networks work best for 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 ⟼TeV would translate $10^{12}$ eV ⟼ 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.

# DATA WRANGLING

- Variable transformations are also useful (can be vital).



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

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

# DATA WRANGLING

- Data wrangling is an important part of ensuring you get the most out of applying machine learning; yet it is often not celebrated as the requirements can be very problem specific.
- If you are interested in this topic you may wish to review the data wrangling presentation and tutorials given by Dr Nick Barlow from the Alan Turing Institute at the GradNet meeting on Machine Learning and AI in January 2020:
  - Dr Barlow's talk and lecture can be found at: https://indico.ph.qmul.ac.uk/indico/conferenceOtherViews.py?view=standard&confId=543

TYPES OF LEARNING
GRID SEARCH
GRADIENT DESCENT
ADAM OPTIMISER
MISCELLANEOUS

# OPTMISATION

A number of optimisation methods exist, and I selectively focus on three. For example see C. Bishop, Neural Networks for Pattern Recognition or I. Goodfellow et. al, Deep Learning for more information on optimisation algorithms.

# OPTIMISATION

## TYPES OF LEARNING

# TYPES OF LEARNING

- Consider a model $y$ that depends on a parameter set $\theta$, we can select a point in the parameter hyperspace $\hat{\theta}$ that has a corresponding estimator of the function $\hat{y}$.
- The $\theta$ are hyper-parameters (HPs) of the model.
- **Unsupervised learning:**
  - Infer the probability distribution $P(\widehat{y})$ from the data without using labels.
  - Widely used in many fields of research.
- **Supervised learning:**
  - Use training examples from labeled data sets with a known type or value, $t$, for each example.
  - Some loss function is used to compare $t$ against model predictions $\widehat{y}$.
  - Can be thought of as computing a conditional probability $P(t\,|\,\widehat{y})$.
  - This is the most commonly used approach in particle physics today.

# TYPES OF LEARNING: SUPERVISED LEARNING

- Define a heuristic based on some figure of merit (FOM) designed to improve the value of that metric through some iterative process.

- The FOM is called the objective function or loss function or cost function in machine learning.

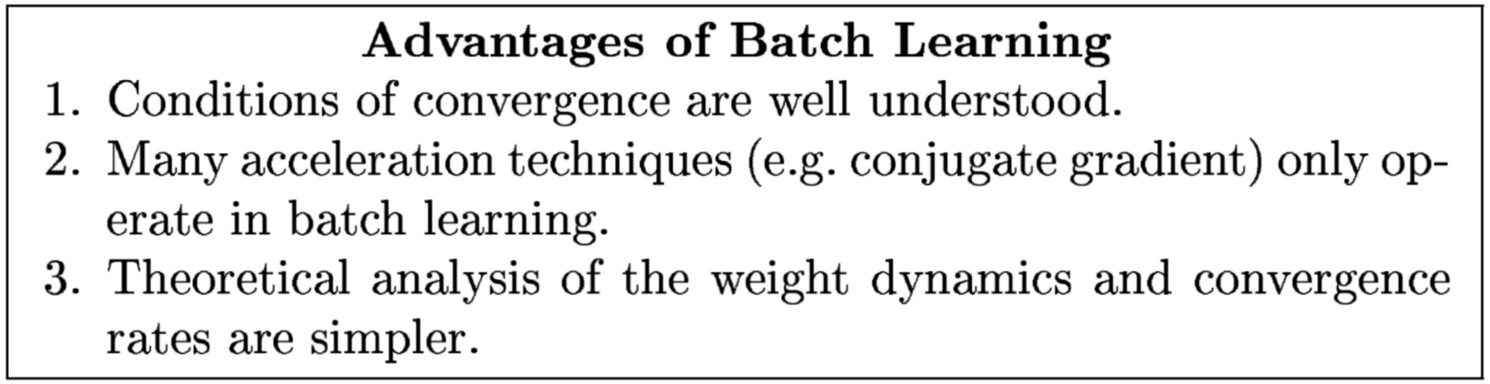
- e.g. the $L_2$ norm loss function: this is like a $\chi^2$, but without the error term:

$$L_2 = \sum_{i=1}^{N_{examples}} \left[ t_i - \hat{y}(\hat{\theta}) \right]^2$$

- The $\theta$ are called HPs as they form a hyperspace; other variables in the optimisation process are often included in the set of HPs (e.g. learning rate for a neural network, cost for a support vector machine, tree depth for a decision tree etc.).

# TYPES OF LEARNING

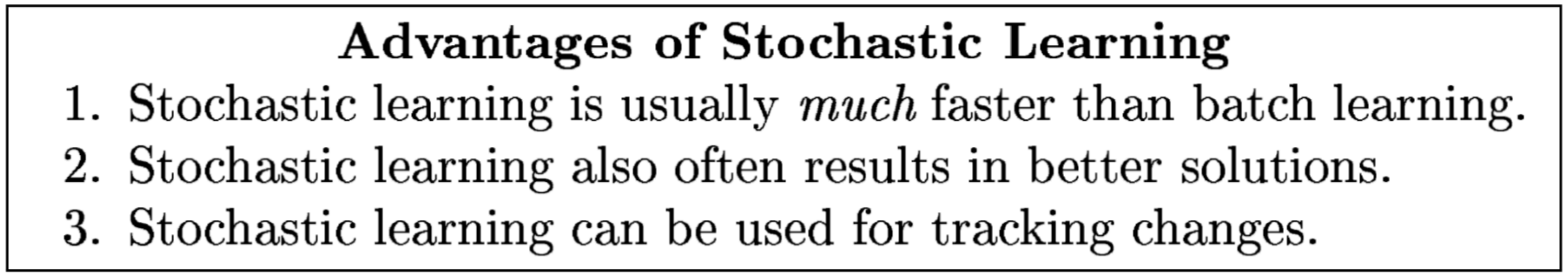
- **Batch learning**
    - Use a sample (batch) of training data to evaluate an estimate of the error and update weights in the optimisation.



**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.

- **Stochastic learning**
    - Use individual training examples to evaluate an estimate of the error and update weights in the optimisation.



**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.

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

- THIS IS A SIMPLE OPTIMISATION ALGORITHM THAT CAN BE USED WITH NO PROGRAMMING EXPERIENCE.
- ONE CAN IMPLEMENT A 1 OR 2 DIMENSIONAL GRID SEARCH USING AN EXCEL SPREADSHEET, AND FROM INSPECTION OF THE TABULATED RESULTS YOU CAN OPTIMISE SIMPLE PROBLEMS.
- THIS IS A BRUTE FORCE OPTIMISATION APPROACH, AND IT IS USED WIDELY IN CERTAIN APPLICATIONS ACROSS A NUMBER OF RESEARCH FIELDS.

# OPTIMISATION

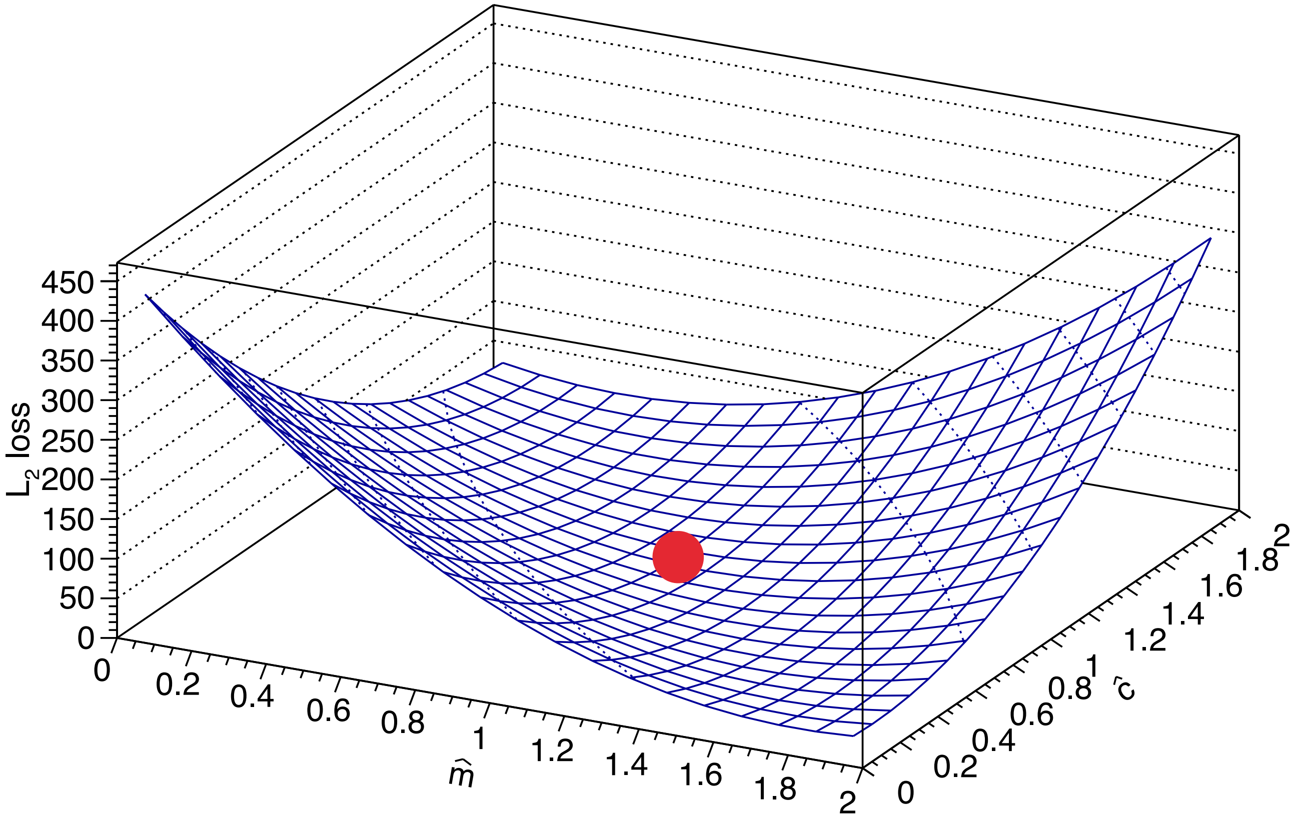
## GRID SEARCH

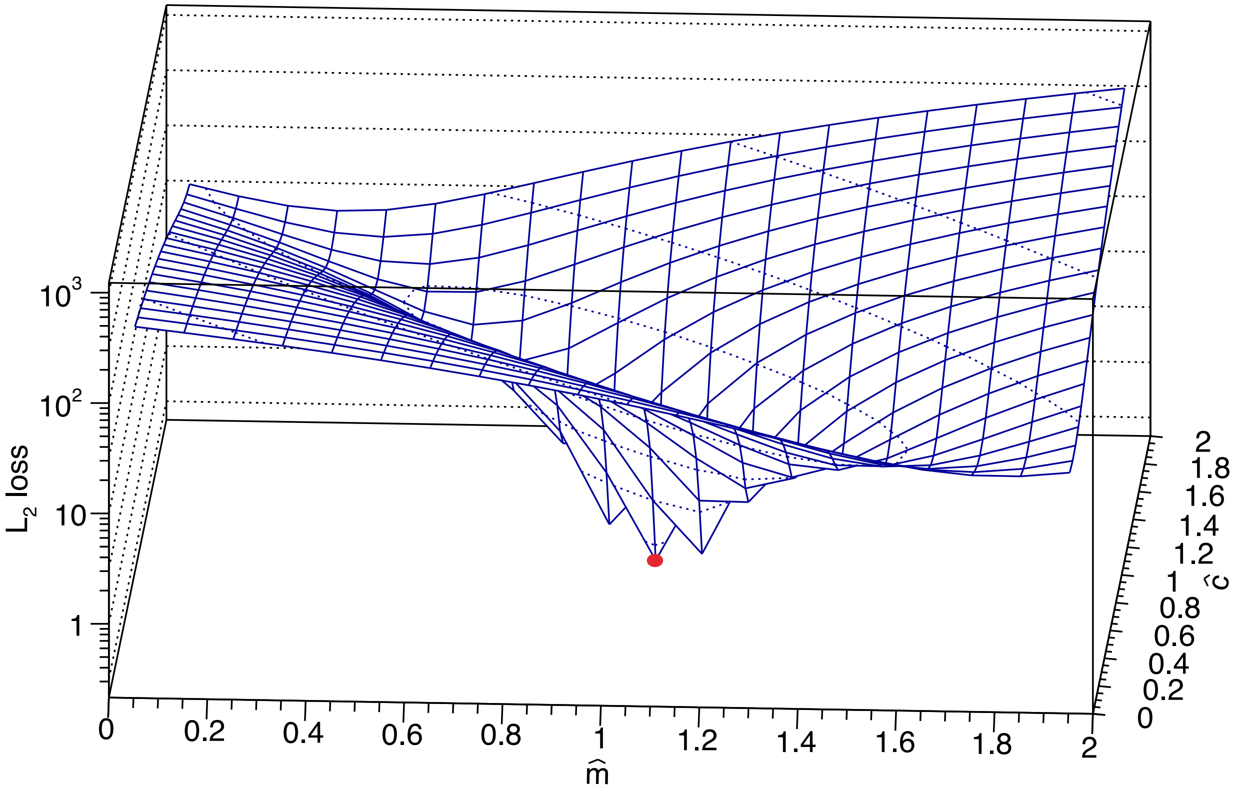
# GRID SEARCH

- Scanning through the hyperspace for each $\hat{\theta}_i$ allows us to compute L$_2$, and the minimum value obtained is the "best fit" or optimal estimate of the parameters $\theta$.
  - Simple heuristic to implement - you can do this in Excel for 1 or 2D problems. ✓
  - Easy to understand. ✓
  - Expensive to compute: scanning $n$ points in a dimension requires $n^M$ computations for $M = dim(\theta)$. ✗
- Heuristic suffers from the curse of dimensionality.

# GRID SEARCH

- e.g. Consider an L2 loss function optimisation of the HPs for $y = mx + c$, i.e. optimise $m$ and $c$. Here $m = 1.0, c = 1.0$





- The contours of the loss function show a minimum.
- The optimal value is selected from a discrete grid of points, only get an exact result if the grid maps onto the problem perfectly (as in this example).

# GRID SEARCH

- e.g. The libsvm[1] package uses a HP grid search for optimisation for the Support Vector Machine algorithm; where the cost C and $\Gamma$ hyper-parameters need to be optimised.
  - The grid search is done efficiently by adapting to whatever step is sensible, in this case the algorithm has $\Gamma$ as the parameter of an exponential (the radial basis function); and so a linear search in C and a logarithmic search in $\Gamma$ is appropriate to sample the parameter hyperspace effectively.

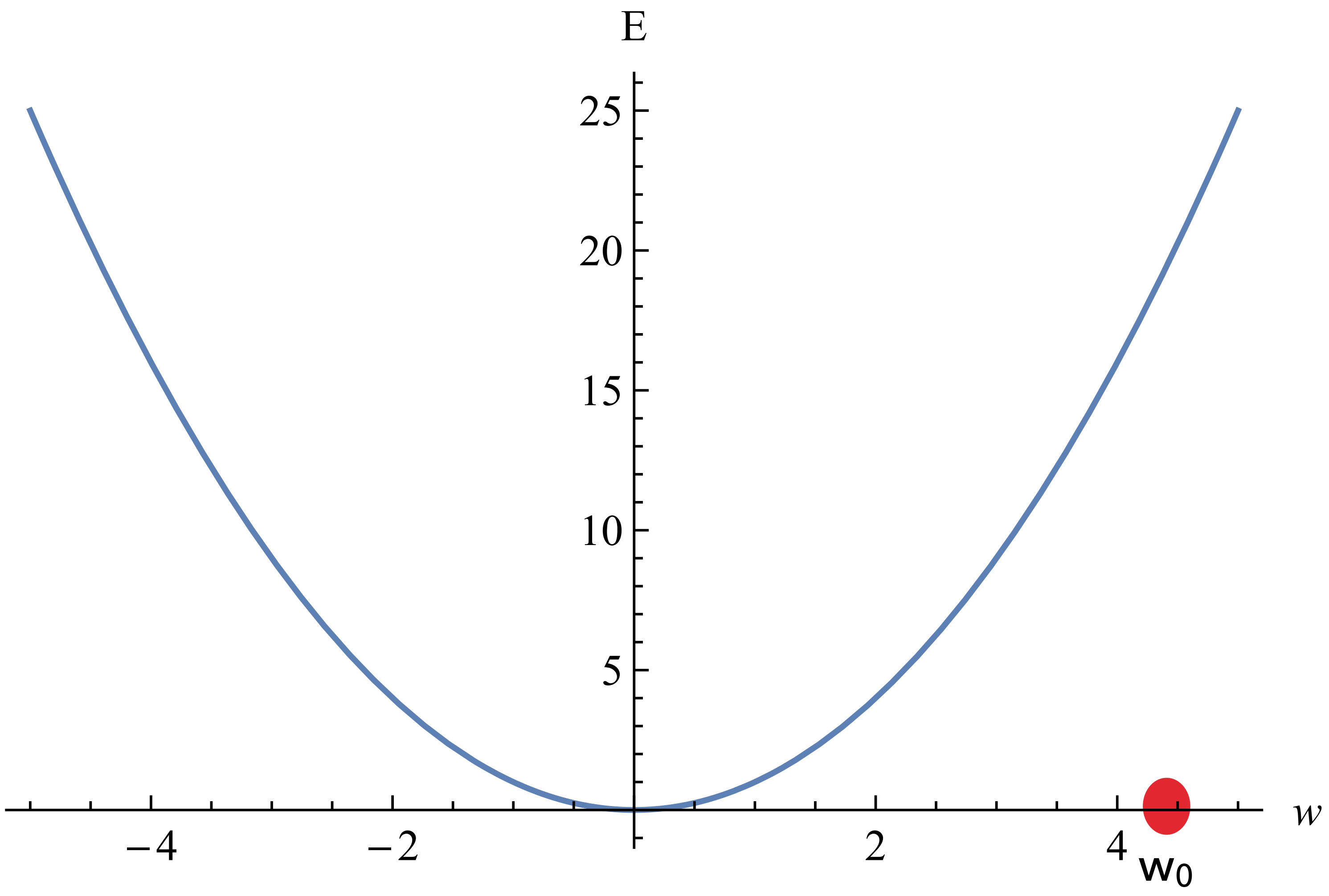
[1] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, **2:**27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

# OPTIMISATION

## GRADIENT DESCENT

# GRADIENT DESCENT

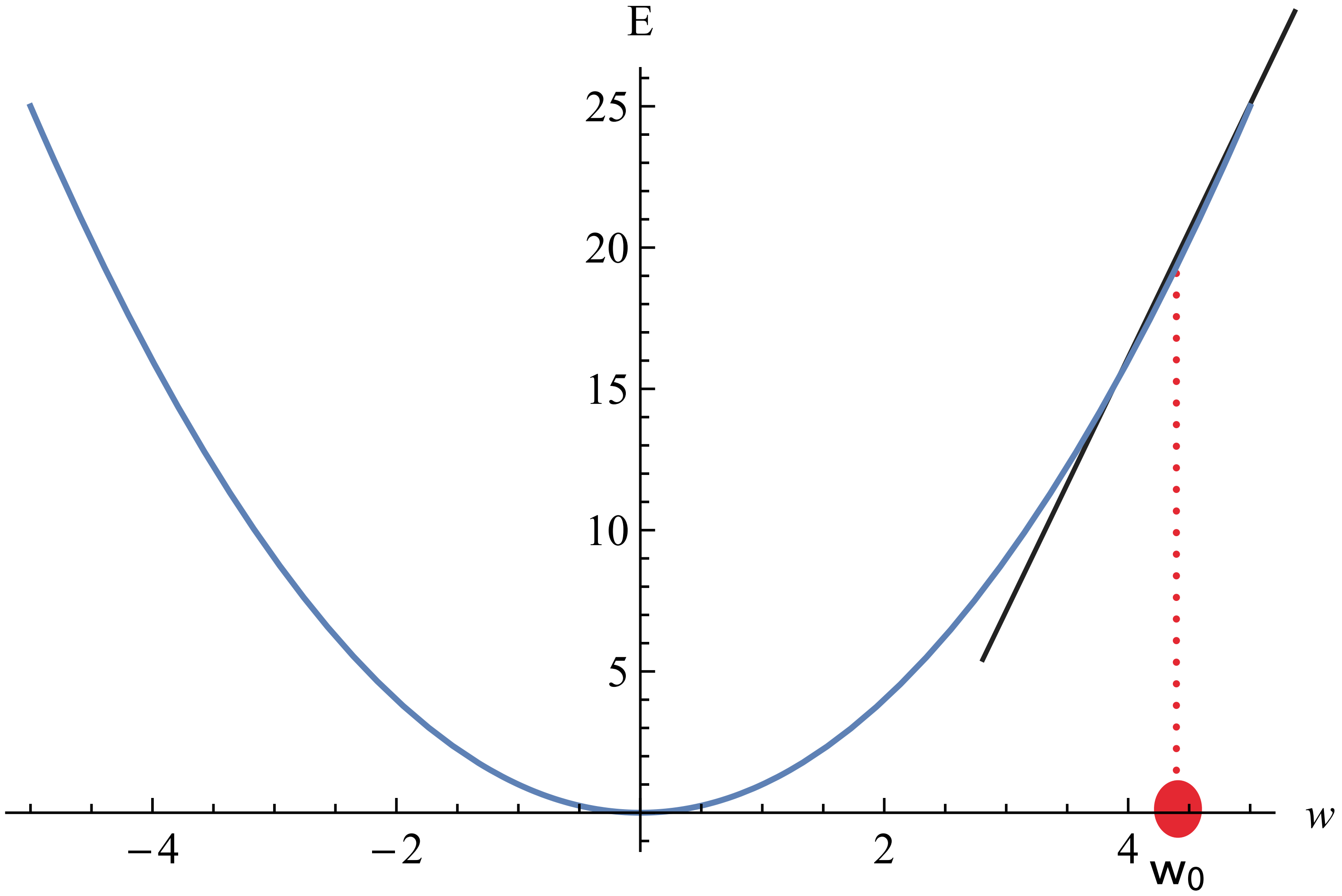
- Guess an initial value for the weight parameter: $w_0$.



e.g. see Bishop, Neural Networks for Pattern Recognition (2013) Oxford University Press

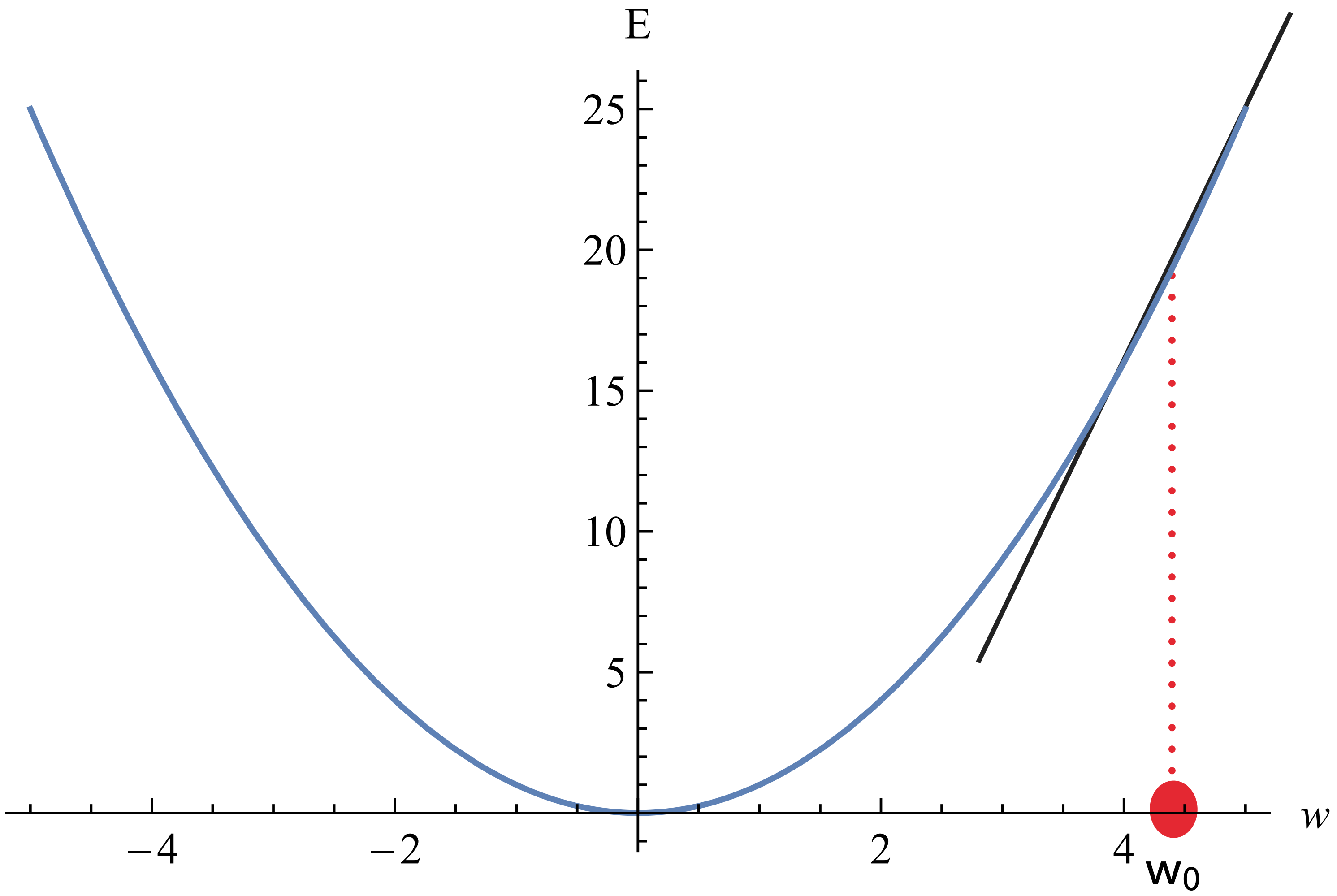
# GRADIENT DESCENT

- Estimate the gradient at that point (tangent to the curve)



# GRADIENT DESCENT

- Compute Δw such that ΔE is negative (to move toward the minimum)



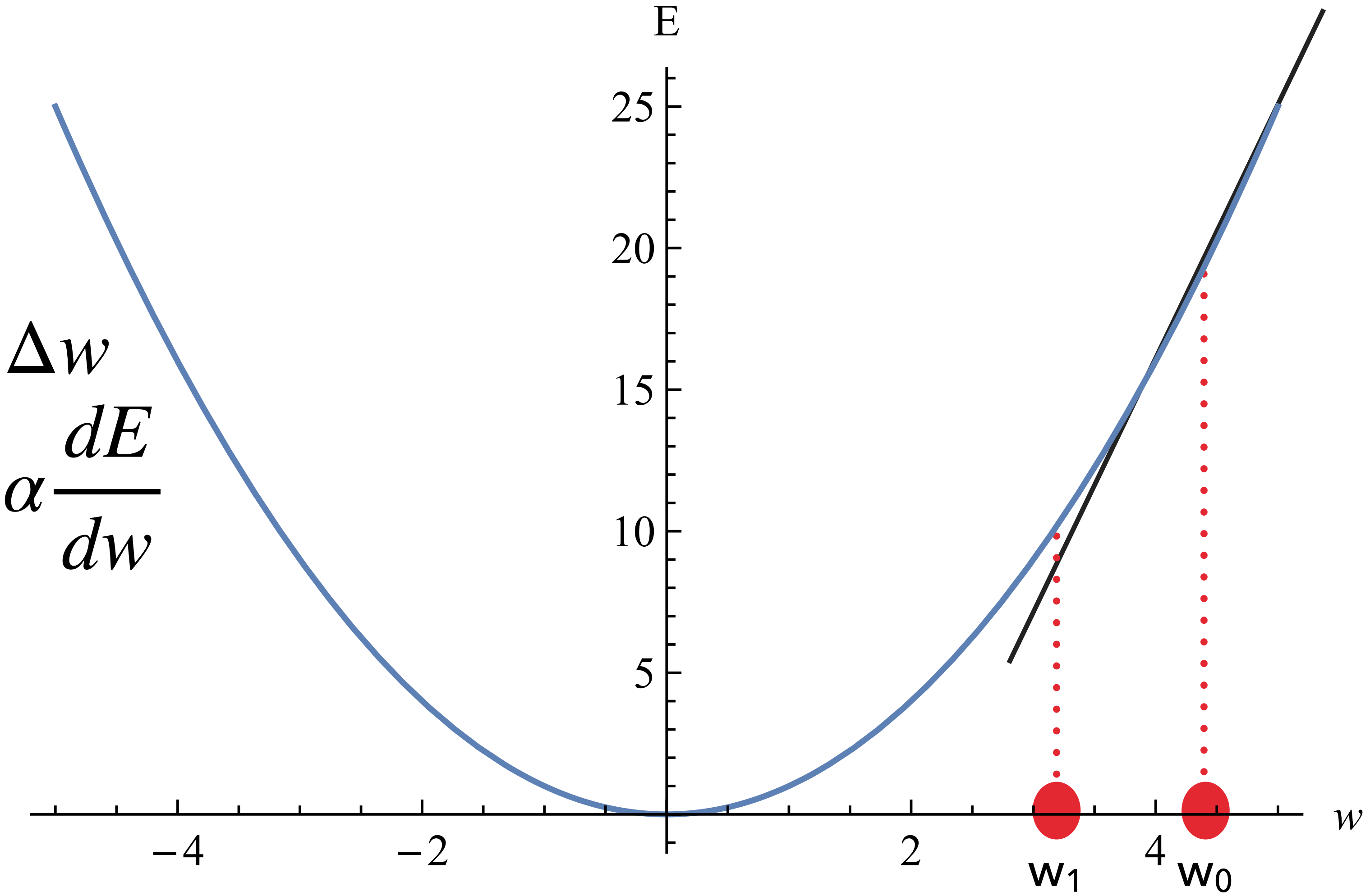
$$\Delta E = \Delta w \frac{dE}{dw}$$
$$= -\alpha \left(\frac{dE}{dw}\right)^2$$

$\alpha$ is the learning rate: a small positive number

Choose $\Delta w = -\alpha \dfrac{dE}{dw}$ to ensure ΔE is always negative.

# GRADIENT DESCENT

- Compute a new weight value: $w_1 = w_0+\Delta w$



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

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

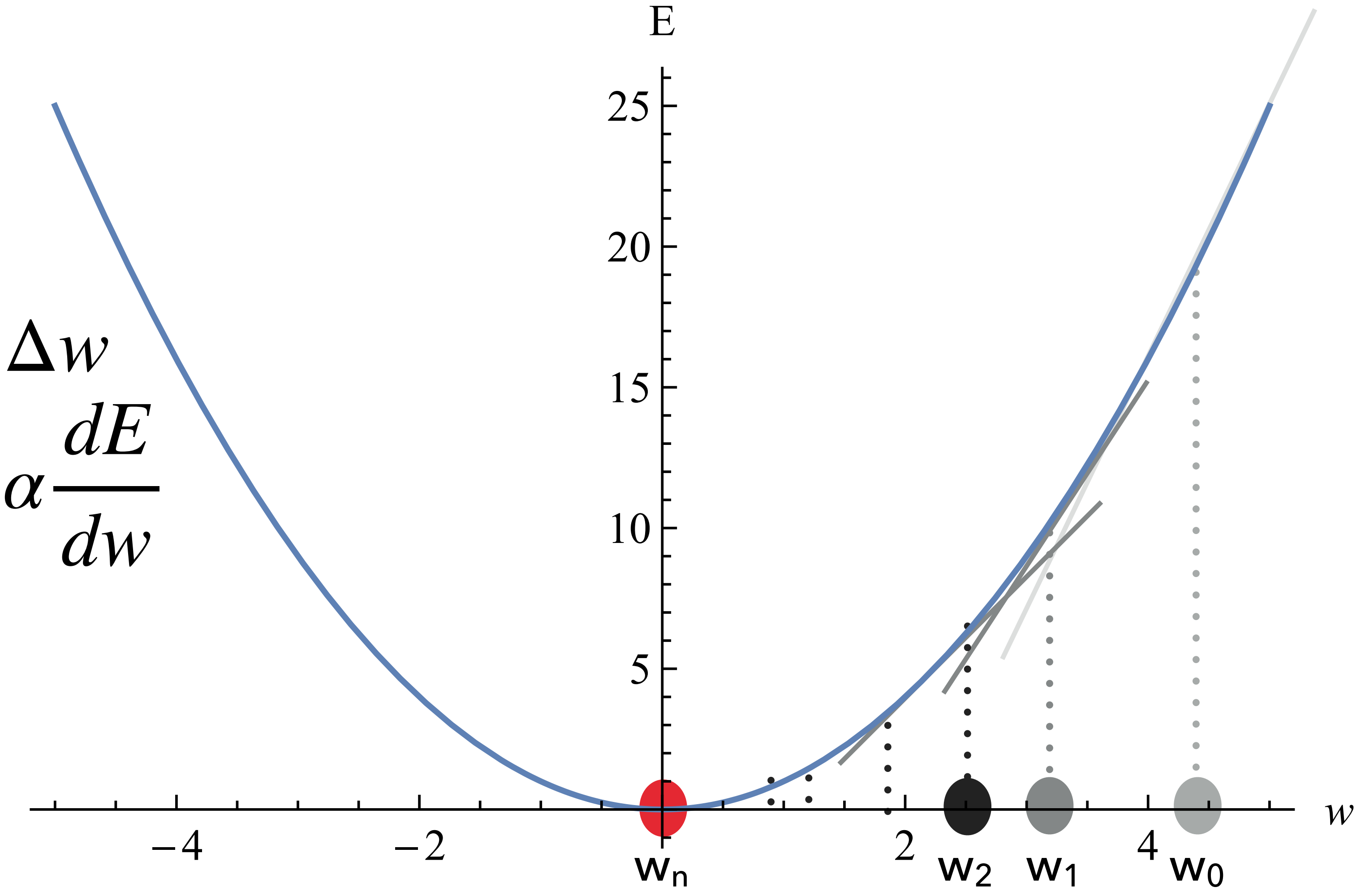
$\alpha$ is the learning rate: a small positive number

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

# GRADIENT DESCENT

- Repeat until some convergence criteria is satisfied.

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



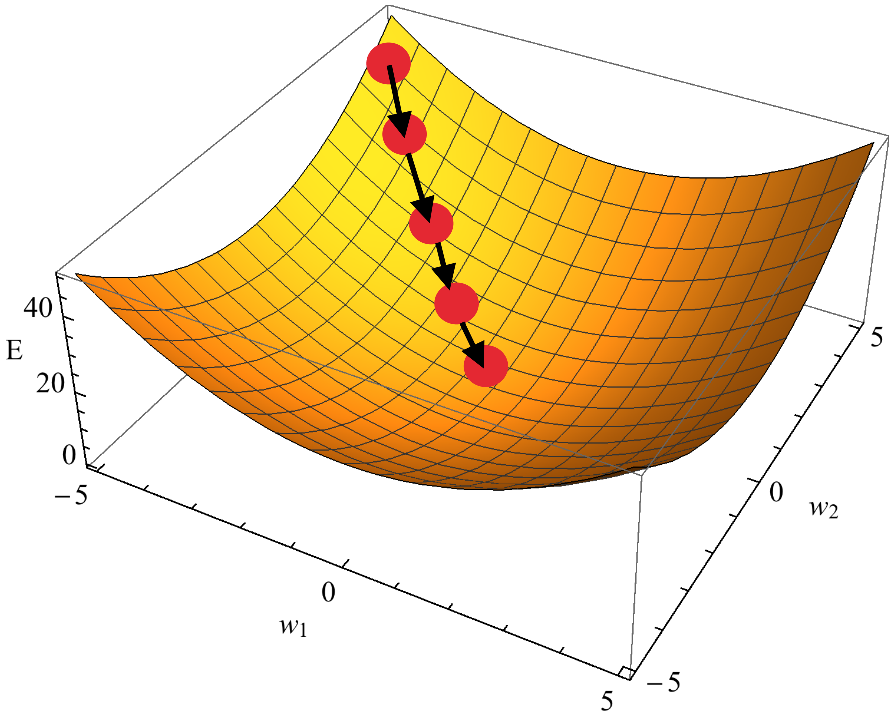
$$\Delta E = \Delta w \frac{dE}{dw}$$
$$= -\alpha \left(\frac{dE}{dw}\right)^2$$

$\alpha$ is the learning rate: a small positive number

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

# GRADIENT DESCENT

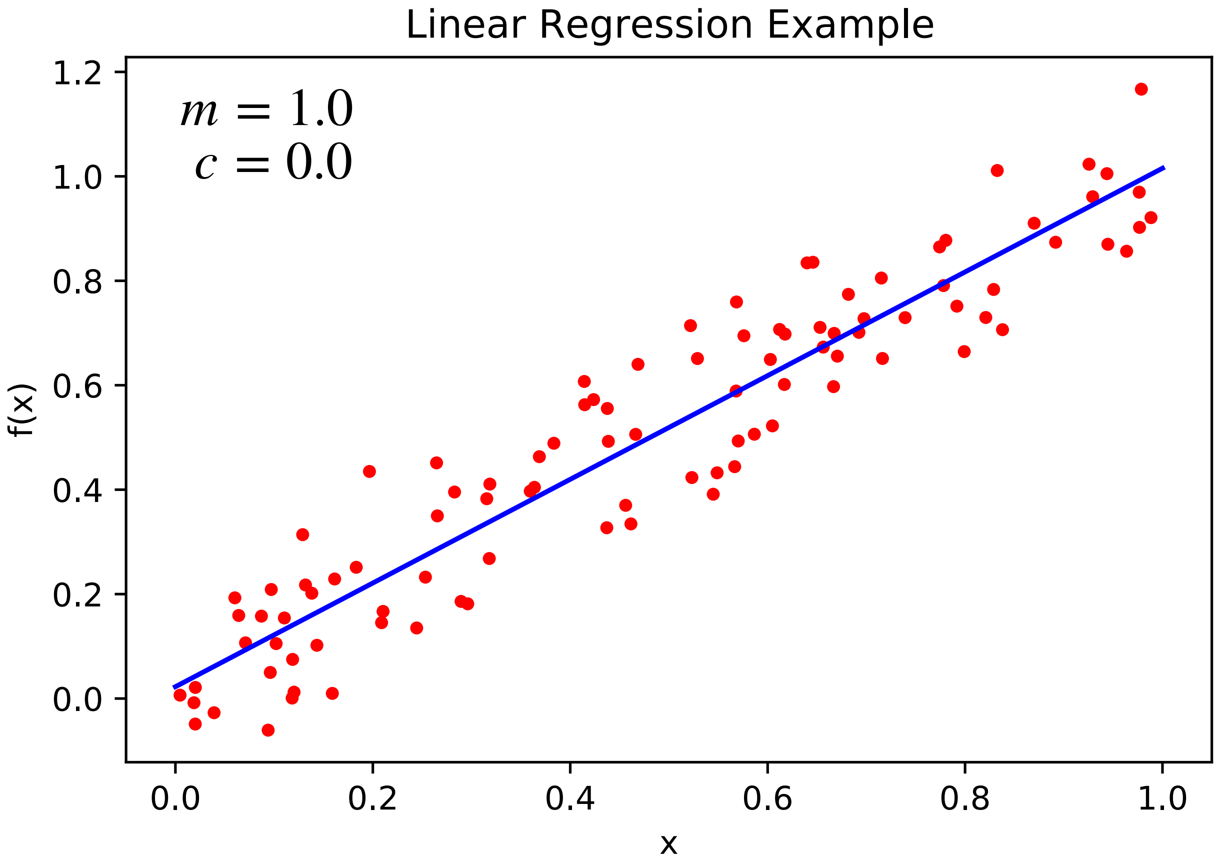
- We can extend this from a one parameter optimisation to a 2 parameter one, and follow the same principles, now in 2D.

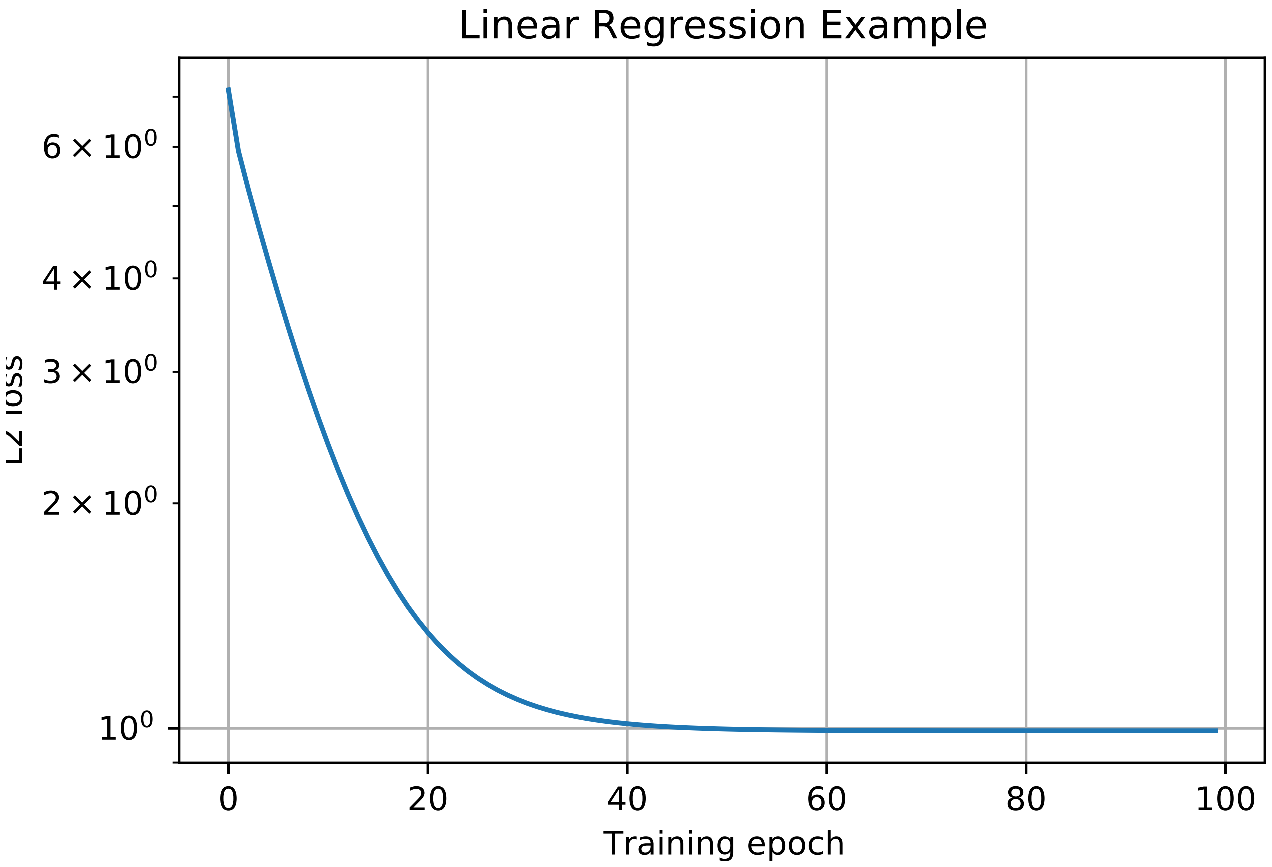


- The successive points $w_i$+1 can be visualised a bit like a ball rolling down a concave hill into the region of the minimum.
- In general update weights such that $\Delta E = \Delta w \nabla E = -\alpha \nabla^2 E$
- and $w_{i+1} = w_i - \alpha \nabla E$.

# GRADIENT DESCENT: EXAMPLE

- Returning to the $y = mx + c$ example, we can optimise this using the Gradient Descent algorithm.





Use N=100 and $\alpha = 0.005$ with the TensorFlow GradientDescent optimiser to obtain:

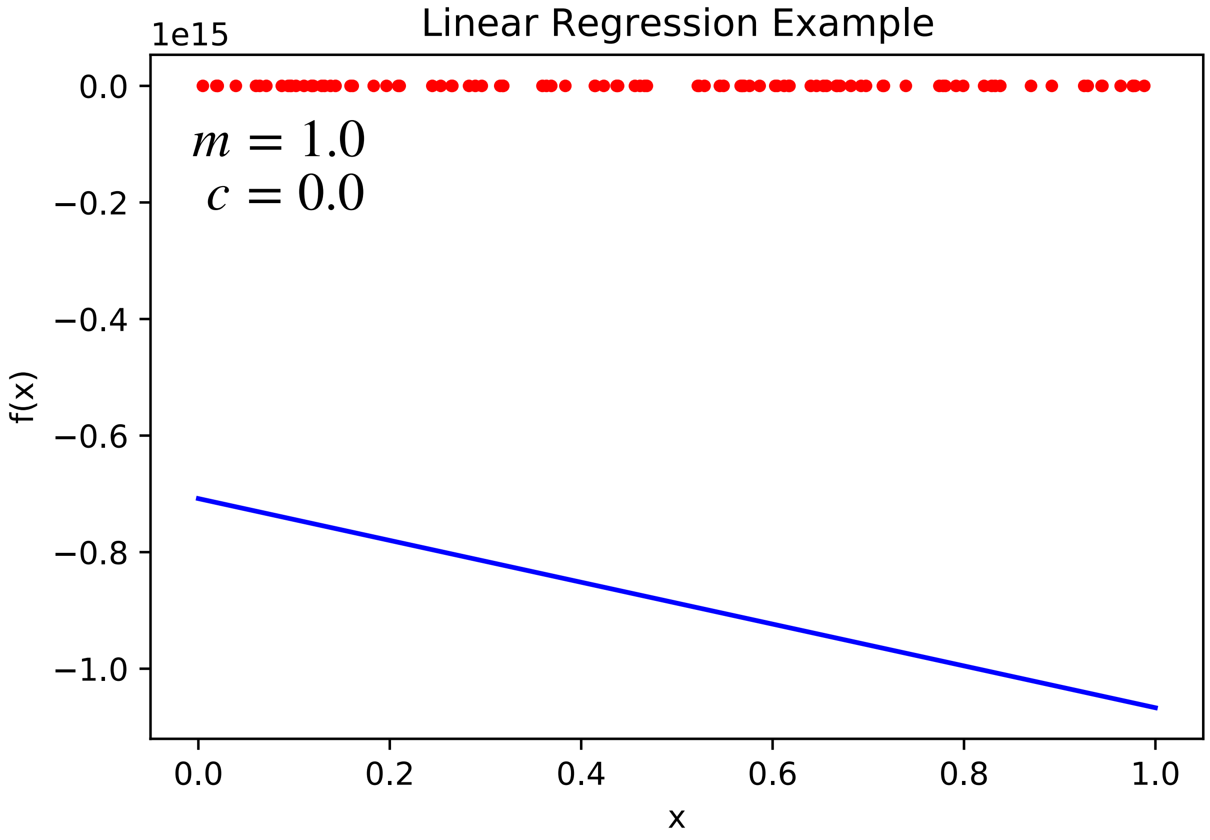
$$\widehat{m} = 0.9928...$$

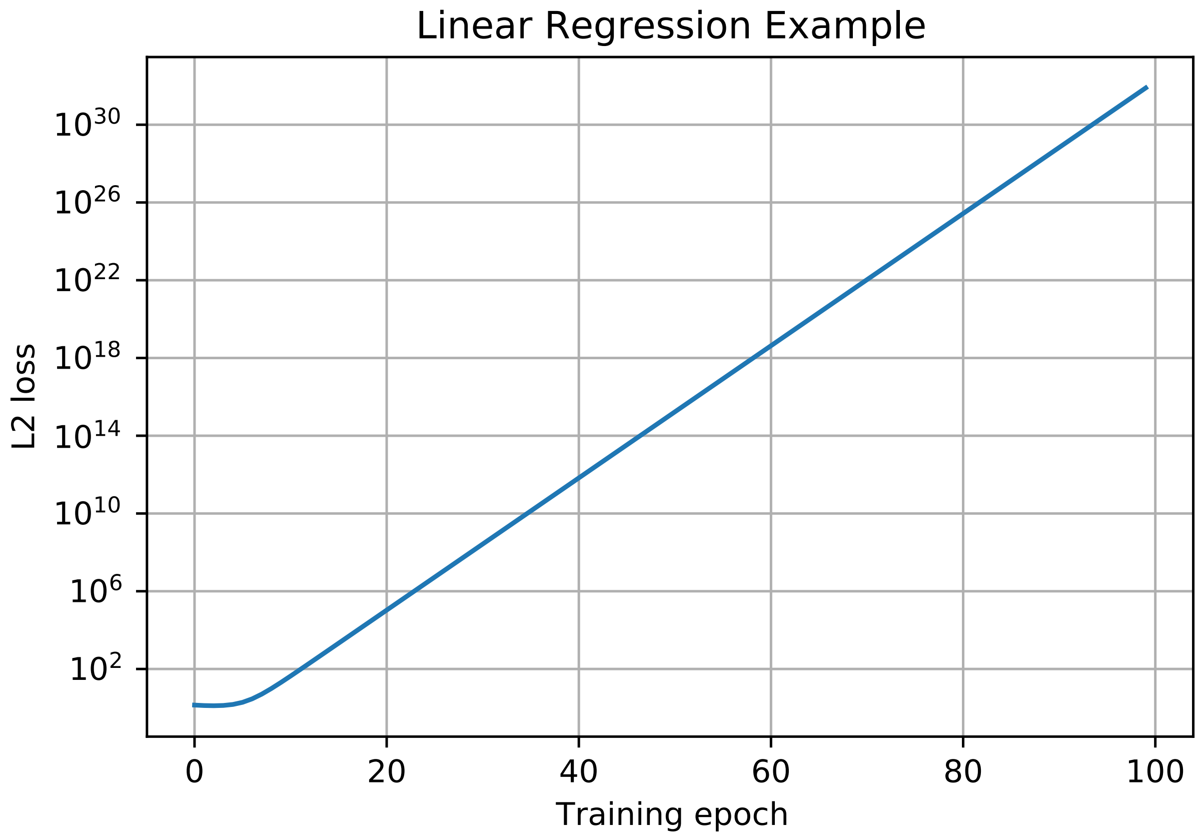
$$\hat{c} = 0.0226...$$

Implementation: TensorFlow 2.2 (using V1 compatibility)
Jupyter Notebook

# GRADIENT DESCENT: EXAMPLE

- Returning to the $y = mx + c$ example, we can optimise this using the Gradient Descent algorithm.





Use N=100 and $\alpha = 0.01$ with the TensorFlow GradientDescent optimiser to obtain:

$$\widehat{m} = -3.5 \times 10^{14}$$

$$\hat{c} = -7.08 \times 10^{14}$$

The minimiser fails; too large a learning rate is being used.

Implementation: TensorFlow 2.2 (using V1 compatibility)
Jupyter Notebook

# GRADIENT DESCENT: REFLECTION

- The examples shown illustrate problems with parabolic minima.
- With selection of an appropriate learning rate, $\alpha$, to fix the step size, we can guarantee convergence to a sensible minimum in some number of steps.
- Translating the distribution to a fixed scale, then we can predict how many steps it will take to converge to the minimum from some distance away from it for a given $\alpha$.
- If the problem hyperspace is not parabolic, this becomes more complicated, and there is no guarantee that we converge to the minimum.
- Modern machine learning algorithms use more refined variants on this method.

# OPTIMISATION

## ADAM OPTIMISER

# ADAM OPTIMISER

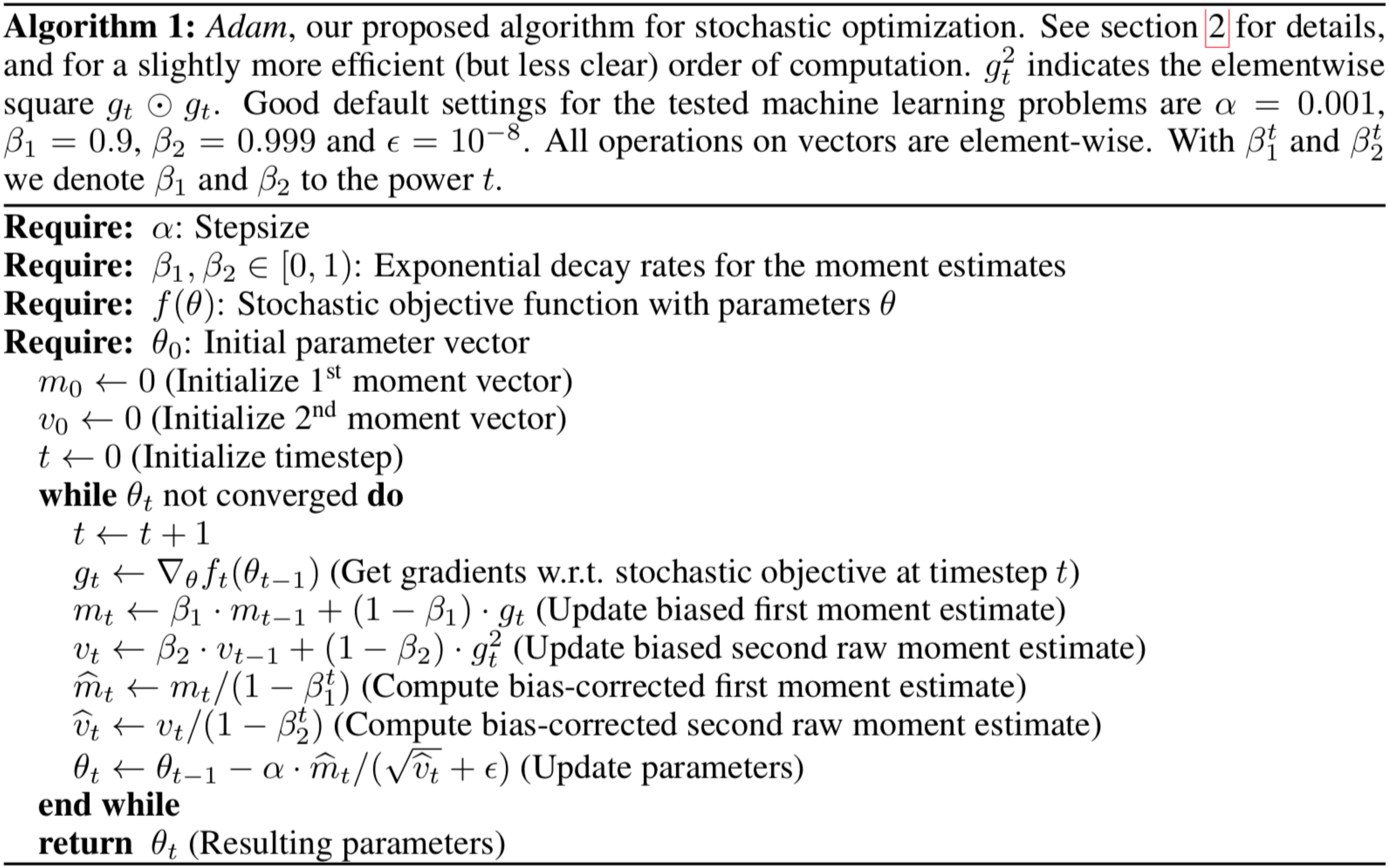
(ADAM: ADAptive Moment estimation based on gradient descent)

- This is a stochastic gradient descent algorithm.
- Consider a model $f(\theta)$ that is differentiable with respect to the HPs $\theta$ so that:
  - $t$ is the training epoch.
  - the gradient $g_t = \nabla f_t(\theta_{t-1})$ can be computed.
  - $m_t$ $(v_t)$ are biased values of the first (second) moment.
  - $\widehat{m}_t$ $(\hat{v}_t)$ are bias corrected estimator of the moments.
  - Some initial guess for the HP is taken: $\theta_0$, and the HPs for a given epoch are denoted by $\theta_t$.
  - $\alpha$ is the step size (i.e. learning rate).
  - $\beta_1$and $\beta_2$ are exponential decay rates of moments.

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

# ADAM OPTIMISER

(ADAM: 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 $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: 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 $\theta_t$ 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 $\theta_t$ (Resulting parameters)
```

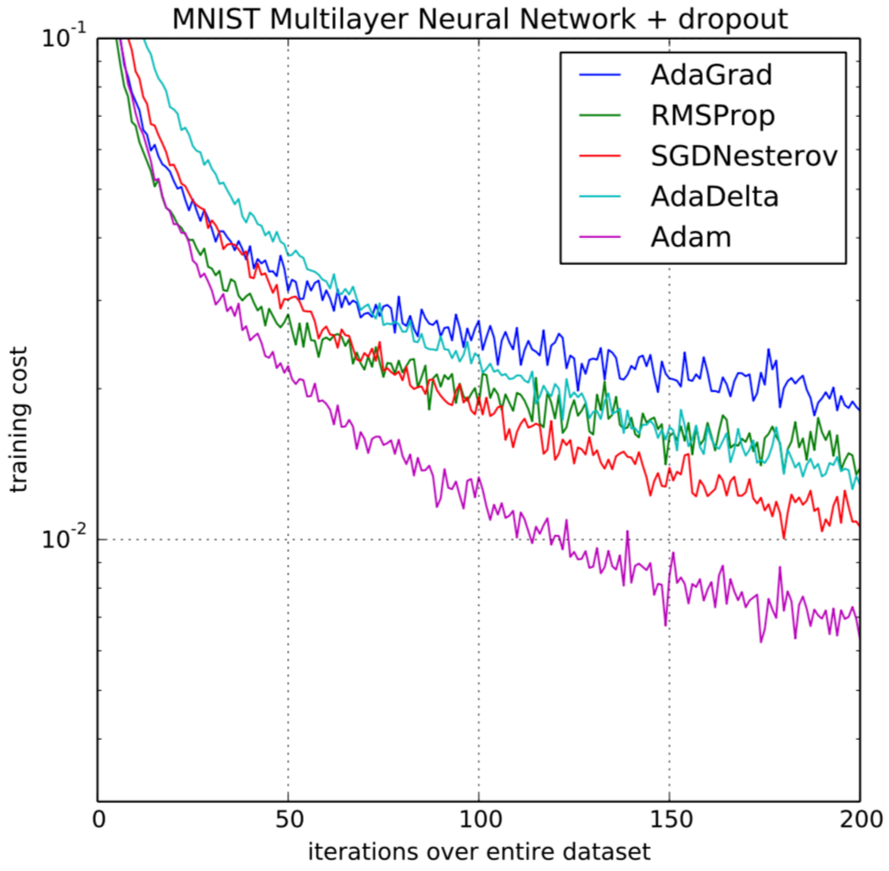
.

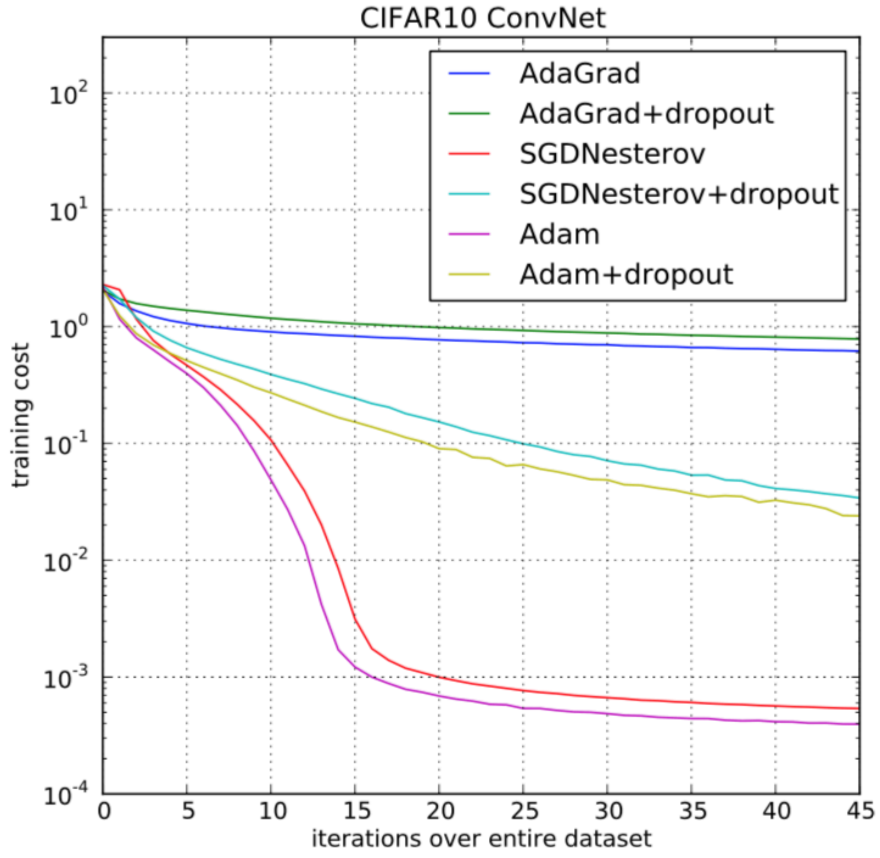
Kingma and Ba, Proc. 3rd Int Conf. Learning Representations, San Diego, 2015

# ADAM OPTIMISER

(ADAM: ADAptive Moment estimation based on gradient descent)

- 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 loss, and lower overall loss obtained vs other algorithms benchmarked at the time.

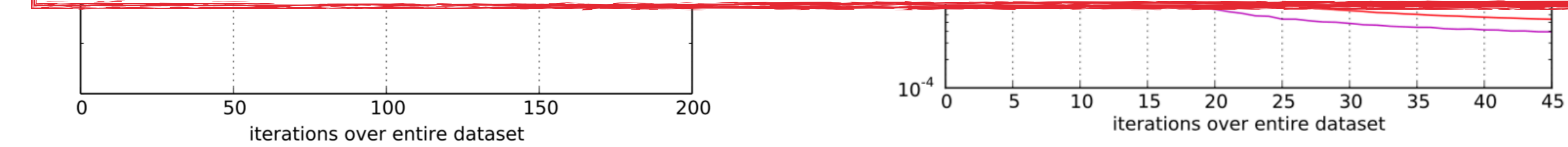
Kingma and Ba, Proc. 3rd Int Conf. Learning Representations, San Diego, 2015

# ADAM OPTIMISER

(ADAM: ADAptive Moment estimation based on gradient descent)

- Benchmarking performance using MNIST and CFAR10 data indicates that ith othe

MNIST and CFAR10 are standard benchmark data sets in CS (see appendix). MNIST is a set of handwritten numbers 0 through 9; CFAR10(0) is an image classification data set (cars, boats etc).



- Faster drop off in loss, and lower overall loss obtained vs other algorithms benchmarked at the time.

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

MULTIPLE MINIMA

MORE ON LOSS FUNCTIONS

# OPTIMISATION

## MISCELLANEOUS

# GRADIENT DESCENT:MULTIPLE MINIMA

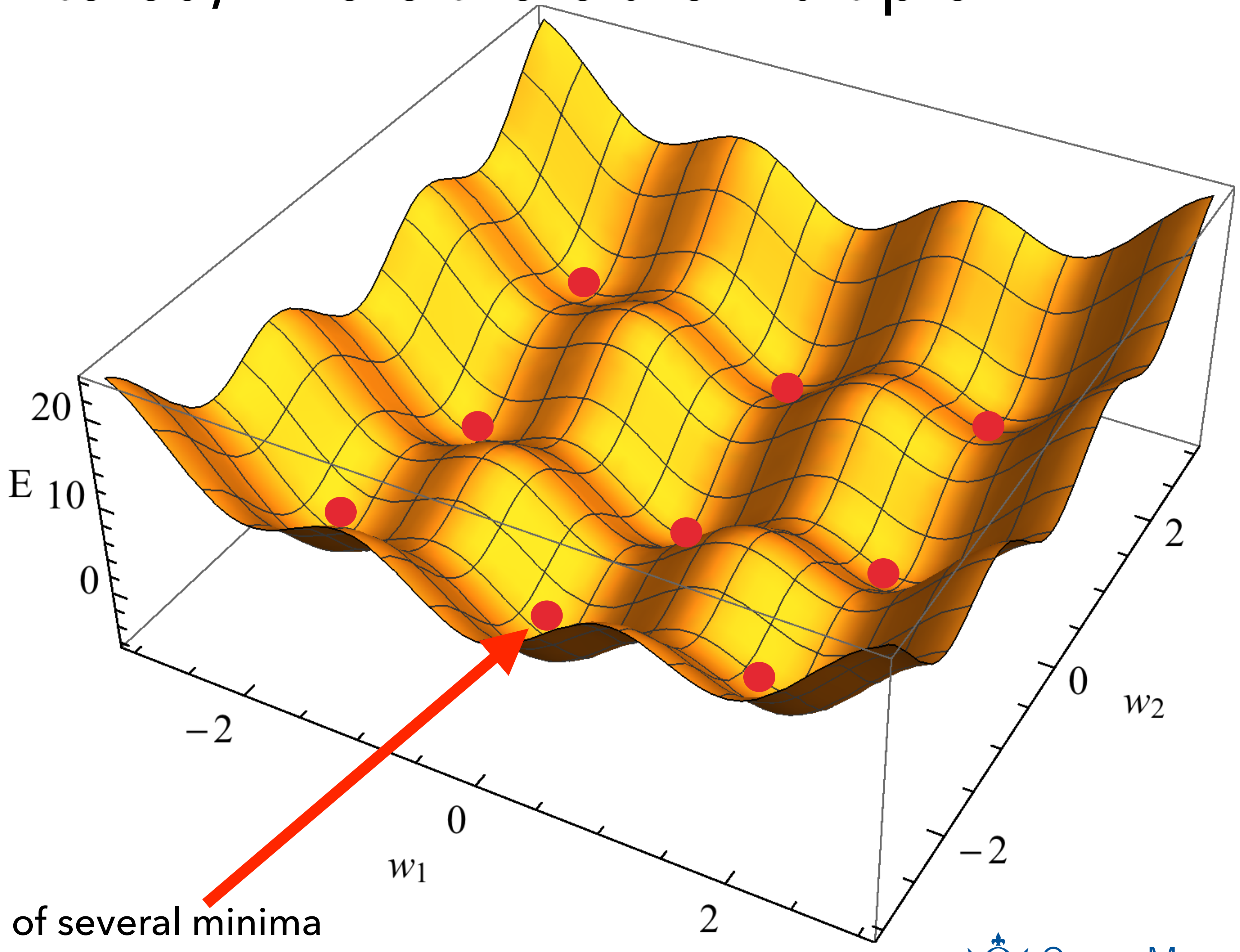
- Often more complication hyperspace optimisation problems are encountered, where there are multiple minima.

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?



One of several minima

# GRADIENT DESCENT:MULTIPLE MINIMA

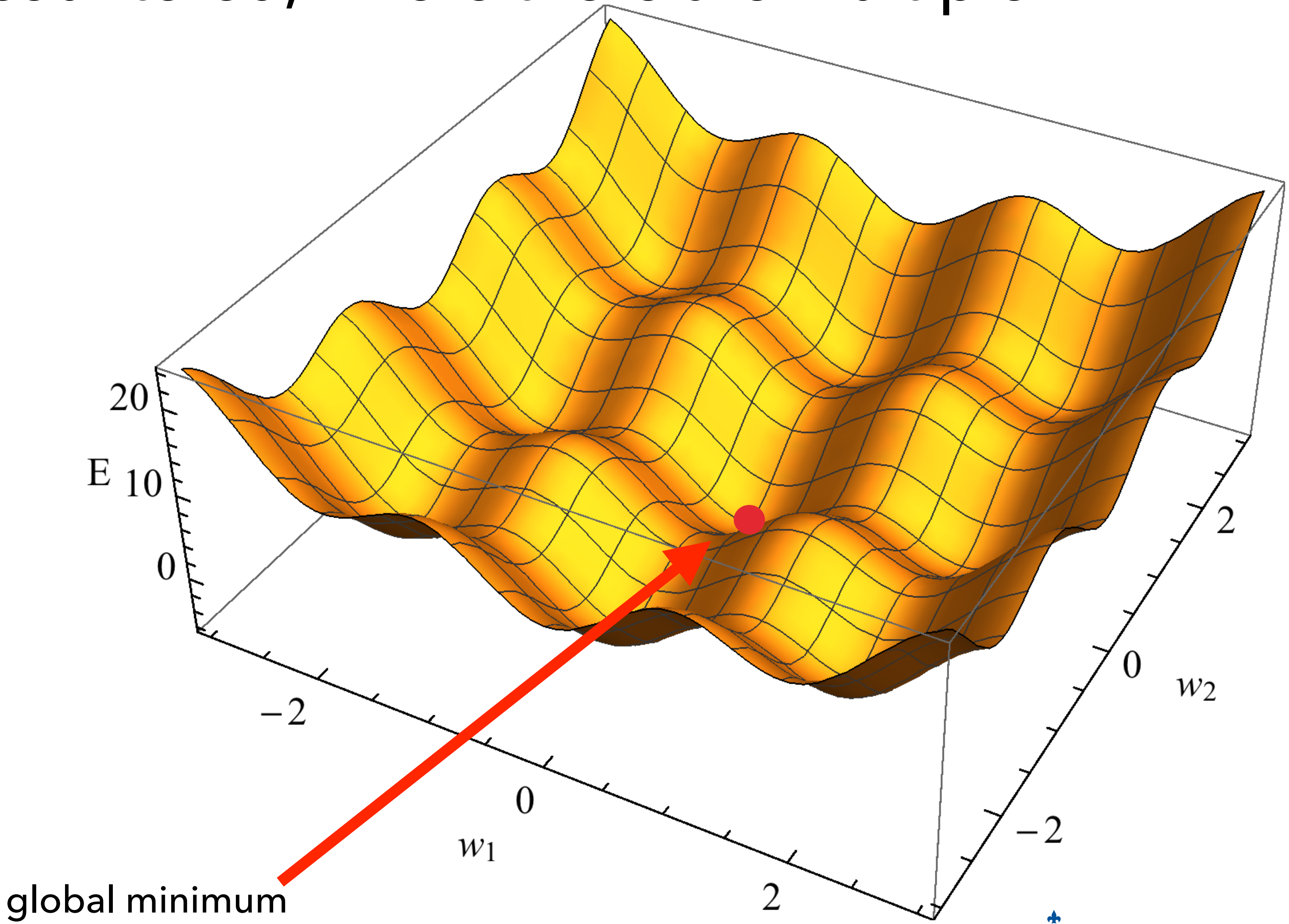
- Often more complication hyperspace optimisation problems are encountered, where there are multiple minima.

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?



# OPTIMISATION: LOSS FUNCTIONS

- There are many types of loss function other than the $L_2$ loss
  - $L_1$ norm loss:

$$L_1 = \sum_{i=1}^{N_{examples}} \left| t_i - \hat{y}(\hat{\theta}) \right|$$

- The mean square error (MSE) loss function:

$$L = \frac{1}{N_{examples}} L_2 = \frac{1}{N_{examples}} \sum_{i=1}^{N_{examples}} \left[ t_i - \hat{y}(\hat{\theta}) \right]^2$$

- Cross entropy

$$L = - \sum_{i=1}^{N_{examples}} \widehat{y}(x_i) \ln t_i$$

The cross entropy can be thought of as the negative log likelihood function for the data $t_i$ under the model $\widehat{y}$

SUPERVISED LEARNING

OVERTRAINING

WEIGHT REGULARISATION

CROSS VALIDATION

DROPOUT

# TRAINING

[1] G. Hinton et al. arXiv:1207.0580

# TRAINING

## SUPERVISED LEARNING

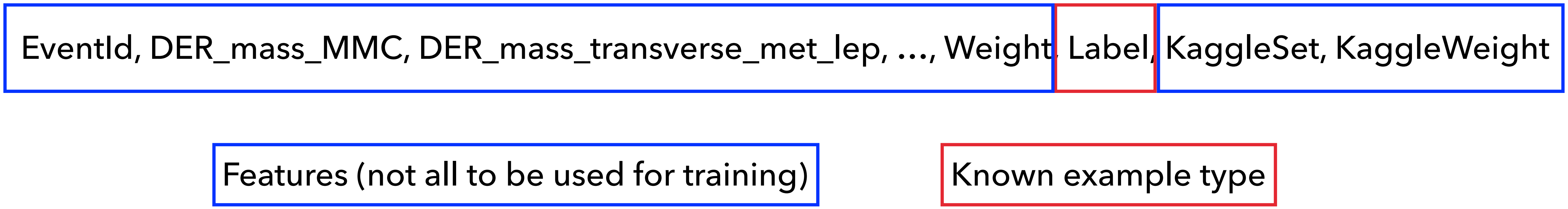
# SUPERVISED LEARNING

- For supervised learning the loss function depends on both model parameters and a known target value $t_i$ for a given example.
  - e.g. the L₂ loss: $L_2 = \sum_{i=1}^{N_{examples}} \left[t_i - \hat{y}(\hat{\theta})\right]^2.$
- This requires (at a minimum) a sample of data with the input feature space of interest, and for each example in the data, the known target output value.
- The loss function is optimised using the data, to obtain a model.
- Unlike fitting however we don't care about the uncertainty on the model parameter estimates, only on the nominal values obtained, $\widehat{\theta}$.
- Due to the complexity of models, it is generally not possible to understand if the optimisation converged to a sensible solution by inspecting marginalised projections of the model on the data.

[1] G. Hinton et al. arXiv:1207.0580

# SUPERVISED LEARNING

- e.g. the Kaggle Higgs data sample:



EventId, DER_mass_MMC, DER_mass_transverse_met_lep, …, Weight, Label, KaggleSet, KaggleWeight

Features (not all to be used for training)

Known example type

- The KaggleSet column in the CVS file for this data allows the user to understand if this is to be used for training or testing, or some other means (e.g. Kaggle Score Board evaluation).

- The Label column, is the known label; this defines the $t_i$ used in the loss function.

- I use this challenge data as an assignment for undergraduate lectures (pdf, zip). N.B. the zip file is 189Mb.

[1] G. Hinton et al. arXiv:1207.0580

WEIGHT REGULARISATION

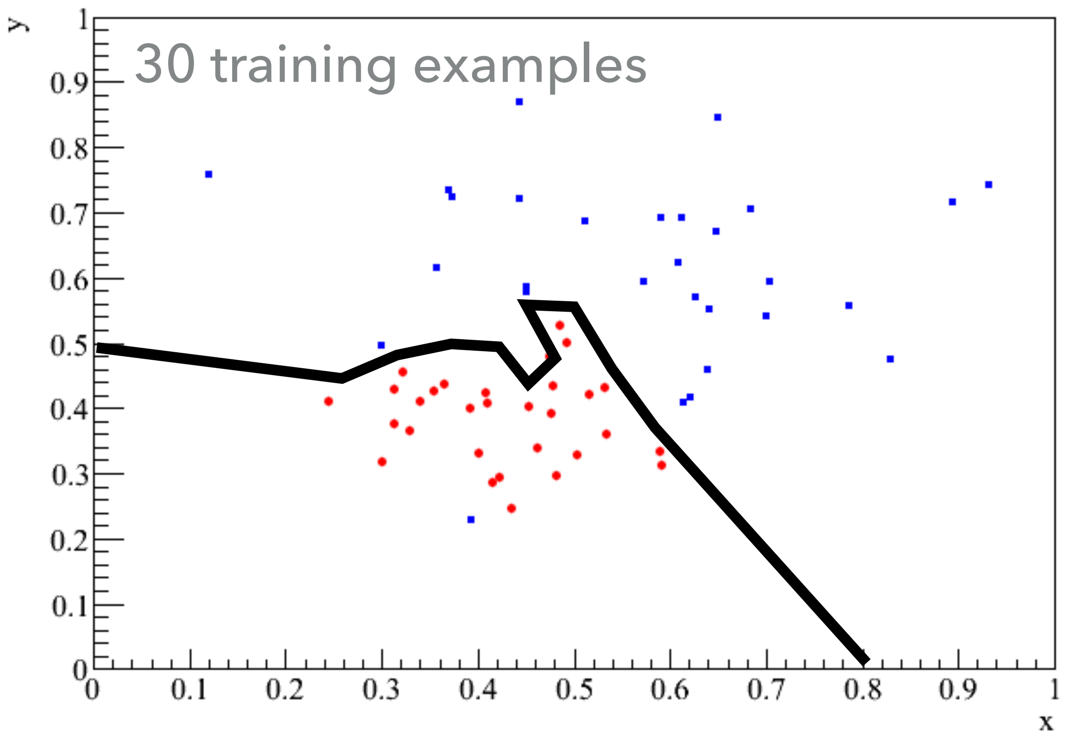
CROSS VALIDATION

# TRAINING
# OVERTRAINING

# OVERTRAINING

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

- Simple illustration of the problem:



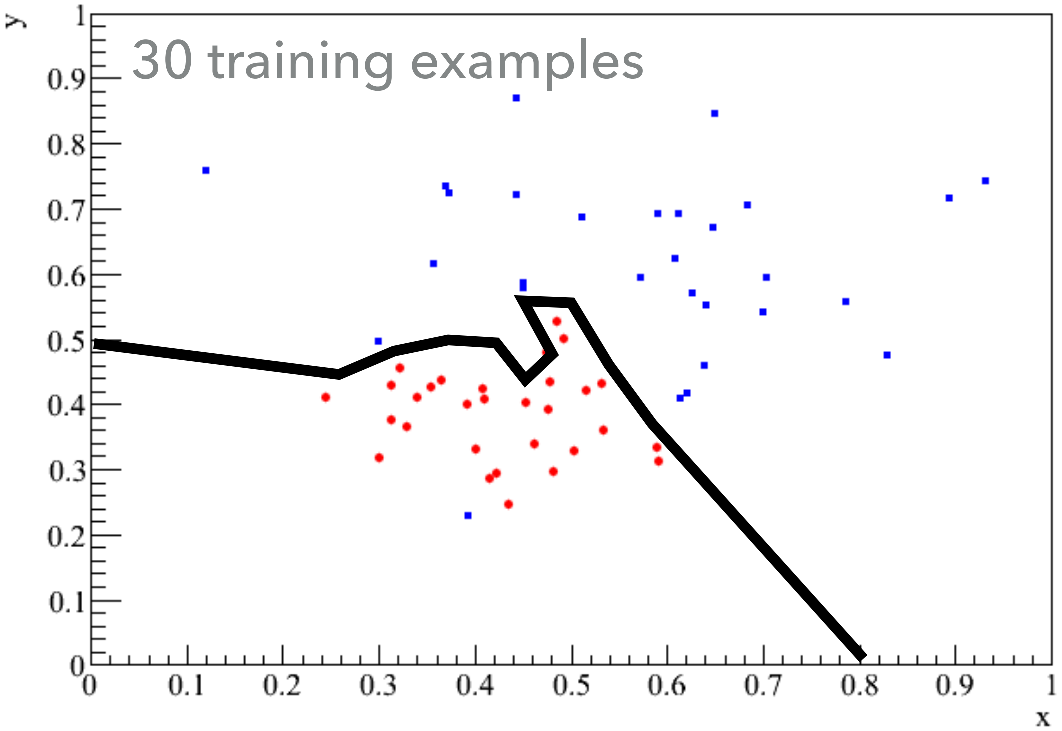
The decision boundary selected here does a good job of separating the red and blue dots.

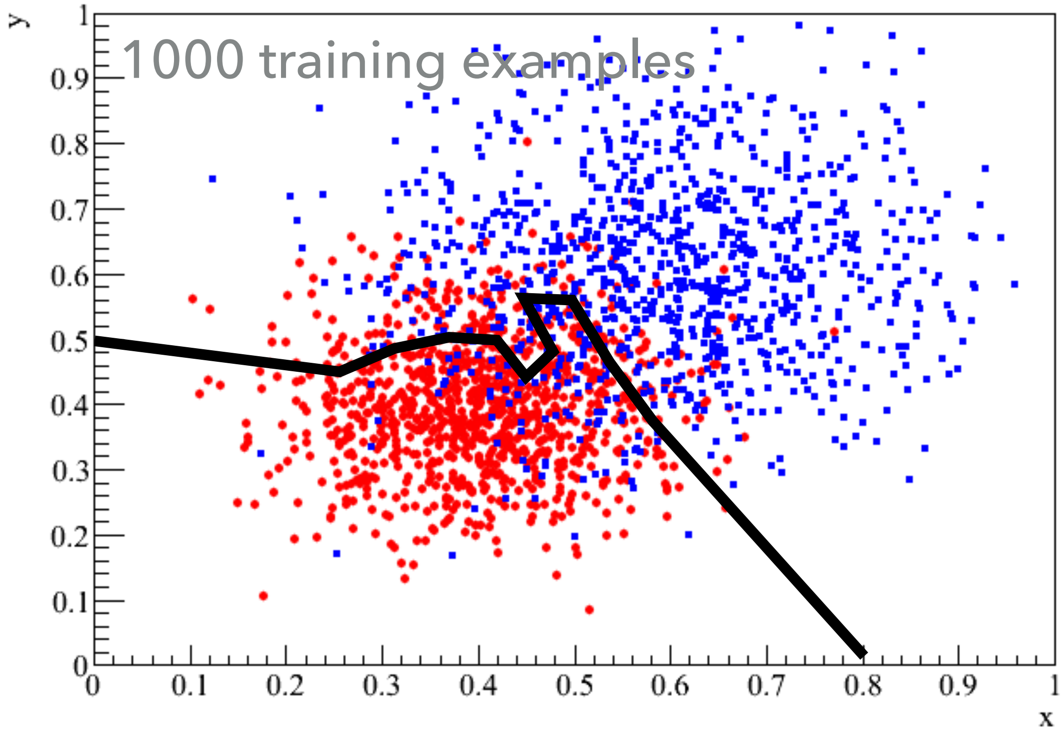
Boundaries like this can be obtained by training models on limited data samples. The accuracies can be impressive.

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

# OVERTRAINING

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

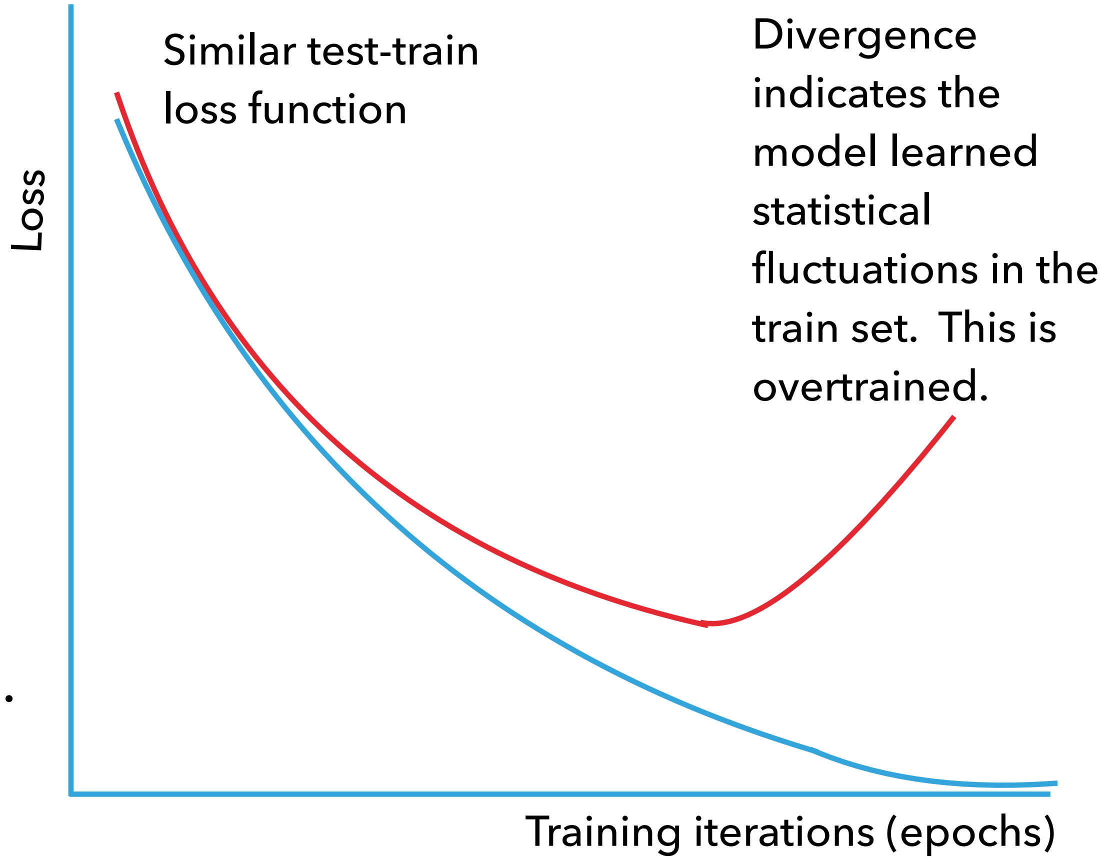




Increasing to 1000 training examples we can see the boundary doesn't do as well. This illustrates the kind of problem encountered when we overfit HPs of a model.

# OVERTRAINING

- 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 test sample.
    - Use the loss evaluated for the training and test samples to check to see if the HPs are over trained.
        - If both samples have similar loss then the model response function is similar on two statistically independent samples.
    - Note: 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.
- Some also recommend a third validation set of data in order to provide a completely independent verification of algorithm performance.



Similar test-train loss function

Divergence indicates the model learned statistical fluctuations in the train set. This is overtrained.

# OVERTRAINING

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

# OVERTRAINING: 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

# OVERTRAINING: WEIGHT REGULARISATION

- For example we can consider extending an MSE loss function to allow for weight regularisation. The MSE loss 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

# OVERTRAINING: CROSS VALIDATION

- A well trained model will provide robust predictions, irrespective of the examples presented to it.
  - The variance of model predictions will be small.
  - The model predictions may be systematically biased independently of this.
- One can divide the training set up into k-folds, and then perform k trainings; each one leaving a single fold out.
- The amount of data used in a fold will depend (generally the more data the better.
- The ensemble of predictions indicates the variance on the model output.

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).

# OVERTRAINING: CROSS VALIDATION

- The use of cross validation to determine the spread of model predictions, and any systematic bias on prediction can be useful.
- e.g. in a physics context: Consider a new particle search at the Large Hadron Collider.
  - Using an overtrained model to suppress background and enhance signal will result in a lack of experimenter understanding as to how the expected and observed limits on the new particle relate to each other.
  - This could result in a false discovery of new physics.
  - It could result in missing out on a discovery of new physics.
  - Some people say it is not wrong to use an over trained model - I argue strongly that it is not scientifically correct to use an over trained model, unless the variance on that model, and hence the implications are well understood.

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).

# TRAINING

## DROPOUT

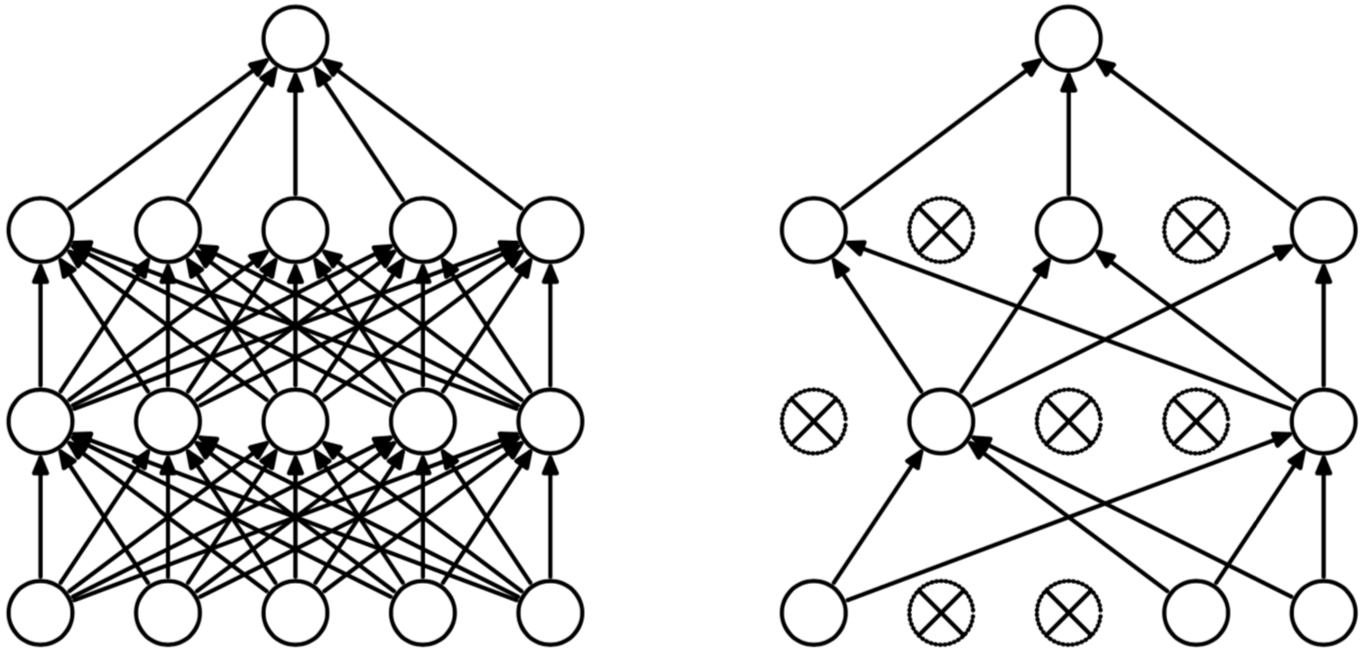
# DROPOUT

- A problem with deep networks is the number of HPs that need to be determined.
- This leads to a requirement for very large data sets to avoid overfitting (or fine-tuning).
- HPs can also learn to "co-adapt" in the training process.
  - co-adapt means that as one parameter is changed, another in the network can be modified in a correlated way to offset that change.
  - This kind of behaviour intentionally exists in some algorithms (Sequential Minimal Optimisation for Support Vector machines, where pairs of HPs are changed to conserve an overall zero sum); but is generally unwelcome behaviour.

G. Hinton et al. arXiv:1207.0580, Srivastava et al., J. Machine Learning Research 15 (2014) 1929-1958

# OVER FITTING: DROPOUT

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



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

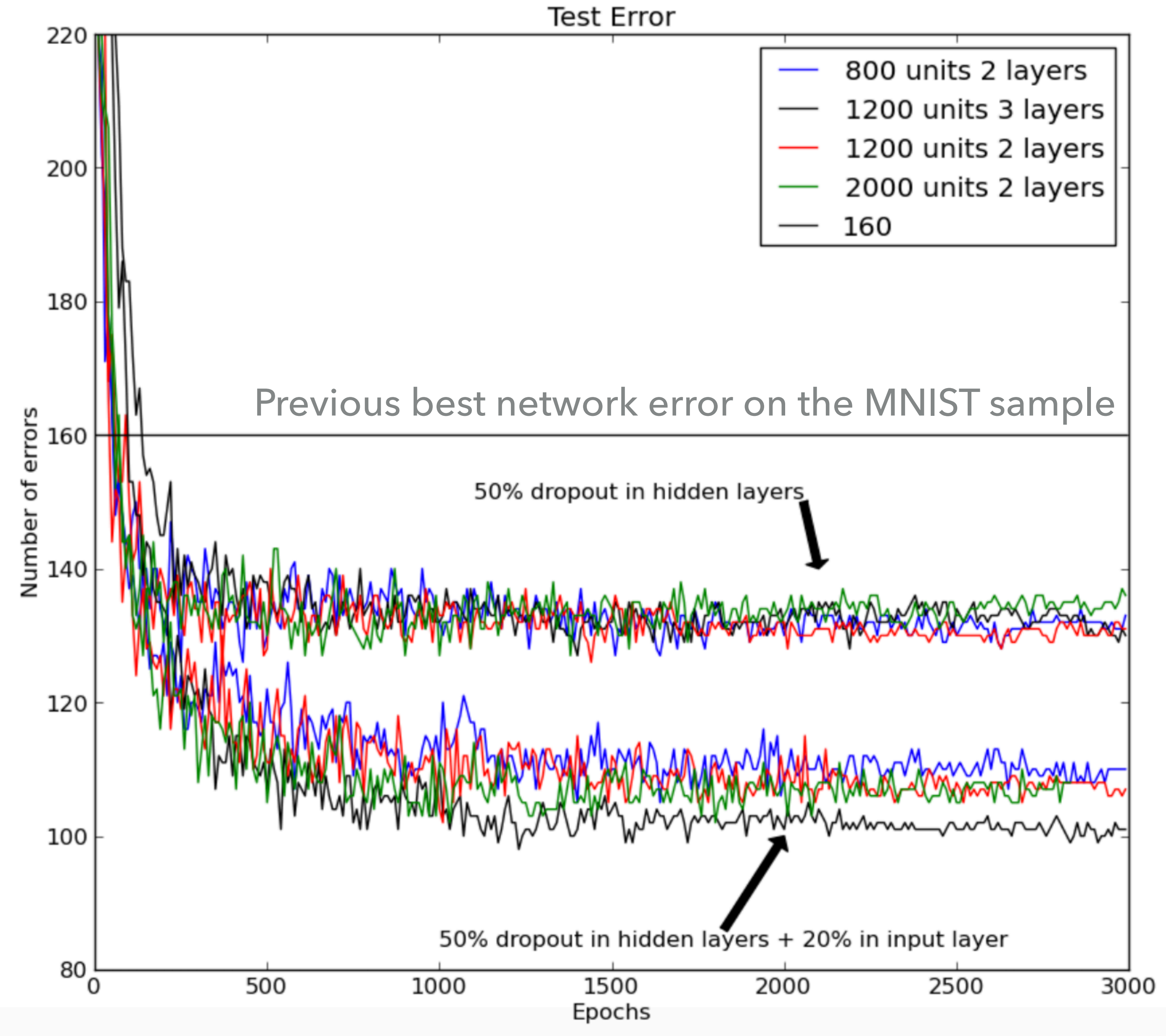
Dropout is used only during training.

The full model is used for making predictions.

- That way the whole model will be effectively trained on a sub-sample of the data with the aim of limiting the ability to learn statistical fluctuations in the data, and mitigating the co-adaption issue.
- This does not remove the possibility that a model is overtrained, as with the previous discussion HP generalisation is promoted by using this method.

G. Hinton et al. arXiv:1207.0580, Srivastava et al., J. Machine Learning Research 15 (2014) 1929-1958

# DROPOUT

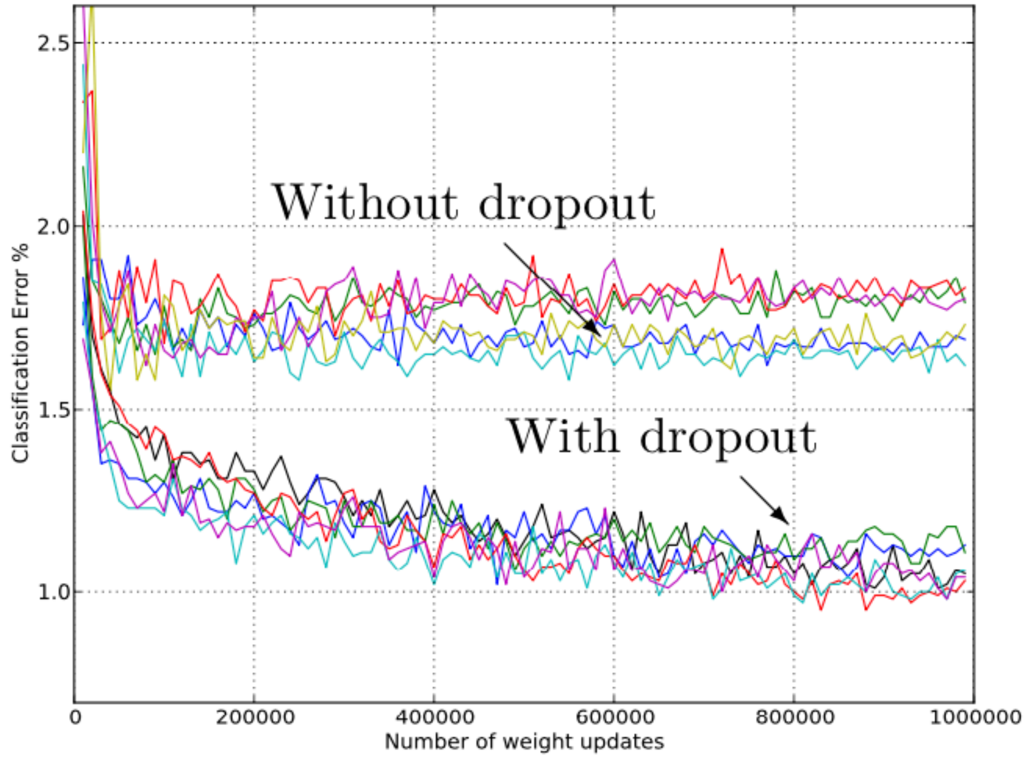


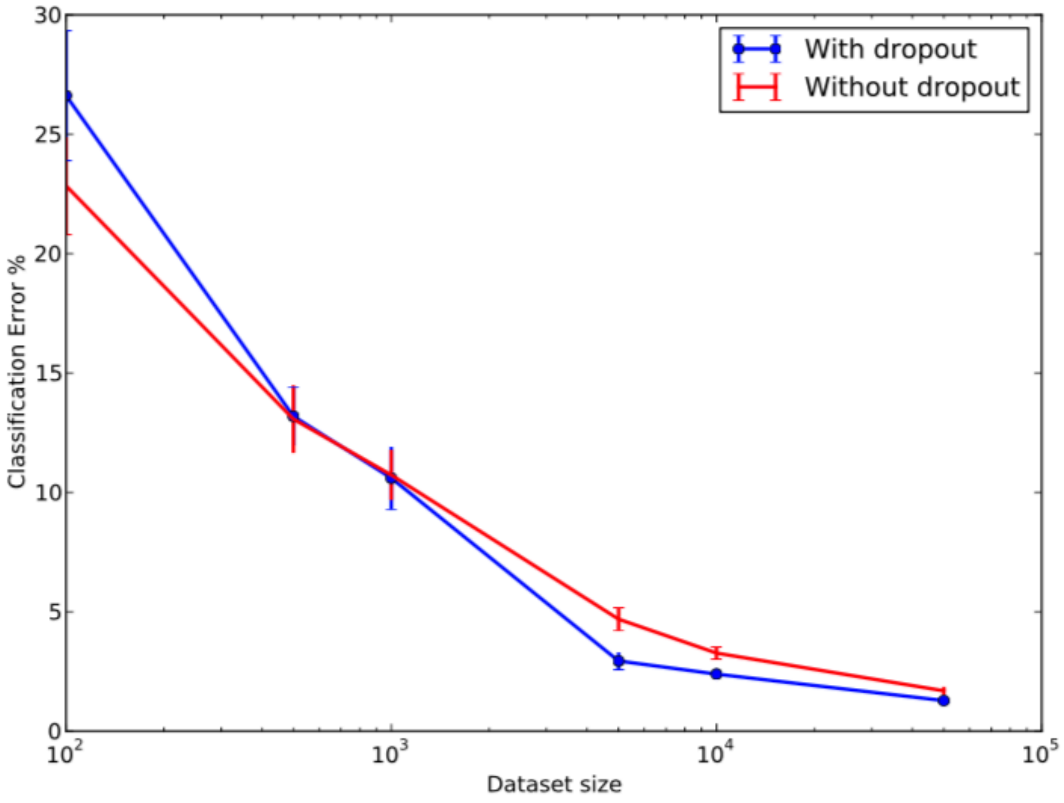
- Example from Hinton et al. for an MNIST sample.
- Using 50% drop out on the hidden layers gives an improvement over the previous best network architecture.
- Adding 20% drop out in the input layer provides further improvement in error.

G. Hinton et al. arXiv:1207.0580, 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 for this technique,

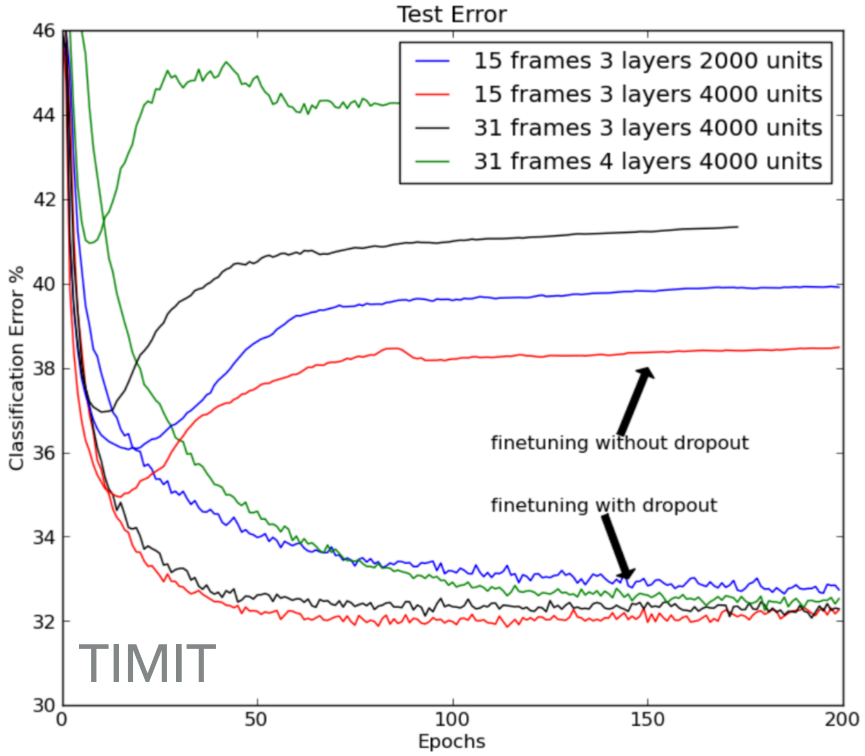


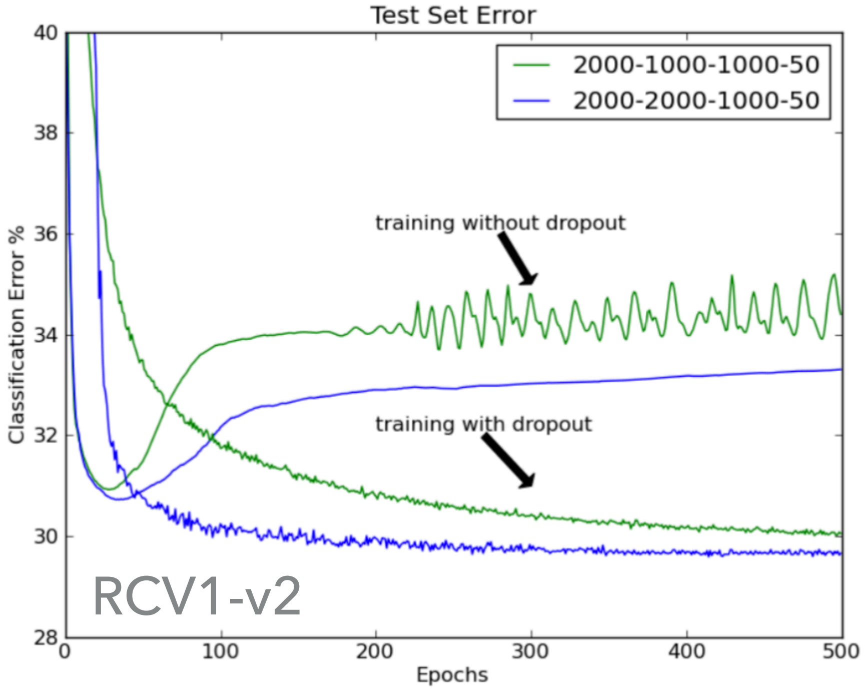


- Dropout can be detrimental for small training samples, however in general the results show that dropout is beneficial.
- For deep networks or typical training samples O(500) examples or more this technique is expected to be beneficial.
- For algorithms with very large training data sets, the benefits are less clear.

G. Hinton et al. arXiv:1207.0580, Srivastava et al., J. Machine Learning Research 15 (2014) 1929-1958

# DROPOUT





- Example from Hinton et al. for a voice recognition sample of data (TIMIT, speech recognition data).
- Illustrates the classification error as a function of epoch with and without dropout.
- Similar results were obtained by Hinton and his team with the Reuters newsfeed data set (RCV1-v2).

G. Hinton et al. arXiv:1207.0580, Srivastava et al., J. Machine Learning Research 15 (2014) 1929-1958

DECISION TREES

BOOSTING

ADABOOST.M1

RANDOM FORESTS

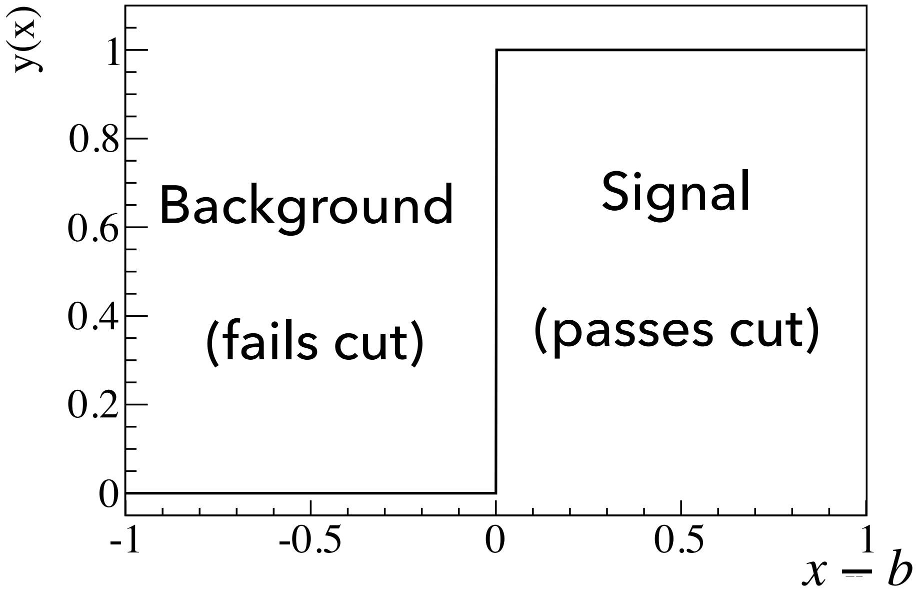
# DECISION TREES

# DECISION TREES

- The cut based [see Data Wrangling] and linear discriminant analysis methods are useful but limited.
- The underlying concepts of applying Heaviside function constraints on data selection and on the use of a decision boundary definition (a plane in hyperspace) of the form of the dot product $\alpha^T x + \beta$ can be applied in more complicated algorithms.
- Here we consider extension to the concept of rectangular cuts to decision trees as a machine learning algorithm.
  - We will have to introduce the concepts of classification and regression; and methods to mitigate mis-classification of data.
  - The issue of overtraining is something we discussed earlier regarding optimisation.

# DECISION TREES

- Consider a data sample with an N dimensional input feature space X.
- X can be populated by examples from two or more different species of event (also called classes, categories or types).
- Consider the two types and call them signal and background, respectively*.
- We can use a Heaviside function to divide the data into two parts:
    - We can use this to distinguish between regions populated signal and background:



For an arbitrary cut position in x we can modify the Heaviside function

$$H(x) = \frac{1}{2}(1 + sign(x - b))$$

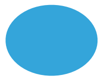
where b is the offset (bias) from zero.

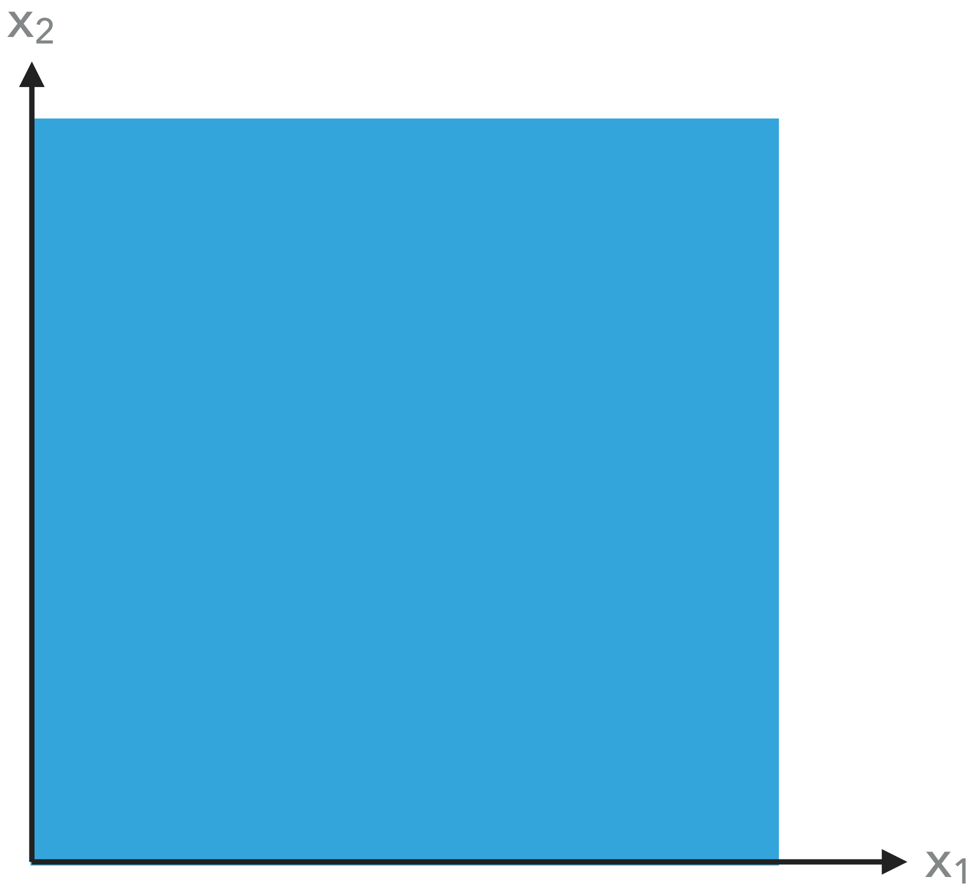
*Can generalise the problem to an arbitrary number of types.

# DECISION TREES

- Decision trees divide the data feature space into a set of hypercubes that are classified as signal (+1) or background (-1) like.
  - Each region can be fitted with a constant to represent the data in that region.
  - We can recursively continue to sub-divide the data until some minimum number of examples are left in each sub-division.

Can describe the data as the root node.



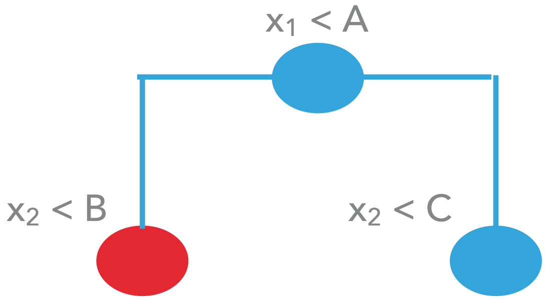


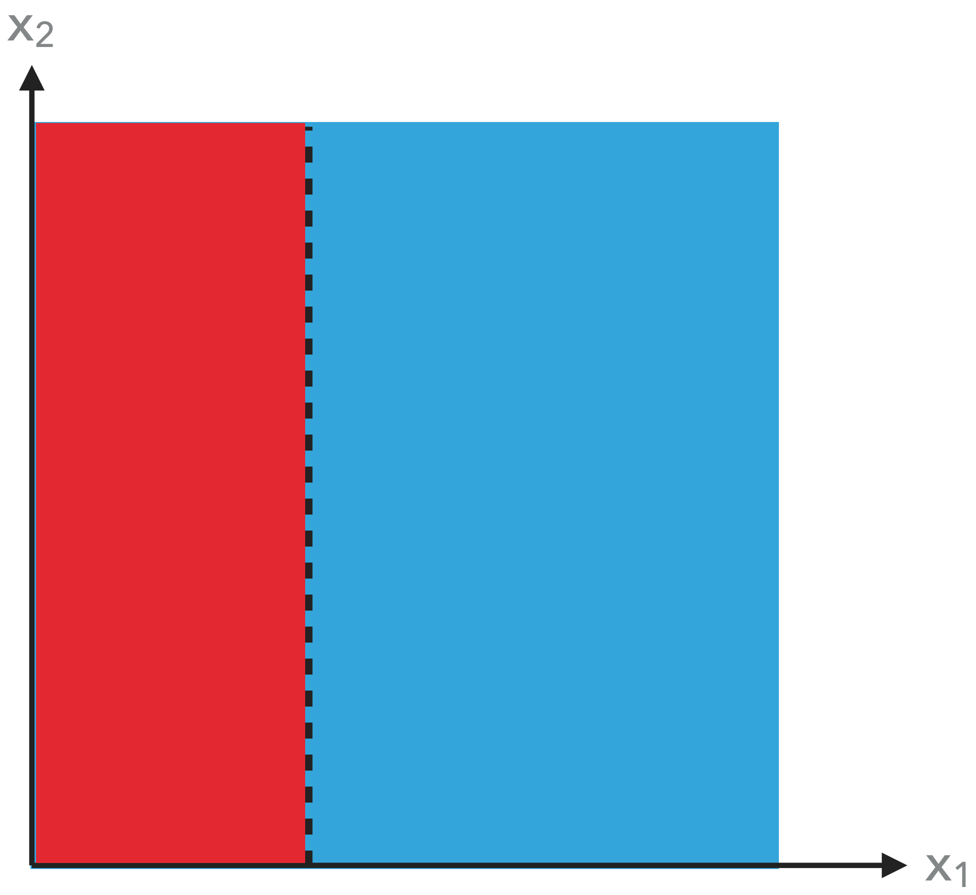
Example feature space described by $X=\{x_1, x_2\}$

# DECISION TREES

- Decision trees divide the data feature space into a set of hypercubes that are classified as signal (+1) or background (-1) like.
  - Each region can be fitted with a constant to represent the data in that region.
  - We can recursively continue to sub-divide the data until some minimum number of examples are left in each sub-division.

The data get divided into two partitions.



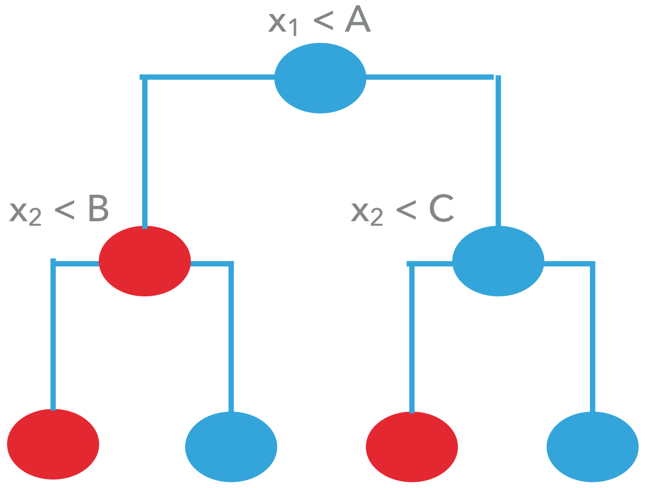


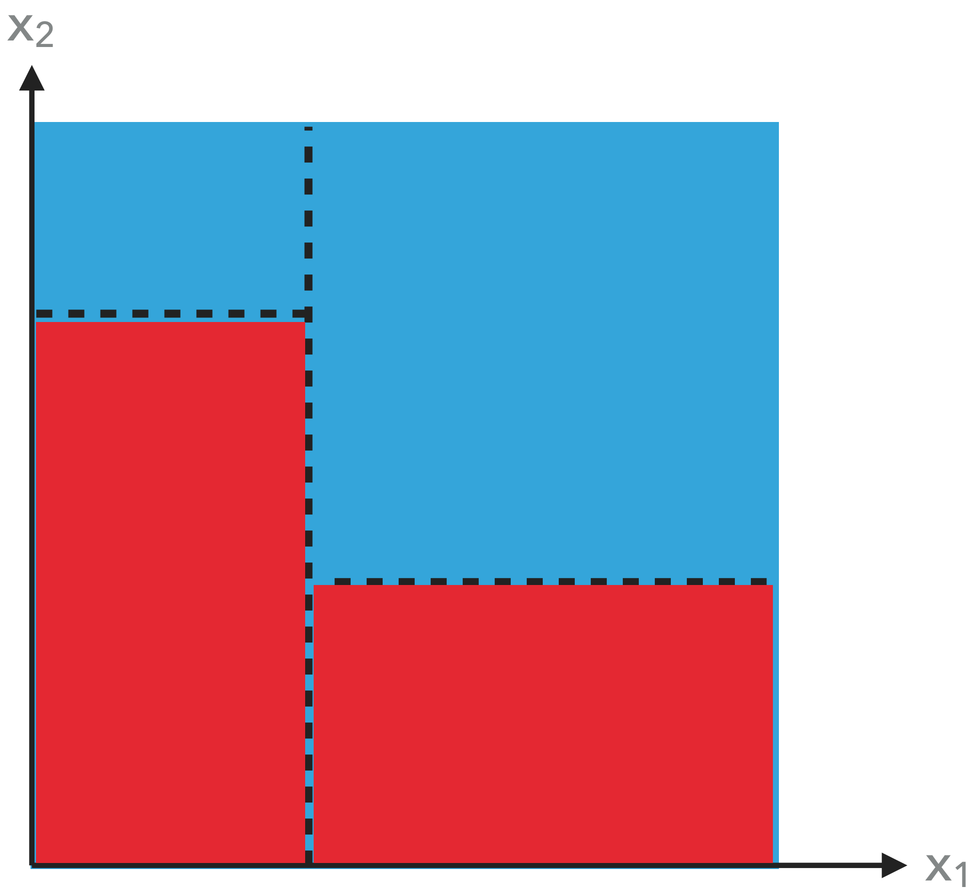
Cut on the feature space to separate the data into two different regions.

# DECISION TREES

- Decision trees divide the data feature space into a set of hypercubes that are classified as signal (+1) or background (-1) like.
    - Each region can be fitted with a constant to represent the data in that region.
    - We can recursively continue to sub-divide the data until some minimum number of examples are left in each sub-division.

Divide the data again



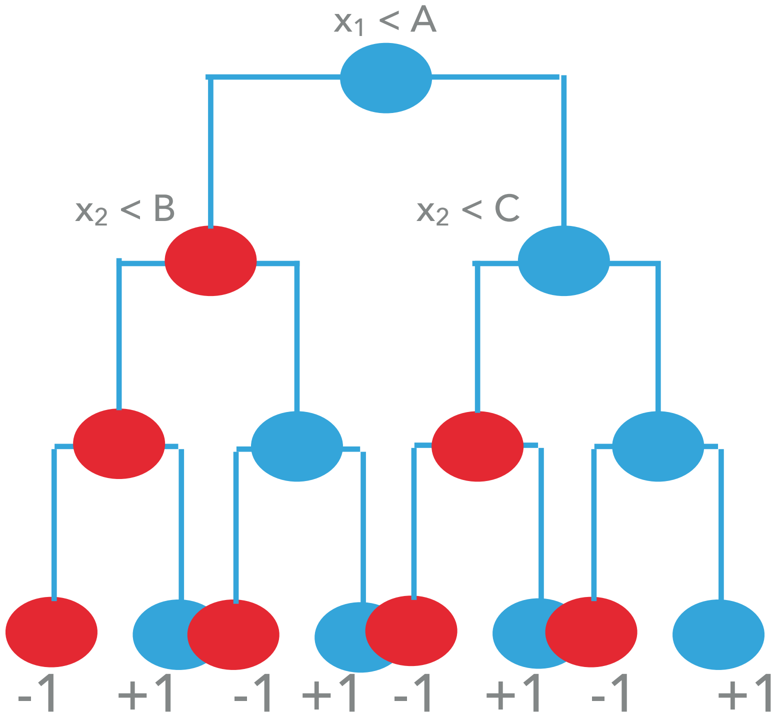


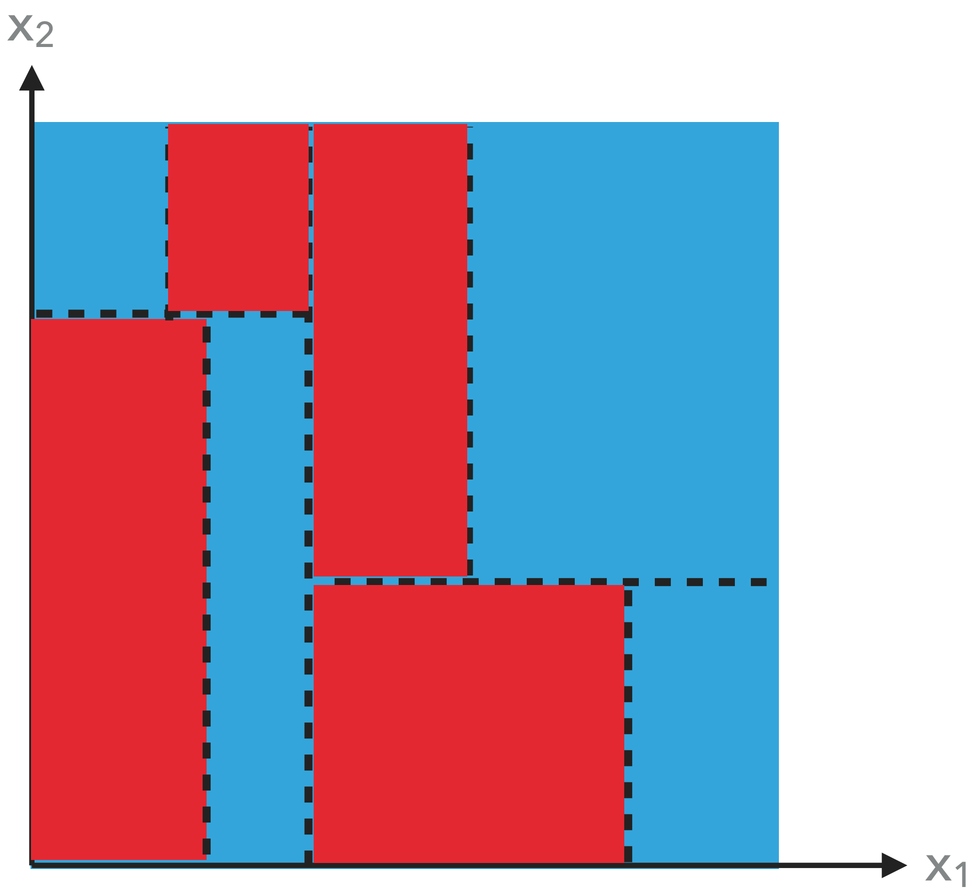
The feature space gets further sub-divided.

# DECISION TREES

- Decision trees divide the data feature space into a set of hypercubes that are classified as signal (+1) or background (-1) like.
    - Each region can be fitted with a constant to represent the data in that region.
    - We can recursively continue to sub-divide the data until some minimum number of examples are left in each sub-division.

Divide the data again

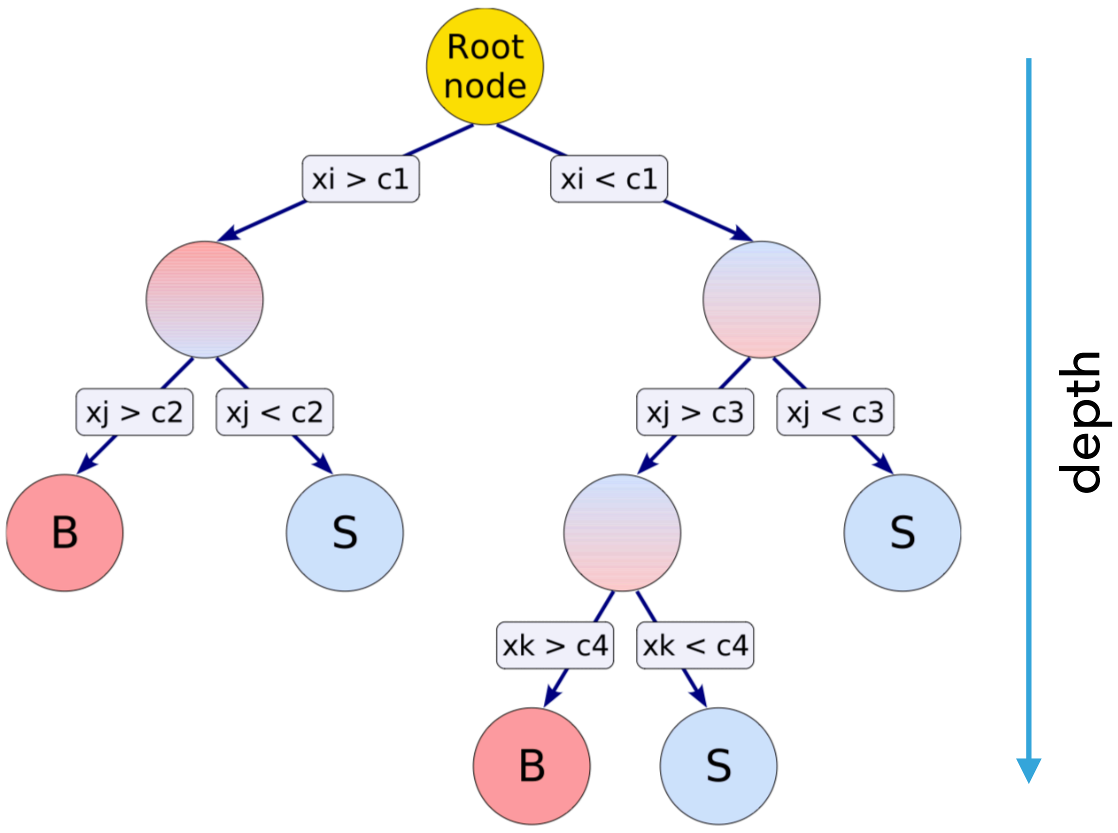




The feature space gets further sub-divided (again).

# DECISION TREES

- The set of rectangular cuts applied to the data allow us to build a tree from the root note.
- We can impose limits on:
  - Tree depth (how many divisions are performed).
  - Node size (how many examples per partition).
- Trees can be extended to more than 2 categories.
- They lend themselves to classifying examples or adapted to make a quantitative prediction (regression)



The decision tree output for a classification problem is

$$G(x) = +1 \text{ or } -1$$

# DECISION TREES

- Decision trees are "weak learners", as they can take input features that only weakly separate types of example and combine those features to increase the separation.
- A single tree is susceptible to overtraining, and there are various methods of reducing this; including limiting the complexity of a tree, or the limiting the minimum number of examples in each node.
- The decision tree can be extended to an oblique decision tree, in which a linear combination of the features (instead of a single feature) is used to classify examples; so the Heaviside function cut becomes a hyperplane cut.

# BOOSTING

- If a training example has been mis-classified in a training epoch, then the weight of that event can be increased for the next training epoch; so that the cost of mis-classification increases.
- The underlying aim is to take a weak learner and try and boost this into becoming a strong learner.
  - This example re-weighting technique is called boosting.
  - There are several re-weighting methods commonly used; here we discuss:
    - AdaBoost.M1 (popular variant of the Adaptive boosting method)
- Boosted Decision Trees are known as BDTs

Freund and Schapire J. Jap. Soc. AI **14** (1999) 771-780

# BOOSTING: AdaBoost.M1

- i is the ith example out of a data set with N examples.
- m is the mth training out of an ensemble of M learners to be trained.

- Step 1:
  - Assign event weights of $w_i$ = 1/N to all of the N examples.

Freund and Schapire J. Jap. Soc. AI **14** (1999) 771-780

# BOOSTING: AdaBoost.M1

- i is the i<sup>th</sup> example out of a data set with N examples.
- m is the m<sup>th</sup> training out of an ensemble of M learners to be trained.

- Step 1:
  - Assign event weights of $w_i$ = 1/N to all of the N examples.

- Step 2: for m=1 through M
  - Train the weak learner (in our case this is a BDT): $G_m(x)$.
  - Compute the error rate $\varepsilon_m$.
  - Calculate the boost factor $\beta_m = \dfrac{\varepsilon_m}{1 - \varepsilon_m}$ .
  - Update weights for misclassified examples $w_i \mapsto w_i e^{\ln(1/\beta_m)}$ .

Freund and Schapire J. Jap. Soc. AI **14** (1999) 771-780

# BOOSTING: AdaBoost.M1

- i is the ith example out of a data set with N examples.
- m is the mth training out of an ensemble of M learners to be trained.

- Step 1:
  - Assign event weights of $w_i$ = 1/N to all of the N examples.

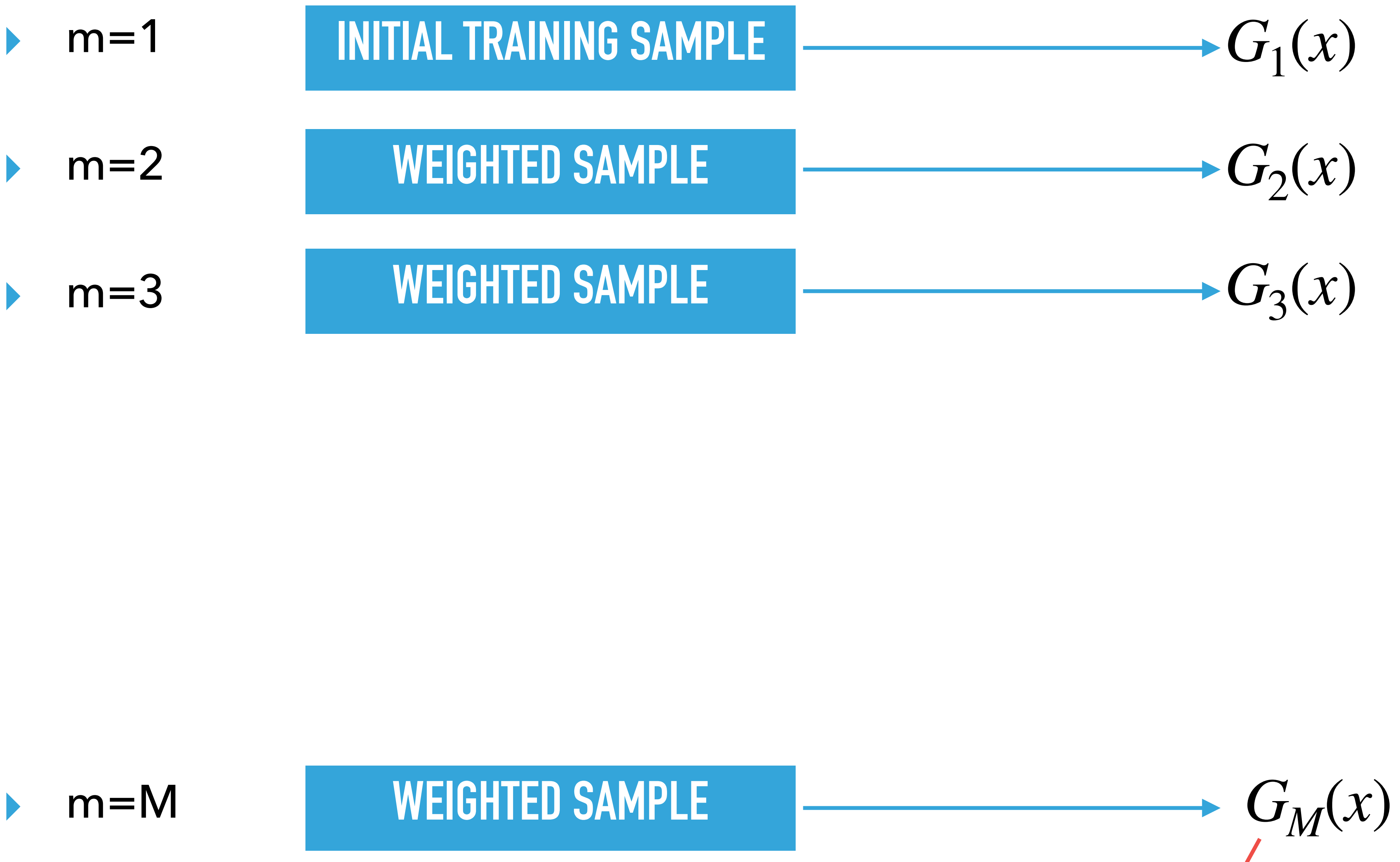
- Step 2: for m=1 through M
  - Train the weak learner (in our case this is a BDT): $G_m(x)$.
  - Compute the error rate $\varepsilon_m$.
  - Calculate the boost factor $\beta_m = \dfrac{\varepsilon_m}{1 - \varepsilon_m}$.
  - Update weights for misclassified examples $w_i \mapsto w_i e^{\ln(1/\beta_m)}$.

- Step 3:
  - Return the weighted committee: a combination of the M trees that have been learned from the data:

$$G(x) = sign\left( \sum_{m=1}^{M} ln \left[ \frac{1 - \varepsilon_m}{\varepsilon_m} \right] G_m(x) \right)$$

Freund and Schapire J. Jap. Soc. AI **14** (1999) 771-780

# BOOSTING: AdaBoost.M1



- m=1 INITIAL TRAINING SAMPLE → $G_1(x)$
- m=2 WEIGHTED SAMPLE → $G_2(x)$
- m=3 WEIGHTED SAMPLE → $G_3(x)$
- m=M WEIGHTED SAMPLE → $G_M(x)$

$$G(x) = sign\left( \sum_{m=1}^{M} ln \left[ \frac{1-\varepsilon_m}{\varepsilon_m} \right] G_m(x) \right)$$

The $G_m(x)$ are individual weak learners; each is derived from a training using the data.

The m=1 training uses the original data; all subsequent trainings use reweighted data.

The final classifier output is formed from a committee that is a weighted majority vote algorithm.

Freund and Schapire J. Jap. Soc. AI **14** (1999) 771-780

# RANDOM FORESTS

- Random Forests are constructed from an ensemble of individual trees.
  - Each tree in the ensemble uses a randomly selected subset of the feature space, and the minimum node size is usually set to 1, so the classifier prediction is almost always accurate.
- The mode (classification) or mean (regression) of the ensemble is the output of the Random Forest.
- The probability that an example $x_i$ is assigned to a given class $c$, is given by

$$P(c|x_i) = \frac{P(c|x_i)}{\sum_{l=1}^{n} P(c_l|x_i)}$$

- and the output score $g_c(x_i)$ is given by the aggregate over $t$ trees in the forest:

$$g_c(x_i) = \frac{1}{t}\sum_{j=1}^{t} P_j(c|x_i)$$

- The classification of $x_i$ is simply the class $c$ that maximises $g_c(x_i)$.

Ho, Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.