

ADRIAN BEVAN

INTRODUCTION TO MACHINE LEARNING

**PRE-SESSION SLIDES FOR THE GRADNET WORKSHOP,
JANUARY 2022**



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 Dictionary:
 - ▶ *“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.



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

- ▶ 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](https://doi.org/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.

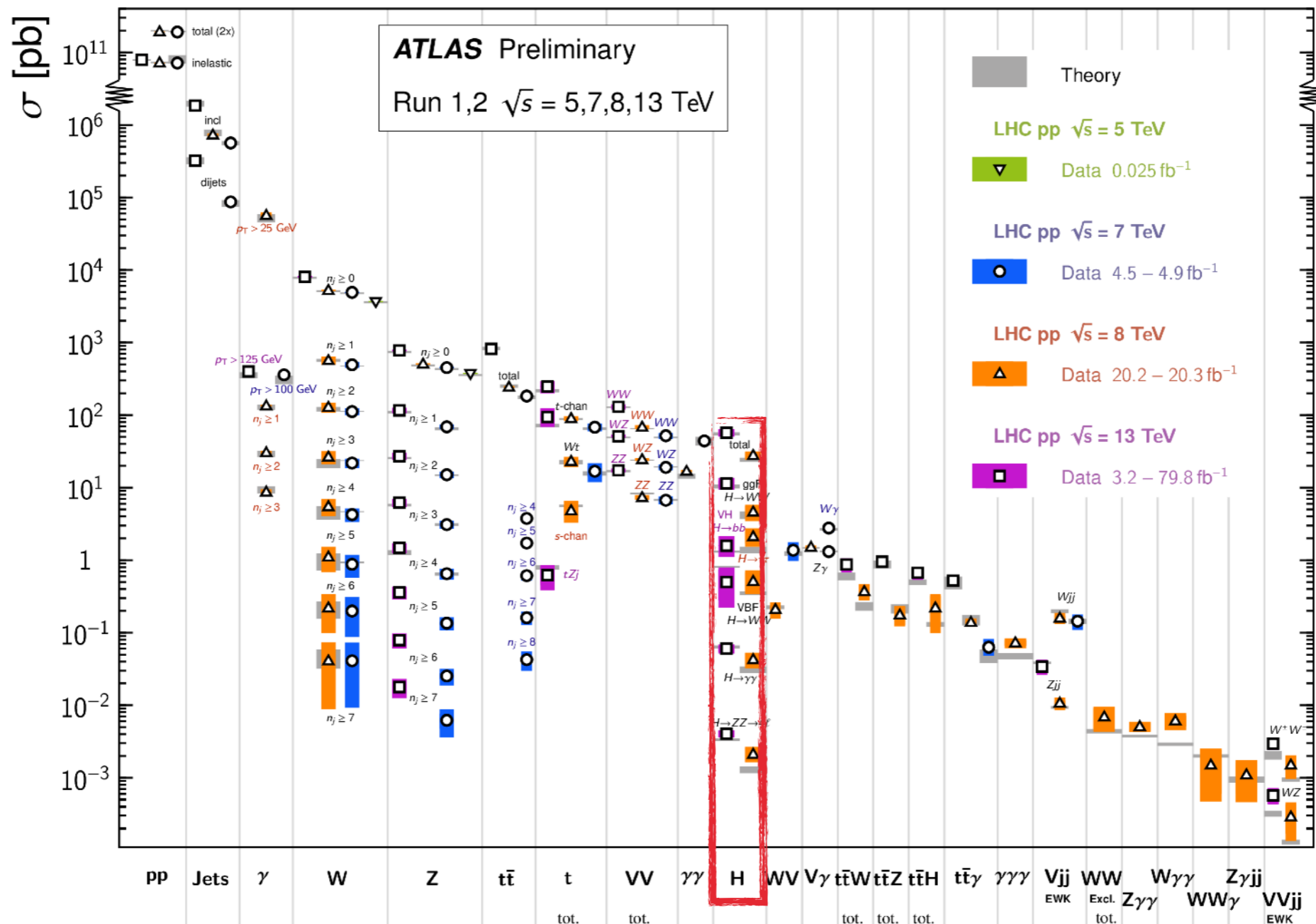


DATA WRANGLING

- ▶ e.g. Higgs physics: Throw away $\sim 10^9$ 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 $pp \rightarrow HH+X$, 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.

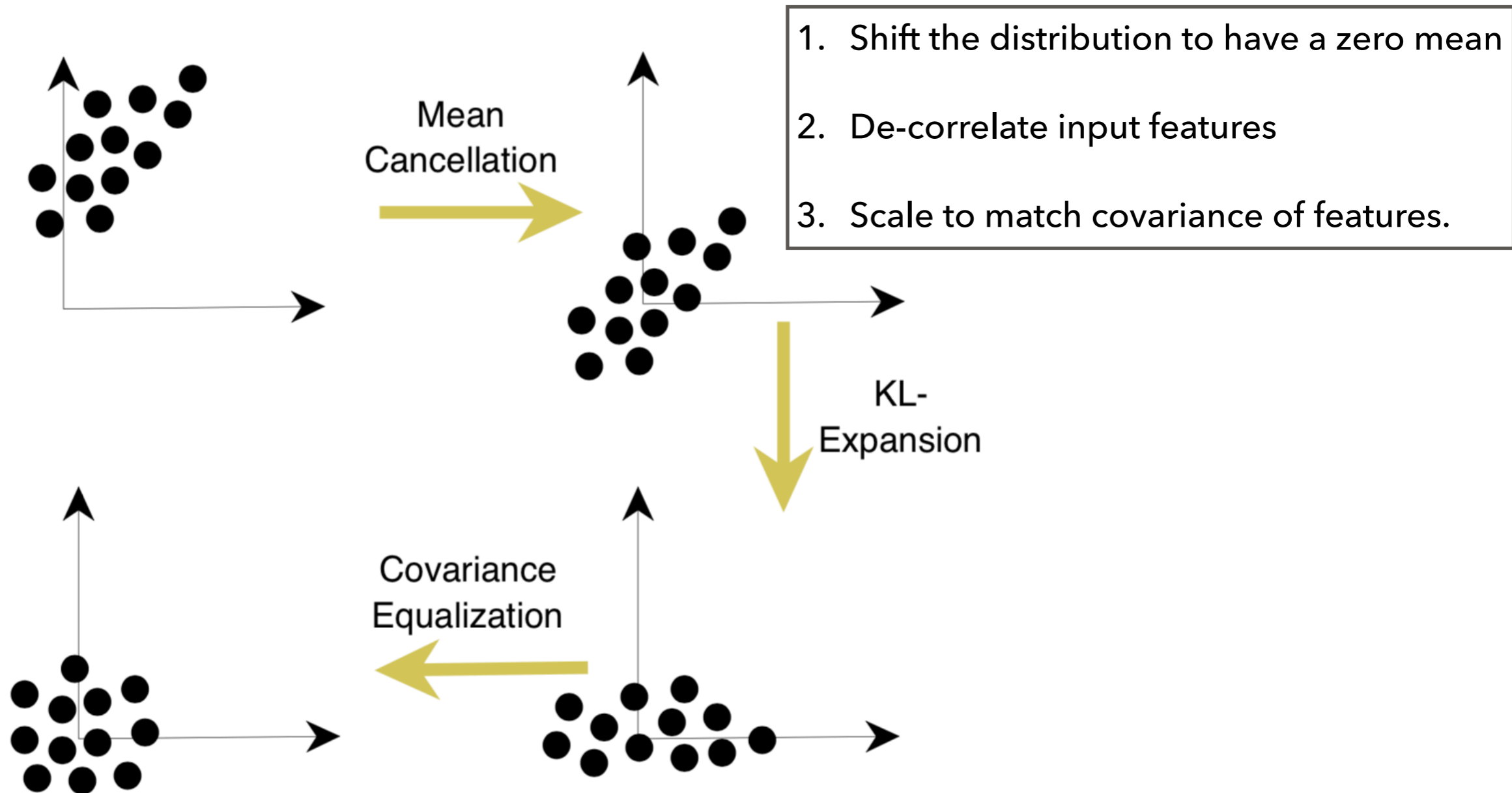


ACTIVATION FUNCTIONS: DATA PREPARATION

- ▶ 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 \mapsto TeV would translate 10^{12} eV \mapsto 1 TeV, 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).





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 θ , we can select a point in the parameter hyperspace $\hat{\theta}$ that has a corresponding estimator of the function \hat{y} .
- ▶ The θ are hyper-parameters (HPs) of the model.
- ▶ **Unsupervised learning:**
 - ▶ Infer the probability distribution $P(\hat{y})$ 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 \hat{y} .
 - ▶ Can be thought of as computing a conditional probability $P(t | \hat{y})$.
 - ▶ 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 χ^2 , but without the error term:

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

- ▶ The θ 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.

- 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

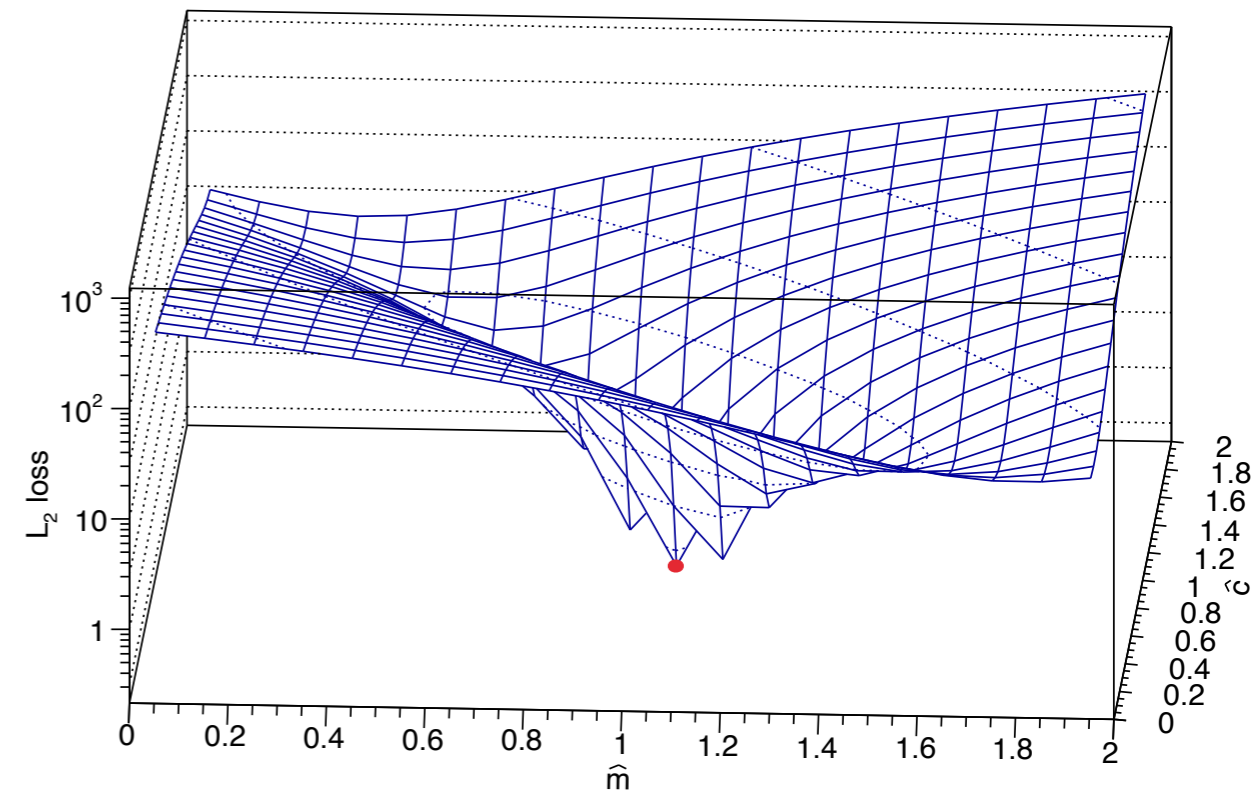
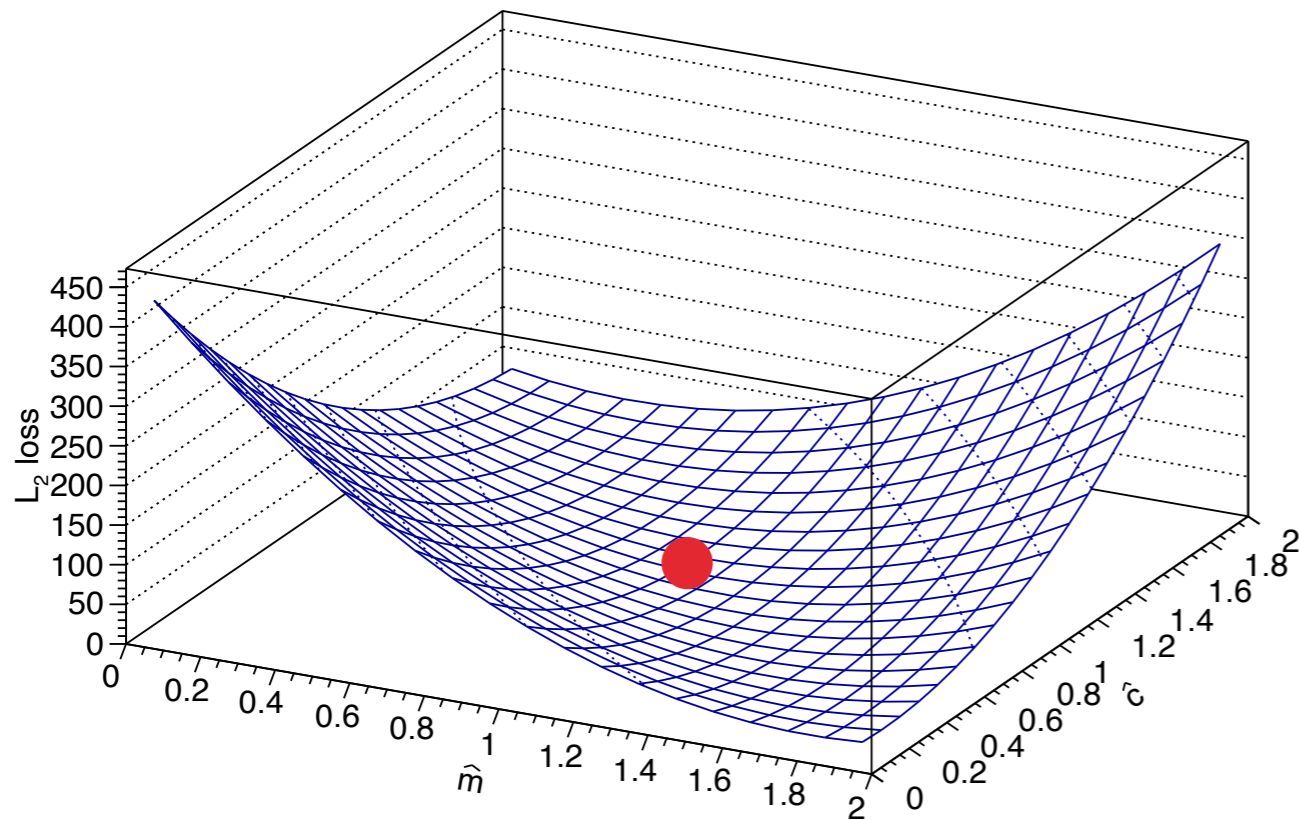


GRID SEARCH

- ▶ 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 θ .
- ▶ 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 Γ 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 Γ as the parameter of an exponential (the radial basis function); and so a linear search in C and a logarithmic search in Γ 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>.

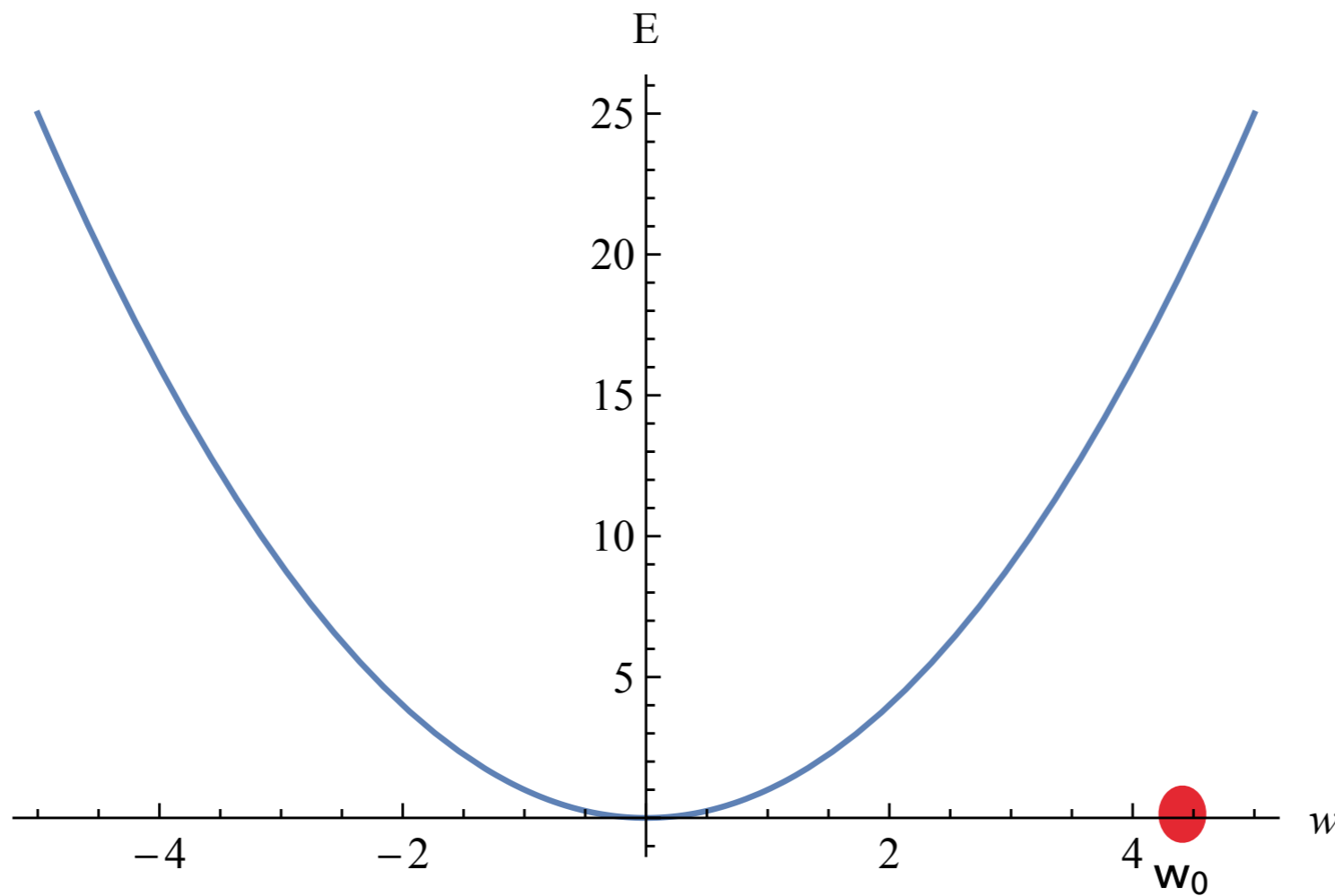
OPTIMISATION

GRADIENT DESCENT



GRADIENT DESCENT

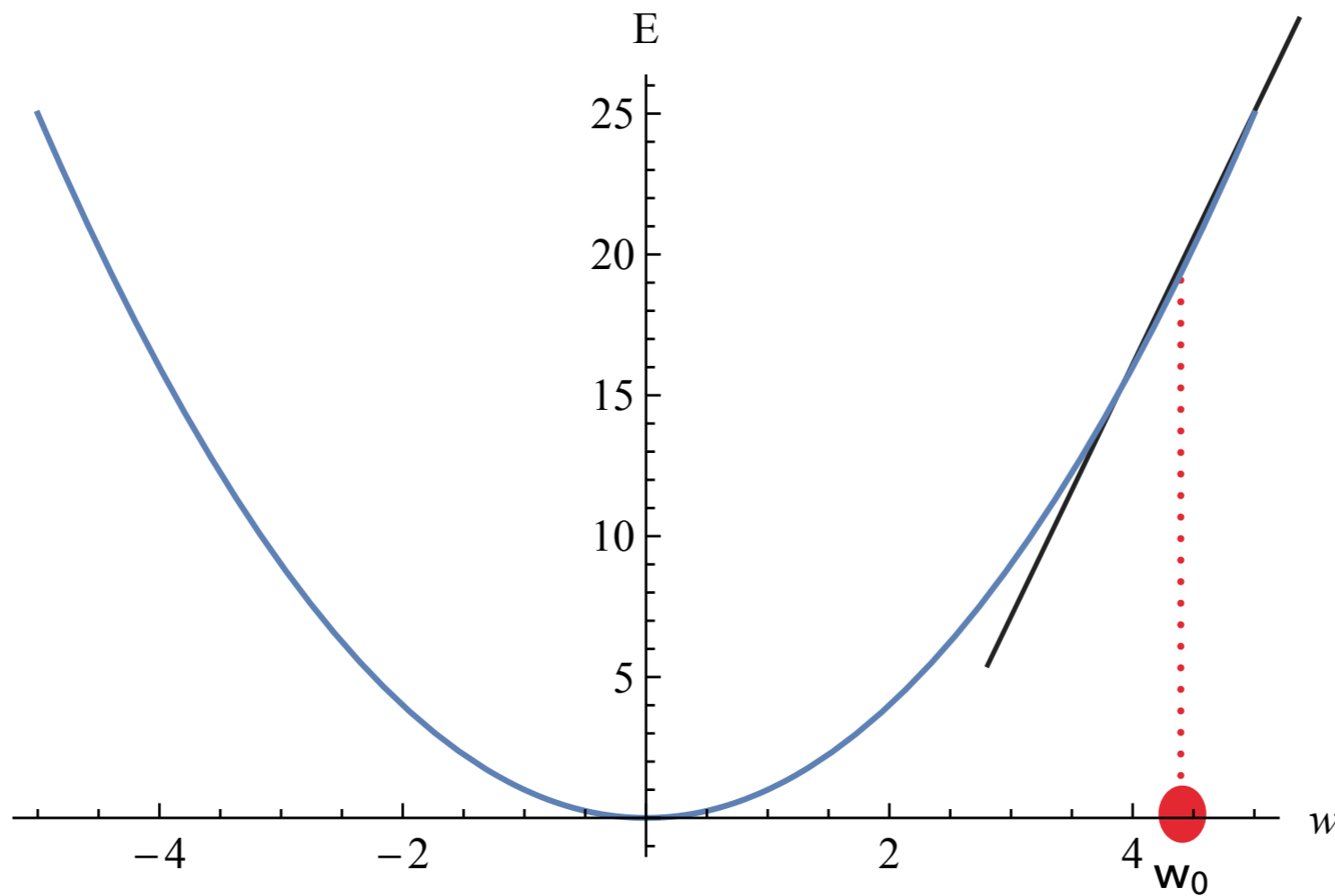
- ▶ Guess an initial value for the weight parameter: w_0 .





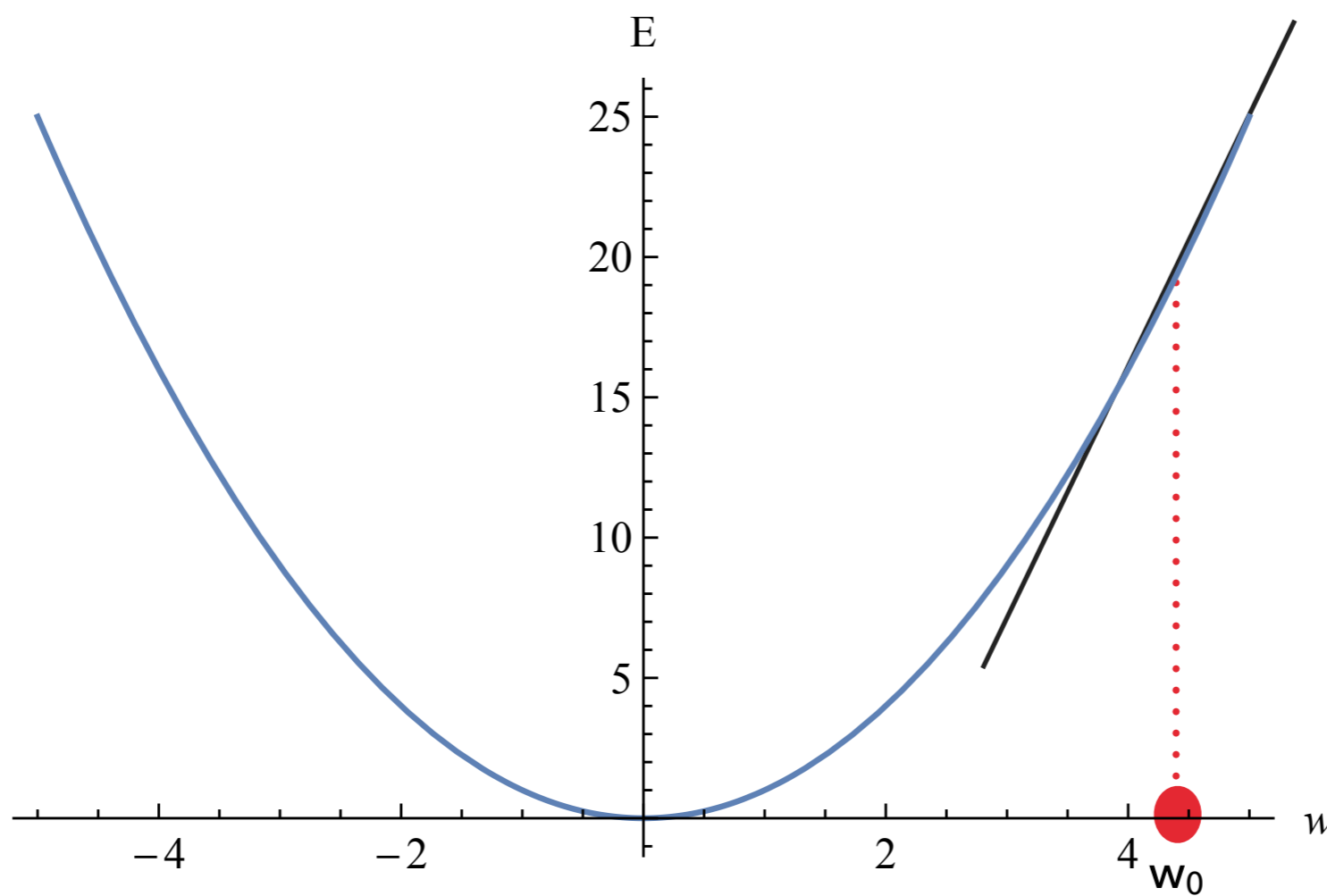
GRADIENT DESCENT

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



GRADIENT DESCENT

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



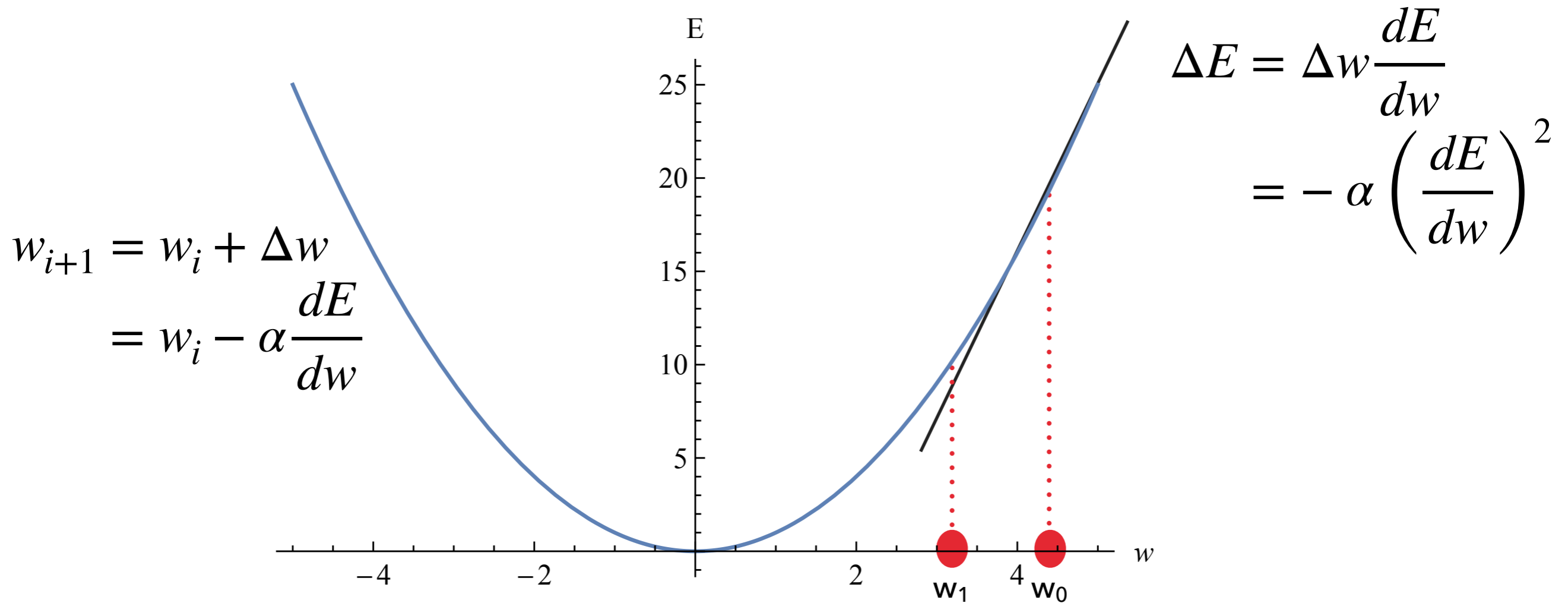
$$\begin{aligned}\Delta E &= \Delta w \frac{dE}{dw} \\ &= -\alpha \left(\frac{dE}{dw} \right)^2\end{aligned}$$

α 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: $w_1 = w_0 + \Delta w$

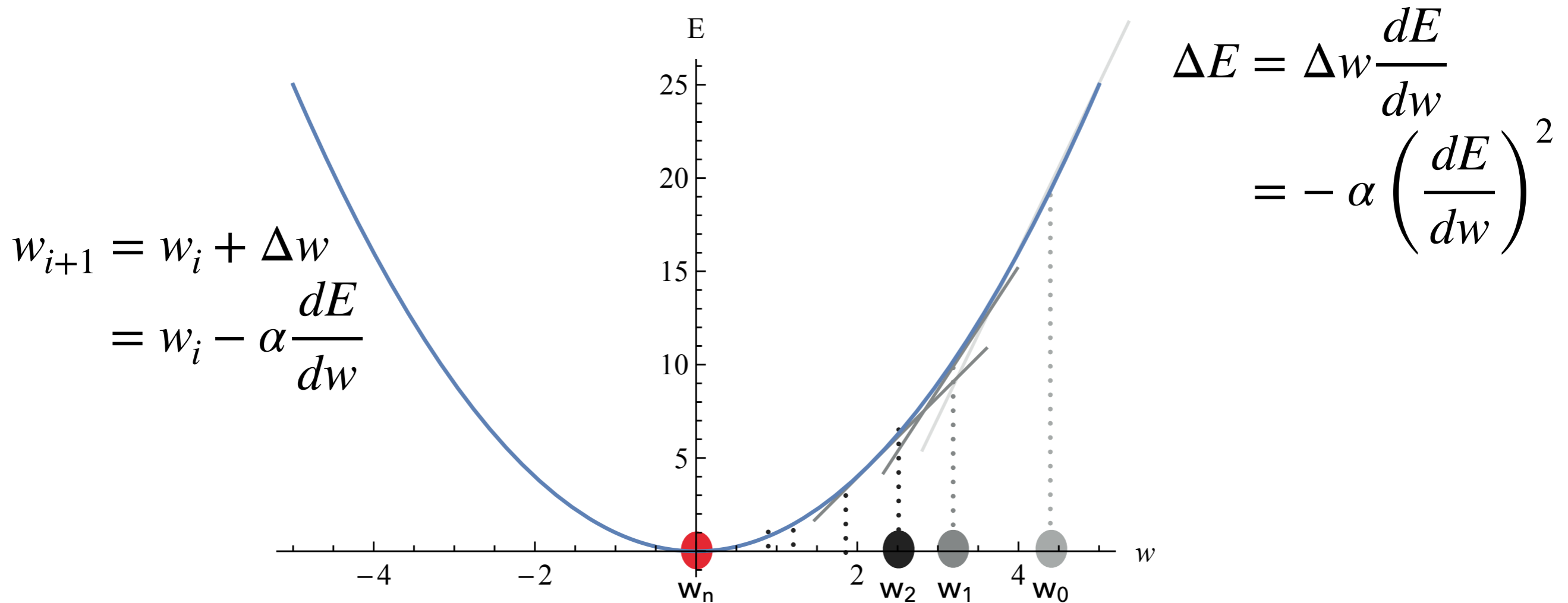


α is the learning rate: a small positive number

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

GRADIENT DESCENT

- ▶ Repeat until some convergence criteria is satisfied.



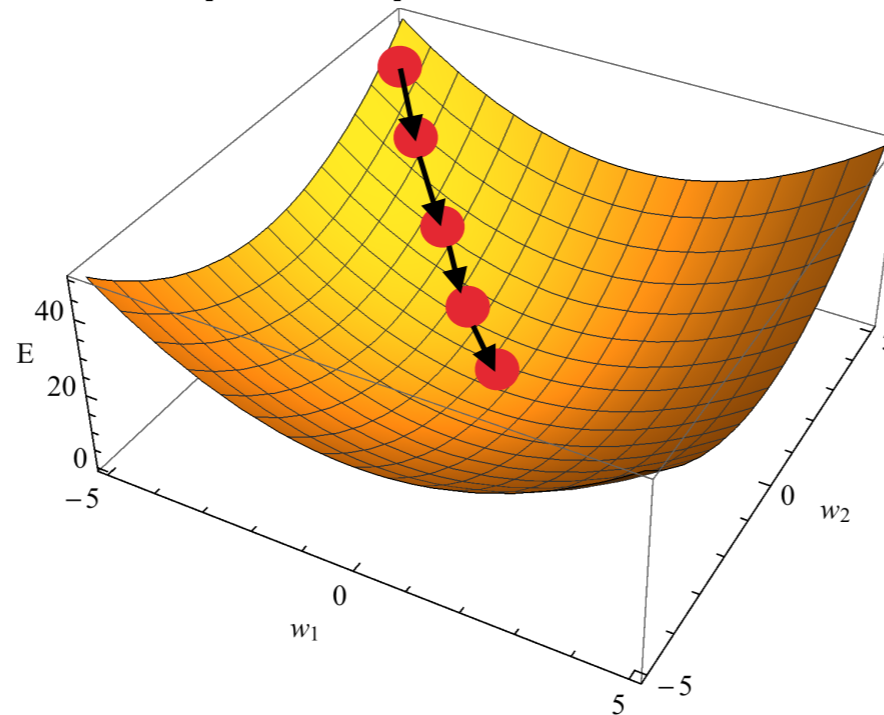
α is the learning rate: a small positive number

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.

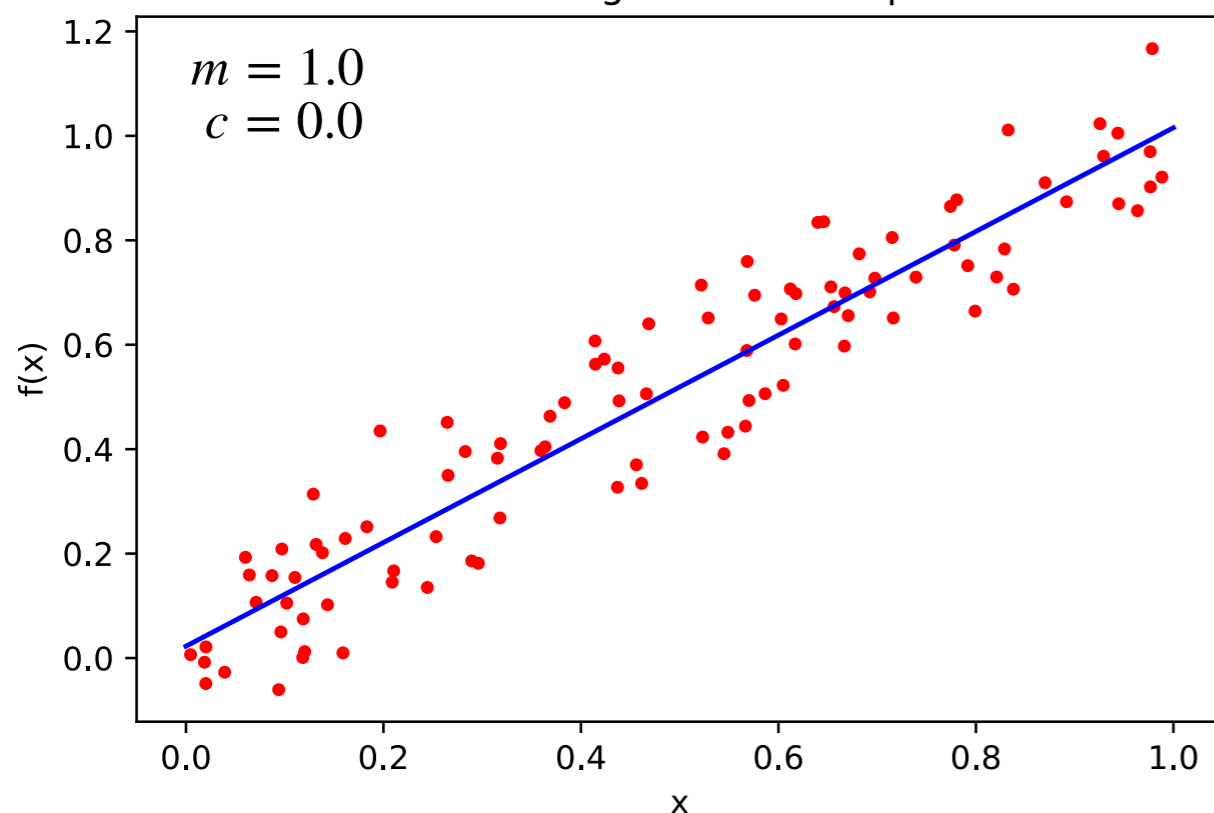


- ▶ The successive points w_{i+1} can be visualised a bit like a ball rolling down a concave hill into the region of the minimum.
- ▶ In general update weights such that $\Delta E = \Delta w \nabla E = -\alpha \nabla^2 E$
- ▶ and $w_{i+1} = w_i - \alpha \nabla E$.

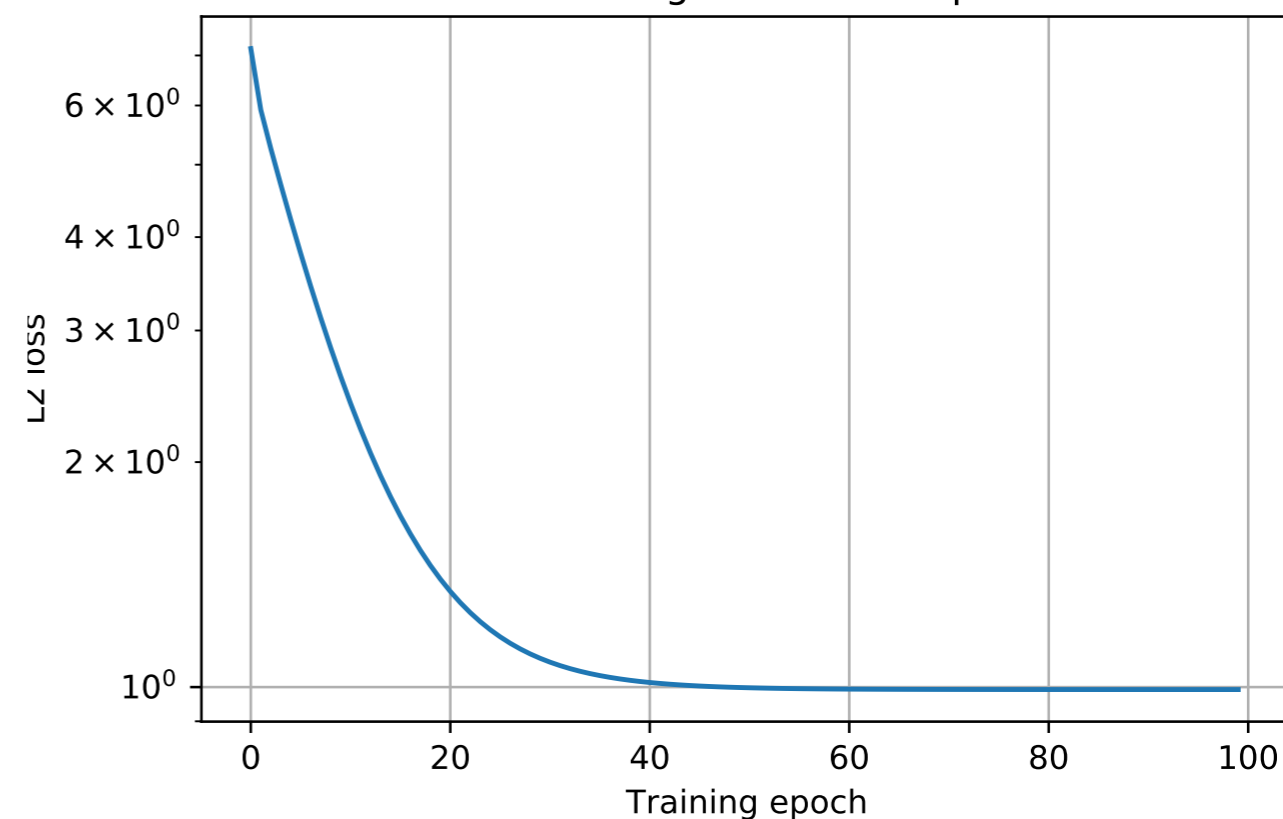
GRADIENT DESCENT: EXAMPLE

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

Linear Regression Example



Linear Regression Example



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

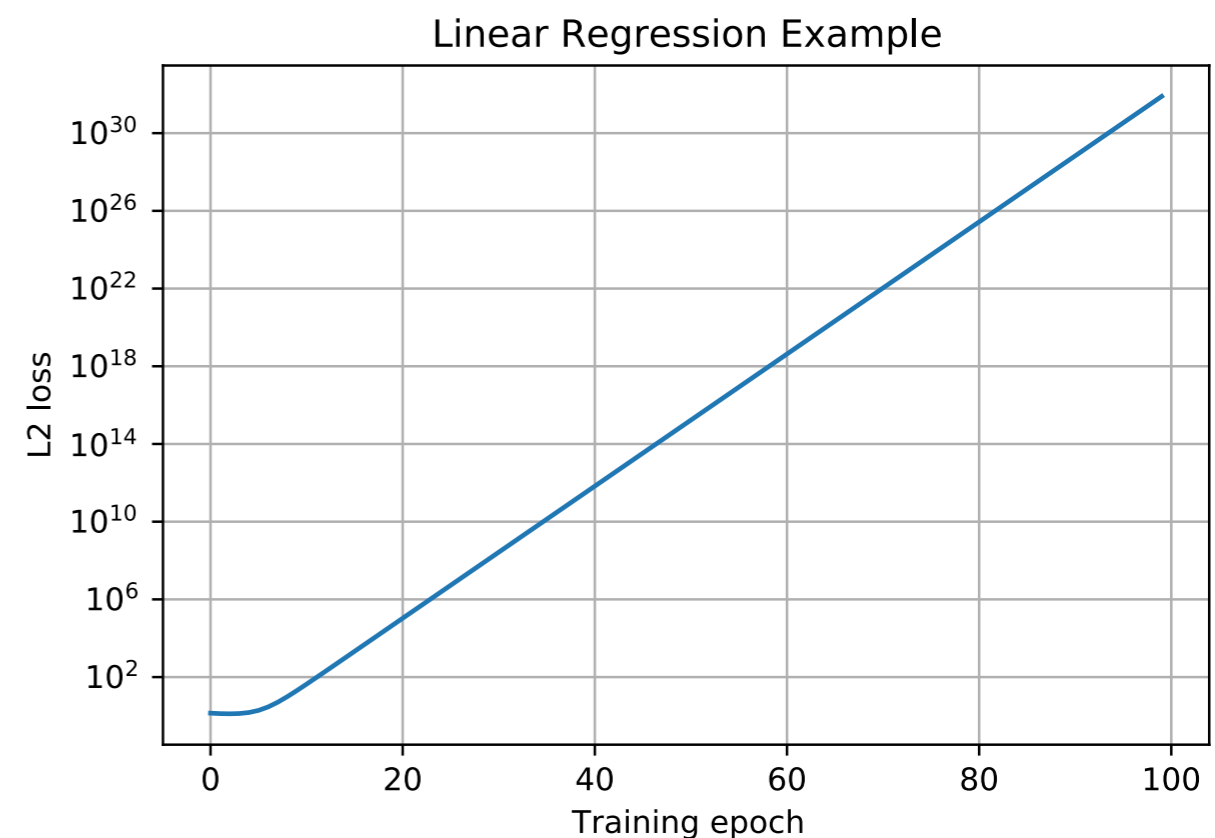
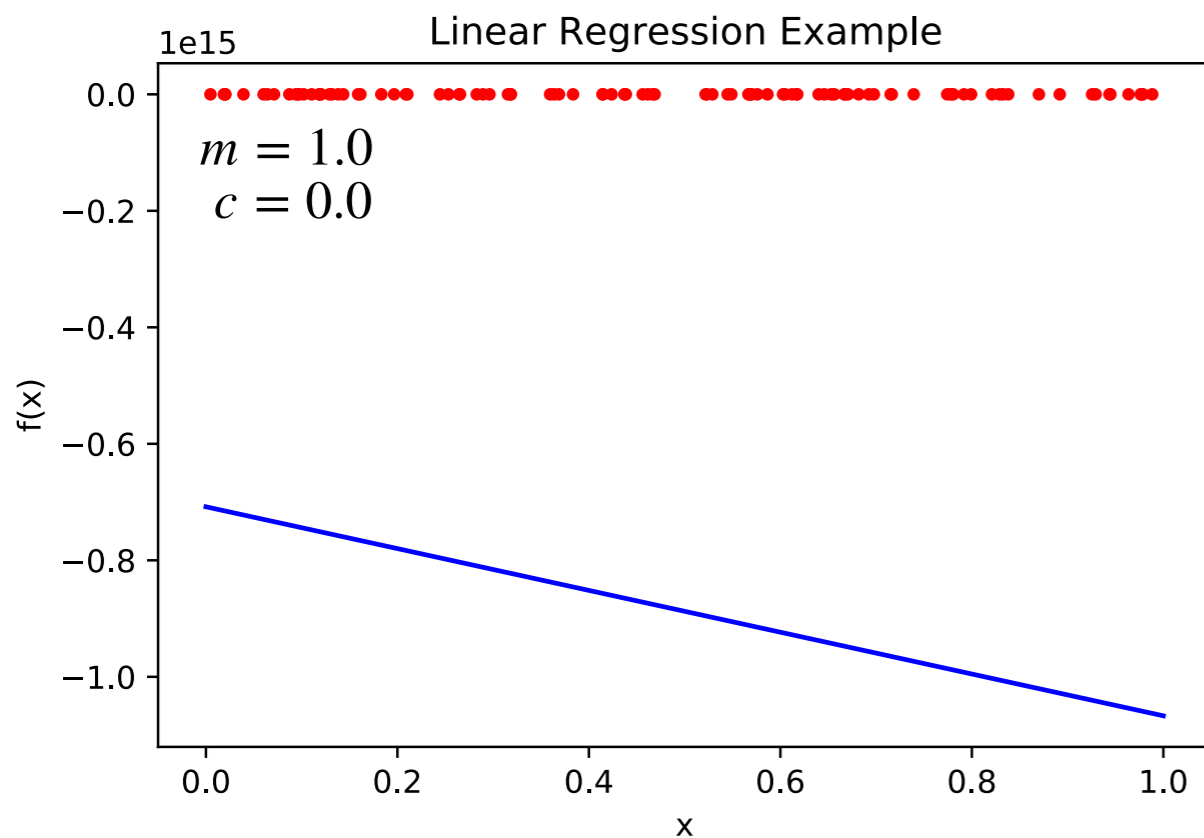
$$\hat{m} = 0.9928\dots$$

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

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:

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

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

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, α , 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 α .
- ▶ 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 θ 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 (\widehat{v}_t) are bias corrected estimator of the moments.
 - ▶ Some initial guess for the HP is taken: θ_0 , and the HPs for a given epoch are denoted by θ_t .
 - ▶ α is the step size (i.e. learning rate).
 - ▶ β_1 and β_2 are exponential decay rates of moments.

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 β_1^t and β_2^t we denote β_1 and β_2 to the power t .

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0 : Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector)

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

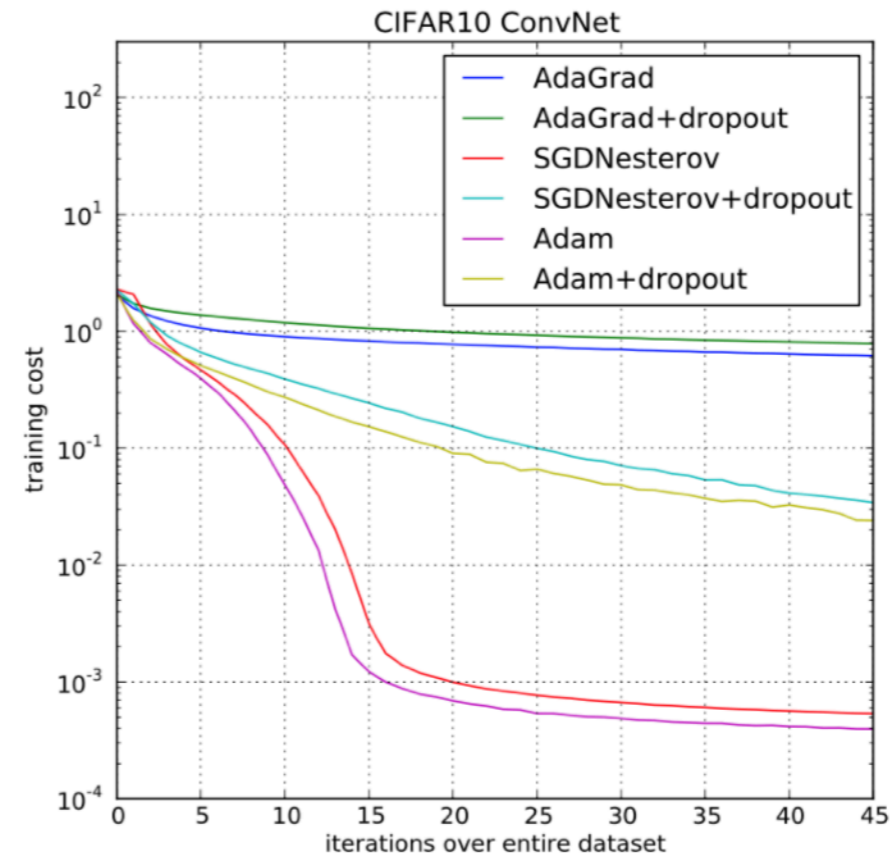
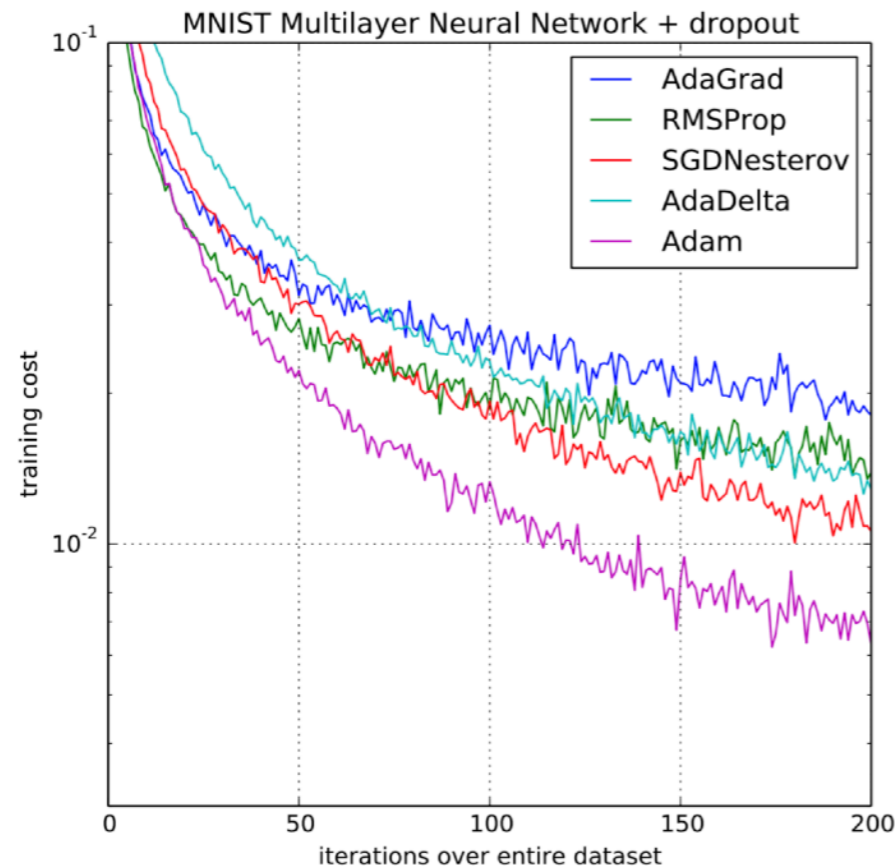
end while

return θ_t (Resulting parameters)

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.

ADAM OPTIMISER

(ADAM: ADaptive Moment estimation based on gradient descent)

- ▶ Benchmarking: MNIST and CFAR10 are standard benchmark data sets in CS (see appendix).

training cost

MNIST is a set of handwritten numbers 0 through 9; CFAR10(0) is an image classification data set (cars, boats etc).

iterations over entire dataset

iterations over entire dataset

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

MULTIPLE MINIMA

MORE ON LOSS FUNCTIONS

OPTIMISATION

MISCELLANEOUS



GRADIENT DESCENT: MULTIPLE MINIMA

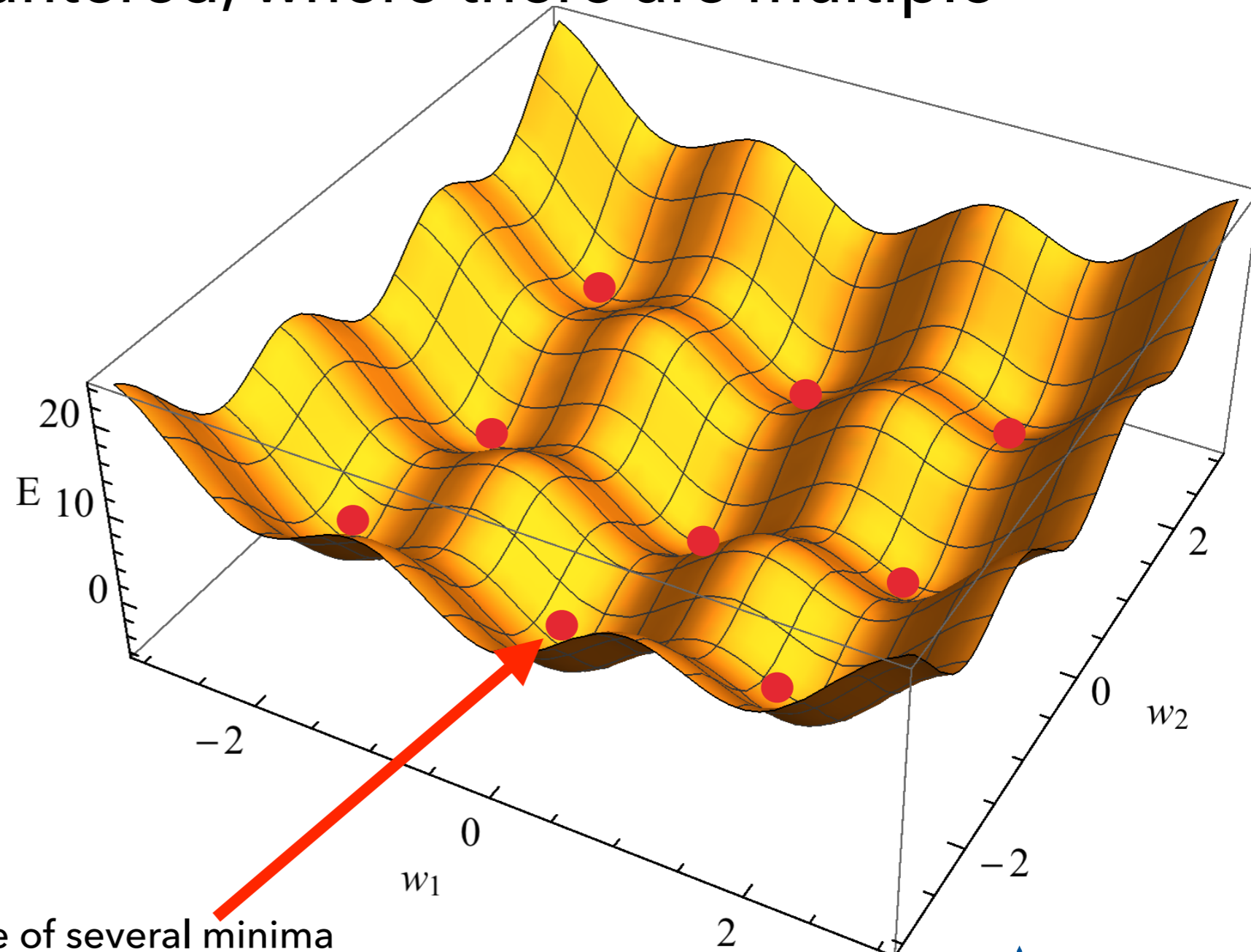
- ▶ Often more complicated hyperspace optimisation problems are encountered, where there are multiple minima.

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

In reality there are often multiple minima.

Sometimes the minima are degenerate, or near degenerate.

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





GRADIENT DESCENT: MULTIPLE MINIMA

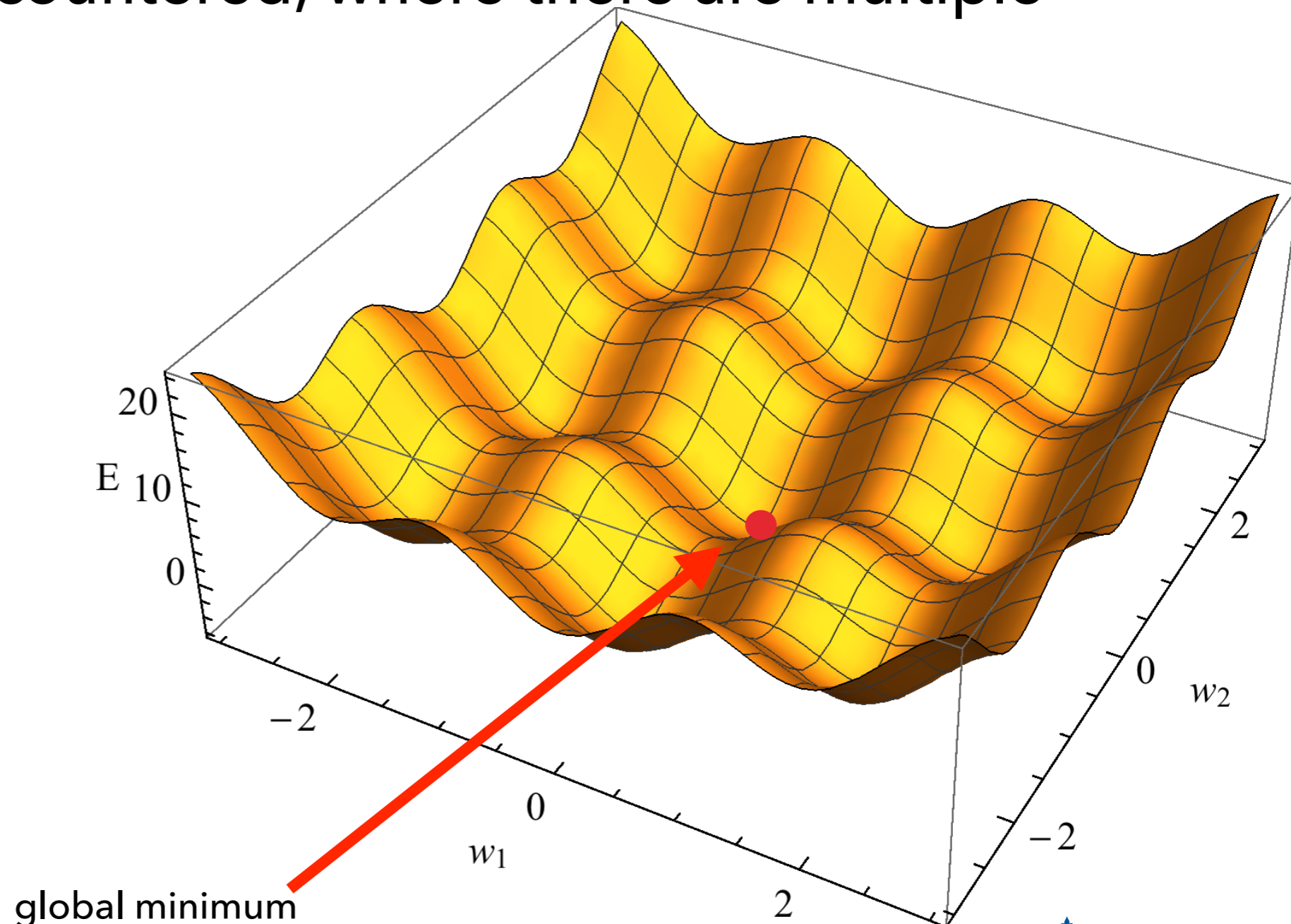
- ▶ Often more complicated hyperspace optimisation problems are encountered, where there are multiple minima.

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

- ▶ The mean square error (MSE) loss function:

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

- ▶ Cross entropy

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

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

- ▶ and many more can be found in the literature and online ...

SUPERVISED LEARNING

OVERTRAINING

WEIGHT REGULARISATION

CROSS VALIDATION

DROPOUT

TRAINING

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

- ▶ 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, $\hat{\theta}$.
- ▶ 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.



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

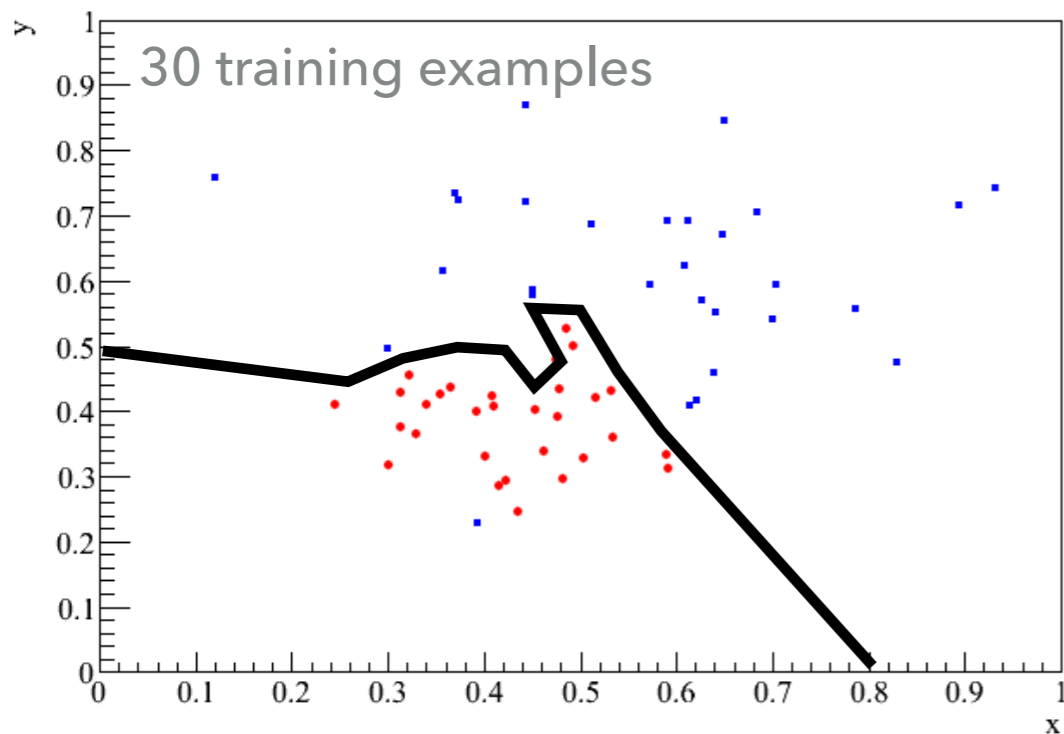
- ▶ The KaggleSet column in the CVS file for this data allows the user to understand if this is to be used for training or testing, or some other means (e.g. Kaggle Score Board evaluation).
- ▶ The Label column, is the known label; this defines the t_i used in the loss function.
- ▶ I use this challenge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

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:



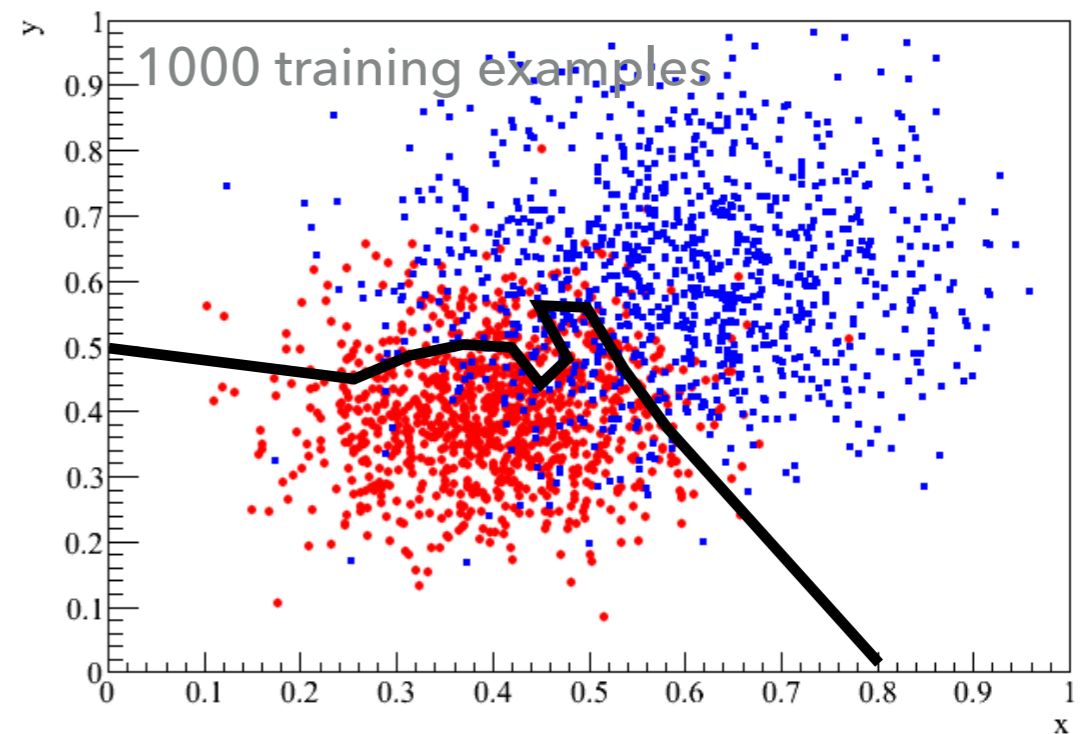
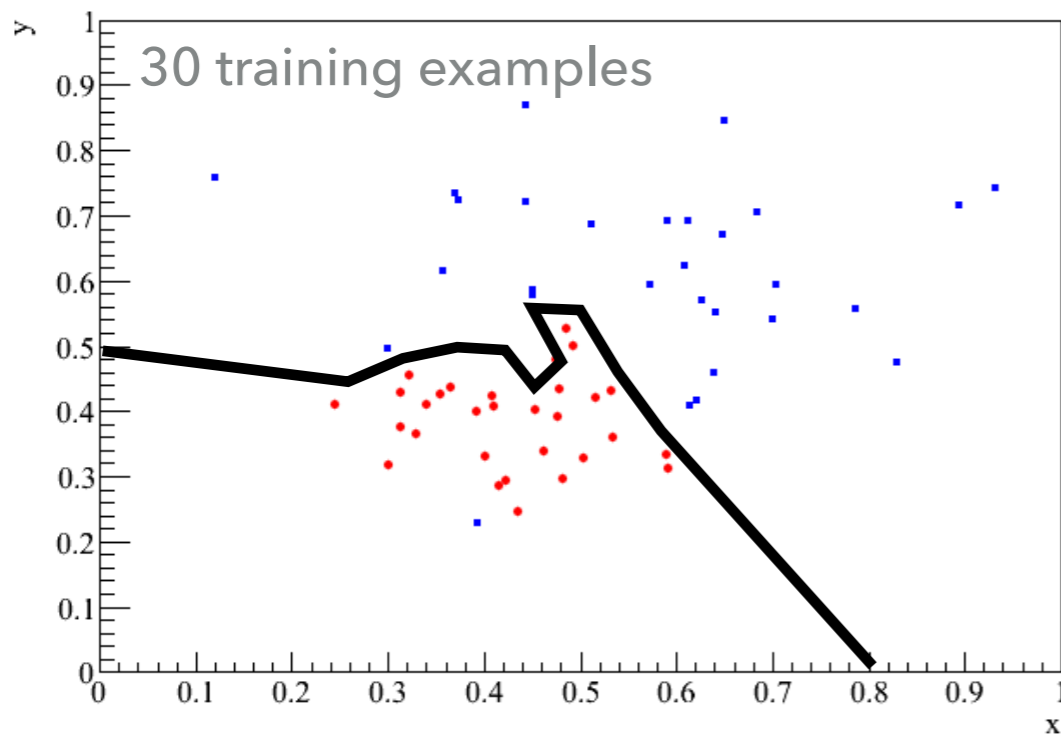
The decision boundary selected here does a good job of separating the red and blue dots.

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

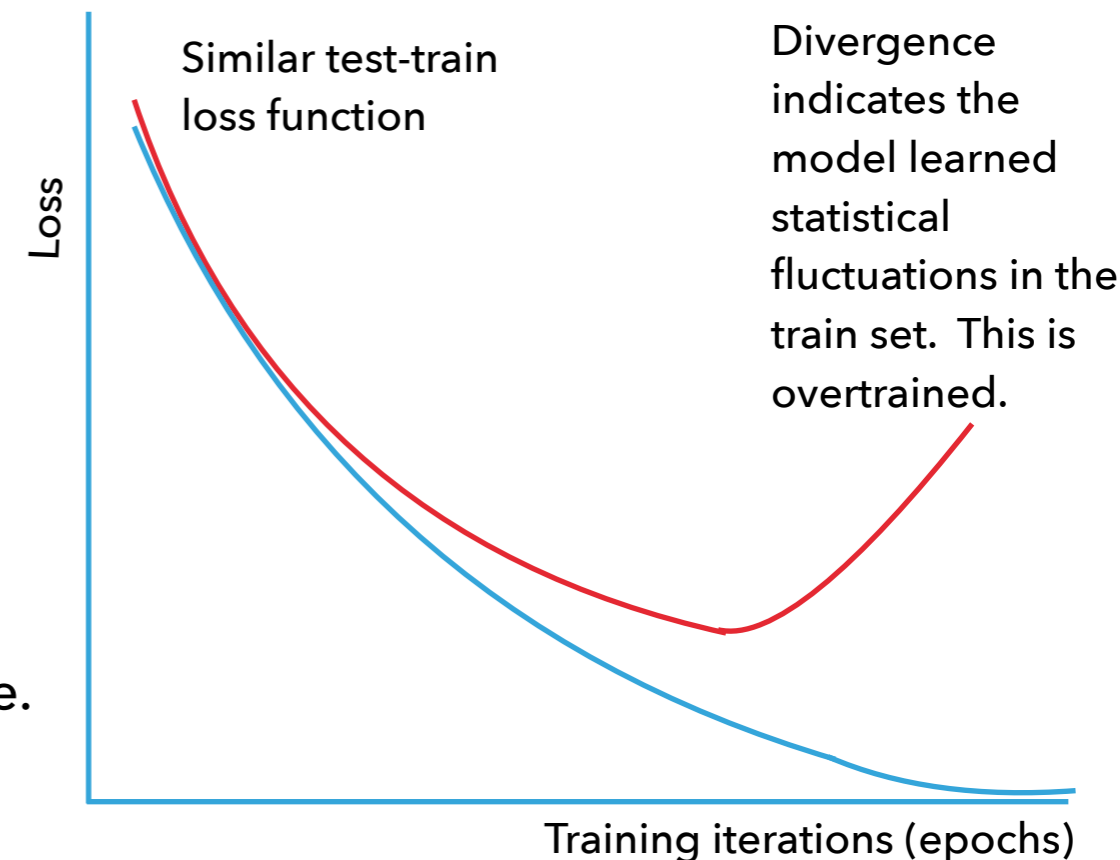
- ▶ A model is over trained if the HPs that have been determined are tuned to the statistical fluctuations in the data set.
- ▶ Simple illustration of the problem:



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.





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 Δ_{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:

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

- ▶ 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.



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.



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.



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.

TRAINING DROPOUT

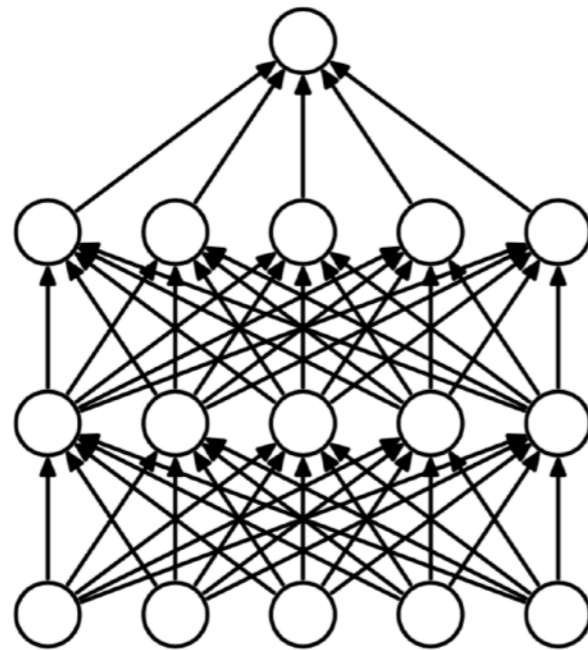


DROPOUT

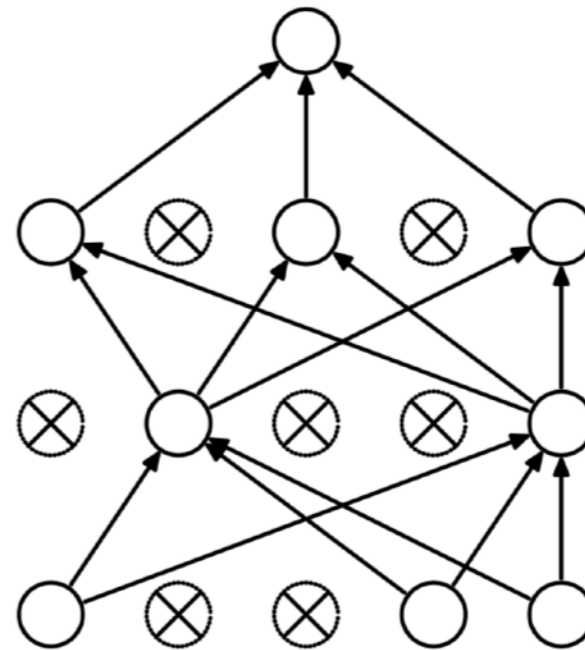
- ▶ A problem with deep networks is the number of HPs that need to be determined.
- ▶ This leads to a requirement for very large data sets to avoid overfitting (or fine-tuning).
- ▶ HPs can also learn to “co-adapt” in the training process.
 - ▶ co-adapt means that as one parameter is changed, another 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.

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



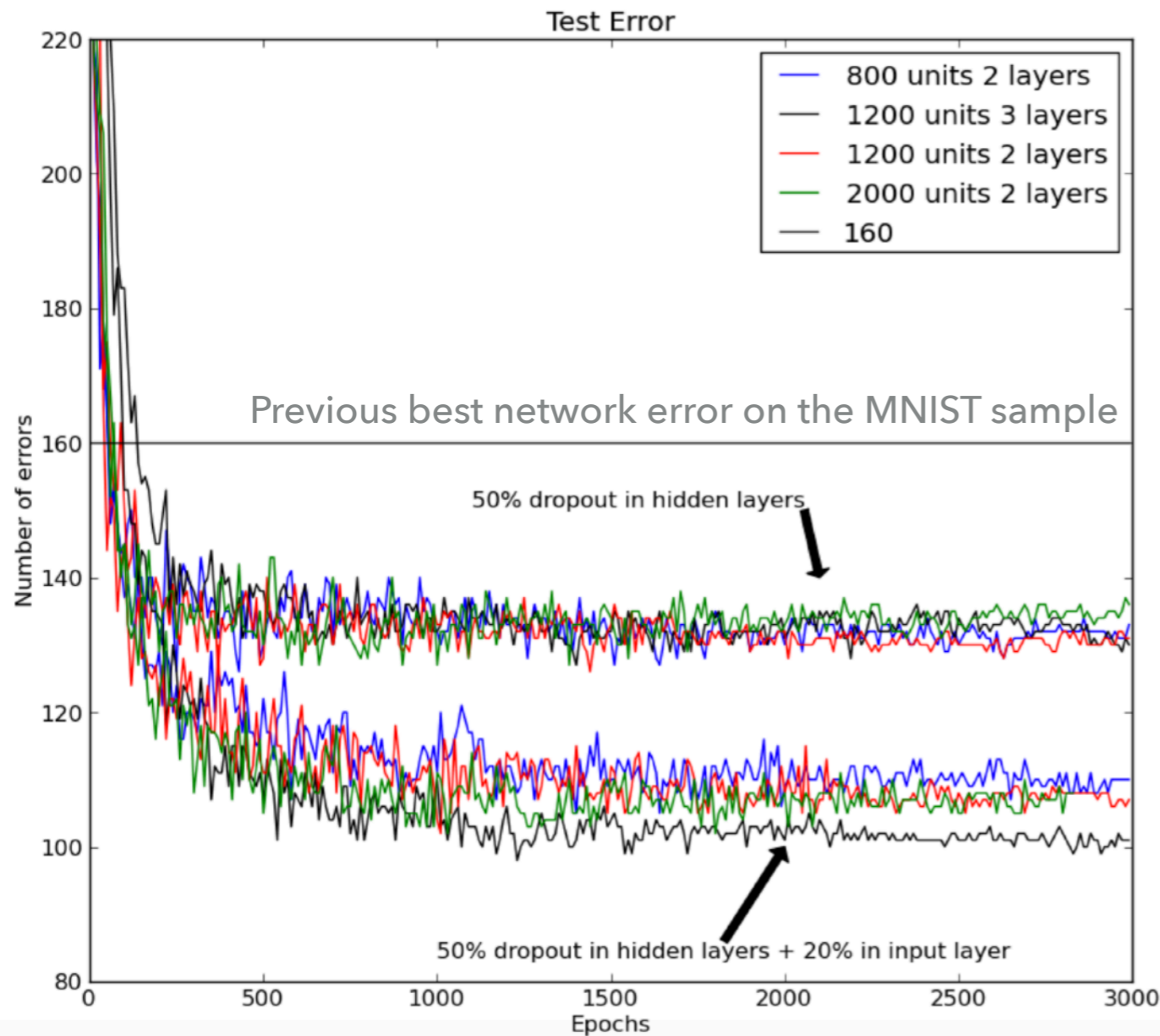
(b) After applying dropout.

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-adaptation 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.

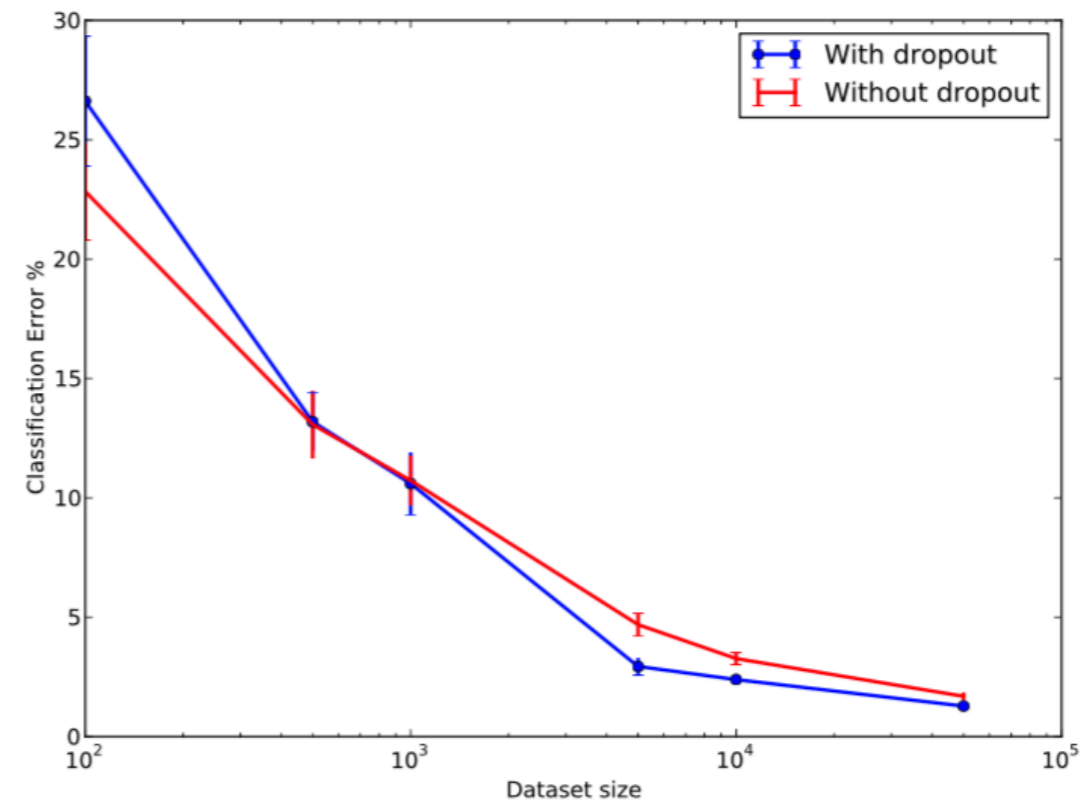
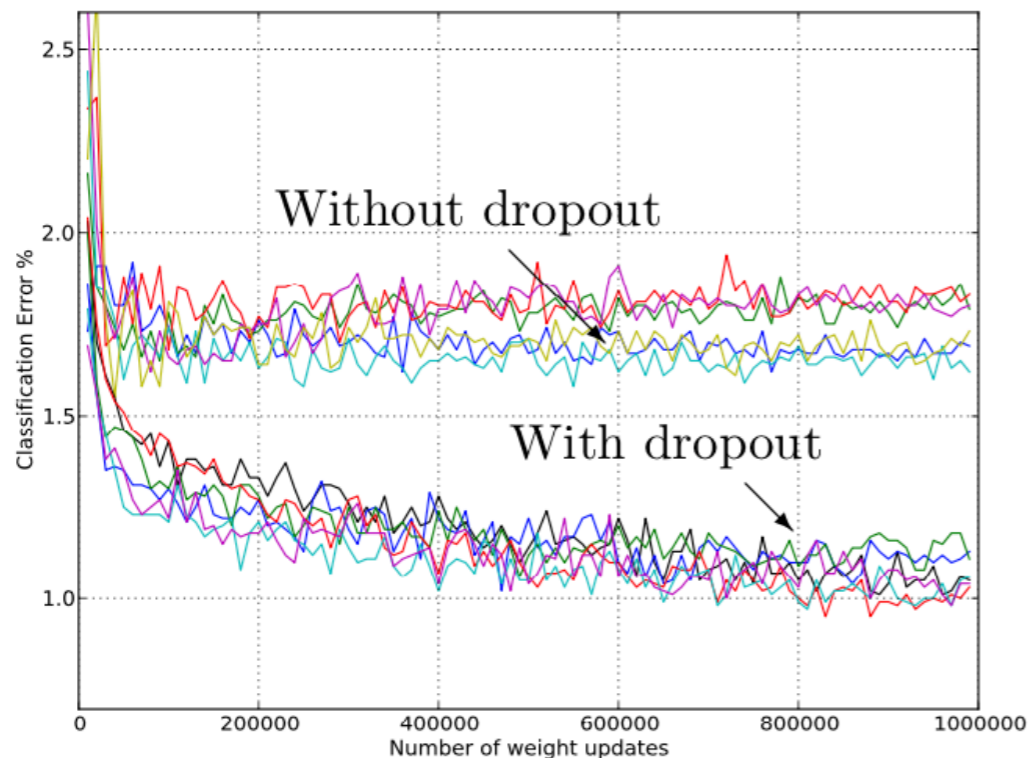
DROPOUT



- ▶ 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.

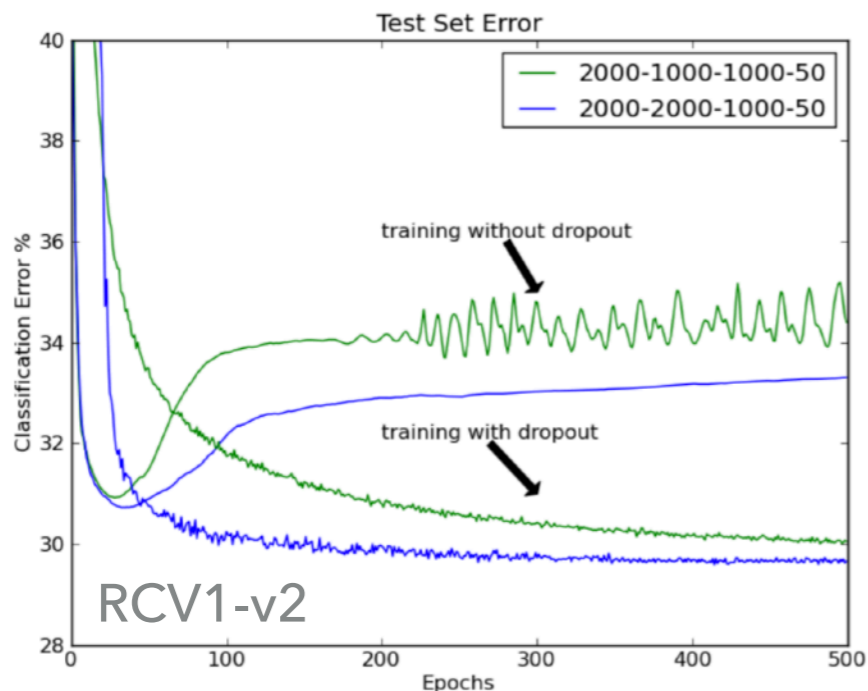
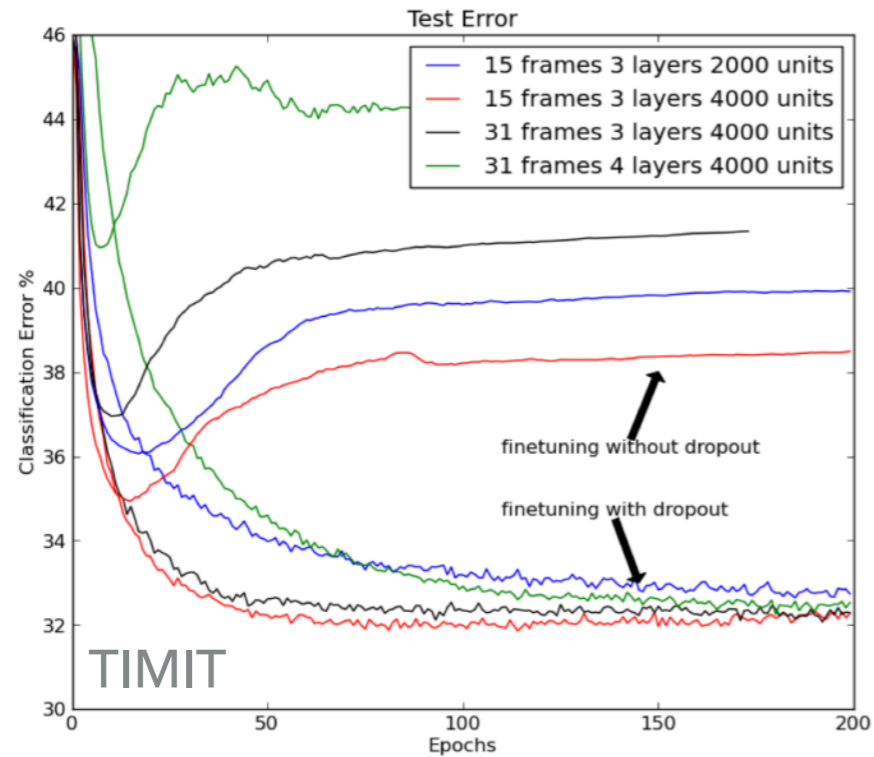
OVER FITTING: DROPOUT

- ▶ 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.

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).

DECISION TREES

BOOSTING

ADABOOST, M1

RANDOM FORESTS

DECISION TREES

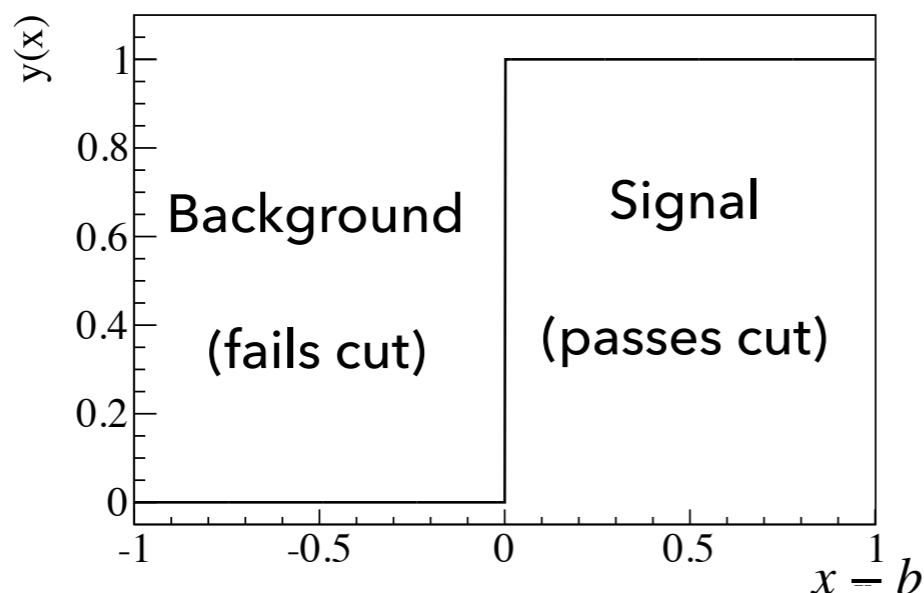


DECISION TREES

- ▶ 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

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

where b is the offset (bias) from zero.

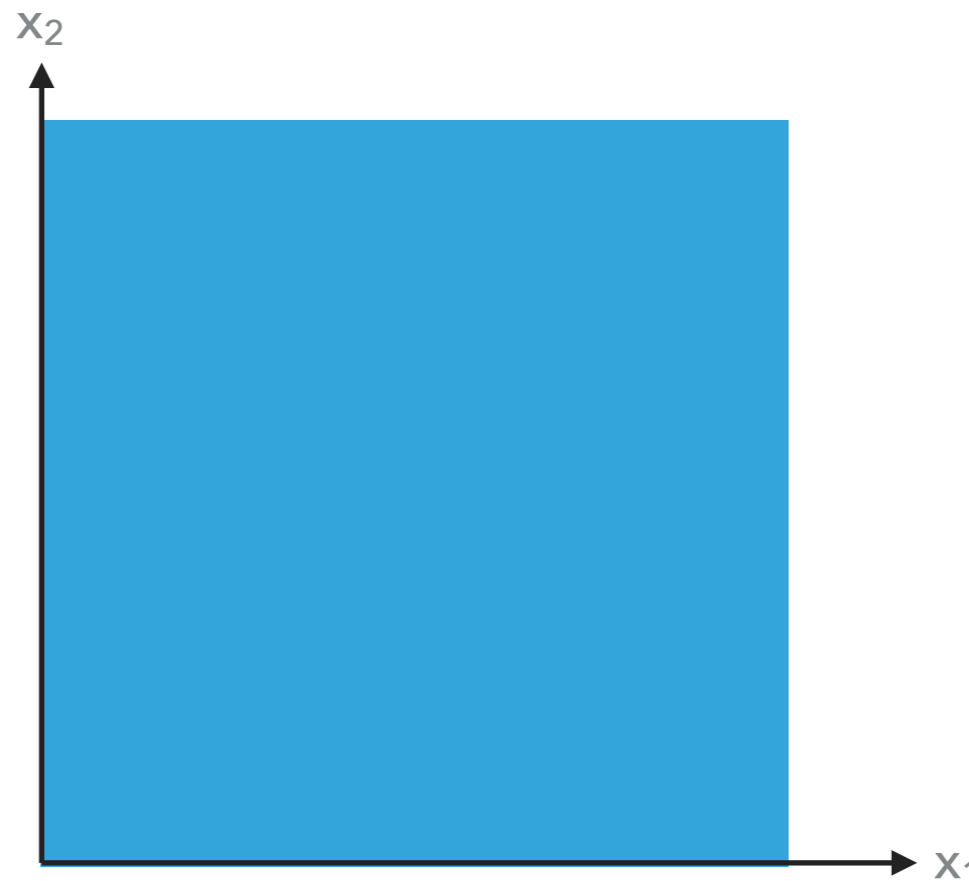
*Can generalise the problem to an arbitrary number of types.



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.

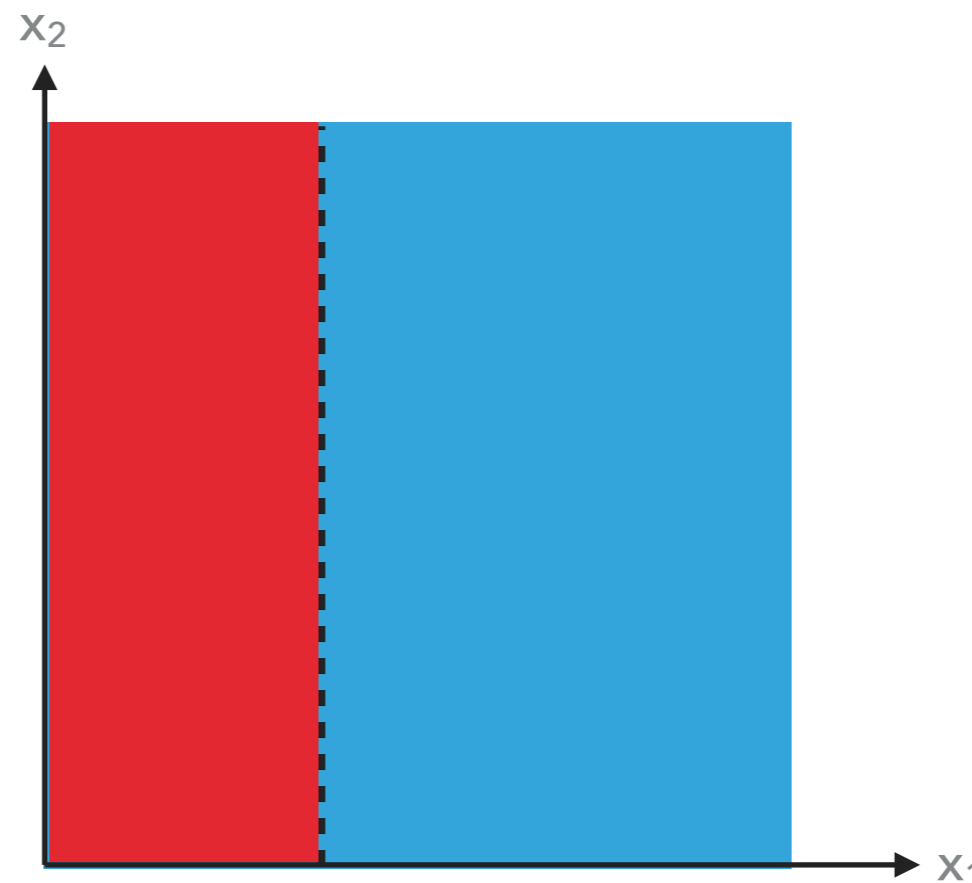
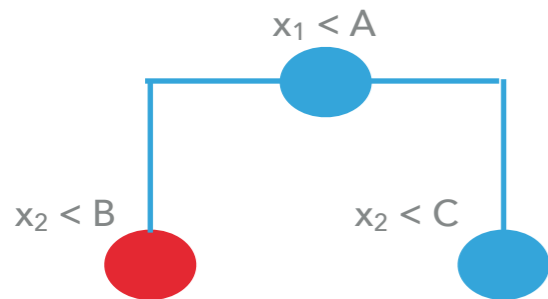


Example feature space
described by $X = \{x_1, x_2\}$

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.



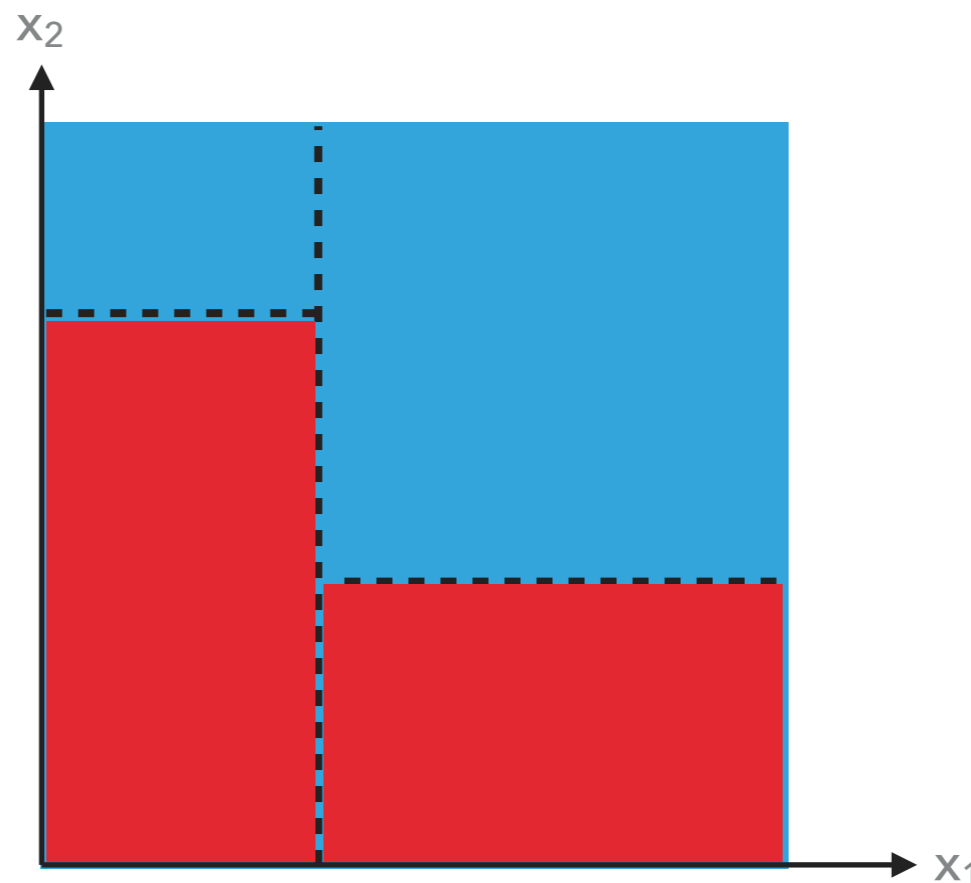
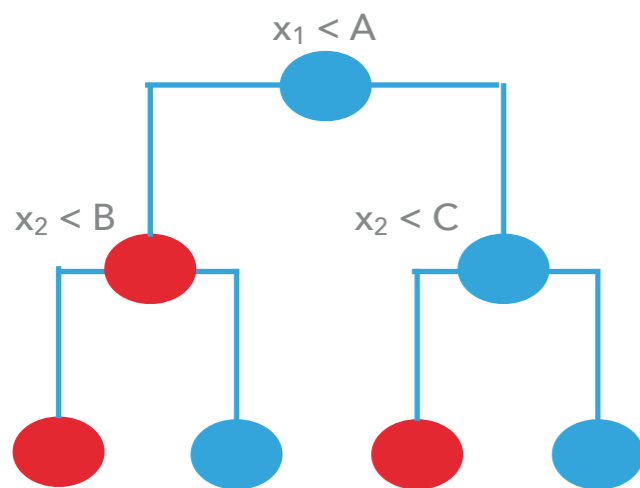
Cut on the feature space to separate the data into two different regions.



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



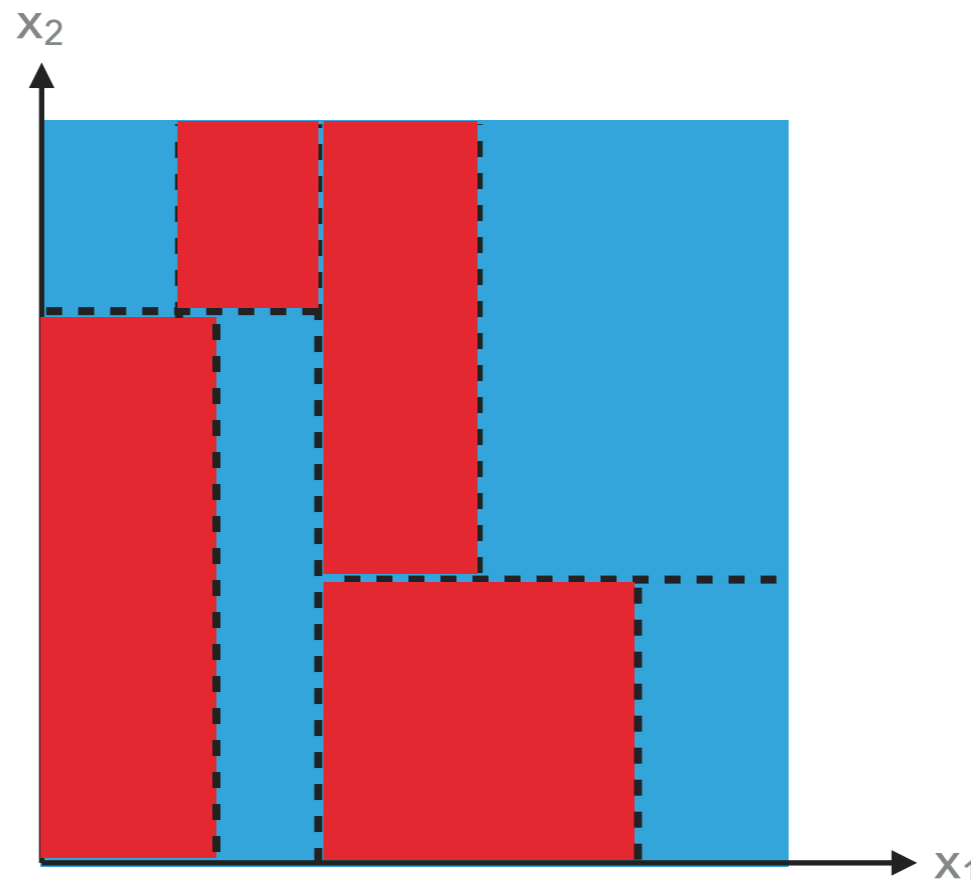
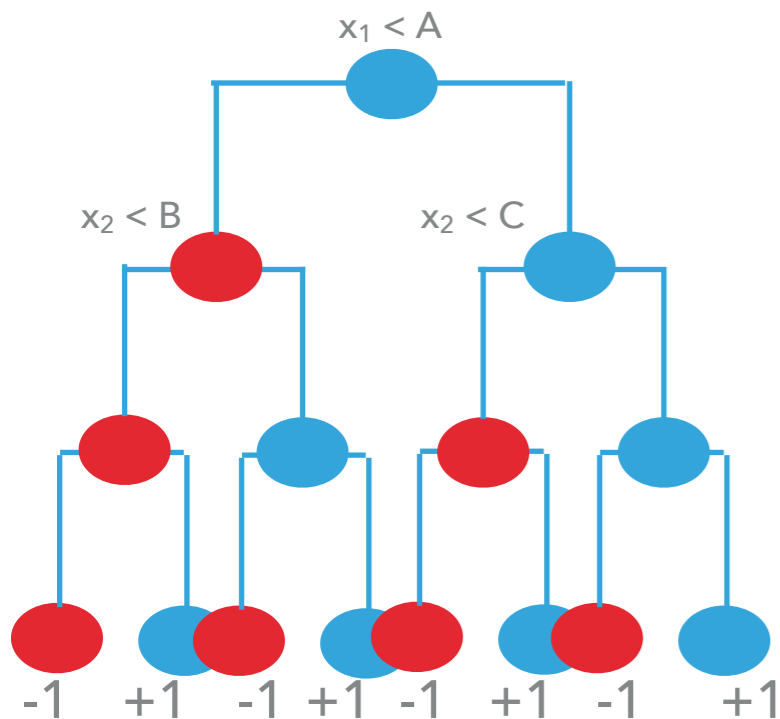
The feature space gets further sub-divided.



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.

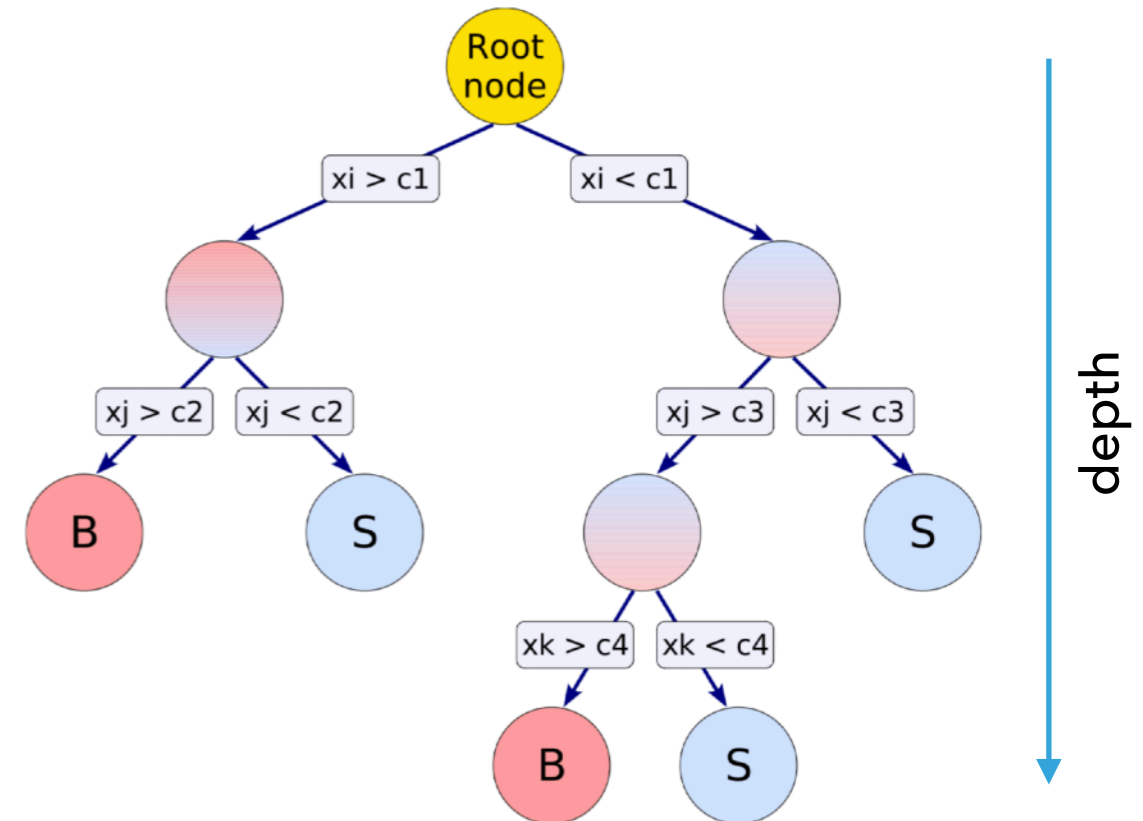
Divide the data again



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 node.
- ▶ 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



BOOSTING: AdaBoost.M1

- ▶ i is the i^{th} example out of a data set with N examples.
- ▶ m is the m^{th} 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.



BOOSTING: AdaBoost.M1

- ▶ i is the i^{th} example out of a data set with N examples.
- ▶ m is the m^{th} 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 ϵ_m .
 - ▶ Calculate the boost factor $\beta_m = \frac{\epsilon_m}{1 - \epsilon_m}$.
 - ▶ Update weights for misclassified examples $w_i \mapsto w_i e^{\ln(1/\beta_m)}$.



BOOSTING: AdaBoost.M1

- ▶ i is the i^{th} example out of a data set with N examples.
- ▶ m is the m^{th} 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 ϵ_m .
 - ▶ Calculate the boost factor $\beta_m = \frac{\epsilon_m}{1 - \epsilon_m}$.
 - ▶ 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:

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

BOOSTING: AdaBoost.M1



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.

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



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 $g_c(x_i)$.

ARTIFICIAL NEURAL NETWORKS

MULTILAYER PERCEPTRONS

AUTO-ENCODERS

CONVOLUTIONAL NEURAL NETWORKS

GENERATIVE ADVERSARIAL NETWORKS

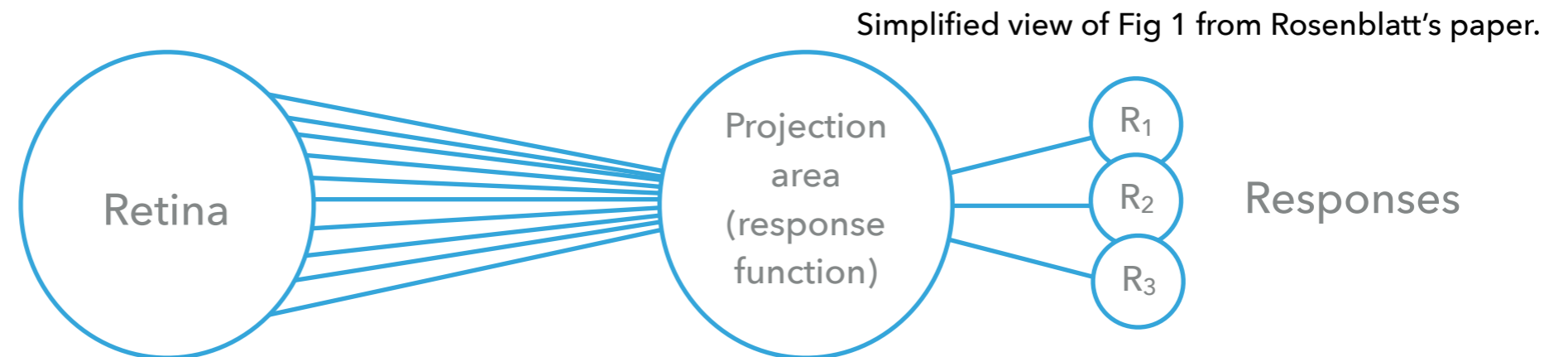
NEURAL NETWORKS

ROSENBLATT'S PERCEPTRON
ACTIVATION FUNCTIONS
ARTIFICIAL NEURAL NETWORKS
BACK PROPAGATION
OTHER REMARKS
EXAMPLE: FUNCTION APPROXIMATION

ARTIFICIAL NEURAL NETWORKS

ROSENBLATT'S PERCEPTRON

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



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

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



ROSENBLATT'S PERCEPTRON

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



ROSENBLATT'S PERCEPTRON

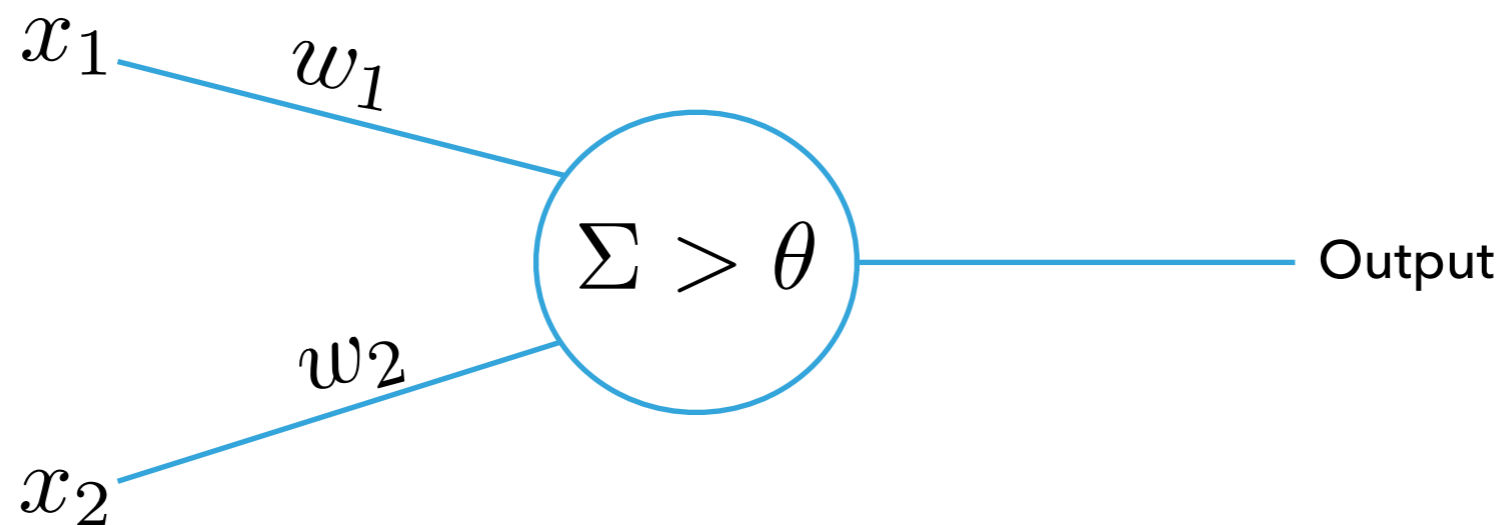
- ▶ Illustrative example:
 - ▶ Consider a measurement of two quantities x_1 , and x_2 .
 - ▶ Based on these measurements we determine if the perceptron is to give an output (value = 1) or not (value = 0).

$$\begin{array}{c} w_1 x_1 \\ + \\ w_2 x_2 \end{array} = \begin{cases} 0 \\ 1 \end{cases}$$



ROSENBLATT'S PERCEPTRON

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



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



ROSENBLATT'S PERCEPTRON

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

$$\text{If } w_1x_1 + w_2x_2 > \theta$$

$$\text{Output} = 1$$

else

$$\text{Output} = 0$$



ROSENBLATT'S PERCEPTRON

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

$$\text{If } w_1x_1 + w_2x_2 > \theta$$

Output = 1

else

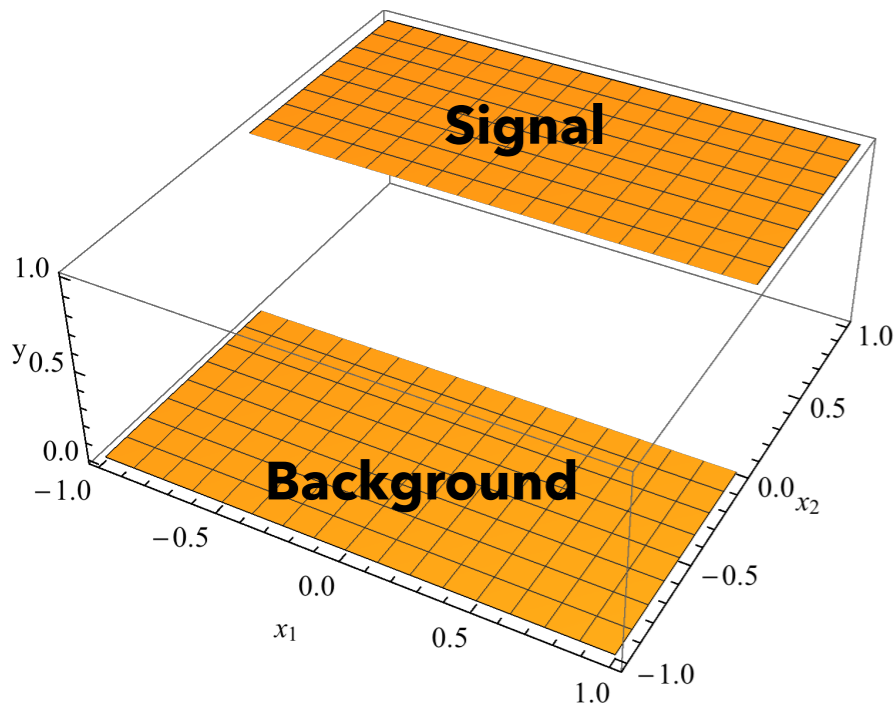
Output = 0

This is called a binary activation function, and is a generalisation of the Heaviside function to a multidimensional feature space.



ROSENBLATT'S PERCEPTRON

► Illustrative examples:

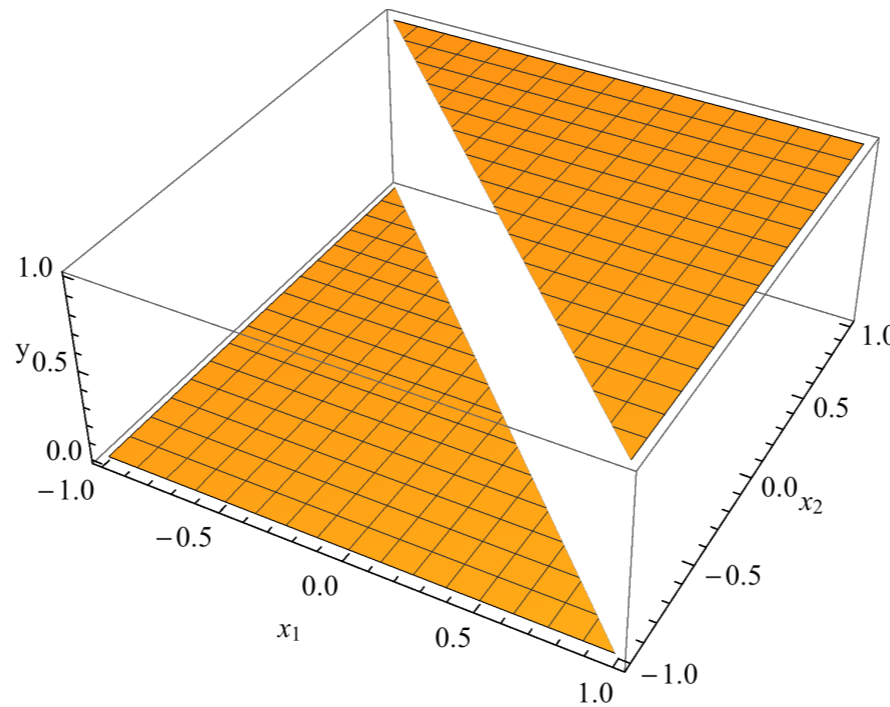


$$w_1 = 0$$

$$w_2 = 1$$

$$\theta = 0$$

Baseline for comparison,
decision only on value of x_2

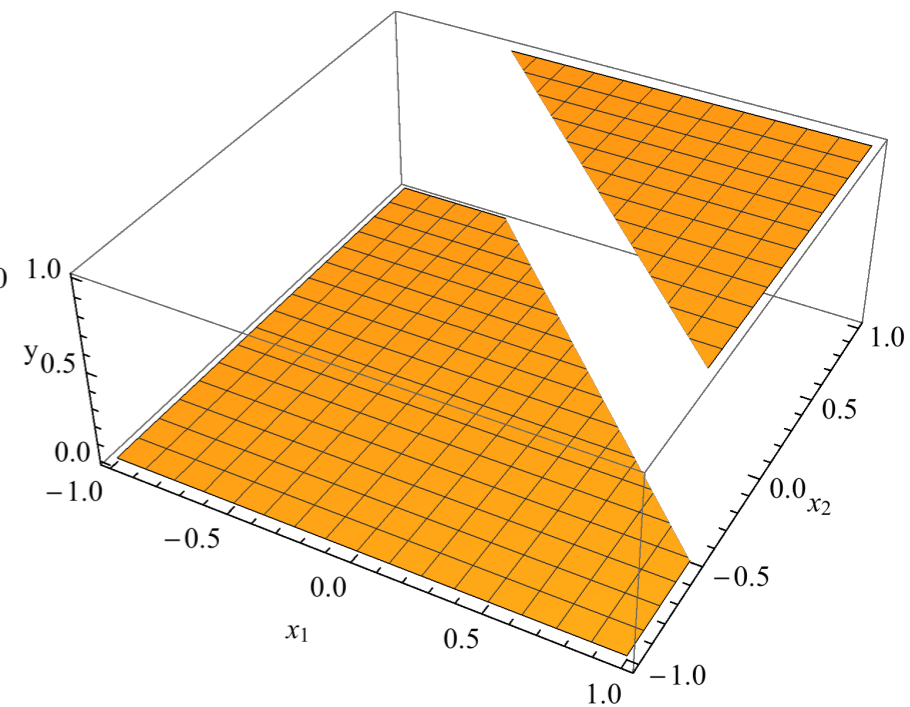


$$w_1 = 1$$

$$w_2 = 1$$

$$\theta = 0$$

Rotate decision
plane in (x_1, x_2)



$$w_1 = 1$$

$$w_2 = 1$$

$$\theta = 0.5$$

Shift decision plane
away from origin

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



ROSENBLATT'S PERCEPTRON

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

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

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

$w^T x + \theta$ is the equation of a hyperplane.



ROSENBLATT'S PERCEPTRON

- ▶ Given some training data, we can learn the hyper parameters θ for this single Rosenblatt perceptron.
- ▶ Step 1:
 - ▶ Choose the loss function and initialise the θ .
- ▶ Step 2:
 - ▶ Optimise the loss function to determine the optimal \hat{y} , corresponding to the optimal hyper parameters $\hat{\theta}$.
- ▶ Step 3:
 - ▶ Evaluate the model performance (e.g. accuracy or some other metric)



ACTIVATION FUNCTIONS

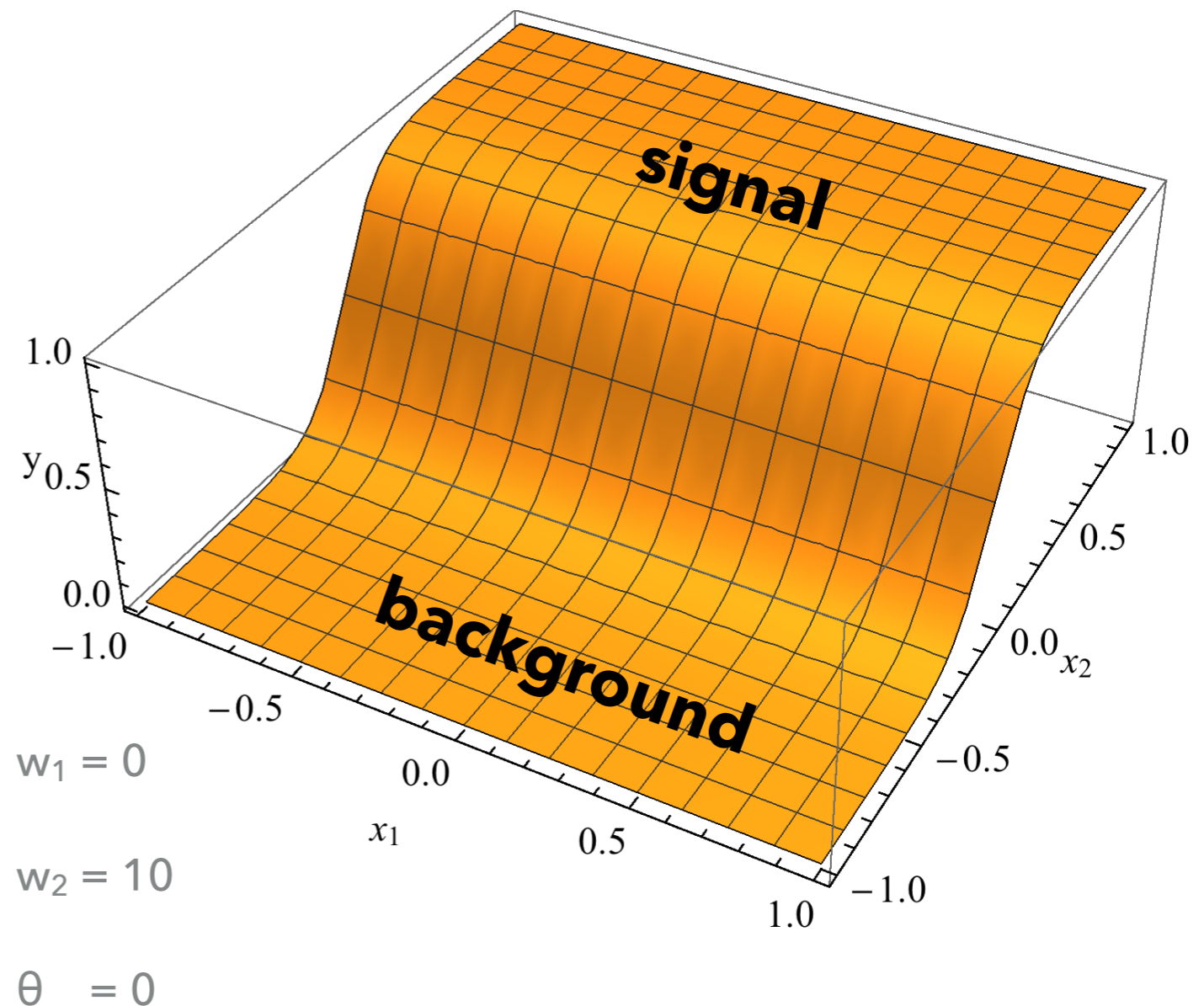
- ▶ The binary activation function of Rosenblatt is just one type of activation function.
 - ▶ This gives an all or nothing response.
- ▶ It can be useful to provide an output that is continuous between these two extremes.
 - ▶ For that we require additional forms of activation function.



ACTIVATION FUNCTIONS: LOGISTIC (OR SIGMOID)

- ▶ A common activation function used in neural networks:

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

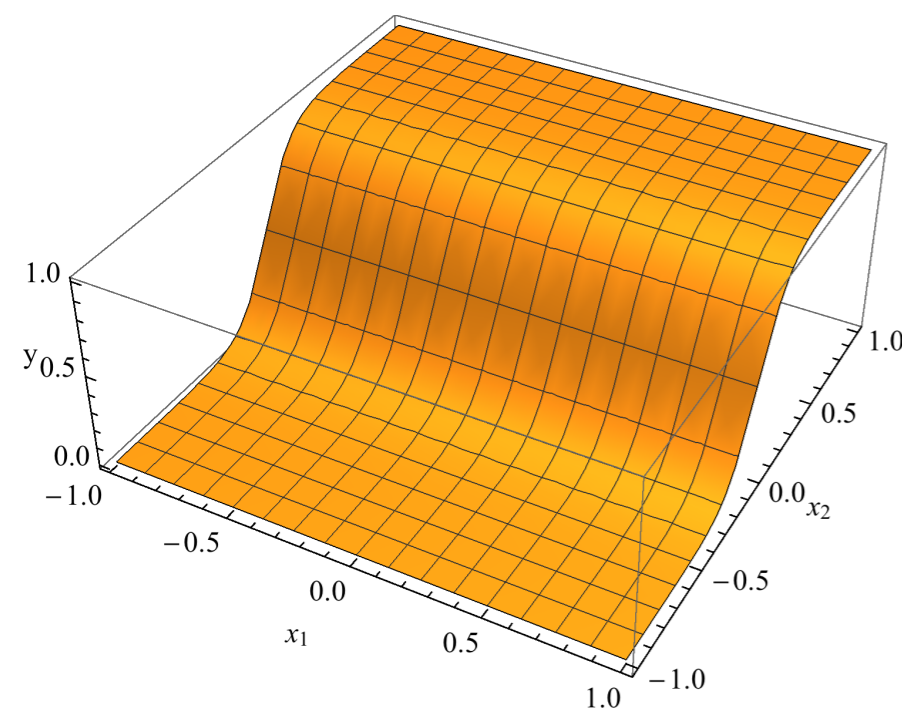




ACTIVATION FUNCTIONS: LOGISTIC (OR SIGMOID)

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

▶ A common activation function used in neural networks:

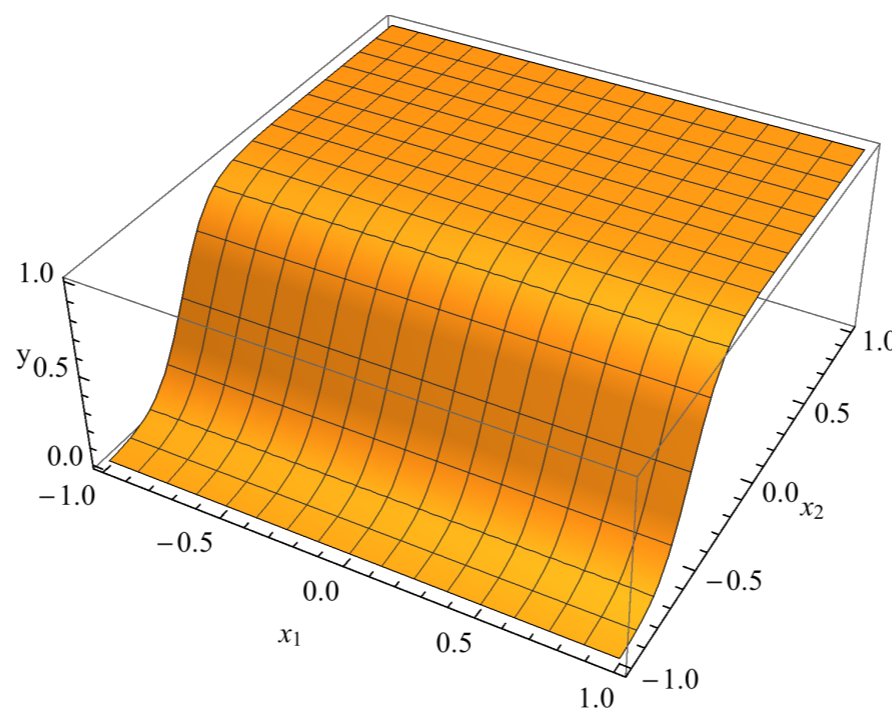


$w_1 = 0$

$w_2 = 10$

$\theta = 0$

Baseline for comparison,
decision only on value of x_2

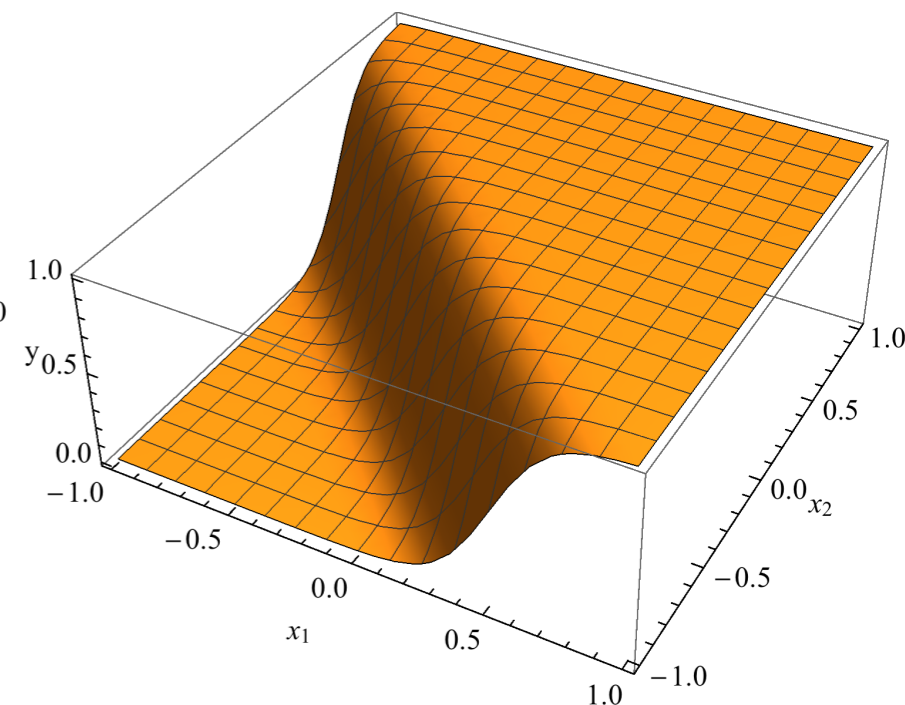


$w_1 = 0$

$w_2 = 10$

$\theta = -5$

Offset from zero using θ



$w_1 = 10$

$w_2 = 10$

$\theta = -5$

rotate "decision
boundary" in (x_1, x_2)

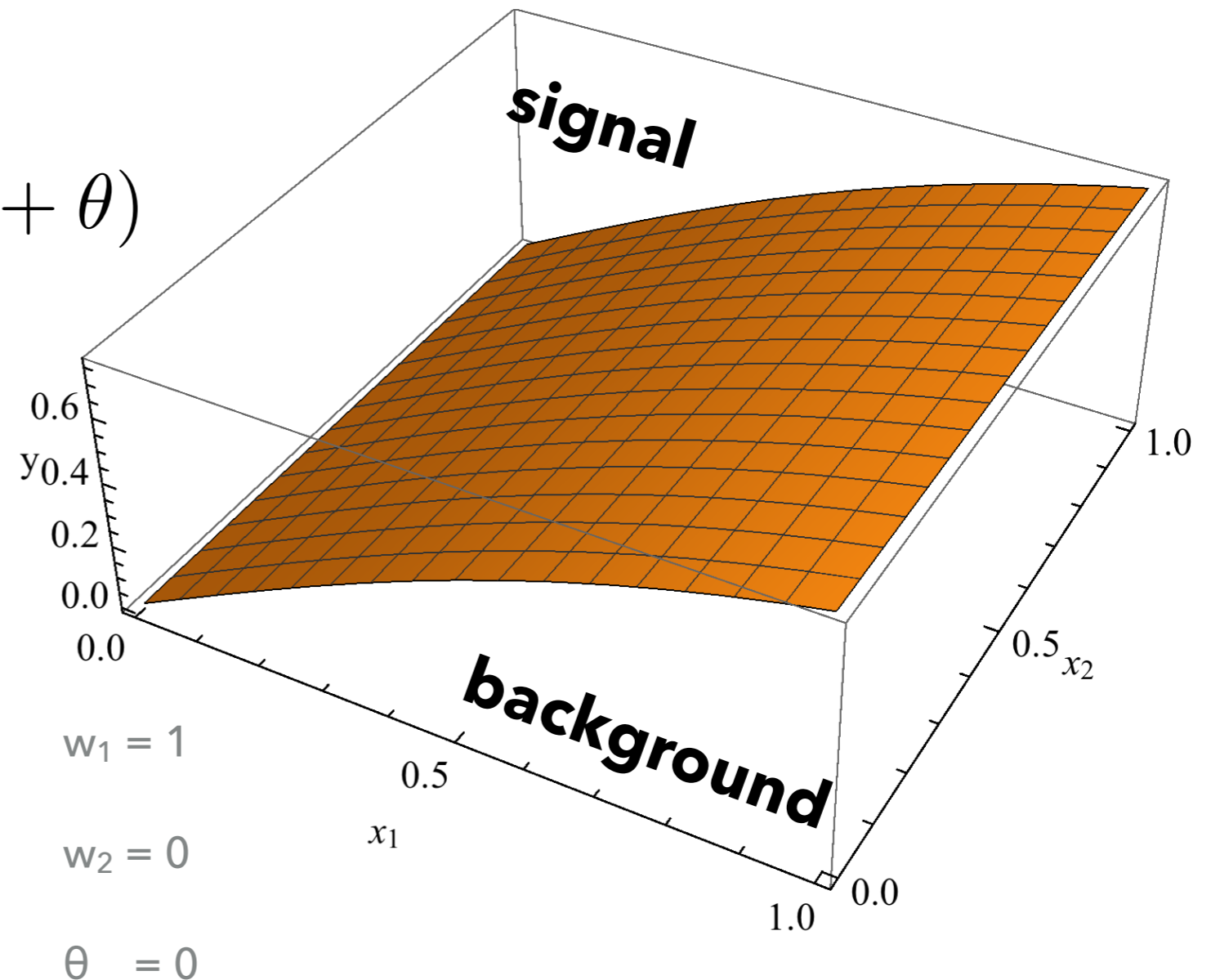


ACTIVATION FUNCTIONS: HYPERBOLIC TANGENT

- ▶ A common activation function used in neural networks:

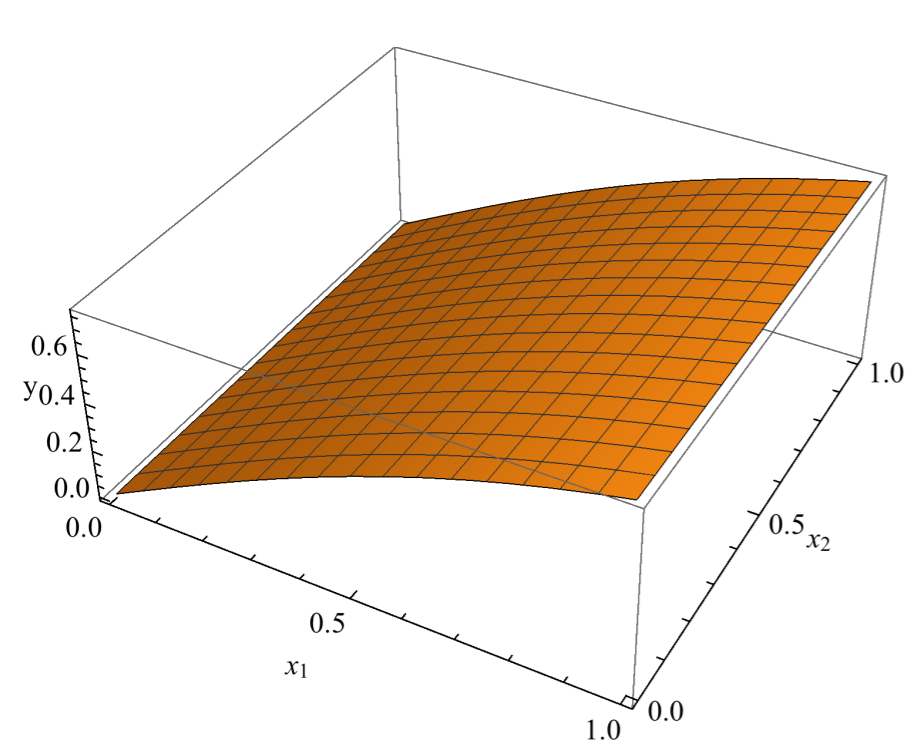
$$\begin{aligned}y &= \tanh(w^T x + \theta) \\ &= \tanh(w_1 x_1 + w_2 x_2 + \theta)\end{aligned}$$

(Often used with $\theta = 0$)



ACTIVATION FUNCTIONS: HYPERBOLIC TANGENT $\tanh(w_1x_1 + w_2x_2 + \theta)$

▶ A common activation function used in neural networks:

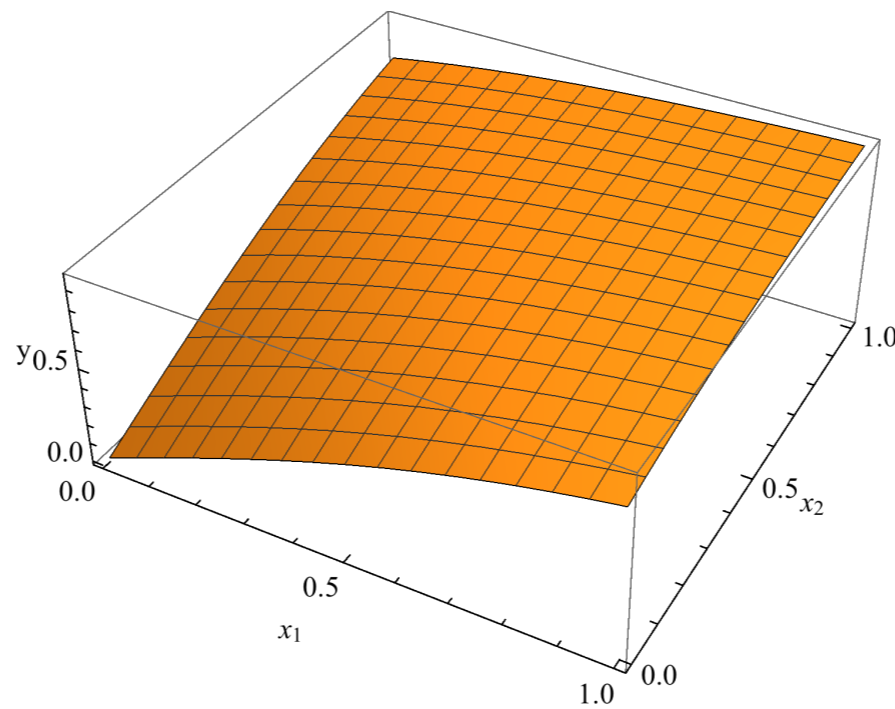


$$w_1 = 1$$

$$w_2 = 0$$

$$\theta = 0$$

Baseline for comparison,
decision only on value of x_1

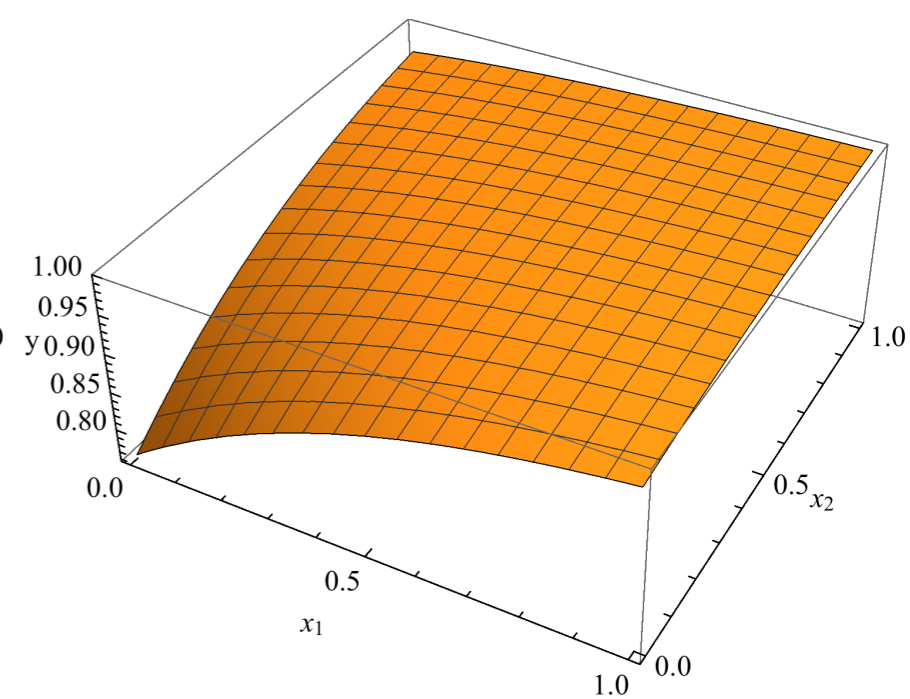


$$w_1 = 1$$

$$w_2 = 1$$

$$\theta = 0$$

rotate "decision
boundary" in (x_1, x_2)



$$w_1 = 1$$

$$w_2 = 1$$

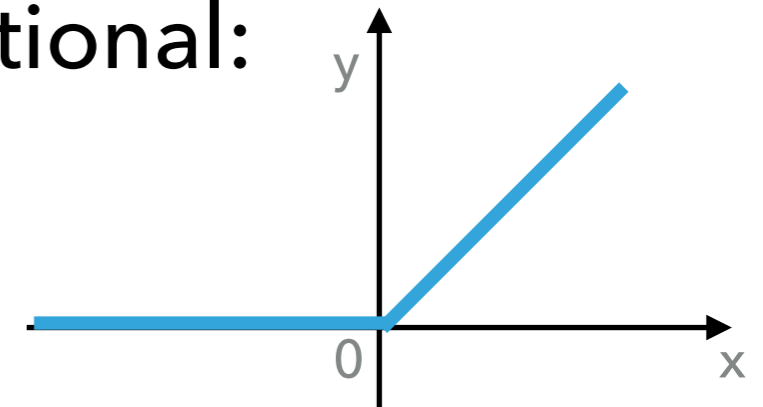
$$\theta = -1$$

Offset (vertically) from
zero using θ



ACTIVATION FUNCTIONS: RELU

- ▶ The **Rectified Linear Unit** activation function is commonly used for CNNs. This is given by a conditional:
 - ▶ If $(x < 0) y = 0$
 - ▶ otherwise $y = x$



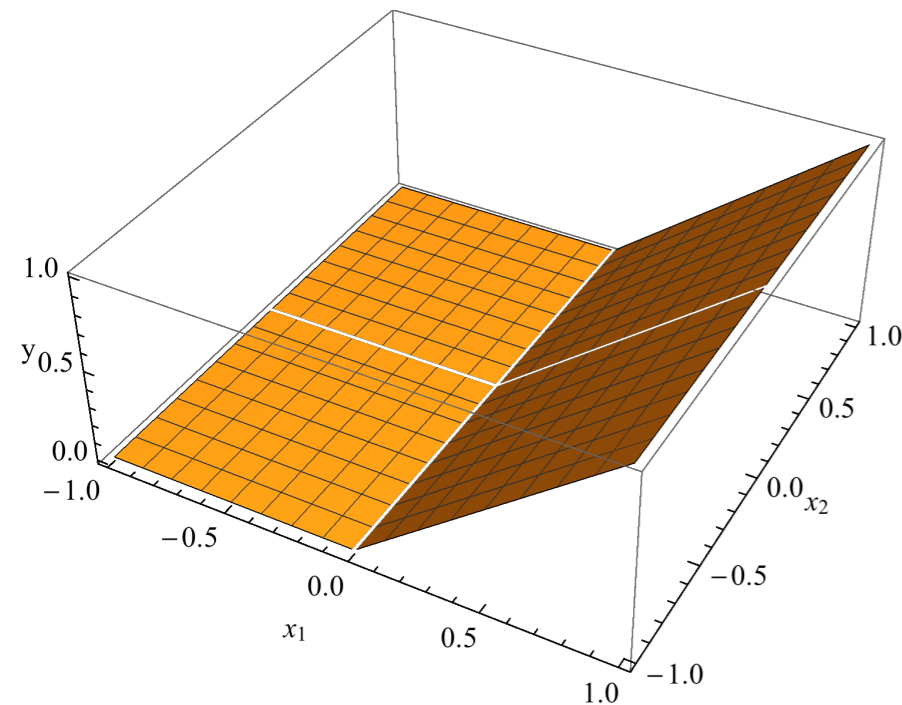
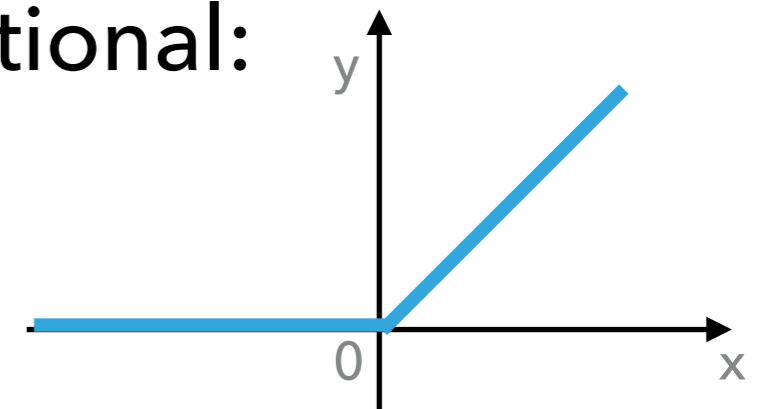


ACTIVATION FUNCTIONS: RELU

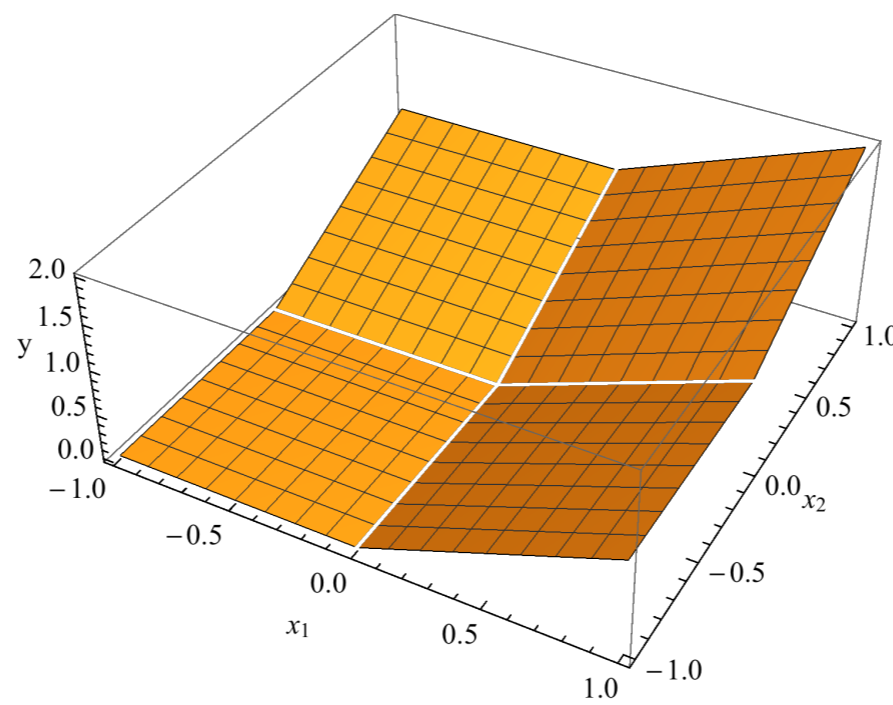
▶ The **Rectified Linear Unit** activation function is commonly used for CNNs. This is given by a conditional:

▶ If $(x < 0) y = 0$

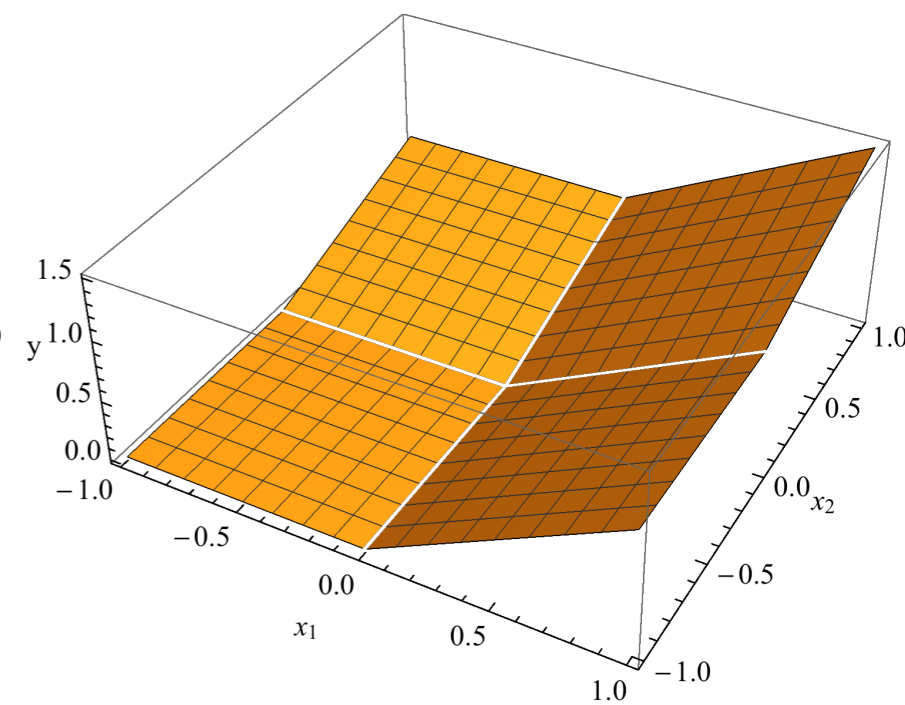
▶ otherwise $y = x$



$$w_1 = 1, w_2 = 0$$



$$w_1 = 1, w_2 = 1$$



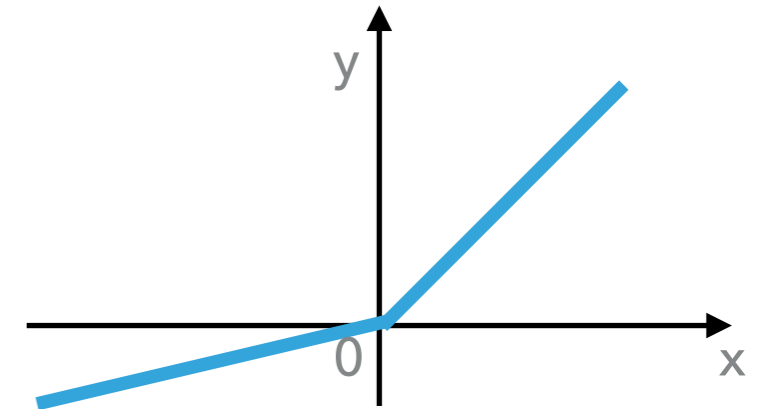
$$w_1 = 1, w_2 = 0.5$$

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

ACTIVATION FUNCTIONS: PRELU VARIANT

- ▶ The ReLU activation function can be modified to avoid gradient singularities.
- ▶ This is the **PReLU** or **Leaky ReLU** activation function

- ▶ If $(x < 0) y = a * x$
- ▶ otherwise $y = x$



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

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

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

[1] Maas, Hannun, Ng, [ICML2013](#).

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

[3] See https://github.com/tensorflow/tensorflow/blob/r2.2/tensorflow/python/ops/nn_ops.py



ACTIVATION FUNCTIONS: RELU

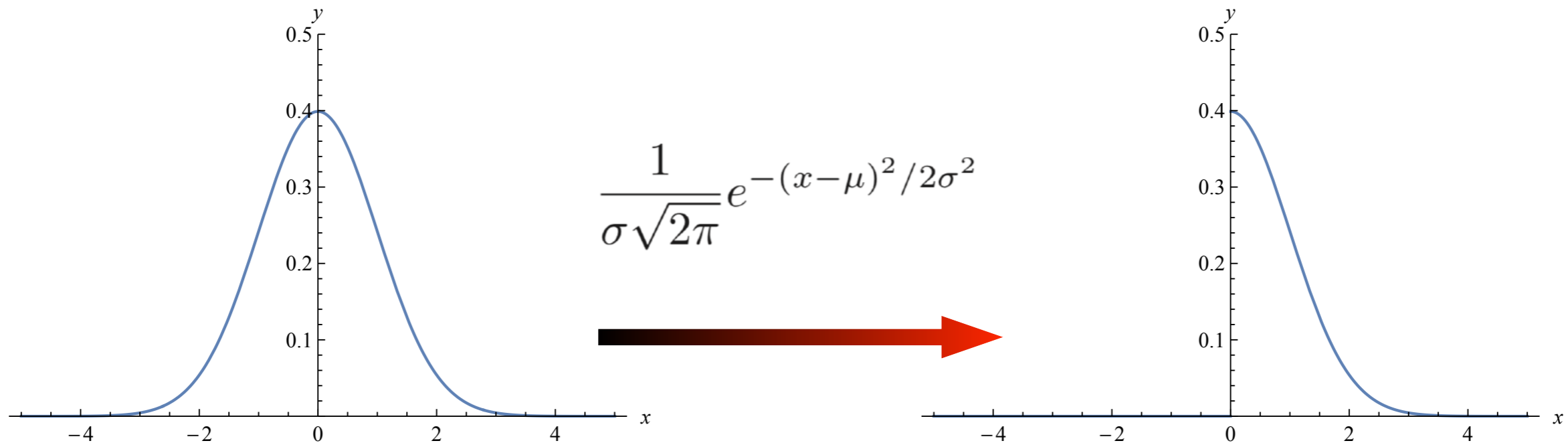
- ▶ Performs better than a sigmoid for a number of applications^[1].
- ▶ Weights for a relu are typically initialised with a truncated normal, OK for shallow CNNs, but there are convergence issues with deep CNNs when using this initialisation approach^[1].
- ▶ Other initialisation schemes have been proposed to avoid this issue for deep CNNs (more than 8 conv layers) as discussed in Ref [2].

^[1] Maas, Hannun, Ng, [ICML2013](#).

^[2] He, Zhang, Ren and Sun, [arXiv:1502.01852](#)

ACTIVATION FUNCTIONS: RELU

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

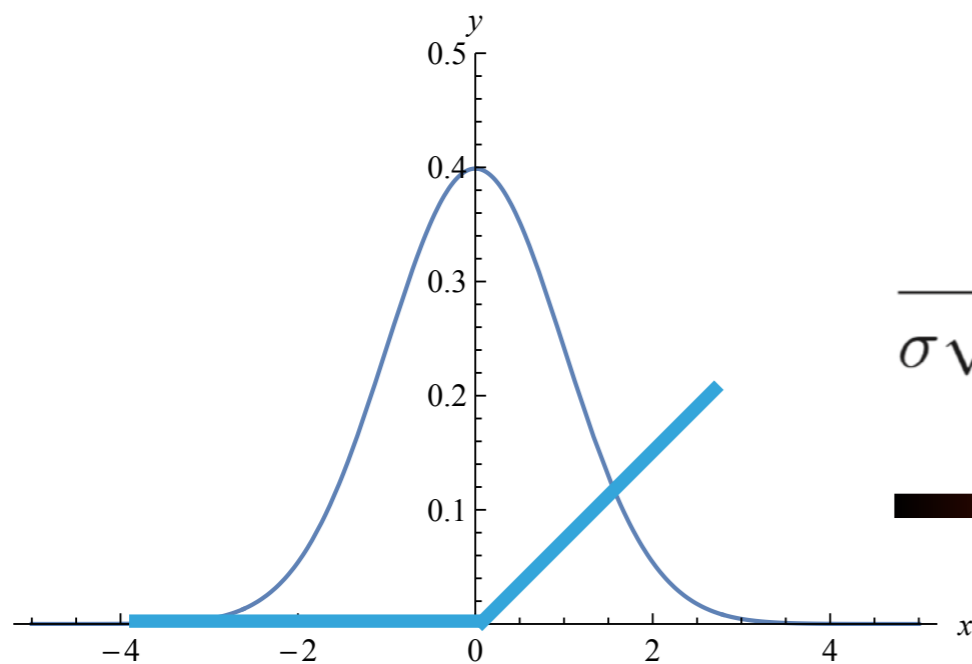


[1] Maas, Hannun, Ng, [ICML2013](#).

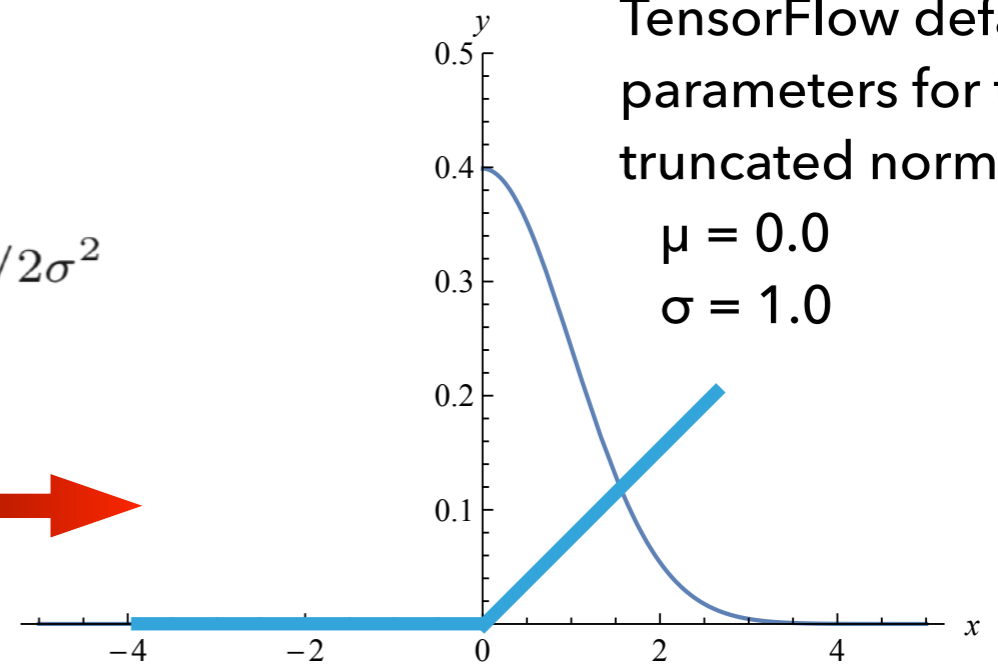
[2] He, Zhang, Ren and Sun, [arXiv:1502.01852](#)

ACTIVATION FUNCTIONS: RELU

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



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



TensorFlow default parameters for the truncated normal are:

$$\mu = 0.0$$

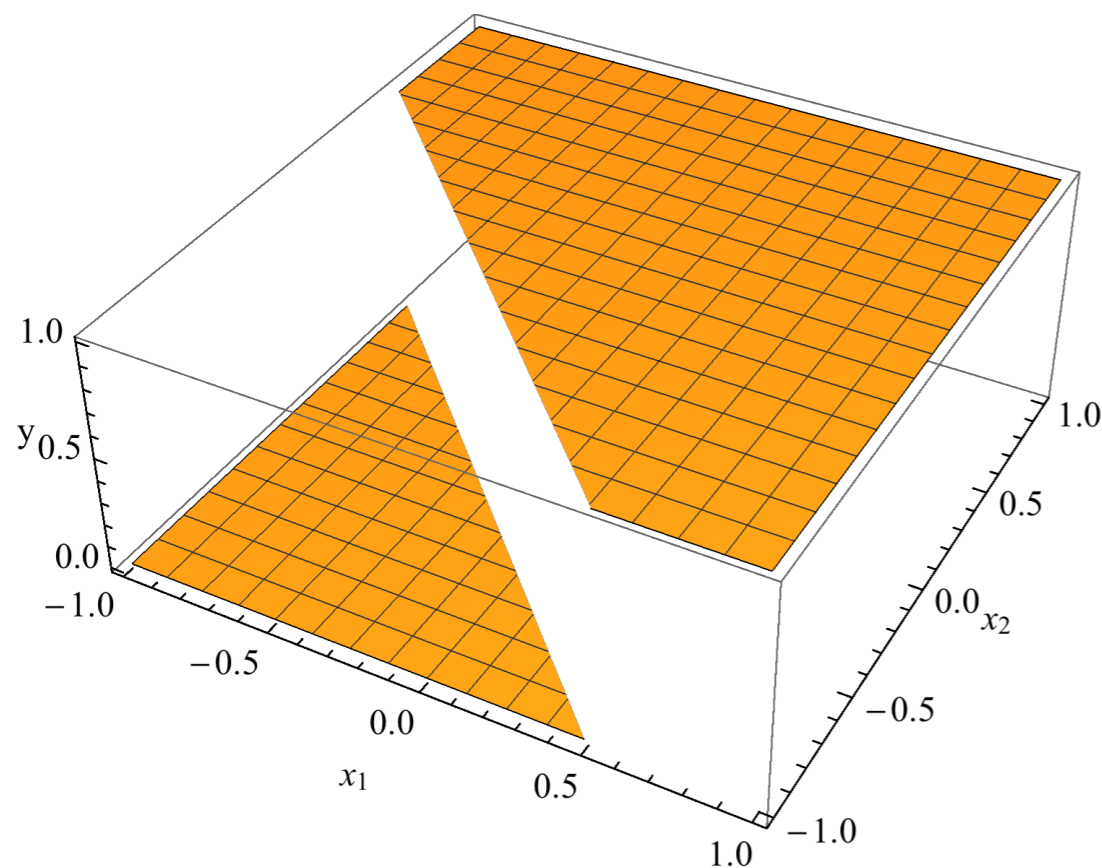
$$\sigma = 1.0$$

[1] Maas, Hannun, Ng, [ICML2013](#).

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

ARTIFICIAL NEURAL NETWORKS (ANNs)

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



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

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

or a node in an oblique decision tree.

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



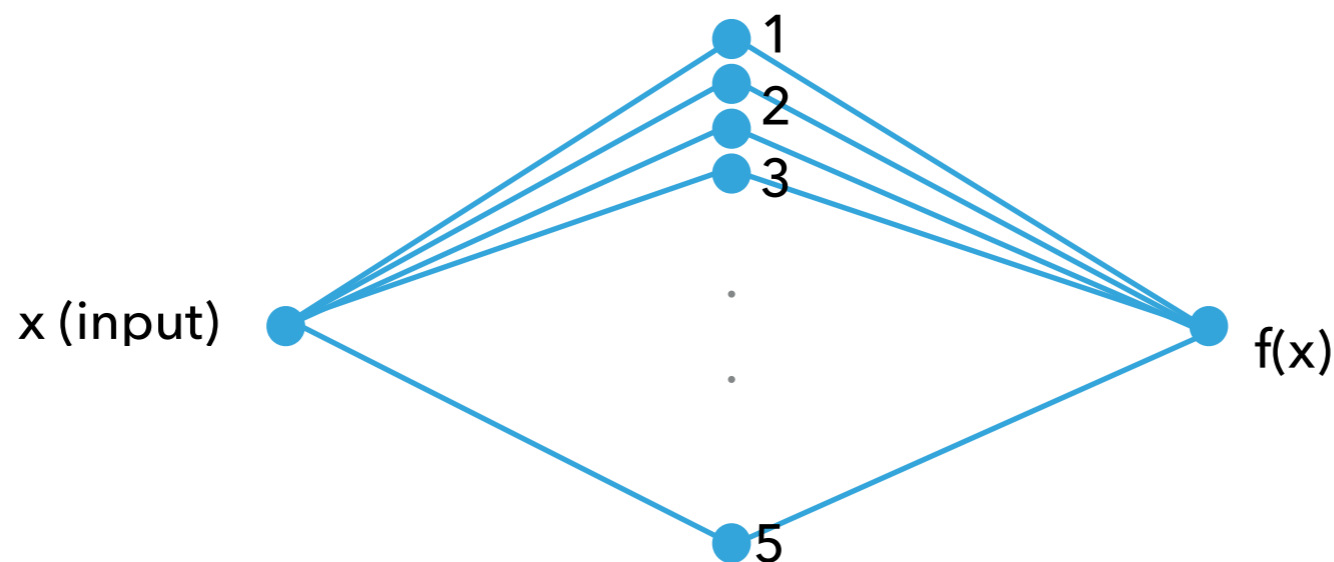
ARTIFICIAL NEURAL NETWORKS (ANNs)

- ▶ A single perceptron can be thought of as defining a hyperplane that separates the input feature space into two regions.
 - ▶ This is a literal illustration for the binary threshold perceptron.
 - ▶ The other perceptrons discussed have a gradual transition from one region to the other.
- ▶ We can combine perceptrons to impose multiple hyperplanes on the input feature space to divide the data into different regions.
- ▶ Such a system is an artificial neural network.



ARTIFICIAL NEURAL NETWORKS (ANNs)

- ▶ The simplest general ANN has one input node, one output node and some number of hidden layer nodes.



Approximation by Superpositions of a Sigmoidal Function*

G. Cybenko†

Abstract. In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. Our results settle an open question about representability in the class of single hidden layer neural networks. In particular, we show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity. The paper discusses approximation properties of other possible types of nonlinearities that might be implemented by artificial neural networks.

Key words. Neural networks, Approximation, Completeness.

The Universal Function Approximation theorem states that neural networks under some assumptions can approximate any continuous functions on compact subsets of an n -dimensional hyperspace of real numbers.

BACK PROPAGATION

- ▶ For feed forward neural networks like the one described here, we can use the back-propagation method to update the weight and bias HPs in the network.
 - ▶ Require differentiable activation functions and loss function (error, E).
 - ▶ Feed the input feature space vector x through the network to compute the output, and hence E .
 - ▶ The derivatives of E with respect to the HPs, ∇E , for this example can then be used to find the set of HPs corresponding to the minimum loss (or error) of the network using some optimisation procedure, e.g. gradient descent:

$$w_{ij}(t + 1) = w_{ij}(t) - \alpha \frac{\partial E}{\partial w_{ij}}$$

t the training epoch
 i is the i^{th} node in layer I of the network
 j is the j^{th} node in layer K of the network

- ▶ The chain rule is used to efficiently compute the set of $\partial E / \partial w_{ij}$ required for the network, so that we can then update the weights in the network.
- ▶ N.B. The term back propagation is also used to describe the process of using gradient descent (also known as the delta rule^[1]) for minimising the L_2 loss with a neural network (Stage 1+2 for training a neural network).

This is for stochastic training, and this is adapted for batch training by considering the sum over the examples in a batch or mini batch.

See, for example, C. Bishop, *Neural Networks for Pattern Recognition*, Ch. 4. and Y. LeCun et al., [Efficient BackProp](#), *Neural Networks Tricks of the Trade*, Springer 1998 (Fig. 3).

[1] Widrow and Hoff (1960),



OTHER REMARKS

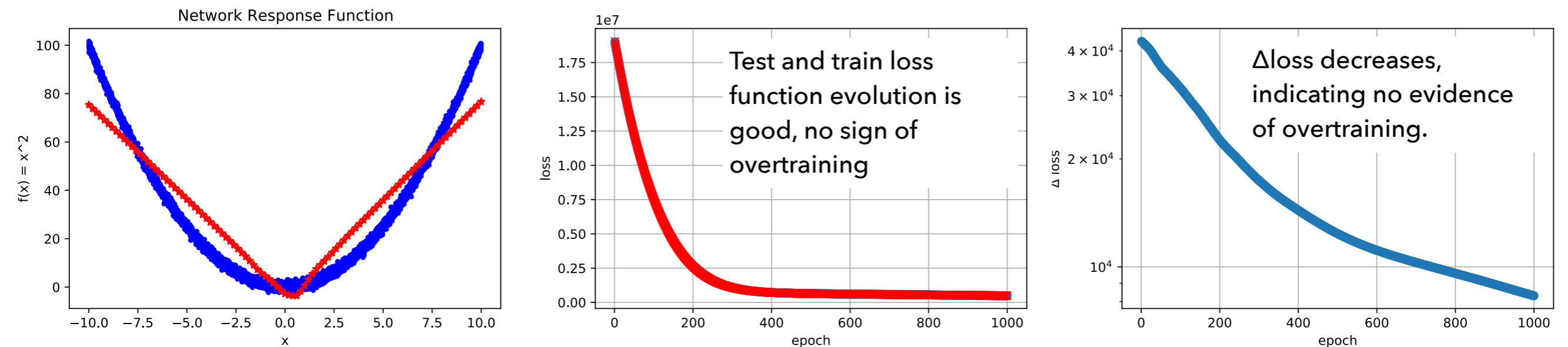
- ▶ When performing stochastic training, the optimisation converges faster when presenting unexpected examples.
- ▶ A pragmatic way to do this is to alternate the presentation of examples from different classes of event to the optimisation.
- ▶ Ensuring equal numbers of different classes are presented to the optimisation will ensure that the learned model has been able to focus equally on distinguishing different types of event.
- ▶ If you feed in different amounts of training data then this may not be helpful: aim to use equal amounts of training samples for the different types.
 - ▶ e.g. a dominant background like $t\bar{t}$ with very little signal, then the model learned will be driven by the dominant contribution to the loss function. This is invariably the example type with the dominant number of examples presented.
 - ▶ Changing the weighting of the different components will affect the way the algorithm learns.

EXAMPLE: FUNCTION APPROXIMATION

- ▶ Consider the model $y = x^2$
- ▶ This can be learned using an ANN, with a single input, x , and a single output y for each example.
- ▶ Model:
 - ▶ ADAM optimiser
 - ▶ 1 hidden layer with 50 nodes
 - ▶ ReLU activation function for nodes in the hidden layer
 - ▶ $w^T x + \beta$ for the output node (to give an unlimited real valued output)
- ▶ Train with:
 - ▶ 1000 epochs.
 - ▶ Gaussian noise overlaid to “simulate” realistic data
 - ▶ $\alpha = 0.001$
 - ▶ 10k examples randomly generated in $x \in [-10,10]$ for training, and 10k for testing
 - ▶ L_2 loss

EXAMPLE: FUNCTION APPROXIMATION

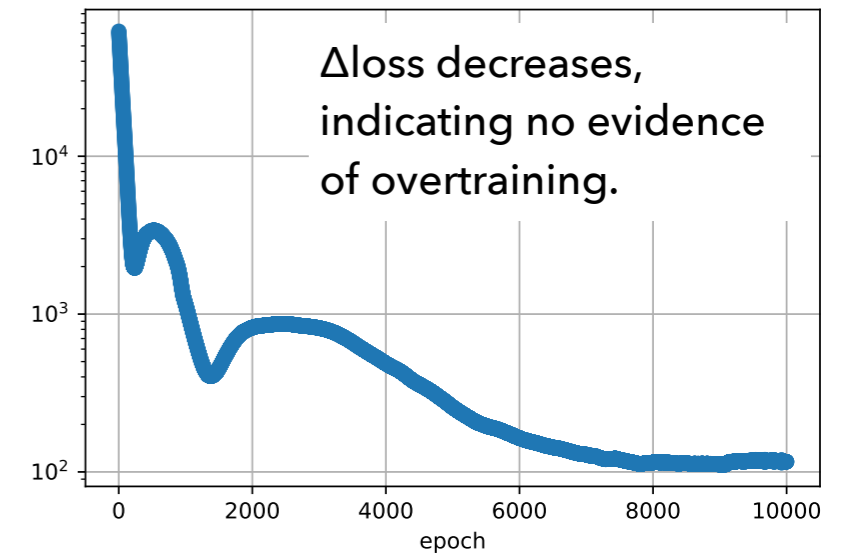
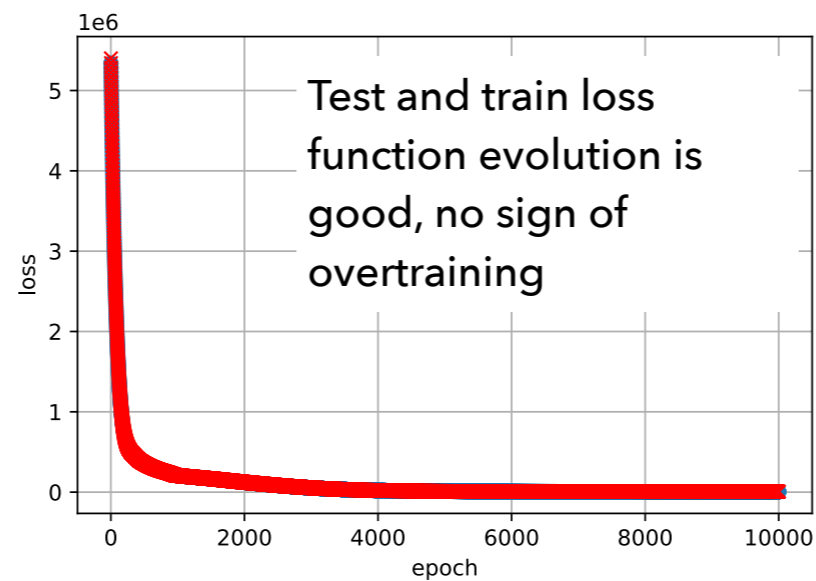
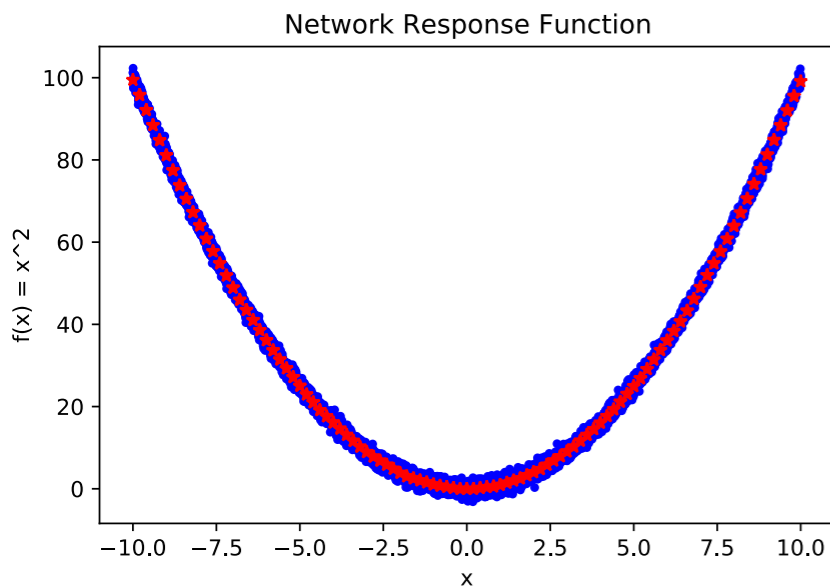
- ▶ The model obtained does not give a good prediction.



- ▶ Increasing or removing the Gaussian noise will not lead to improvement.
- ▶ Increasing the number of epochs or number of training data can yield an improvement.
- ▶ Changing the network architecture can lead to an improvement.

EXAMPLE: FUNCTION APPROXIMATION

- ▶ As before, but now with 10k training epochs.



- ▶ Able to learn a good model to approximate the target function.
- ▶ Can increase the learning rate to learn faster.
- ▶ ADAM has built in learning rate decay parameters, so a learning rate of 0.1 will yield a good training with only 1000 epochs; c.f. the previous page with $\alpha = 0.001$.

MULTILAYER PERCEPTRONS

... AS DEEP NETWORKS

EXAMPLE: JET FLAVOUR CLASSIFICATION

EXAMPLE: FUNCTION APPROXIMATION

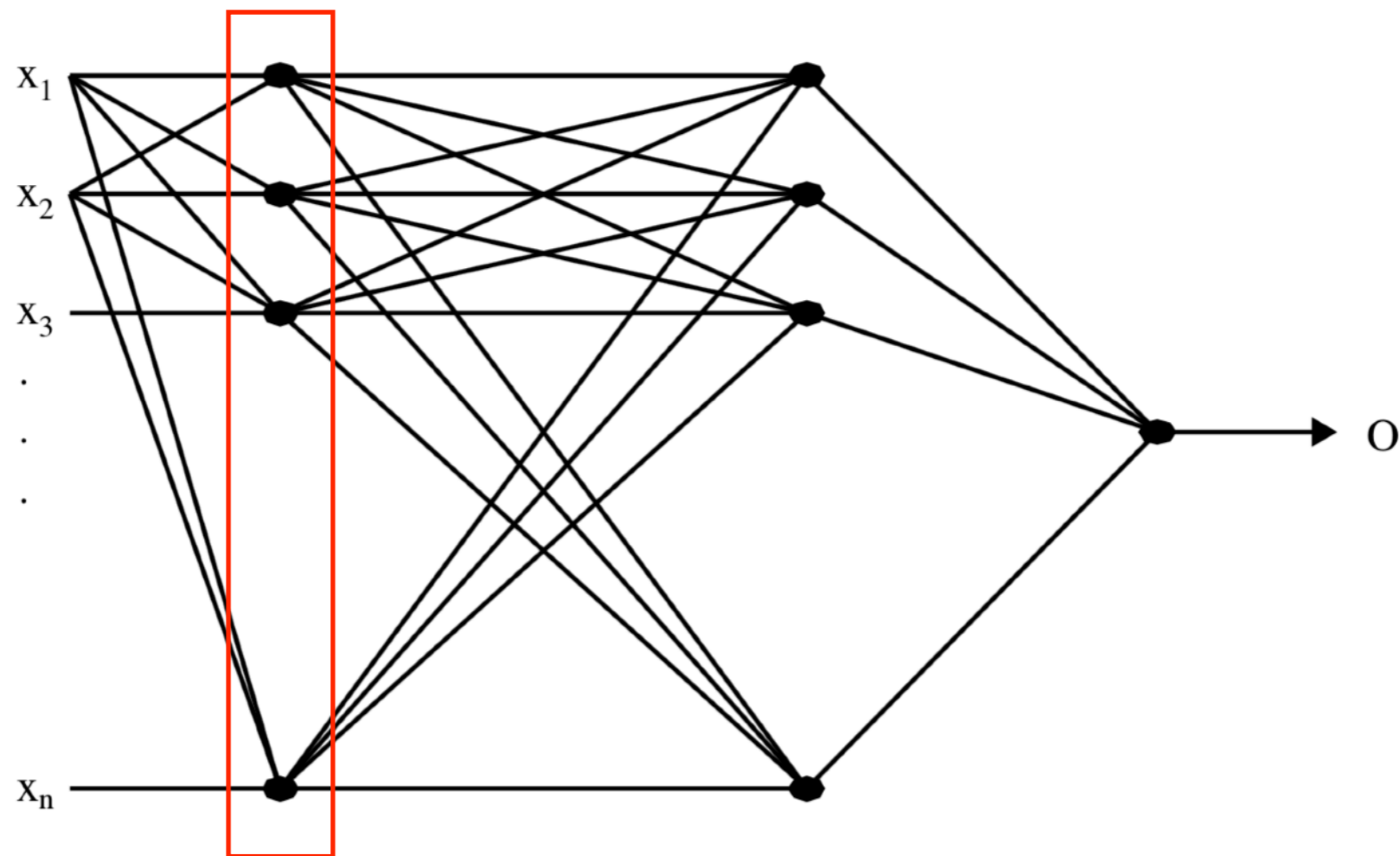
NEURAL NETWORKS

MULTI LAYER PERCEPTRONS



MULTI LAYER PERCEPTRONS

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

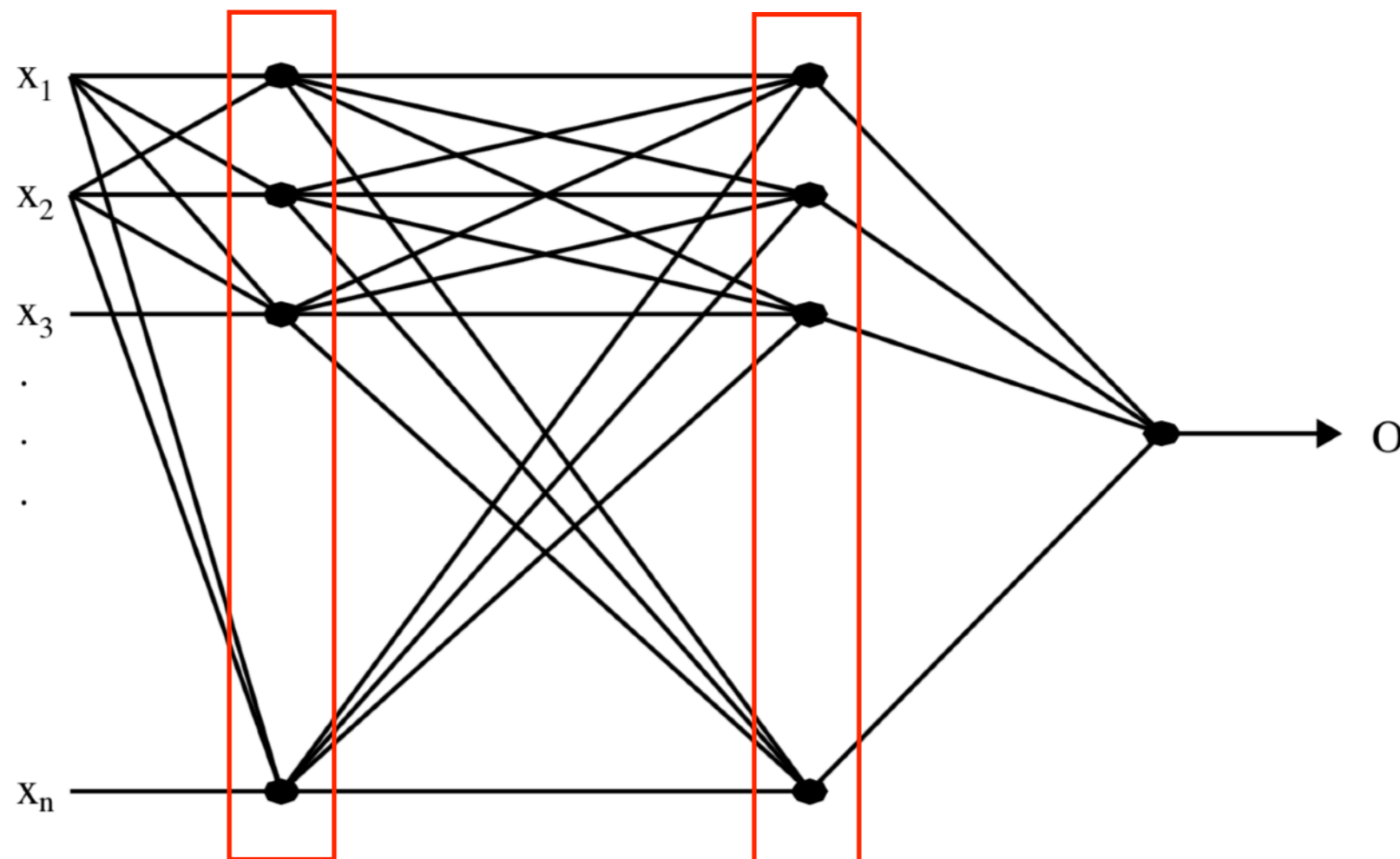


Input layer of n perceptrons;
one for each dimension of the
input feature space



MULTI LAYER PERCEPTRONS

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

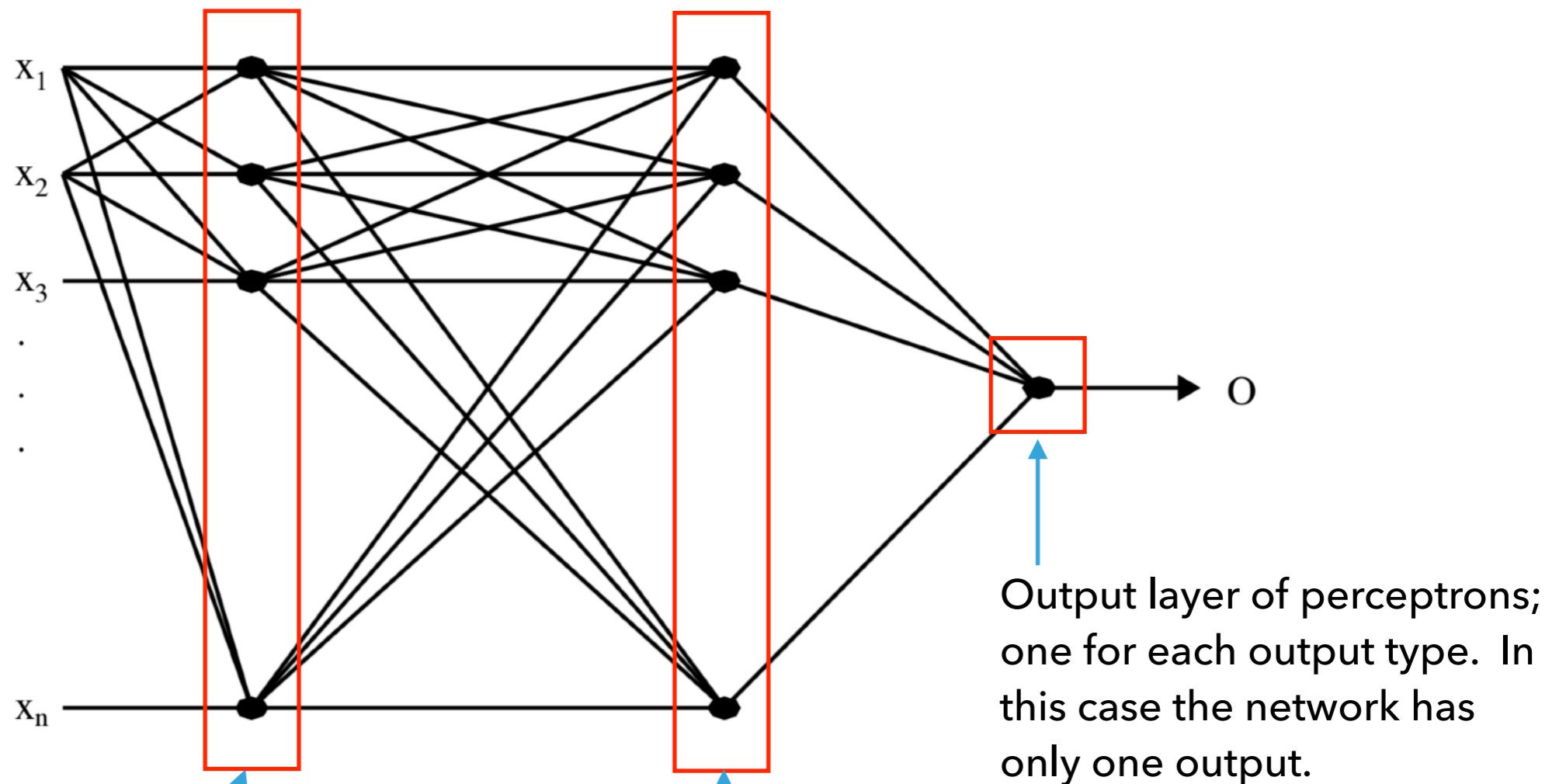


Input layer of n perceptrons; one for each dimension of the input feature space

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

MULTI LAYER PERCEPTRONS

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



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

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



MULTI LAYER PERCEPTRONS AS DEEP NETWORKS

- ▶ The term deep network does not have a fixed definition.
- ▶ Examples in the literature range from having more than 2 hidden layers, or having a very large number of nodes in a network.
 - ▶ We have been using deep network model configurations in HEP for decades.
- ▶ The key point, that is sometimes overlooked, is this:
 - ▶ The network is trained to a large number of epochs, so that the model learned can take advantage of subtleties of the data.
- ▶ To avoid overtraining the model, deep networks generally require large training sets, and hence have a significant computational expense.



EXAMPLE: JET FLAVOUR CLASSIFICATION

- ▶ Quark colour confinement leads to the creation of jets as pairs of quarks and anti-quarks are pulled apart.
 - ▶ As a result we don't see bare quarks.
 - ▶ However we do see many hadrons eliminating from some underlying quark.
 - ▶ These hadrons form objects called jets that are reconstructed in our detectors.
 - ▶ The nature of the underlying quark is of interest as knowing that allows us to infer something about an underlying interaction; e.g. the decay of a Higgs boson to two b-quarks requires that we accurately identify events with two (or more) b jets.

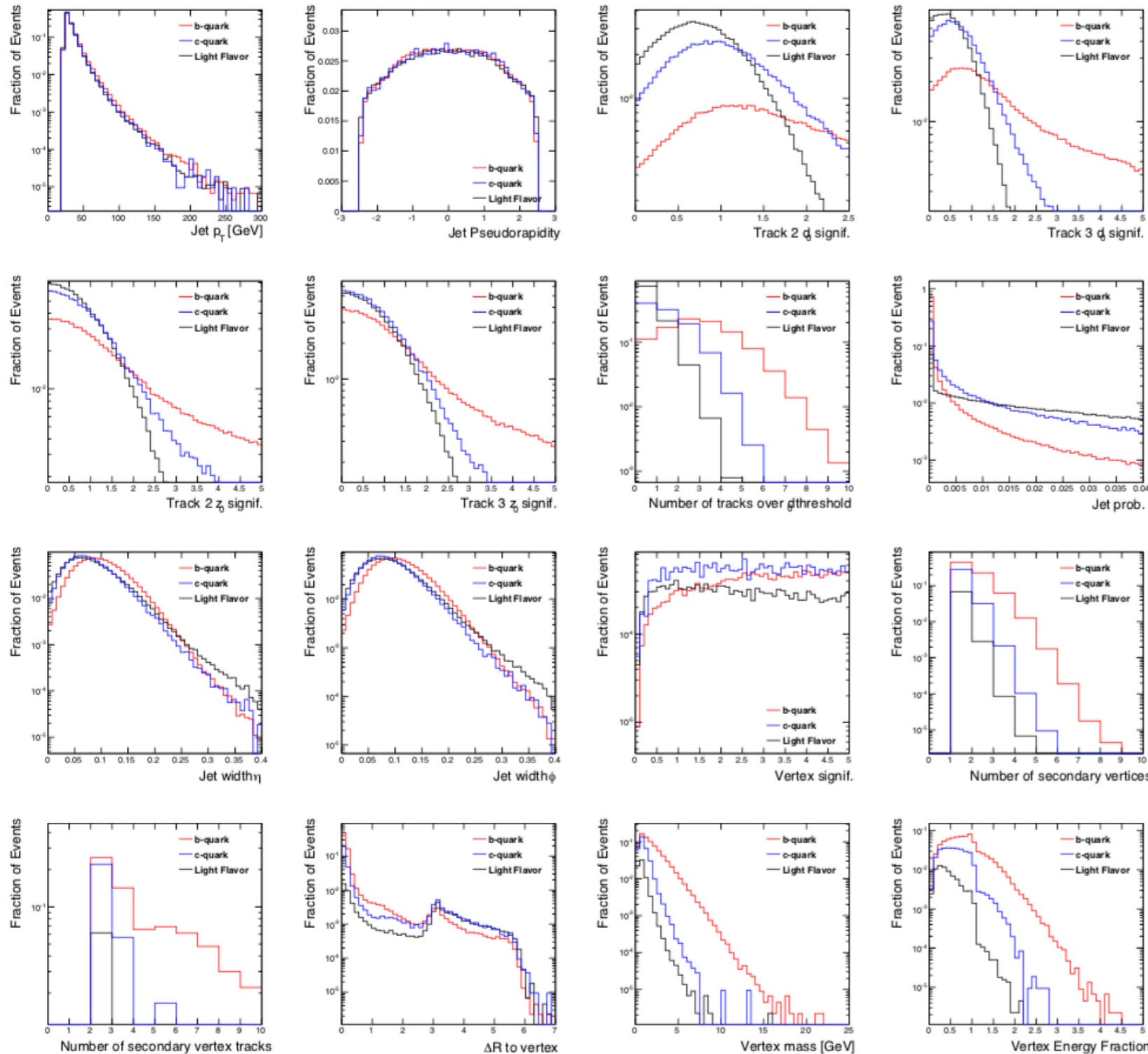


EXAMPLE: JET FLAVOUR CLASSIFICATION

- ▶ Consider jet identification in pp collisions at the LHC.
- ▶ Vertex, track and calorimeter information are used to identify jets.
- ▶ Aim: separate jets into:
 - ▶ light quarks (u, d, s);
 - ▶ charm;
 - ▶ beauty.
- ▶ Guest's study uses the anti-kt algorithm for jet reconstruction and FastJet.
- ▶ 8 million jets for training, 1 million for testing and 1 million for validation.



EXAMPLE: JET FLAVOUR CLASSIFICATION



- The d_0 and z_0 significance of the 2nd and 3rd tracks attached to a vertex, ordered by d_0 significance.
- The number of tracks with d_0 significance greater than 1.8σ .
- The JETPROB [32] light jet probability, calculated as the product over all tracks in the jet of the probability for a given track to have come from a light-quark jet.

- The width of the jet in η and ϕ , calculated for η as

$$\left(\frac{\sum_i p_{Ti} \Delta\eta_i^2}{\sum_i p_T} \right)^{1/2}$$

and analogously for ϕ .

- The combined vertex significance,

$$\frac{\sum_i d_i / \sigma_i^2}{\sqrt{\sum_i 1 / \sigma_i^2}}$$

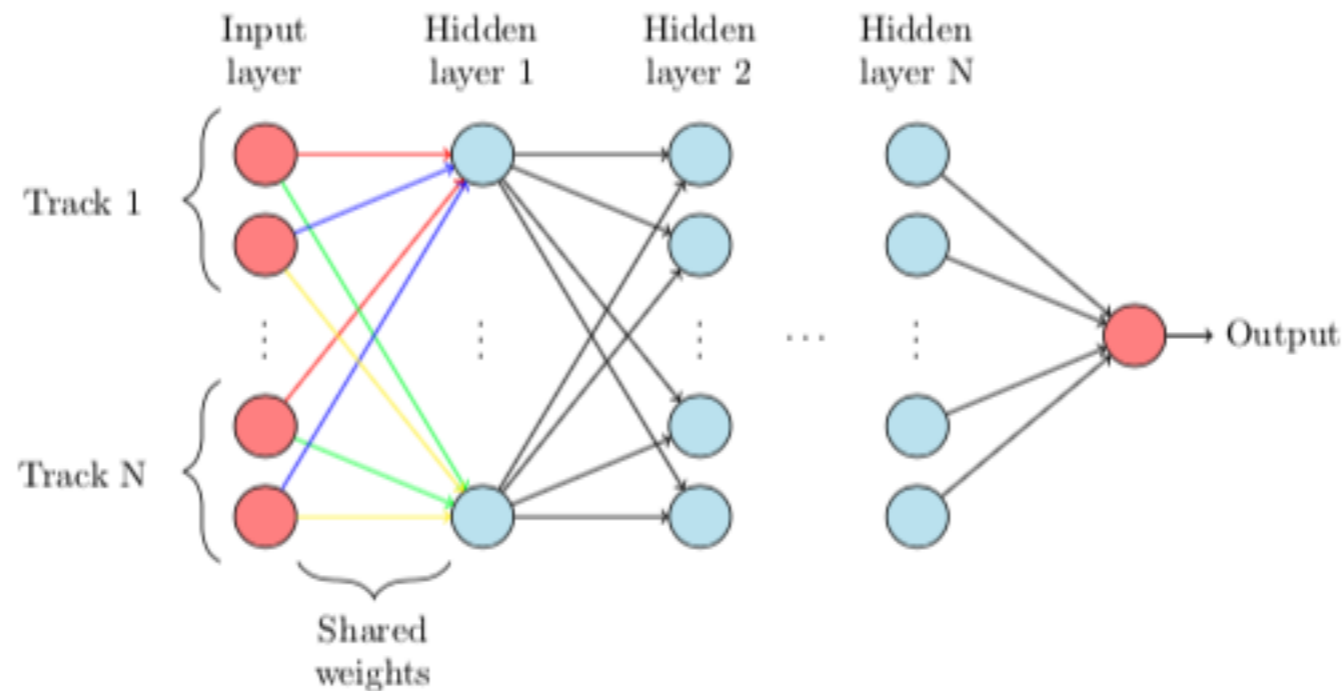
where d is the vertex displacement and σ is the uncertainty in vertex position along the displacement axis.

- The number of secondary vertices.
- The number of secondary-vertex tracks.
- The angular distance ΔR between the jet and vertex.
- The decay chain mass, calculated as the sum of the invariant masses of all reconstructed vertices, where particles are assigned the pion mass.
- The fraction of the total track energy in the jet associated to secondary vertices ¹



EXAMPLE: JET FLAVOUR CLASSIFICATION

- ▶ Several different algorithms are used including MLPs

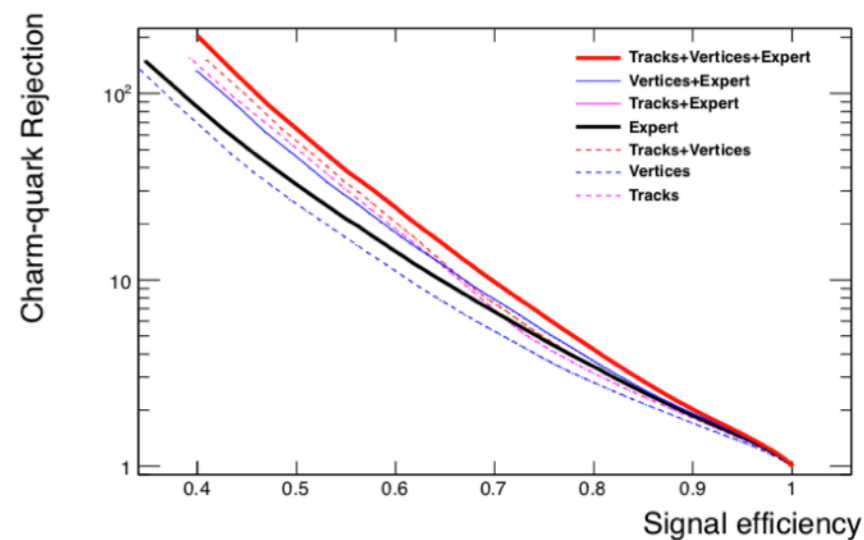
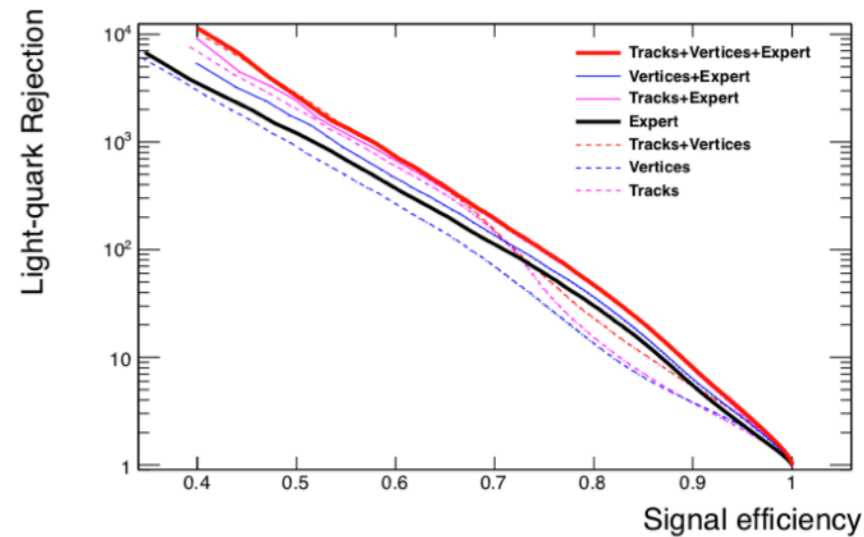


- ▶ “Experts” are networks that are trained to address a specific issue. This study constructs “Experts” that are used as inputs to a final network.
- ▶ This is an example of a committee machine.

EXAMPLE: JET FLAVOUR CLASSIFICATION

- ▶ Several different algorithms are used including MLPs

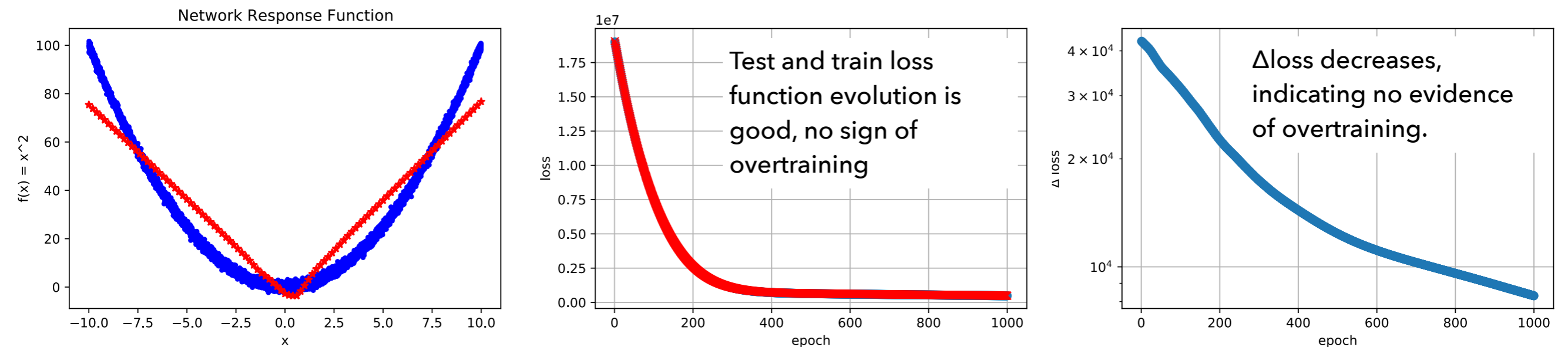
Inputs		Technique	AUC
Tracks	Vertices		
✓		Feedforward	0.916
✓		LSTM	0.917
✓		Outer	0.915
	✓	Feedforward	0.912
	✓	LSTM	0.911
	✓	Outer	0.911
✓	✓	Feedforward	0.929
✓	✓	LSTM	0.929
✓	✓	Outer	0.928
		✓ Feedforward	0.924
		✓ LSTM	0.925
		✓ Outer	0.924
✓		✓ Feedforward	0.937
✓		✓ LSTM	0.937
✓		✓ Outer	0.936
	✓	✓ Feedforward	0.931
	✓	✓ LSTM	0.930
	✓	✓ Outer	0.929
✓	✓	✓ Feedforward	0.939
✓	✓	✓ LSTM	0.939
✓	✓	✓ Outer	0.937



- ▶ They give similar performance.
- ▶ CMS has followed a deep network approach to jet tagging in their latest work (e.g. $H \rightarrow bb$) [see A.M. Sirunyan et al 2018 JINST 13 P05011, CMS PAS HIG-18-016].

EXAMPLE: FUNCTION APPROXIMATION (RECAP)

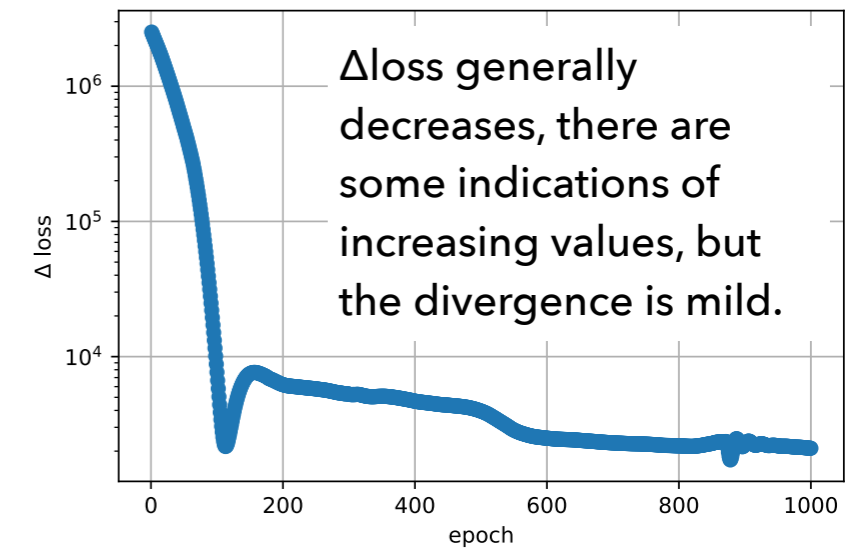
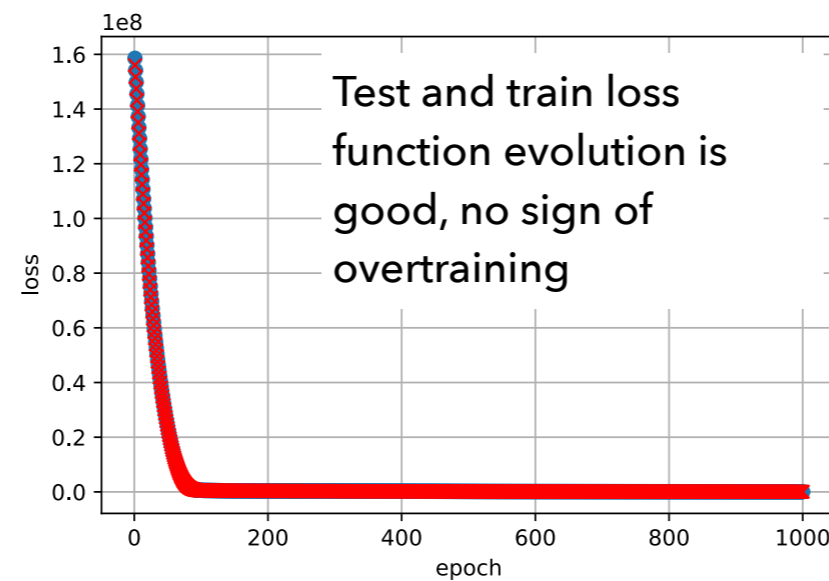
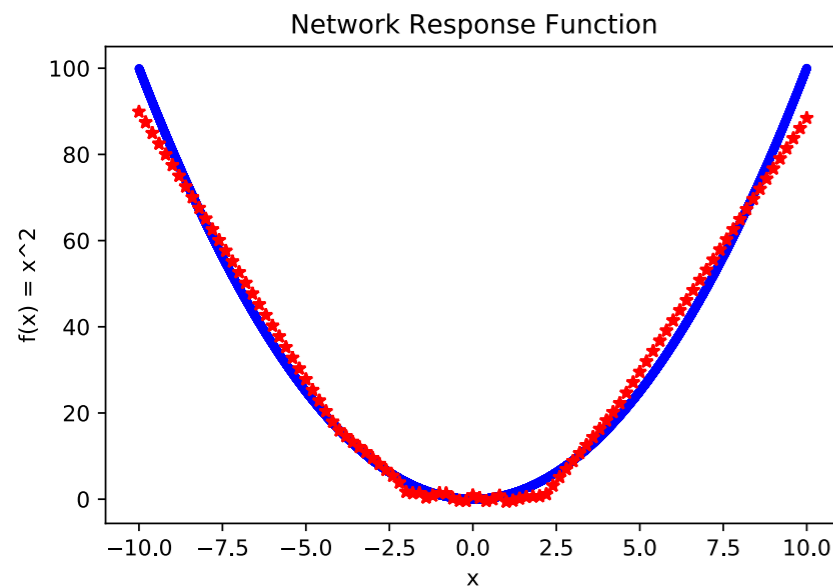
- ▶ The model obtained does not give a good prediction.



- ▶ Increasing or removing the Gaussian noise will not lead to improvement.
- ▶ Increasing the number of epochs or number of training data can yield an improvement.
- ▶ Changing the network architecture can lead to an improvement.

EXAMPLE: FUNCTION APPROXIMATION

- ▶ Now repeat with an MLP: 1:50:50:1 (2 hidden layers each with 50 nodes)



- ▶ The 2-hidden layer architecture is able to fit the function much better than the single layer architecture.
- ▶ Increasing the number of epochs or number of training data can yield a further improvement.

AUTO-ENCODERS



AUTO-ENCODERS

- ▶ Auto-encoders (AEs) Can be used to implicitly learn fundamental representations of underlying features of the data to facilitate:
 - ▶ Noise removal (e.g. de-noising auto-encoder)
 - ▶ Dimensional reduction



USING AUTO-ENCODERS

- ▶ Sometimes it is difficult to specify what features need to be extracted from input data to solve a particular problem, as the type of interest can manifest itself differently under different scenarios.
- ▶ e.g. Special Relativity: Properties depend on the frame of reference;
 - ▶ Data represented via Lorentz invariants provide a clear picture of the existence of underlying particles via the mass spectrum.
- ▶ In analogy AEs can learn representations of the data.
- ▶ They have two parts:
 - ▶ An **encoder** that maps input features into a different representation;
 - ▶ A **decoder** that is used to convert back into the original format.



USING AUTO-ENCODERS

- ▶ The aim is to learn the mapping $x \mapsto h \mapsto r$
$$h = f(x)$$
$$r = g(h) = g(f(x))$$
 - x**: input feature space
 - h**: hidden layer giving an alternate representation of the data
 - r**: reconstruction of x computed by the auto-encoder
- ▶ If the AE learns to copy the input feature to the reconstructed output perfectly then r is not particularly useful.
- ▶ The representation given by h can be useful:
 - ▶ If $\dim(h) < \dim(x)$: as the AE is under-complete and learns how to copy x to r using the most important subset of input features.
 - ▶ Under-complete AEs can learn something interesting about the input data, which can be extracted from h .
 - ▶ AEs with too large a $\dim(h)$ can learn to copy x without extracting any interesting information about the data.



USING AUTO-ENCODERS

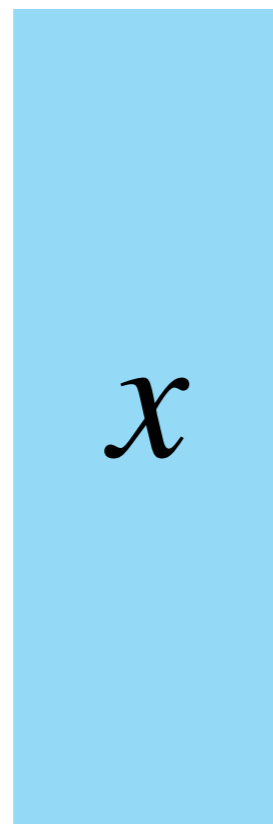
$$x \mapsto h \mapsto r$$

x: input feature space

h: hidden layer giving an alternate representation of the data

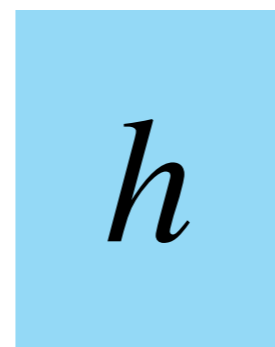
r: reconstruction of x computed by the auto-encoder

Input
feature
space

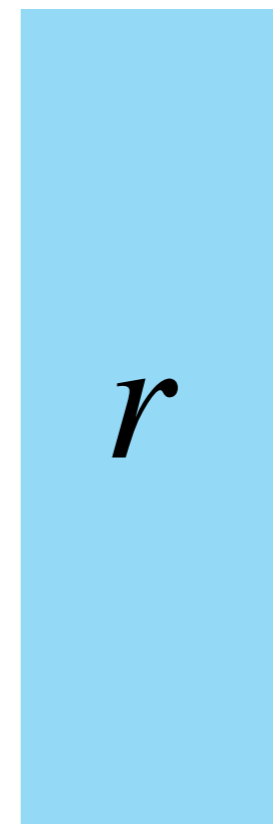


Encoder

(e.g. layer of an MLP)



$$h = f(x)$$



Decoder

(e.g. layer of an MLP)

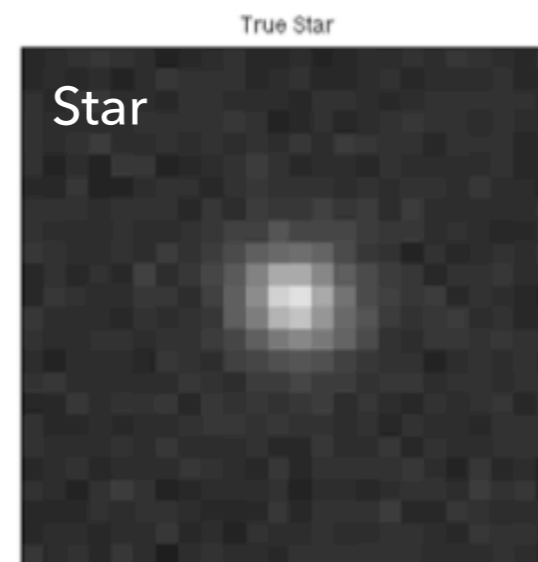
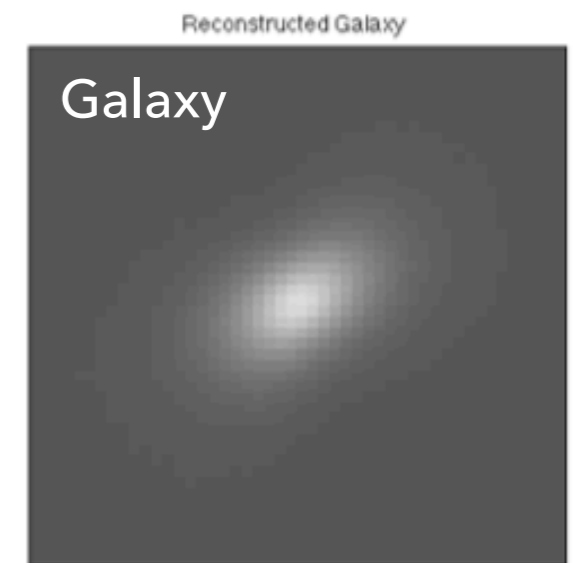
Output prediction
of the model

$$r = g(h) = g(f(x))$$



EXAMPLE: AUTO-ENCODER

- ▶ Dimensional reduction using an AE configuration of 10 nodes in a hidden layer:
 - ▶ Galaxies can be described by 4 parameters (two ellipticities, a position angle and an amplitude).
 - ▶ Stars can be described by 2 parameters (radius and amplitude)
 - ▶ This auto-encoder configuration is able to reconstruct an image of the galaxy and star with noise removed.



INPUT DATA

CONVOLUTION LAYERS

PADDING

FILTERS

POOLING

MODEL ARCHITECTURES (& INPUT DATA)

DROPOUT

EXAMPLE: PARTICLE IDENTIFICATION AT MICROBOONE

NEURAL NETWORKS

CONVOLUTIONAL NEURAL NETWORKS



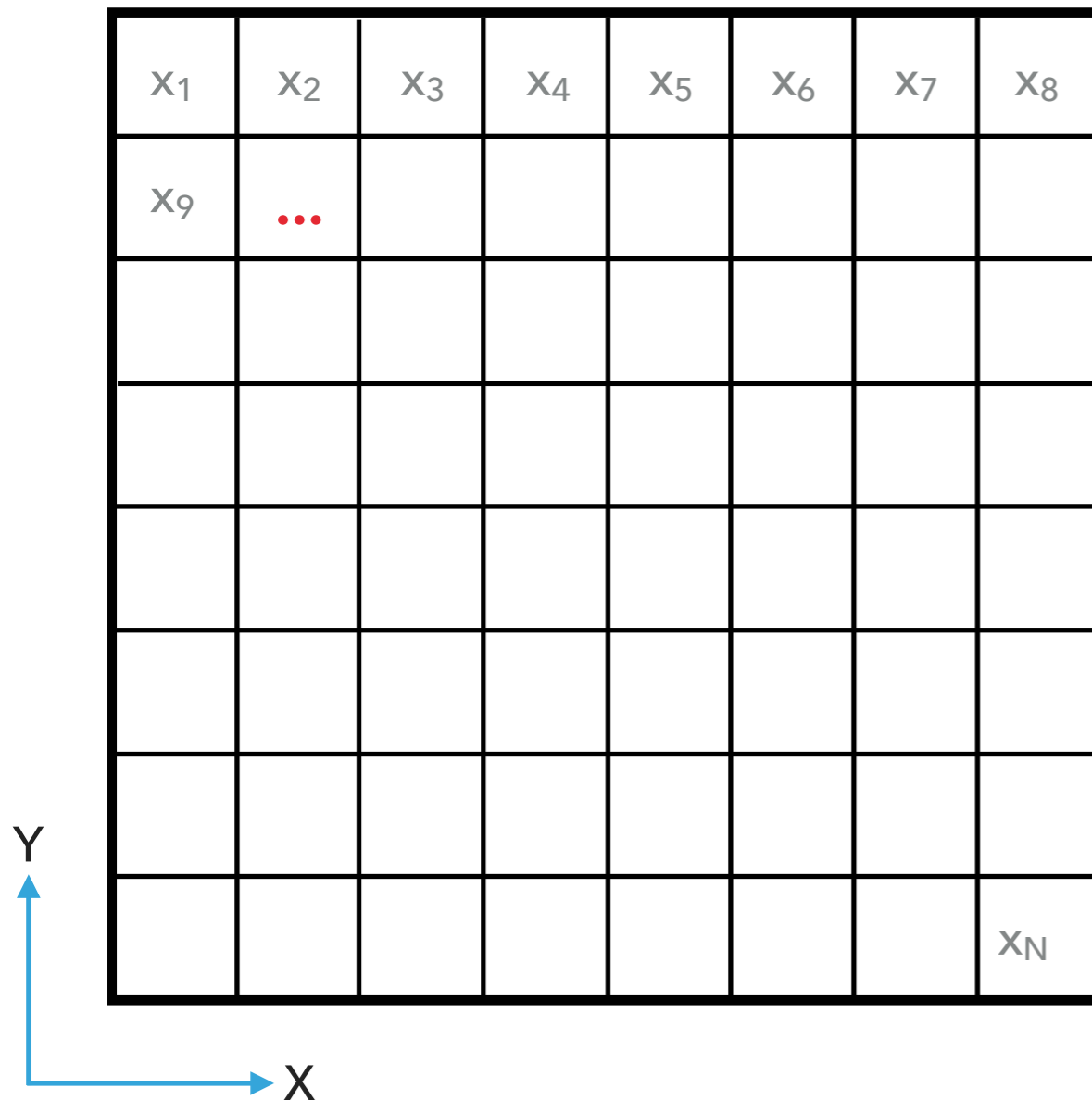
INPUT DATA

- ▶ CNNs take advantage of spatial correlations of the input feature space.^[1]
- ▶ This is typically in the form of image data.
 - ▶ Each pixel corresponds to a feature for each colour that is encoded in it.
 - ▶ Greyscale images have a depth of 1, and so the dimensionality of the feature-space is $n_{\text{pixels}} \times m_{\text{pixels}}$.¹
 - ▶ Colour images have a depth of 3 (R, G, B); so the dimensionality of the feature space is $3 \times n_{\text{pixels}} \times m_{\text{pixels}}$.¹

[1] K Fukushima, Bio. Cybernetics **36** p193-202, 1980.

¹Typically CNNs are applied to square images.

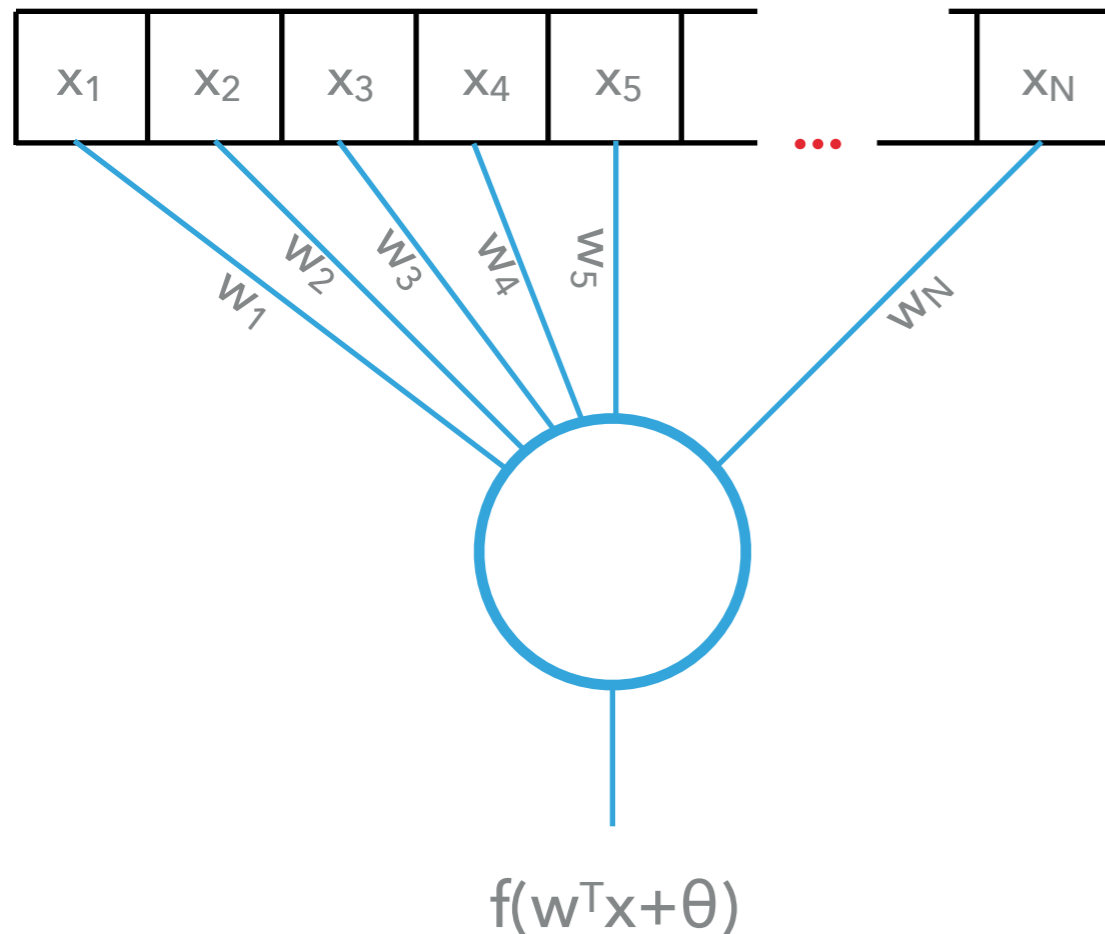
INPUT DATA



This is an 8 by 8 array of pixels, that corresponds to a 64 dimensional feature space.

- ▶ An MLP can be used to process this data, but you lose the spatial correlations between information in the image.
- ▶ The image can be represented by a line of features.
- ▶ Doing this removes the spatial correlations and would naturally lend itself to being processed by a perceptron; i.e. $f(w^T x + \theta)$.

INPUT DATA

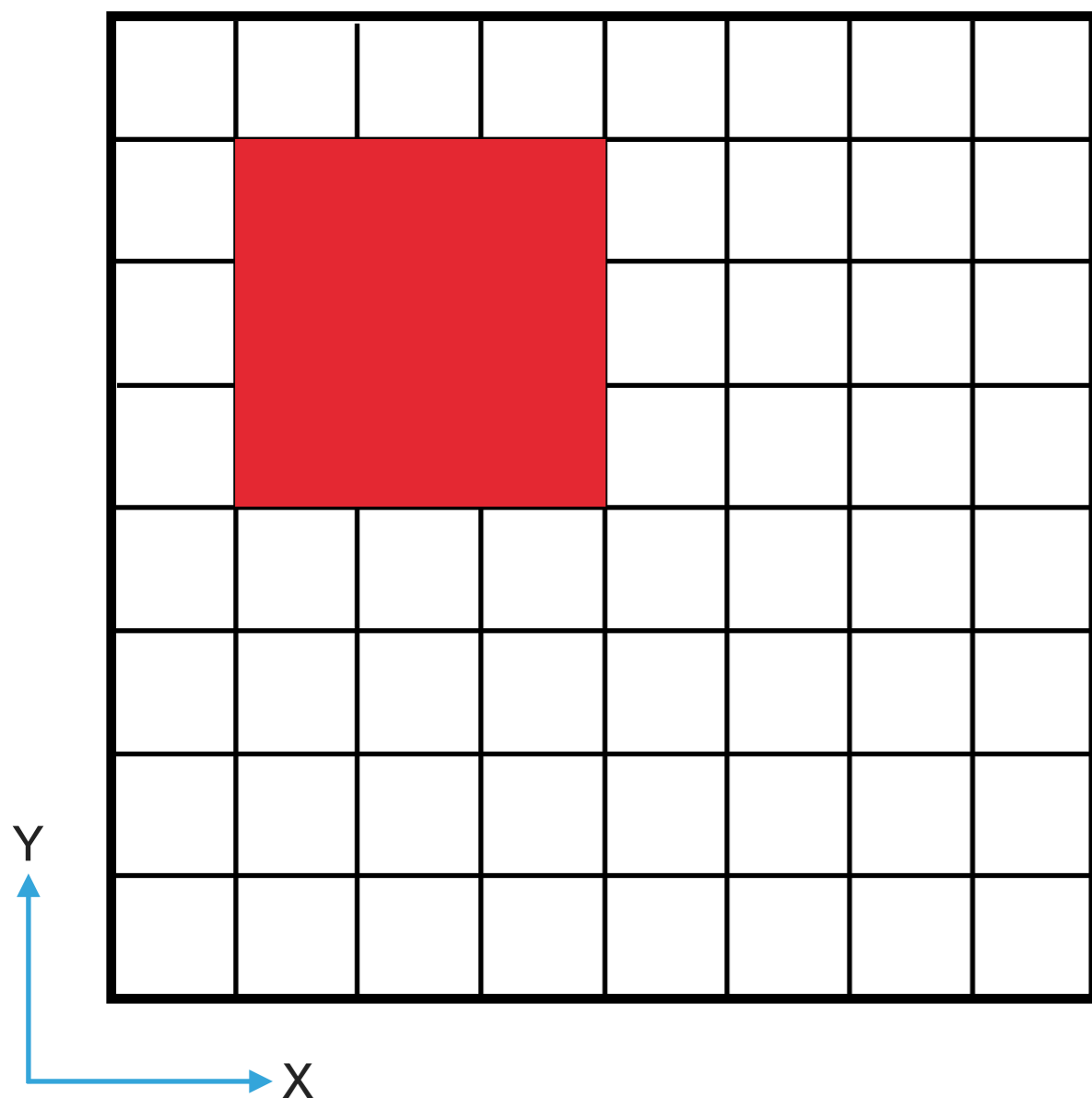


This is an 8 by 8 array of pixels, that corresponds to a 64 dimensional feature space.

- ▶ An MLP can be used to process this data, but you lose the spatial correlations between information in the image.
- ▶ The image can be represented by a line of features.
- ▶ But doing this removes the spatial correlations and would naturally lend itself to being processed by a perceptron; i.e. $f(w^T x + \theta)$.



CONVOLUTION LAYERS

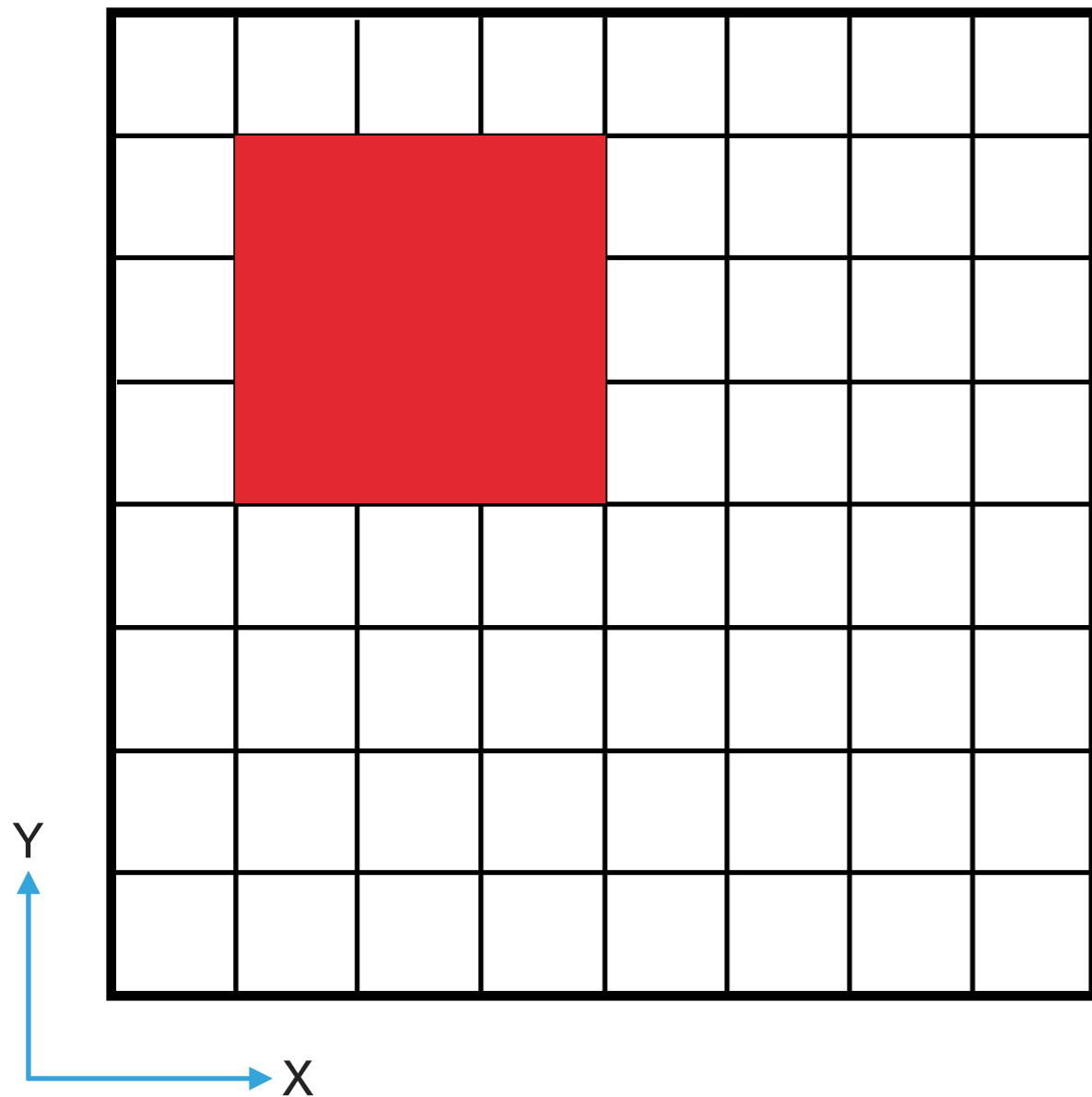


This is an 8 by 8 array of pixels, that corresponds to a 64 dimensional feature space.

- ▶ We can split the image up into a smaller grid of pixels (filter), and search for a pattern in that grid.
 - ▶ In this example we take a 3x3 grid of pixels.
 - ▶ We can compute a numerical convolution of these 9 pixels using $f(w^T x + \theta)$.
- ▶ Spatial correlations within this grid of pixels are used when computing the numerical convolution.
- ▶ Larger filters can be used; where odd numbers of pixels are normally used:
 - ▶ 1x1: identity transformation preserve the input image;
 - ▶ 3x3, 5x5, ... ; compute convolution image.



CONVOLUTION LAYERS



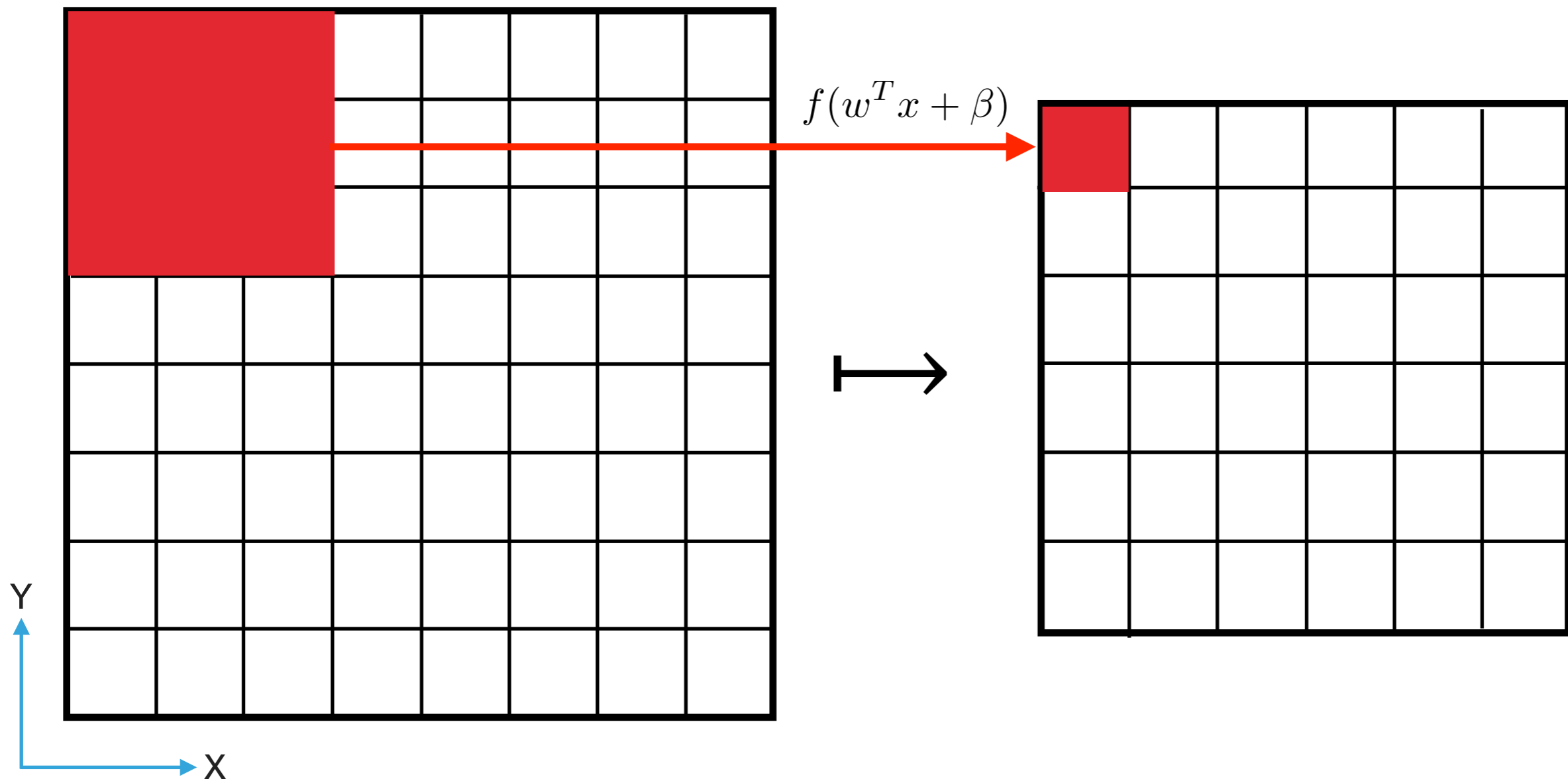
This is an 8 by 8 array of pixels, that corresponds to a 64 dimensional feature space.

- ▶ That same “convolution filter” can be used iteratively over the whole input image.
- ▶ The output values for each iteration are just the value of the output of a perceptron.
- ▶ The set of outputs from running the convolution filter across the input forms a new “convolution image”.



CONVOLUTION LAYERS

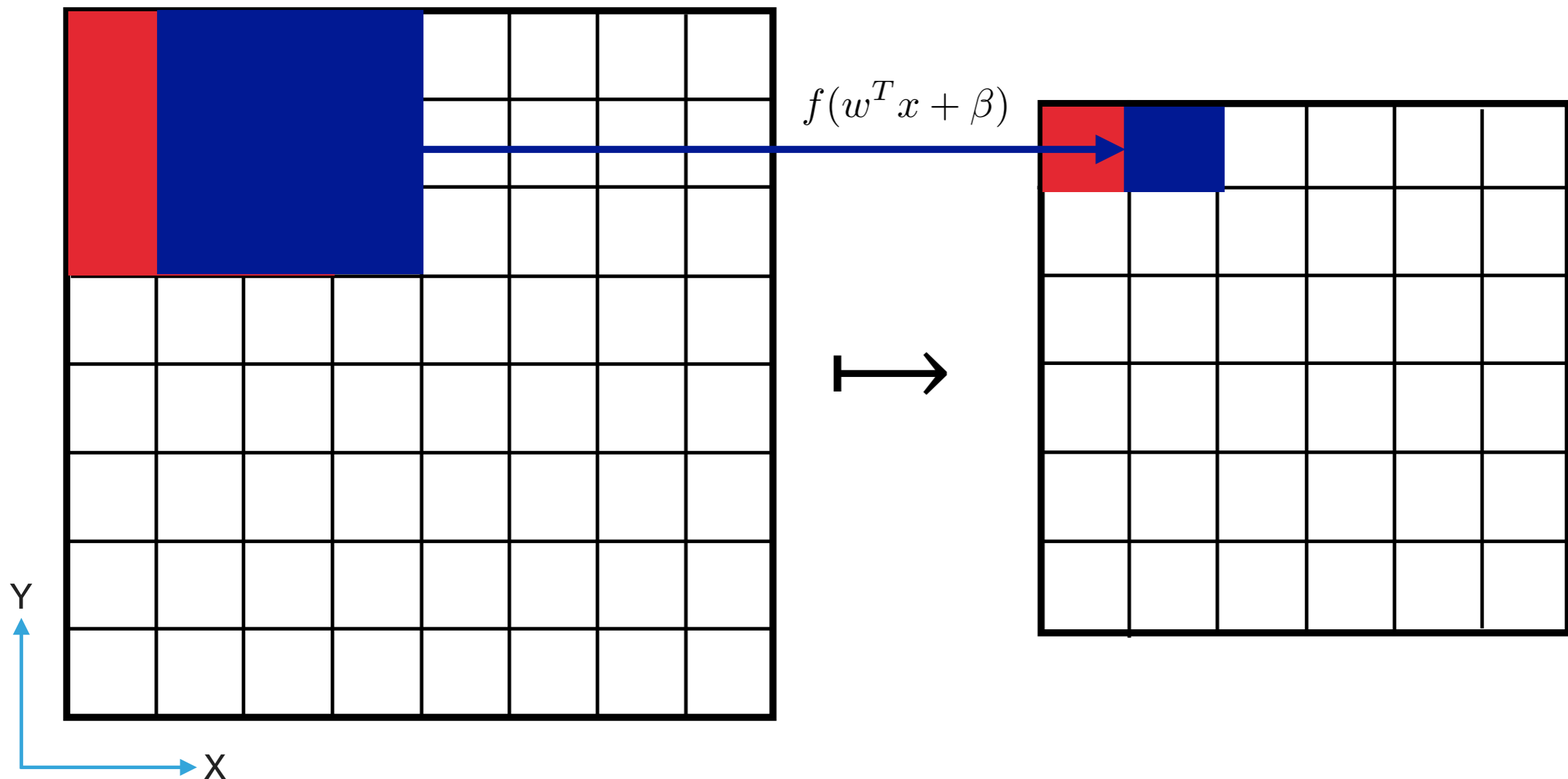
- ▶ If you use an $M \times M$ filter on an $N \times N$ image, the convolved image is smaller than the original.





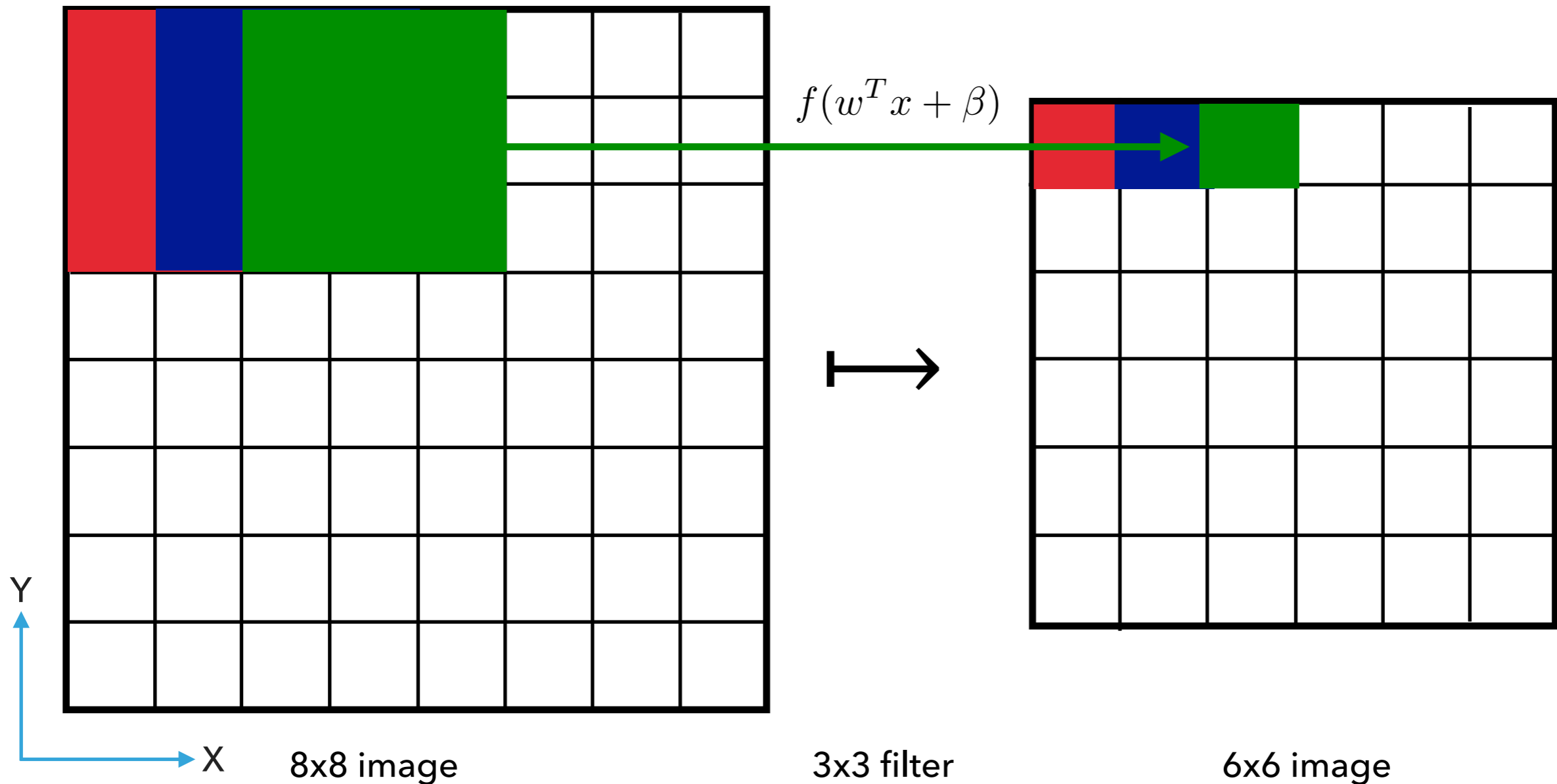
CONVOLUTION LAYERS

- ▶ If you use an $M \times M$ filter on an $N \times N$ image, the convolved image is smaller than the original.



CONVOLUTION LAYERS

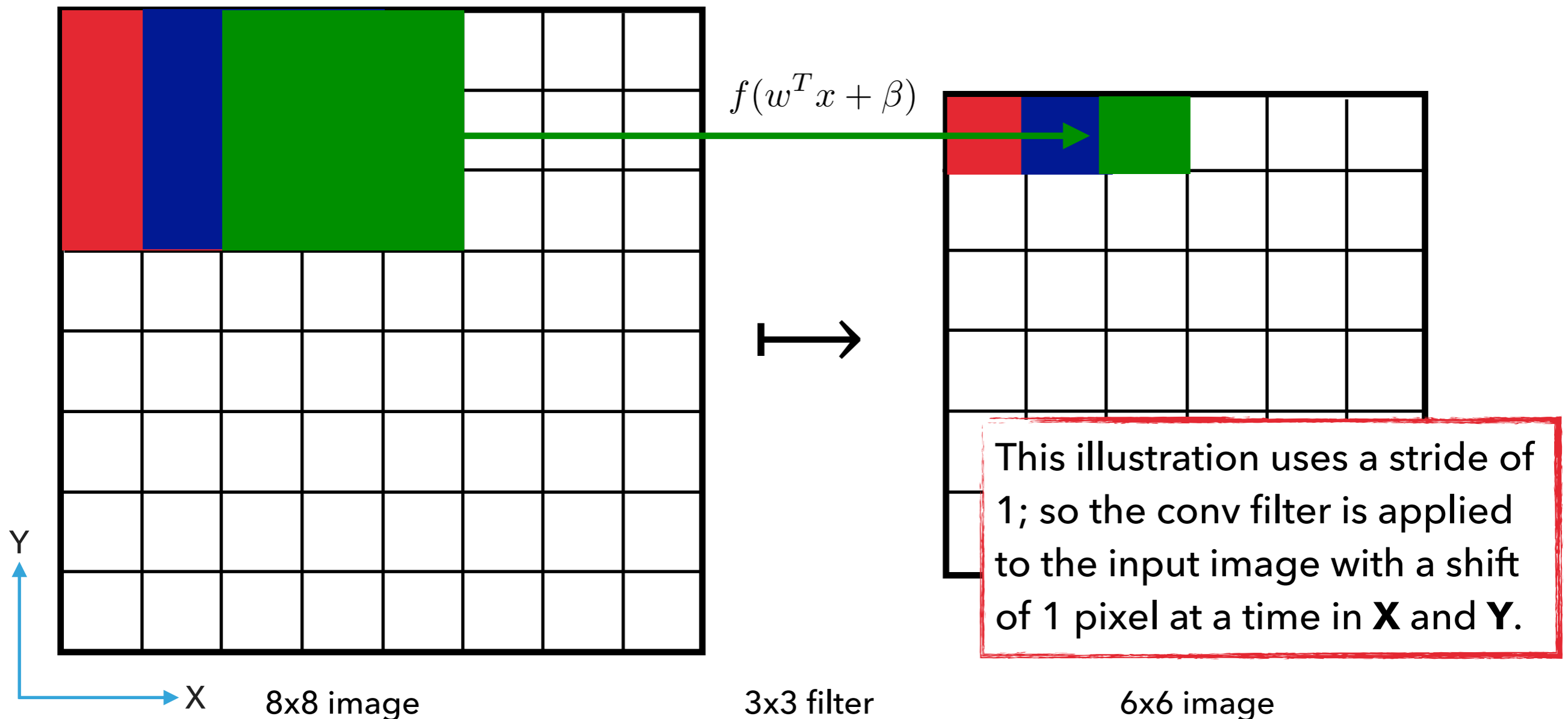
- ▶ If you use an $M \times M$ filter on an $N \times N$ image, the convolved image is smaller than the original.





CONVOLUTION LAYERS

- ▶ If you use an $M \times M$ filter on an $N \times N$ image, the convolved image is smaller than the original.



CONVOLUTION LAYERS

- ▶ $(M-1)/2$ pixels are lost from the border of the input image in order to create the convolution image.

Image Size	Filter Size	Convolution image size
8x8	3x3	6x6
8x8	5x5	4x4
8x8	7x7	2x2
10x10	3x3	8x8
10x10	5x5	6x6
10x10	7x7	4x4
10x10	9x9	2x2

This illustration uses a stride of 1

- ▶ An $N \times N$ image becomes a $(N-M+1) \times (N-M+1)$ image*.
- ▶ Repeatedly convoluting the image reduces the dimensionality of the feature space; which can be undesirable.

*The border is both sides of the image so you loose $(M-1)/2$ pixels twice; once for each side.



CONVOLUTION LAYERS: PADDING

- ▶ The dimensional reduction of feature space can be mitigated by padding the original image with a border of width $(M-1)/2$ pixels.
- ▶ The values of the border padding are set to zero (no information provided to the convolution layer).
- ▶ Now the original image can be convolved with the filter any number of times (within resource limitations) ***without reducing the dimensionality of the input feature space that contains non-trivial information.***



CONVOLUTION LAYERS: FILTERS

- ▶ Each convolution filter is tuned to identify a given shape when scanning through an image.
 - ▶ These can be edge, line or other shape filters.
- ▶ By using a set of convolution filters, one can pick out a set of different features in an image.
- ▶ The weights for these filters are usually initialised randomly using a truncated Gaussian distribution with output >0 .
 - ▶ This is to avoid negative weights.



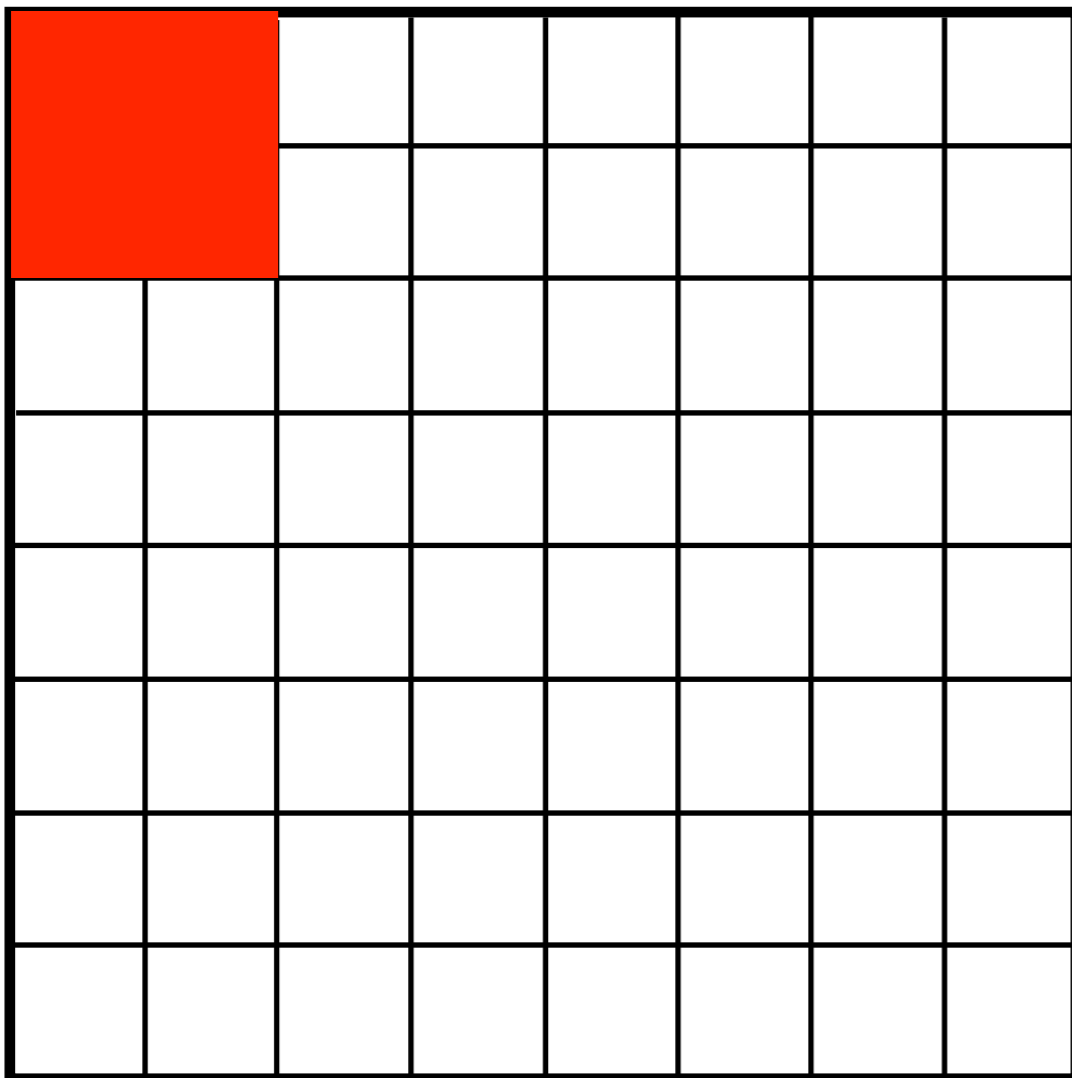
CONVOLUTION LAYERS: POOLING

- ▶ The dimensionality of the features in a convolutional layer is large; numerically it is convenient to reduce the dimensionality for further processing.
 - ▶ High resolution images are not required to be able to identify shapes of objects;
 - ▶ Can make a lower resolution representation and still reach the same conclusion.
 - ▶ Pooling is a mechanism that allows you to achieve this.^[1]
- ▶ Define a filter size for pooling (e.g. 2x2) and then perform an operation on the pixels to compute:
 - ▶ Maximum value (max pooling): useful to suppress noise when information is sparse and the number of pixels having a significant value is expected to be low.
 - ▶ Average value (average pooling): Averaging pixels values can give a smaller variance on the information contained in those pixels.
- ▶ Ref. [1] provides an analysis of these two approaches.

[1] Boureau, Y. L., Ponce, J., & Lecun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In ICML 2010 - Proc., 27th Int. Conf. on Machine Learning (pp. 111-118)

CONVOLUTION LAYERS: POOLING

- ▶ The pooling process is applied to each individual part of the image (i.e. dimensional reduction)



Average pooling is just applying the following to each set of pixels.

$$f = \frac{1}{N} \sum_i x_i$$

Max pooling is equivalent but taking

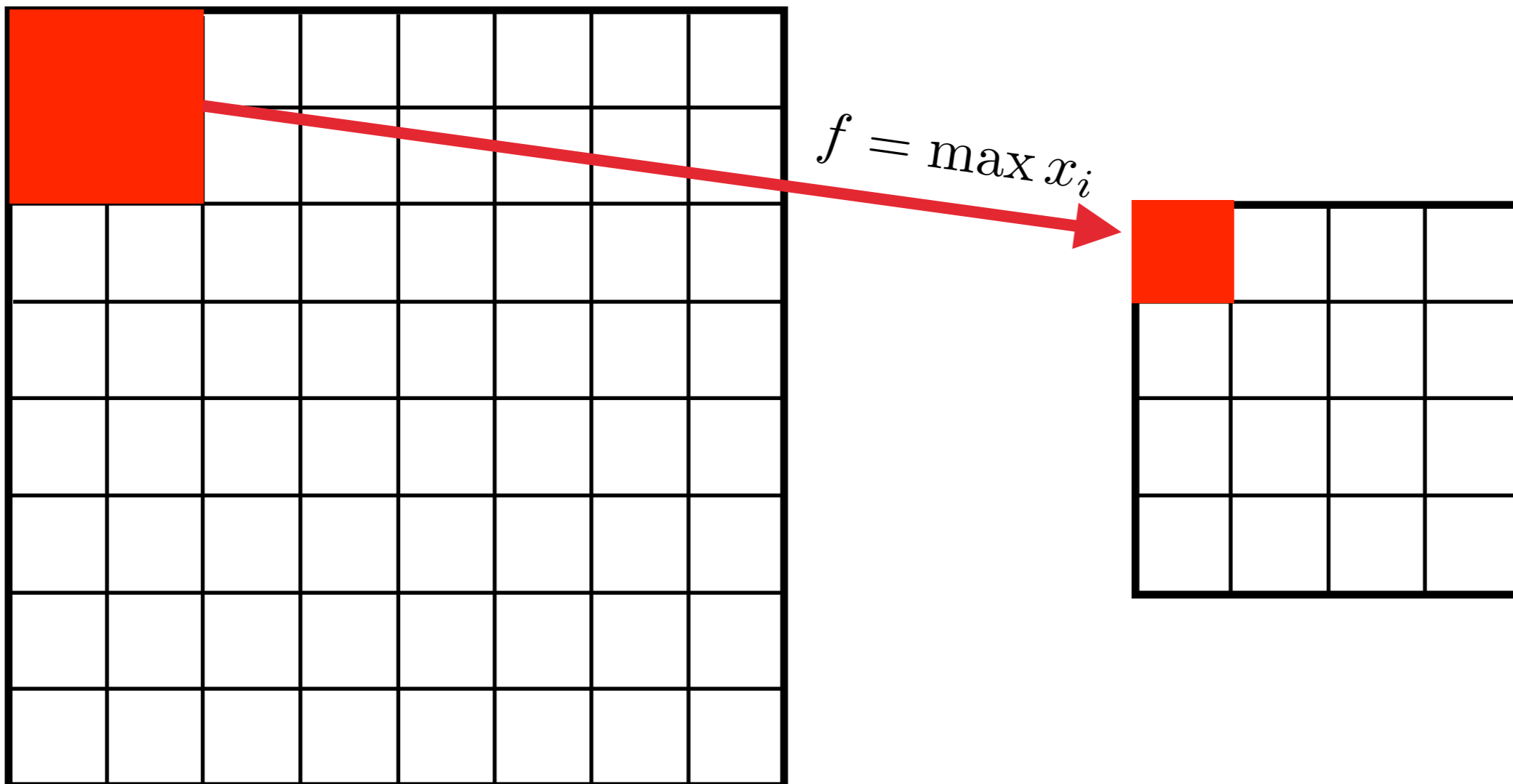
$$f = \max x_i$$

The output is a smaller image.



CONVOLUTION LAYERS: POOLING

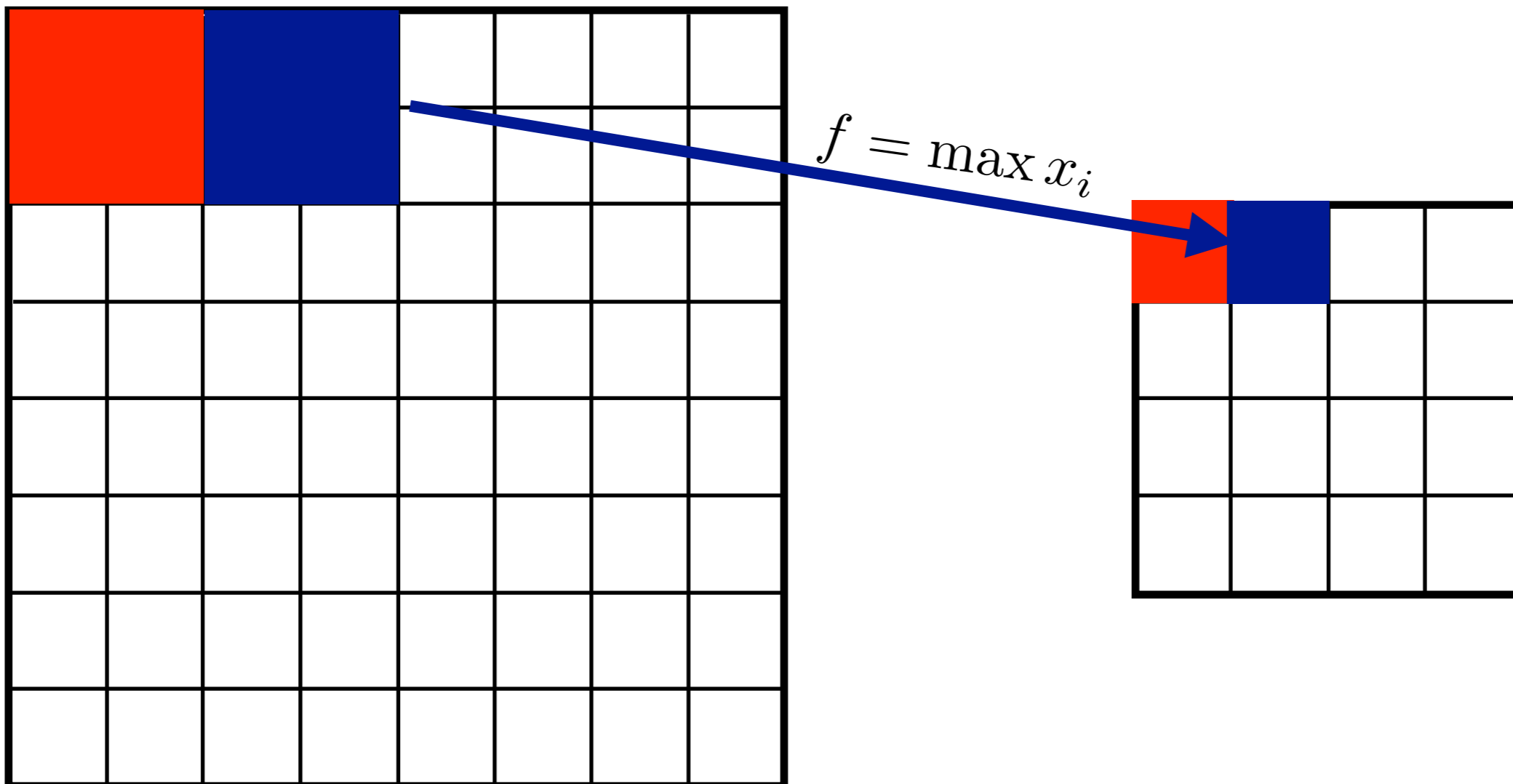
- ▶ The pooling process is applied to each individual part of the image (i.e. dimensional reduction)





CONVOLUTION LAYERS: POOLING

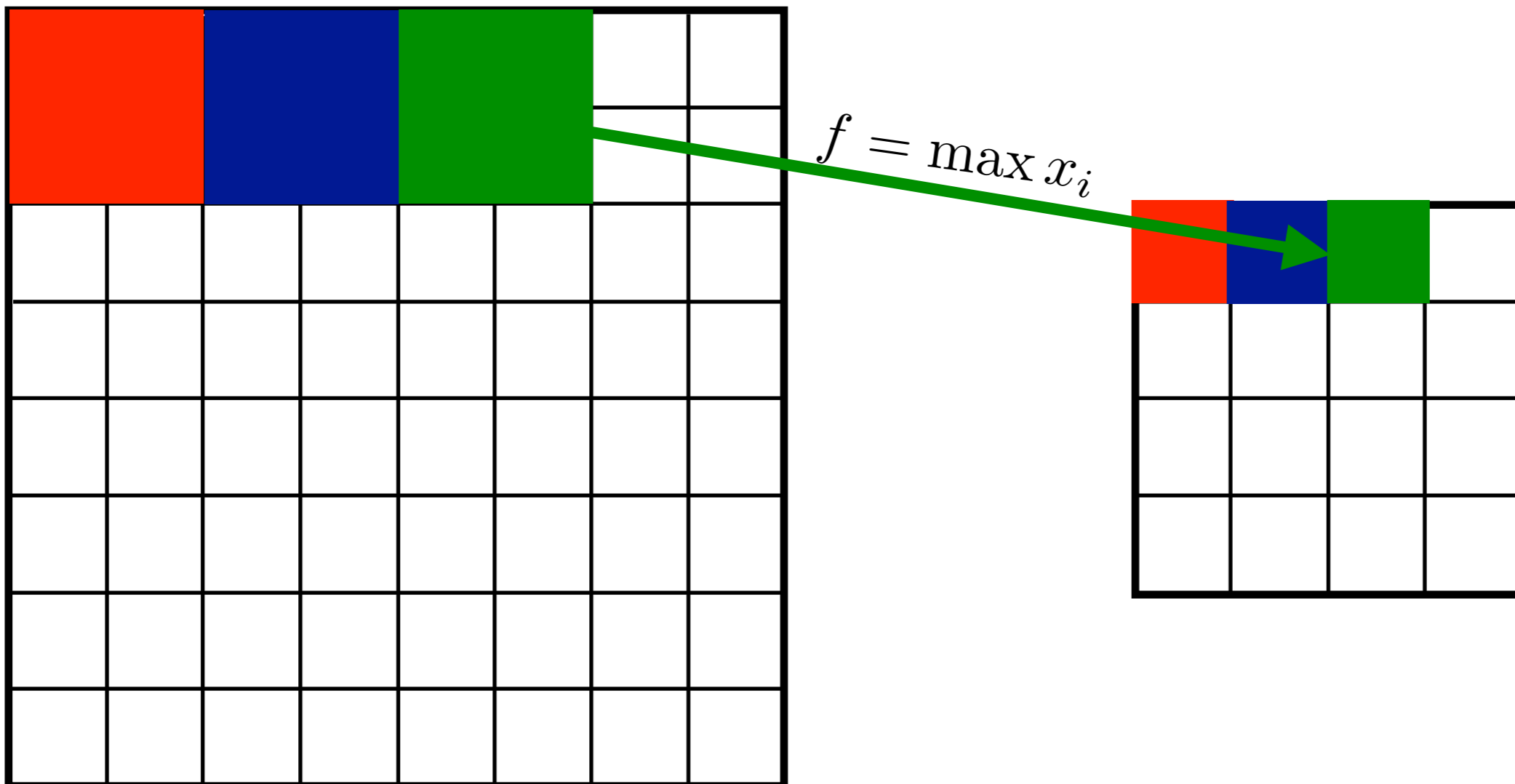
- ▶ The pooling process is applied to each individual part of the image (i.e. dimensional reduction)





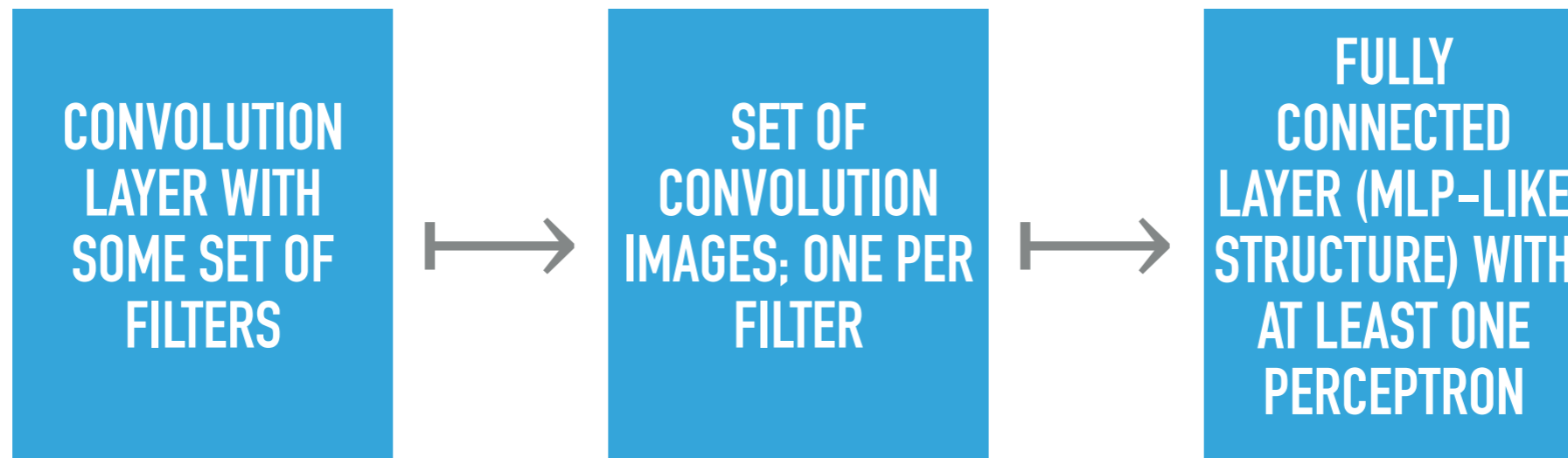
CONVOLUTION LAYERS: POOLING

- ▶ The pooling process is applied to each individual part of the image (i.e. dimensional reduction)



CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES

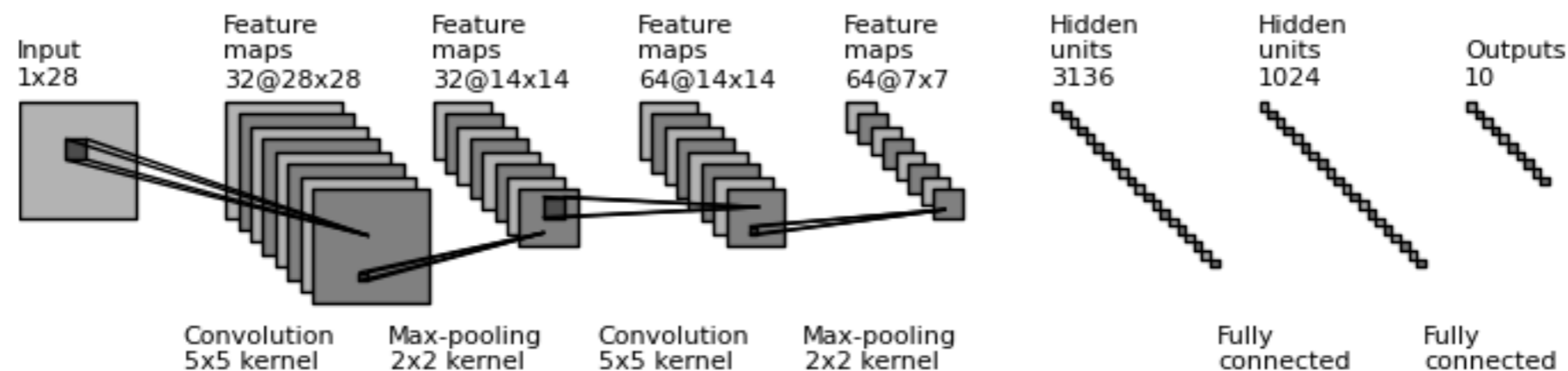
- ▶ The simplest convolutional neural network (CNN) architecture is:



- ▶ The convolution layer takes an image and applies a set of k filters to the image.
- ▶ Each filter results in a new convolution image as its output.
- ▶ All of the features in all of the convolution images are combined to make a final combined output of the information.

CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES

- ▶ We can include multiple convolution layers layers that may add to the information extracted from the image.
- ▶ We can add pooling layers to reduce the dimensionality.
- ▶ e.g. MNIST: handwritten numbers from 0 to 9.



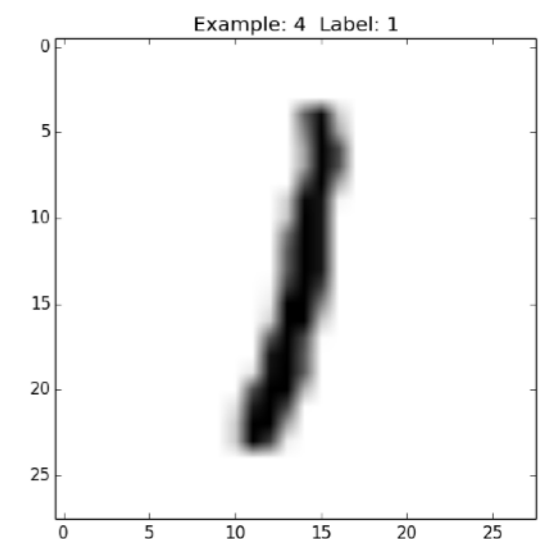
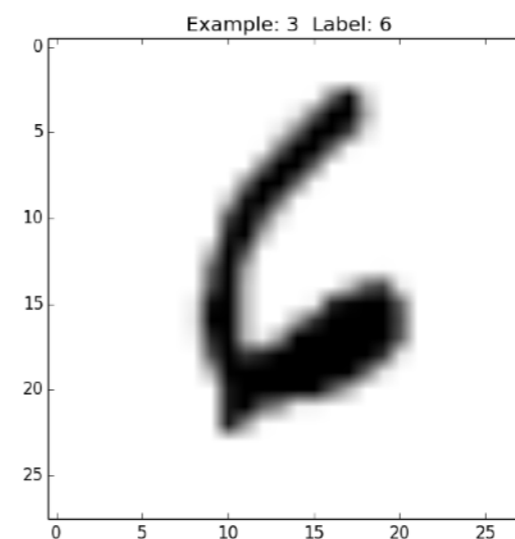
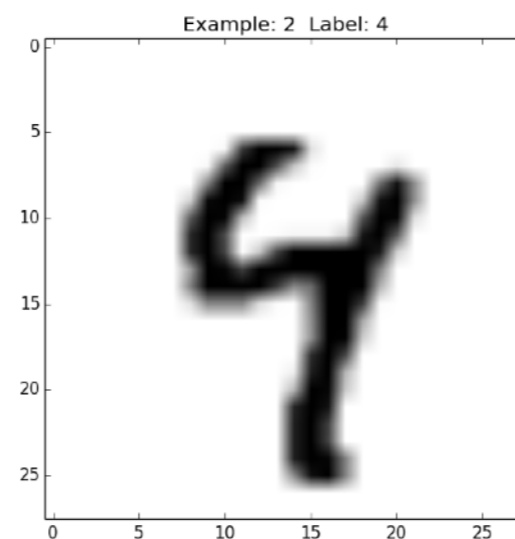
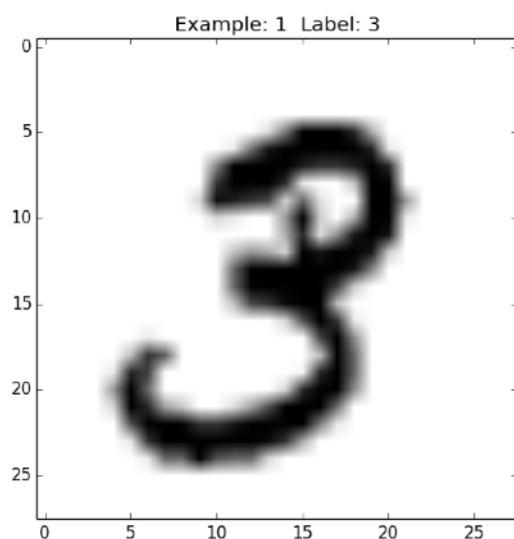
- 28x28 input image (e.g. MNIST example).
- 2 convolution layers using 5x5 filter kernels.
- Each convolution layer followed by a 2x2 max-pooling layer.
- 2 fully connected layers leading to 10 outputs.



CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES

▶ MNIST

- ▶ Standard library of hand written numbers for benchmarking algorithms: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.
- ▶ Images are 28x28 pixels (greyscale).
- ▶ Several examples are shown below





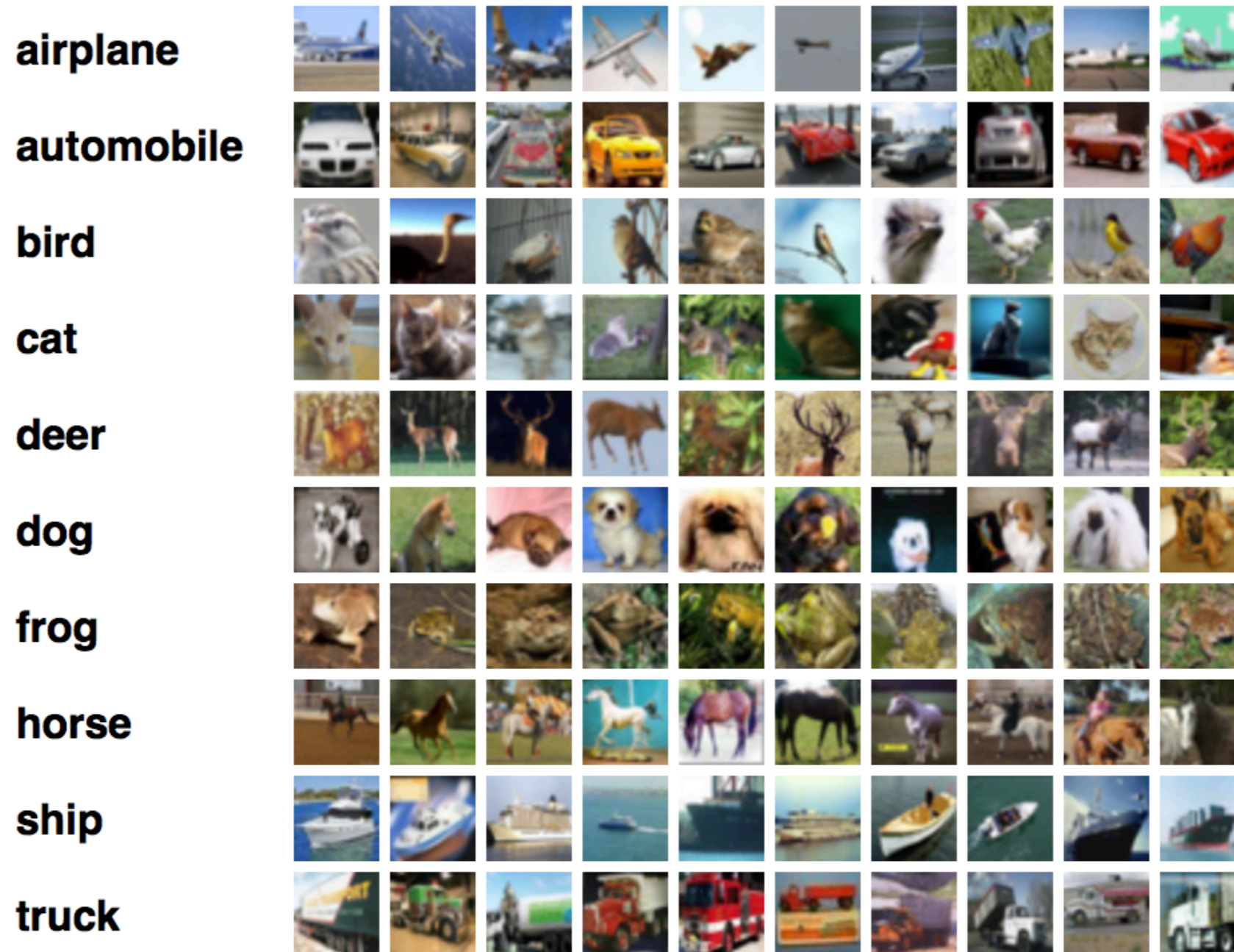
CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES & INPUT DATA

- ▶ So far we have focussed on monochrome images with a single number representing each pixel.
- ▶ What about colour images?
 - ▶ These have 3 numbers (r, g, b) describing each pixel.
 - ▶ Trivial to extend the convolution and pooling processes to work on images of some arbitrary depth D ($=3$ for colour).
 - ▶ 3-fold increase in weight parameters to determine.
- ▶ e.g. CIFAR10 benchmark training set^[1].

[1] <https://www.cs.toronto.edu/~kriz/cifar.html>

CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES & INPUT DATA

▶ CIFAR 10 examples:



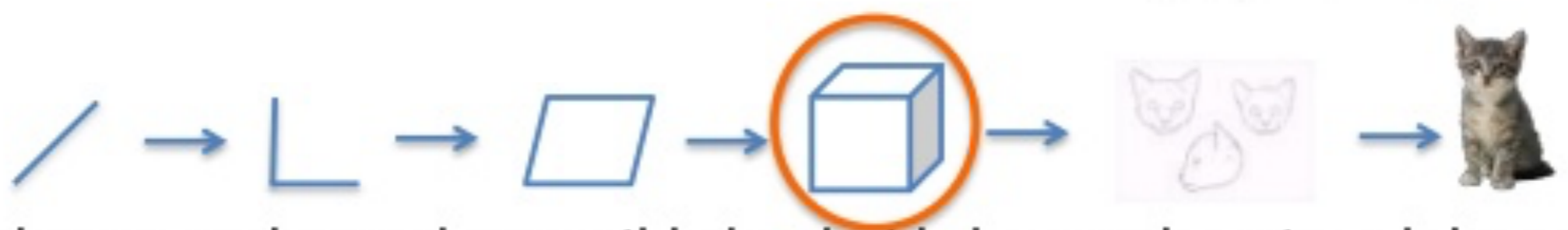
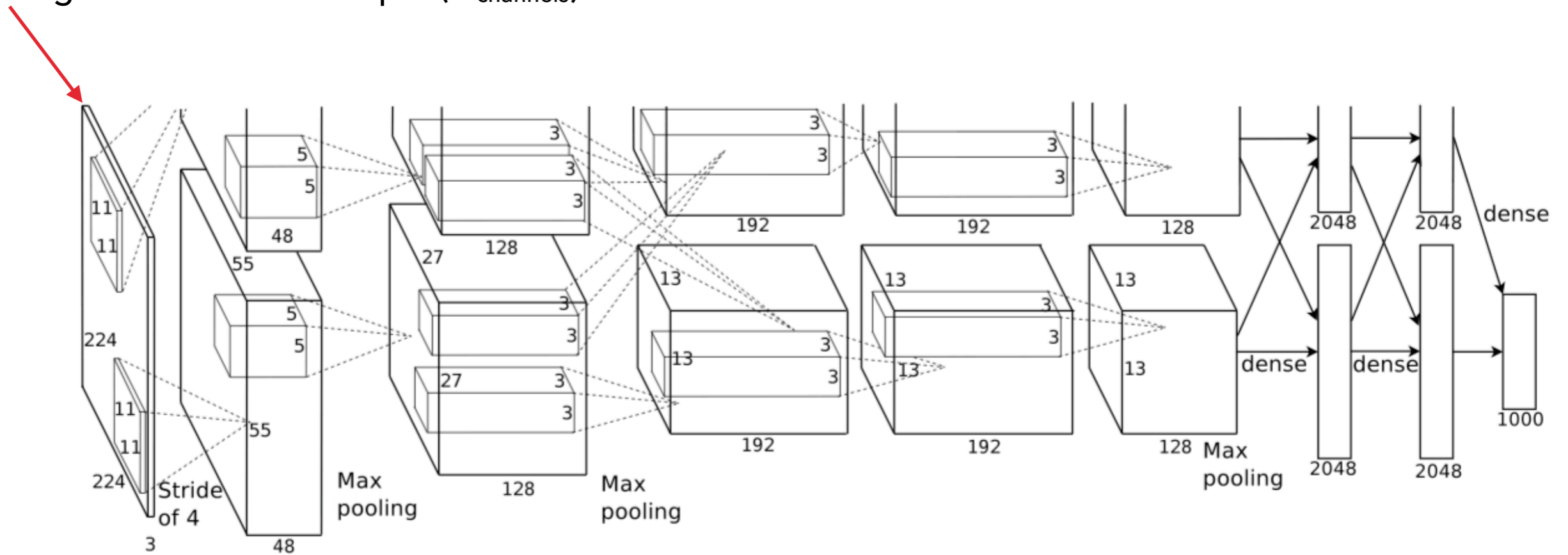
[1] <https://www.cs.toronto.edu/~kriz/cifar.html>



CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES & INPUT DATA

The DNN selects the image class with the highest likelihood.

Input images have a colour depth ($N_{channels}$) of 3



When AlexNet is processing an image, this is what is happening at each layer.

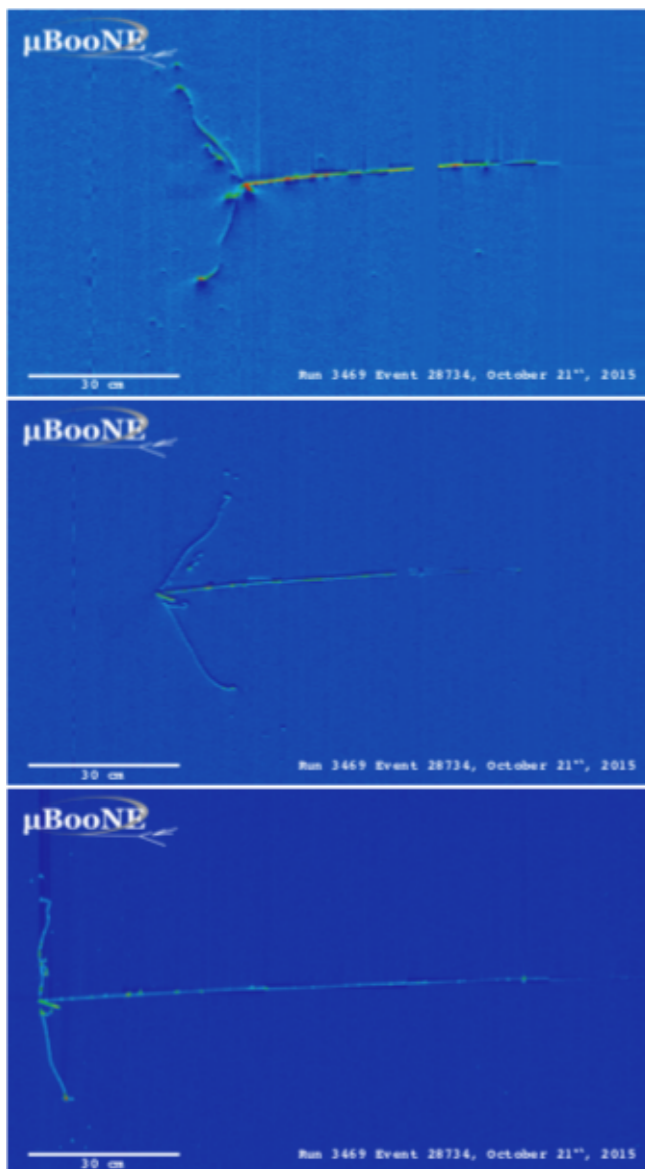


CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES & INPUT DATA

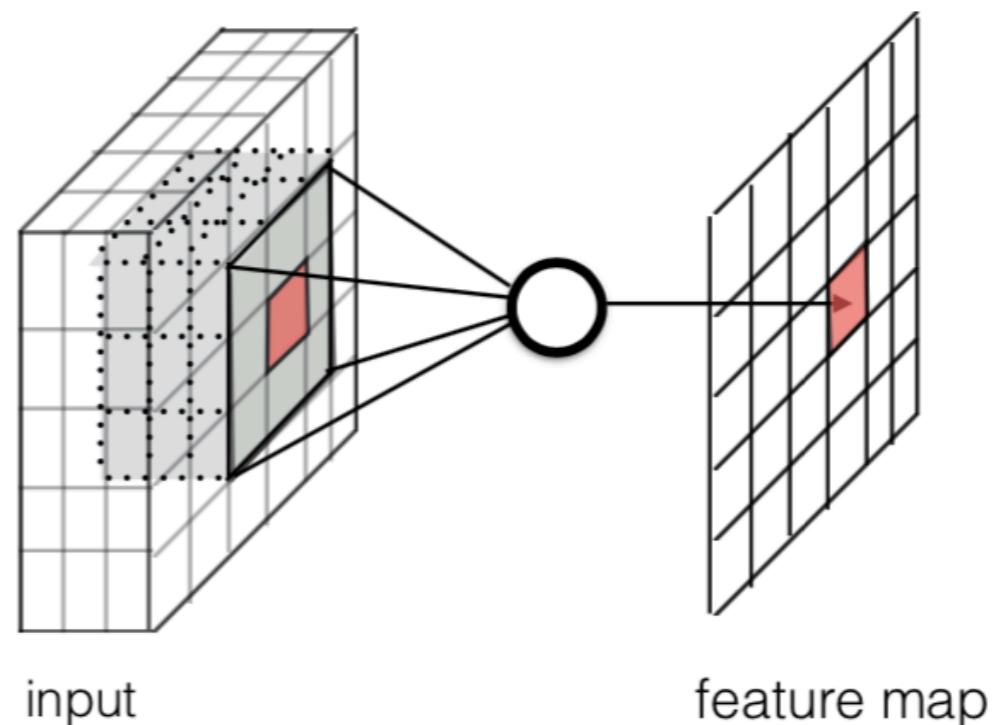
- ▶ More abstract problems can be addressed in the same way.
- ▶ Some examples are given below:
 - ▶ Transient searches can be addressed by stacking images together to form an image of depth D .
 - ▶ Tracking problems can be addressed by stacking measurement data from successive layers.
 - ▶ More arbitrary problems can be addressed by feeding pixelised images of 2D correlation plots between pairs of input "features". Stacks of these can be fed into a CNN.

EXAMPLE: PARTICLE IDENTIFICATION AT MICROBOONE

- ▶ MicroBooNE LAr TPC produces images that need to be interpreted in terms of the underlying neutrino interaction physics:

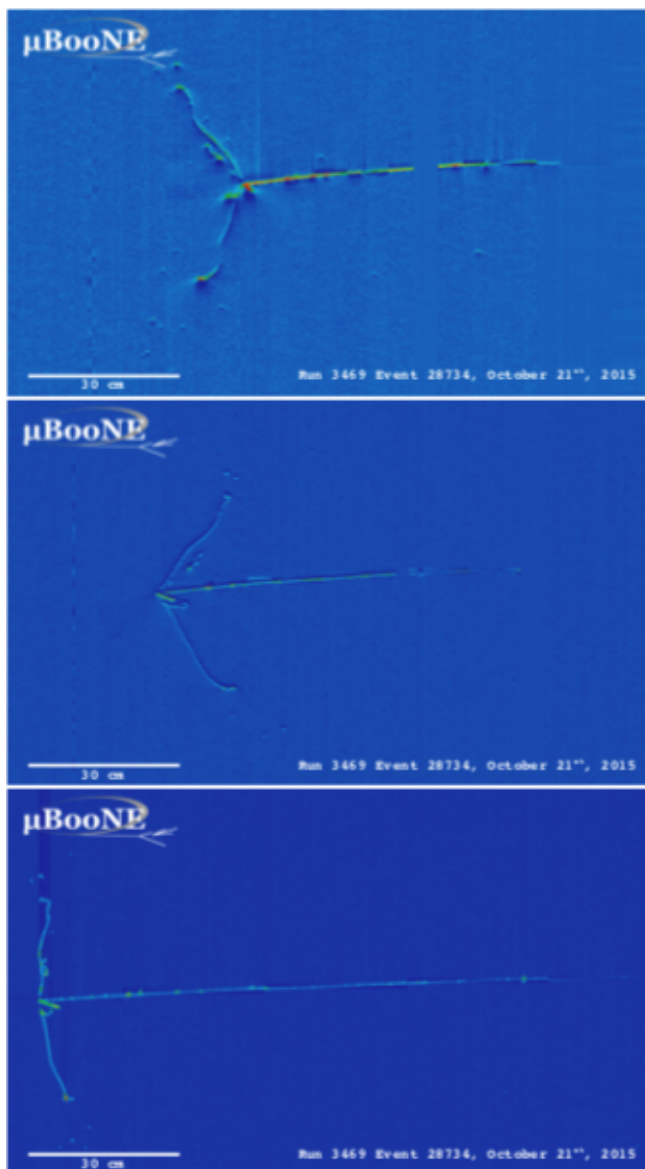


- ▶ 3 different views of the same ν interaction.
- ▶ Use existing software available from the web with many of the techniques discussed in these lectures.



EXAMPLE: PARTICLE IDENTIFICATION AT MICROBOONE

- ▶ MicroBooNE LAr TPC produces images that need to be interpreted in terms of the underlying neutrino interaction physics:

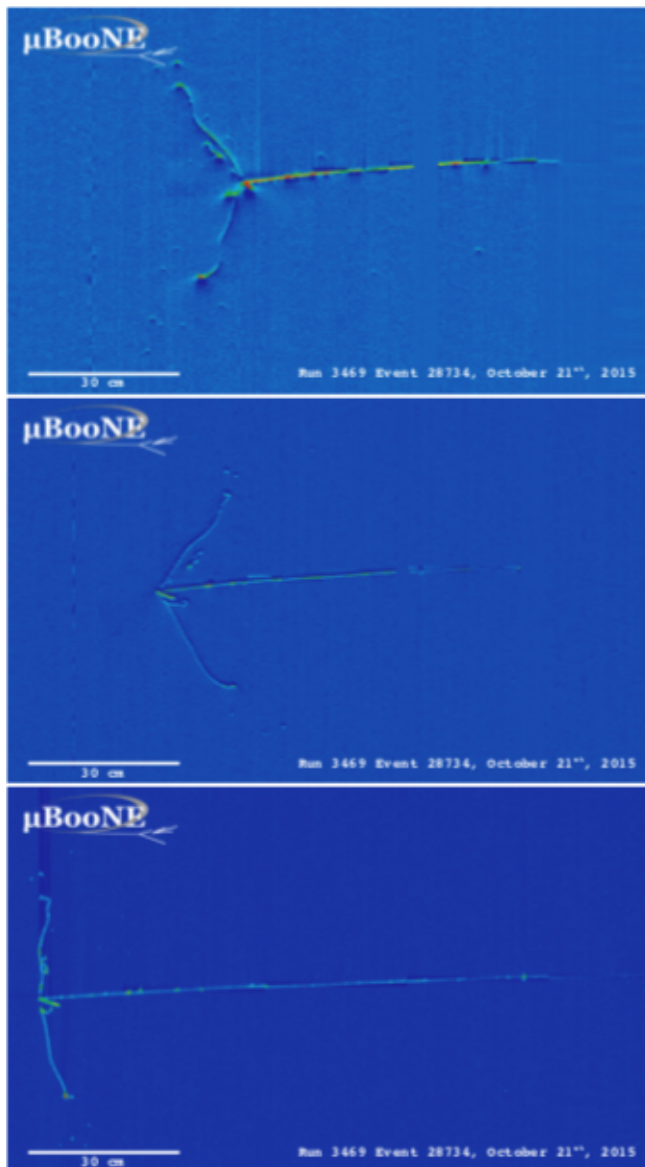


- ▶ 3 different views of the same ν interaction.
- ▶ Use existing software available from the web with many of the techniques discussed in these lectures.

Software	ref.	Purpose	Used in Demonstrations
LArSoft	[7]	Simulation and Reconstruction	1-3
uboonecode	[8]	Simulation and Reconstruction	1-3
LArCV	[9]	Image Processing and Analysis	1-3
Caffe	[10]	CNN Training and Analysis	1-3
AlexNet	[1]	Network Model	1,2
GoogLeNet	[11]	Network Model	1
Faster-RCNN	[12]	Network Model	1,2
Inception-ResNet-v2	[13]	Network Model	2
ResNet	[14]	Network Model	3

EXAMPLE: PARTICLE IDENTIFICATION AT MICROBOONE

- ▶ MicroBooNE LAr TPC produces images that need to be interpreted in terms of the underlying neutrino interaction physics:



- ▶ Image resolution matters for the performance of this convolutional neural network.

Image, Network	Classified Particle Type				
	e^- [%]	γ [%]	μ^- [%]	π^- [%]	proton [%]
HiRes, AlexNet	73.6 ± 0.7	81.3 ± 0.6	84.8 ± 0.6	73.1 ± 0.7	87.2 ± 0.5
LoRes, AlexNet	64.1 ± 0.8	77.3 ± 0.7	75.2 ± 0.7	74.2 ± 0.7	85.8 ± 0.6
HiRes, GoogLeNet	77.8 ± 0.7	83.4 ± 0.6	89.7 ± 0.5	71.0 ± 0.7	91.2 ± 0.5
LoRes, GoogLeNet	74.0 ± 0.7	74.0 ± 0.7	84.1 ± 0.6	75.2 ± 0.7	84.6 ± 0.6

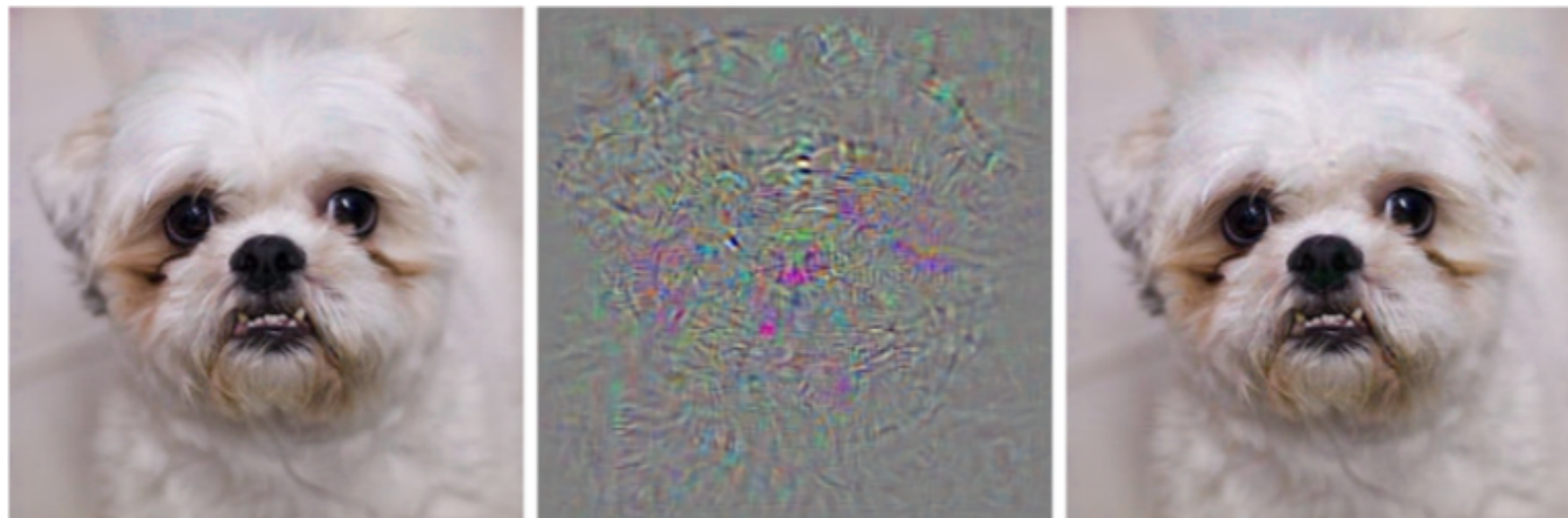
Table 2. Five particle classification performances. The very left column describes the image type and network where *HiRes* refers to a standard 576 by 576 pixel image while *LowRes* refers to a downsized image of 288 by 288 pixels. The five remaining columns denote the classification performance per particle type. Quoted uncertainties are purely statistical and assume a binomial distribution.

GENERATIVE ADVERSARIAL NETWORKS



GENERATIVE ADVERSARIAL NETWORKS

- ▶ Szegedy et al found that small perturbations in the example were sufficient to lead to example mis-classification.
- ▶ The inclusion of some training example that has a small amount of noise added to it resulting in a change in classification is counter to the aim of obtaining a generalised performance from a network.



Correctly classified image

Perturbation of image

Incorrectly classified resultant image

From Szegedy et al,



GENERATIVE ADVERSARIAL NETWORKS

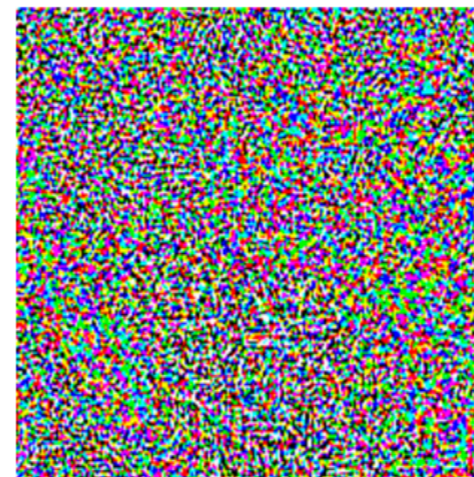
- ▶ Szegedy et al found that small perturbations in the example were sufficient to lead to example mis-classification.
- ▶ The inclusion of some training example that has a small amount of noise added to it resulting in a change in classification is counter to the aim of obtaining a generalised performance from a network.

 x

“panda”

57.7% confidence

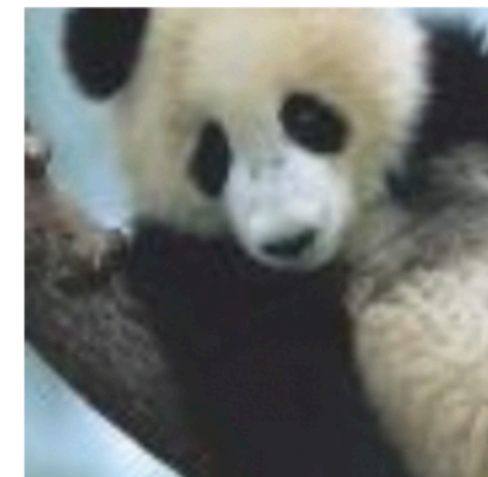
+ .007 ×

 $\text{sign}(\nabla_x J(\theta, x, y))$

“nematode”

8.2% confidence

=

 $x +$ $\epsilon \text{sign}(\nabla_x J(\theta, x, y))$

“gibbon”

99.3 % confidence

From Goodfellow et al,



GENERATIVE ADVERSARIAL NETWORKS

- ▶ Adversarial examples with small perturbations in the input are difficult for networks to classify because of their linear nature in high dimensional feature spaces.
 - ▶ e.g. for $w^T x \mapsto w^T (x + \eta)$ a large value of $\dim(x)$ will result in a large change in the contribution of the perturbed dot product.
- ▶ Adversarial training relies on a modification of the cost function with the intention that the use of adversarial examples in training regularise the optimisation process by identifying flaws in the model that is being learned.
- ▶ This in turn leads to an improved training performance.
 - ▶ Exploiting the nature of adversarial examples allowed Goodfellow et al., to reduce the error rate for image classification with MNIST data; beyond the benefits of using dropout.
 - ▶ The interpretation of this procedure is that one is “minimising the worst case error when the data are perturbed by an adversary”.



GENERATIVE ADVERSARIAL NETWORKS

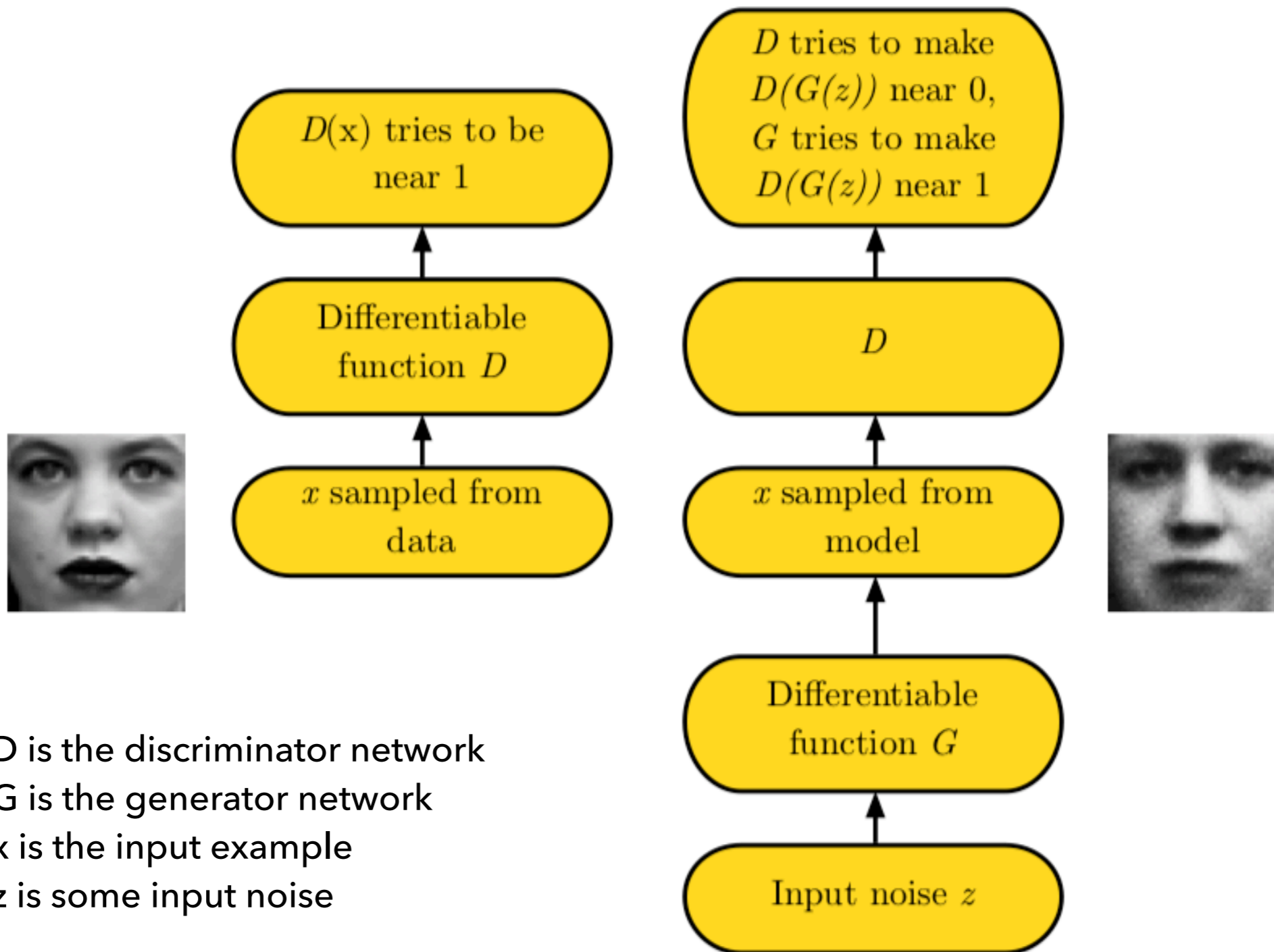
- ▶ The idea behind adversarial networks is to find some way to present adversarial examples alongside data to improve the ability of the model to recognise both the data and its adversarial counterpart.
- ▶ Train two models simultaneously:
 - ▶ **G: a generative model** (the model used to generate adversarial examples for training)
 - ▶ **D: a discriminative model** (the model used to make a prediction that an example is either data or from the generative model)
 - ▶ Train D to maximise the rate of correct outcomes for training examples and samples from the generative model.
 - ▶ Train G to minimise $\ln(1 - D[G(z)])^*$.
- ▶ Over some number of training epochs the generative model G will improve so that it mimics D better.

* It can be problematic to train G in early epochs as it is possible for D to reject samples from G with high confidence; so for early epochs one can maximise $\ln(D[G(z)])$ to overcome this limitation.

Szegedy et al, ICLR, [abs/1312.6199](https://arxiv.org/abs/1312.6199)

Goodfellow et al, CoRR, [abs/1412.6572](https://arxiv.org/abs/1412.6572), [arXiv:1406.2661](https://arxiv.org/abs/1406.2661) also see Goodfellow's Neural Information Processing Systems proceedings on Generative Adversarial Networks: [arxiv:1701.00160](https://arxiv.org/abs/1701.00160).

Adversarial Nets Framework



D is the discriminator network
 G is the generator network
 x is the input example
 z is some input noise

(Goodfellow 2016)



GENERATIVE ADVERSARIAL NETWORKS

- ▶ Use two sets of examples to train: training examples and generated noise examples.
- ▶ Simultaneously optimise the two networks in a combined loss function as a min-max game (zero sum) to identify the saddle point corresponding to minimising the loss contribution from the discriminator while maximising the ability of the generator to fake the data.

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\log D(\mathbf{x}) - \frac{1}{2}\mathbb{E}_{\mathbf{z}}\log(1 - D(G(\mathbf{z})))$$

$$J^{(G)} = -J^{(D)}$$

- ▶ This allows us to optimise the model parameters for the discriminator, $\theta^{(D)}$, and generator, $\theta^{(G)}$.
- ▶ Use normal optimisation algorithms (e.g. ADAM or some other stochastic gradient descent algorithm).

GENERATIVE ADVERSARIAL NETWORKS

- ▶ GAN's are difficult to train, compared with other simpler models (this is simultaneous training of two models).
- ▶ e.g. Spot the generated image example:



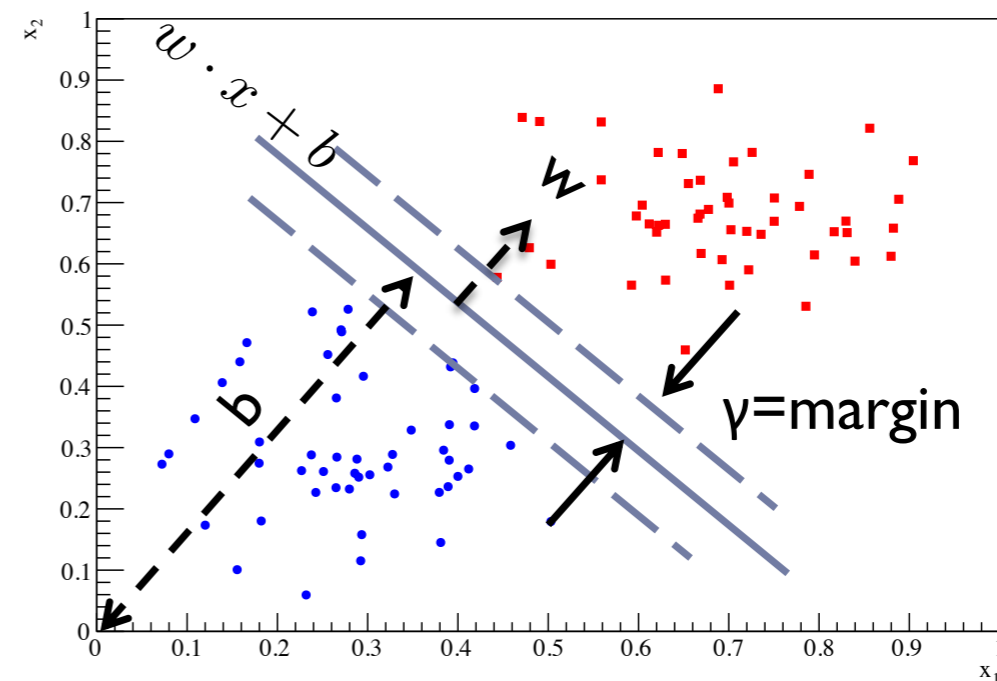
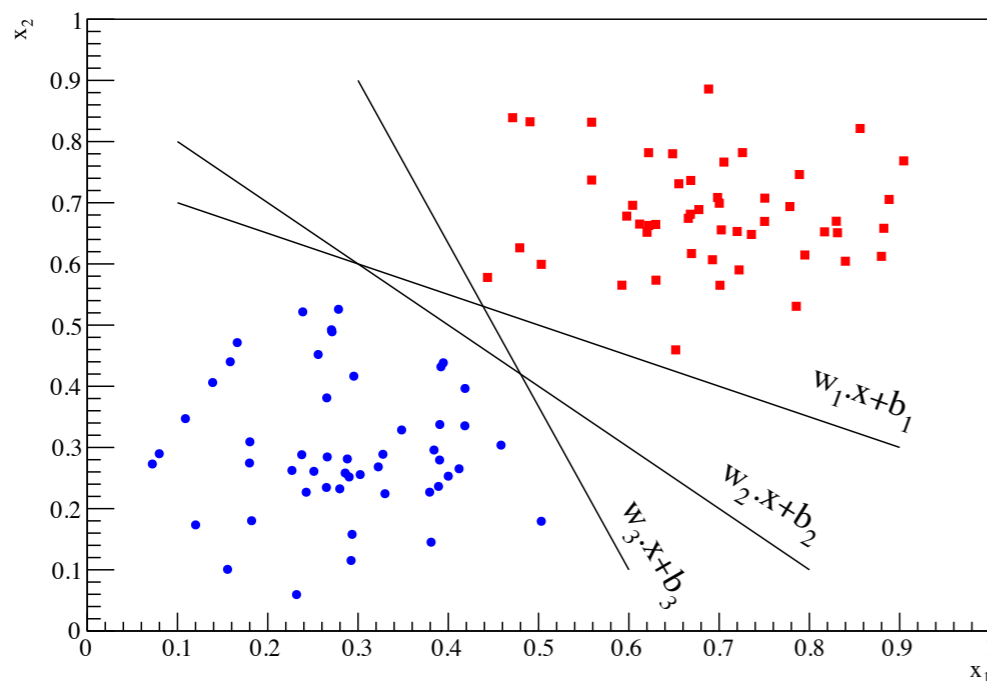
- ▶ A well written discussion of GANs in the context of HEP can be found in: Konstantin and Shyamsundar <https://arxiv.org/abs/2002.06307>.

SUPPORT VECTOR MACHINES



HARD MARGIN SVM

- ▶ Identify the support vectors (SVs): these are the points nearest the decision boundary.
- ▶ Use these to define the hyperplane that maximises the margin (distance) between the optimal plane and the SVs.



- ▶ If we can do this with a SVM - we would simply cut on the data to separate classes of event.

HARD MARGIN SVM: PRIMAL FORM

- ▶ Optimise the parameters for the maximal margin hyperplane with:

$$\arg \min_{w,b} \frac{1}{2} \|w\|^2$$

- ▶ such that $y_i(w \cdot x_i - b) \geq 1$ (y_i is called the functional margin)

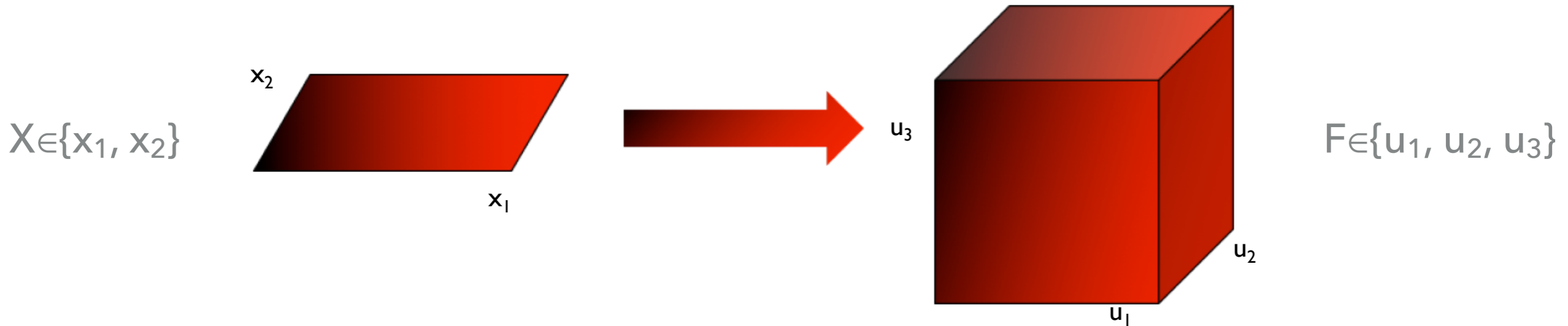
- ▶ Equivalent to solving the following optimisation problem:

$$\arg \min_{w,b} \max_{\alpha \geq 0} \left[\frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i [y_i(w \cdot x_i - b) - 1] \right]$$

- ▶ Where: $w = \sum_{i=1}^n \alpha_i y_i x_i$ and $b = \frac{1}{N_{SV}} \sum_{i=1}^n (w \cdot x_i - y_i)$

HARD MARGIN SVM: KERNEL FUNCTIONS

- ▶ We can introduce the use of a Kernel Function (KF) to implicitly map from our input feature space X to some potentially higher dimensional dual feature space F .
- ▶ Define the function: $K(x, y) = \langle \phi(x) \cdot \phi(y) \rangle$



- ▶ We don't need to know the details of the mapping; this is the "kernel trick".

B. Scholkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002.



HARD MARGIN SVM: KERNEL FUNCTIONS

- ▶ We can introduce the use of a Kernel Function (KF) to implicitly map from our input feature space X to some potentially higher dimensional dual feature space F .
- ▶ Define the function: $K(x, y) = \langle \phi(x) \cdot \phi(y) \rangle$

e.g.

$$x \in \mathbb{R}^n \quad \longrightarrow \quad F \in \{\phi(x) | x \in X\}$$

- ▶ We don't need to know the details of the mapping; this is the "kernel trick".

B. Scholkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002.

HARD MARGIN SVM: DUAL FORM

- ▶ The problem can be solved in the dual space by minimising the Lagrangian for the Lagrange multipliers α_i :

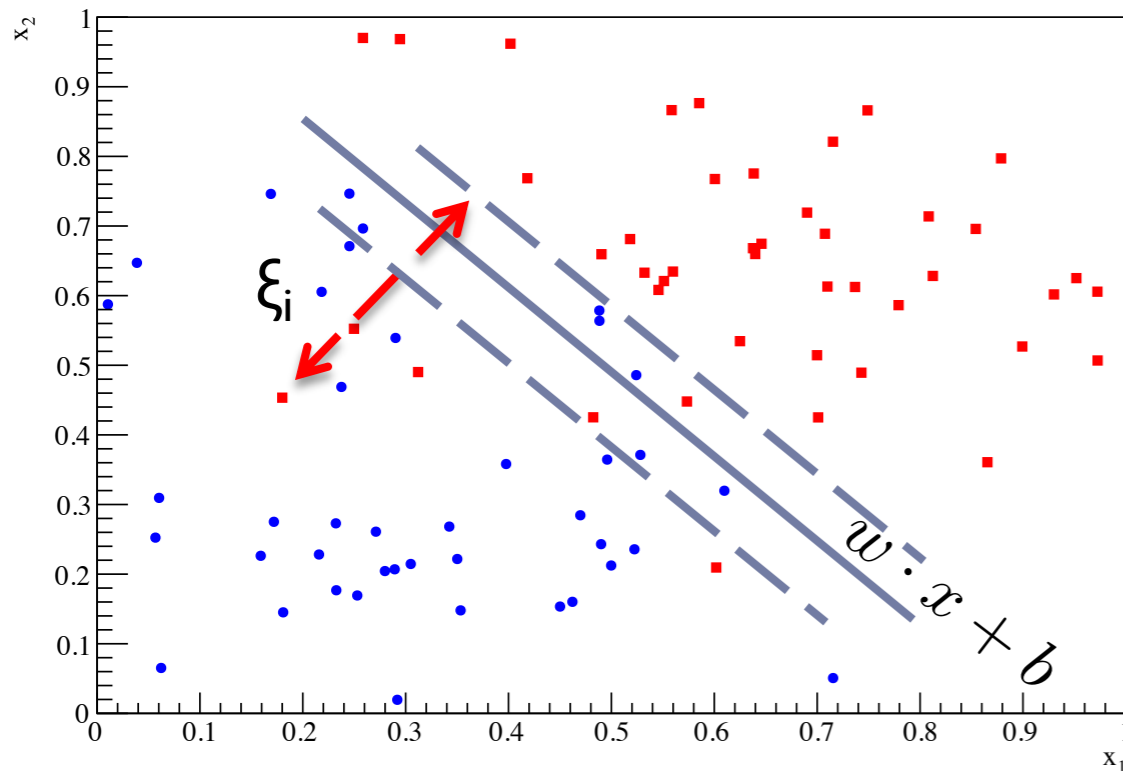
$$\begin{aligned}\tilde{L}(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j).\end{aligned}$$

Dot product KF

- ▶ Such that: $\alpha_i \geq 0$ and $\sum_{i=1}^n \alpha_i y_i = 0$.
- ▶ α_i are non-zero for SVs only.
- ▶ The sum provides a constraint equation for optimisation.

SOFT MARGIN SVM

- ▶ Relax the hard margin constraint by introducing mis-classification:
 - ▶ Describe by slack (ξ_i) and cost (C) parameters.
 - ▶ Alternatively describe mis-classification in terms of loss functions.
 - ▶ These are iust ways to describe the error rate.



ξ_i = distance between the hyper-plane defined by the margin and the i th SV (i.e. now this is a mis-classified event).

Cost (C) multiplies the sum of slack parameters in optimisation.

MVA architecture complexity is encoded by the KF.

- ▶ These are much more useful!

SOFT MARGIN SVM

- ▶ The Lagrangian to optimise simplifies when we introduce the slack parameters:

$$\tilde{L}(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

- ▶ Where $0 \leq \alpha_i \leq C$
- ▶ and as before we constrain:

$$\sum_{i=1}^n \alpha_i y_i = 0$$

The optimisation problem in dual space is essentially the same for the hard and soft margin SVMs.

- ▶ The algorithm is designed to focus on reducing the impact of misclassified events; again using those closest to the decision boundary to determine that boundary.



KERNEL FUNCTIONS

- ▶ The KF, $K(x,y)$, extends the use of inner products on data in a vector space to a transformed space where

$$K(x, y) = \langle \phi(x) \cdot \phi(y) \rangle$$

- ▶ The book by
 - ▶ *Nello Cristianini and John Shawe-Taylor, called Support Vector Machines and other kernel-based learning methods. Cambridge University Press, 2000 (and references therein)*
- ▶ *discusses a number of KFs and the conditions required for these to be valid in the geometrical representation that SVMs are constructed from.*
- ▶ Here I'll focus on the main points and give a few examples of KFs (ones that are implemented in TMVA).



KERNEL FUNCTIONS: RADIAL BASIS FUNCTION (RBF)

- ▶ Commonly used KF that maps the data from X to F .
- ▶ Distance between two support vectors is computed and used as an input to a Gaussian KF.
- ▶ For two data x and y in X space we can compute $K(x, y)$ as
$$K(x, y) = e^{-\|x-y\|^2/\sigma^2}$$
- ▶ One tuneable parameter in mapping from X to F ; given by $\Gamma = 1/\sigma^2$.



KERNEL FUNCTIONS: MULTI GAUSSIAN KERNEL

- ▶ Extend the RBF function to recognise that the bandwidth of data in problem space can differ for each input dimension; i.e. the norm of the distance between two support vectors can result in loss of information.

- ▶ Overcome this by introducing a $\Gamma_i=1/\sigma_i$ for each dimension:

$$K(x, y) = \prod_{i=1}^{\dim(X)} e^{-\|x_i - y_i\|^2 / \sigma_i^2}$$

- ▶ Down side ... we increase the number of parameters that need to be optimally determined for the map from X to F .



KERNEL FUNCTIONS: MULTI GAUSSIAN KERNEL

- ▶ The multi-gaussian kernel does not include off-diagonal terms that would allow for accommodation of correlations between parameters.
 - ▶ De-correlate the input feature space to overcome this deficiency, or alternatively one could implement a variant of this kernel function using:

$$K(x, y) = e^{-(x-y)^T \Sigma^{-1} (x-y)}$$

- ▶ Here Σ is an $n \times n$ matrix corresponding to the covariance matrix for the problem.
- ▶ However this would be very computationally expensive to optimise (and is **not** implemented in TMVA).



KERNEL FUNCTIONS: POLYNOMIAL

- ▶ There are many different types of polynomial kernel functions that one can study.

- ▶ A common variant is of the form:

$$K(x, z) = (\langle x \cdot z \rangle + c)^d = \left(\sum_{i=1}^{\ell} x_i z_i + c \right)^d$$

- ▶ c and d are tuneable parameters.
- ▶ The sum is over support vectors (i.e. events in the data set for a soft margin SVM).



KERNEL FUNCTIONS: PRODUCTS AND SUMS

- ▶ Valid (Mercer) kernels satisfy Mercer's conditions^(*). This allows us to construct new kernels from known Mercer kernels that are products and sums.
- ▶ The sum of Mercer KFs is a valid KF.
- ▶ The product of Mercer KFs is a valid KF.

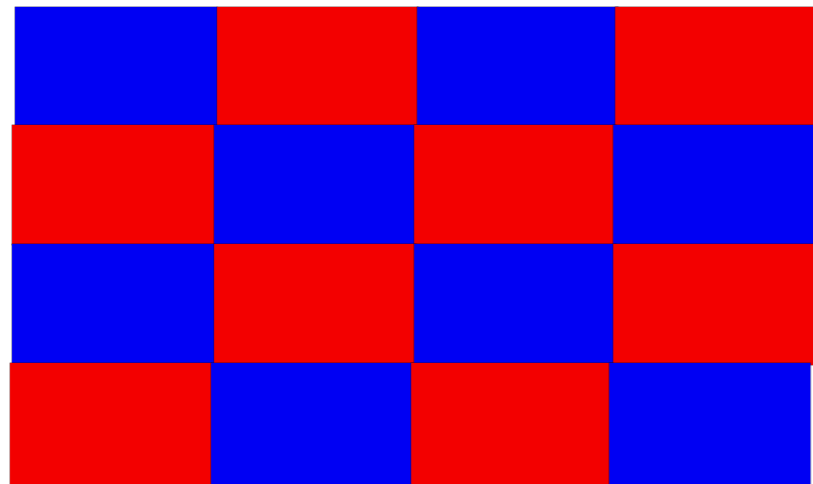
* Mercer's conditions require that the Gram matrix formed from SVs is positive semi-definite. This is a consequence of the geometric interpretation of SVMs given x is real. Modern extensions of the SVM construct allow for complex input spaces, and for example can be based on Clifford algebra to accommodate this extension.

Complex input spaces are of interest for electronic engineering problems.

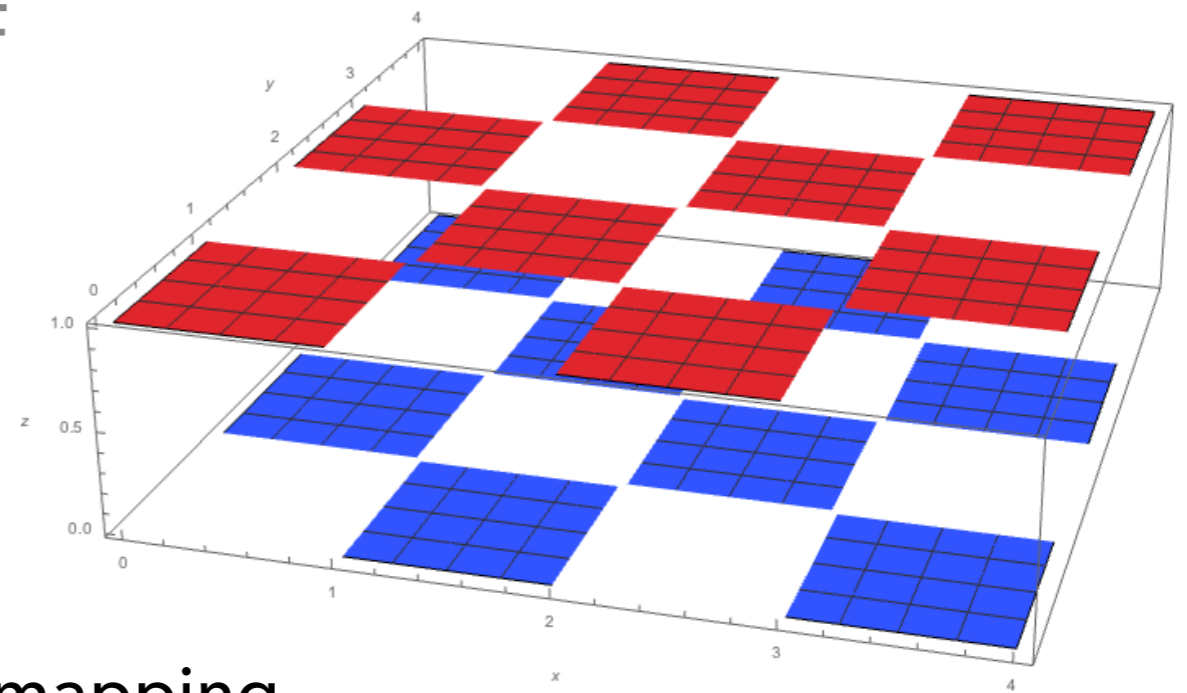
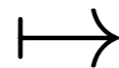
N.B. It is conceivable that one could be interested in using these if an amplitude analysis were to be written using SVMs to directly extract phase and magnitudes... but that could also be incorporated by mapping the complex feature space element into a doublet of reals.

EXAMPLES: CHECKER BOARD

- ▶ Generate squares of different colour.
- ▶ Use SVM to classify the pattern into +1 and -1 targets.
- ▶ Hard margin SVM problem; but can be solved for using soft margin SVM.
- ▶ Not easy to solve in 2D (x, y) with a linear discriminant, but e.g. a 3D space of (x, y, colour) allows us to separate the squares.



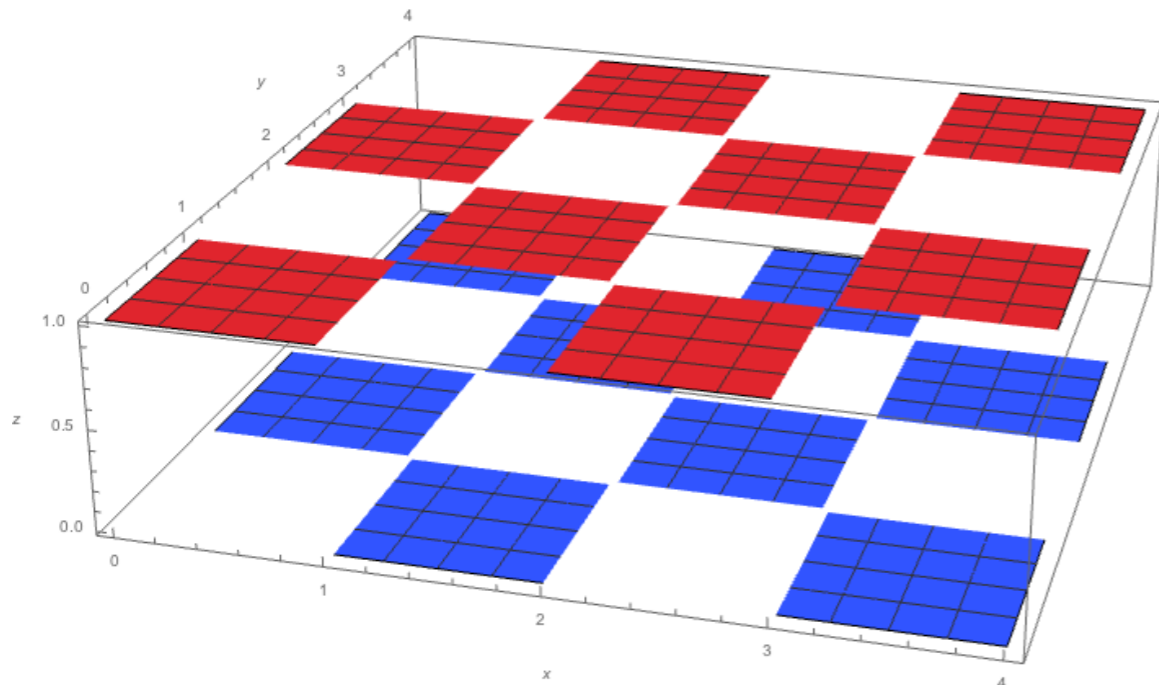
$X \mapsto F$



- ▶ Want to find a KF that approximates this mapping.

EXAMPLES: CHECKER BOARD

- ▶ Generate 1000 events in the blue and red squares and give each event x and y values.



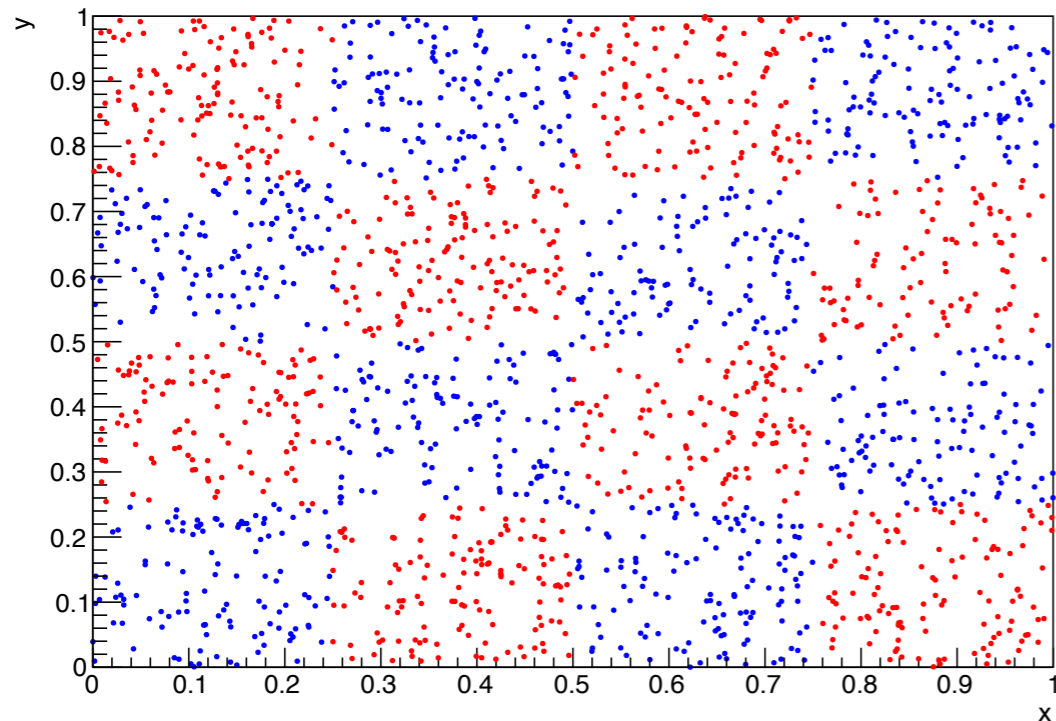
This is the ideal feature space that we would like to implicitly map into.

Because we implicitly do the mapping via choice of KF, in practice we don't explicitly map into this space; but we implicitly map into another space that we hope will be approximately topologically equivalent.

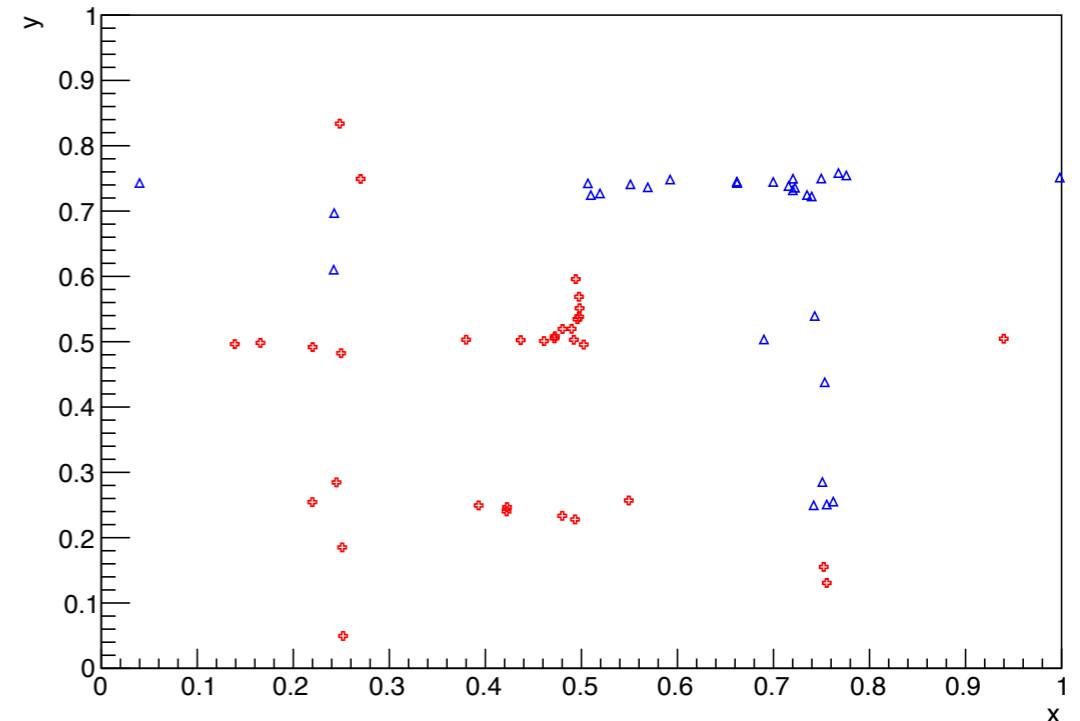
- ▶ e.g. Use a multi-Gaussian kernel function with $\Gamma_1=1$, $\Gamma_2=2$ and cost of 10^4 (not optimised) to see what separation we can obtain.

EXAMPLES: CHECKER BOARD

▶ Correctly classified events



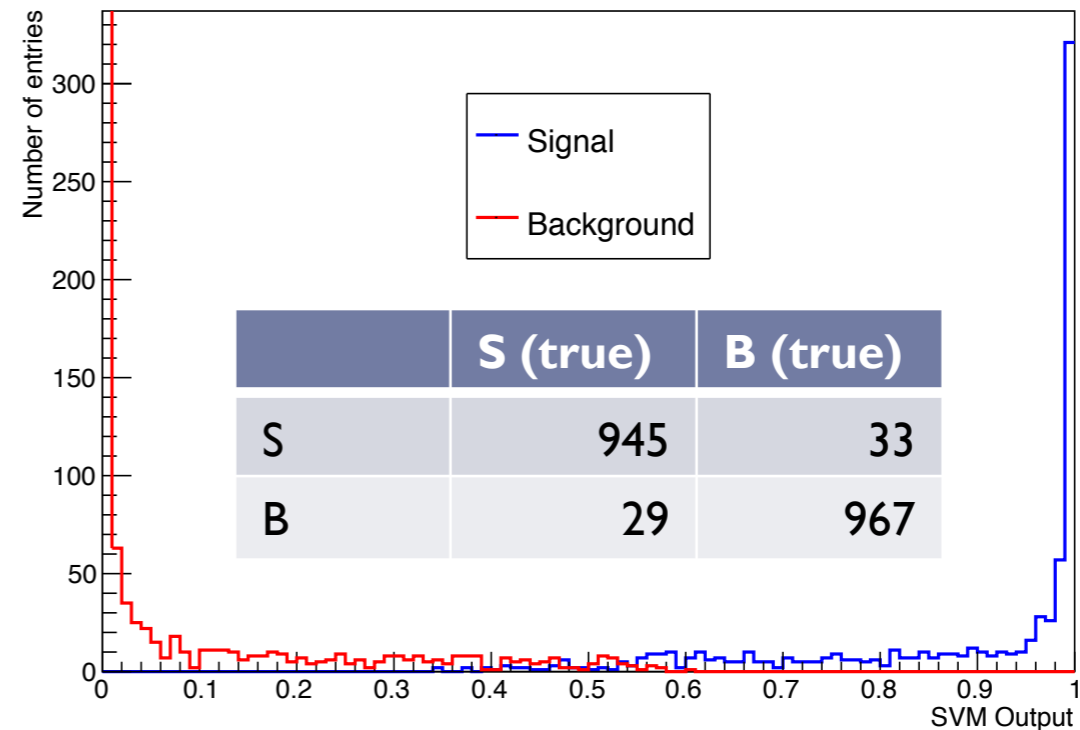
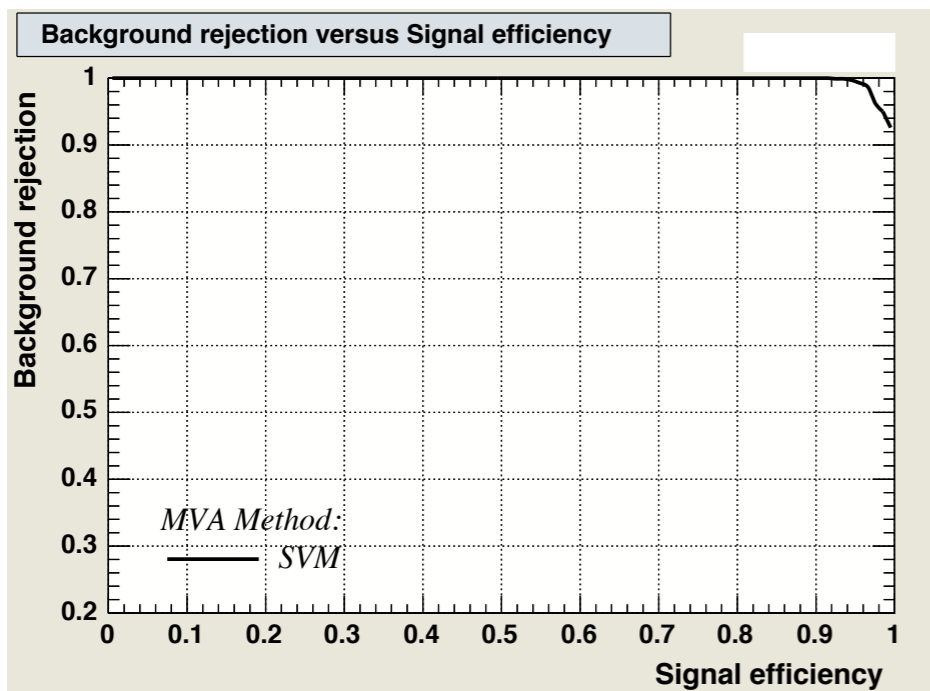
Incorrectly classified events



- ▶ Signal mis-classification rate $\sim 3.3\%$.
- ▶ Background mis-classification rate $\sim 3.7\%$.

EXAMPLES: CHECKER BOARD

- ▶ The confusion matrix ([in-]correctly classified events) for this example shows a high level of correct classification:

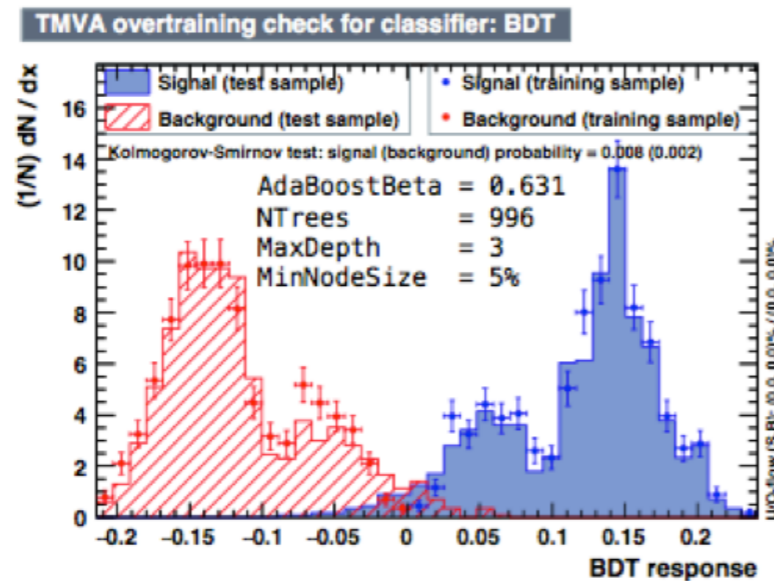
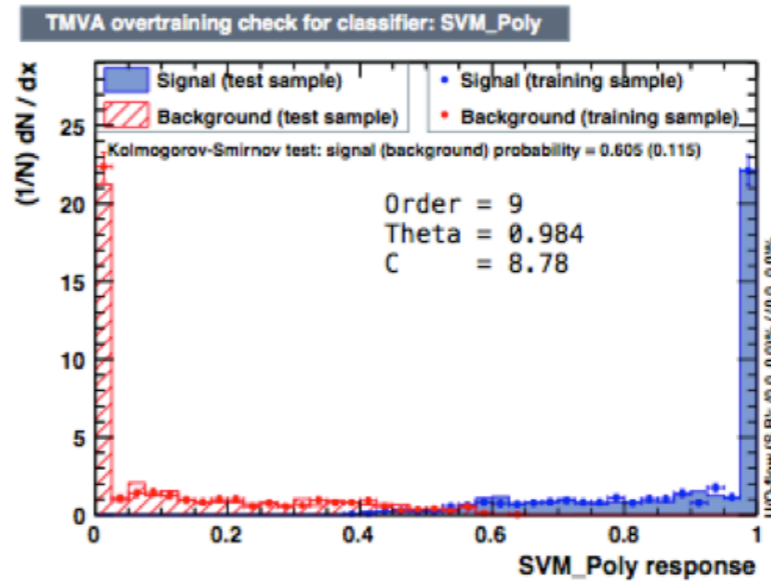
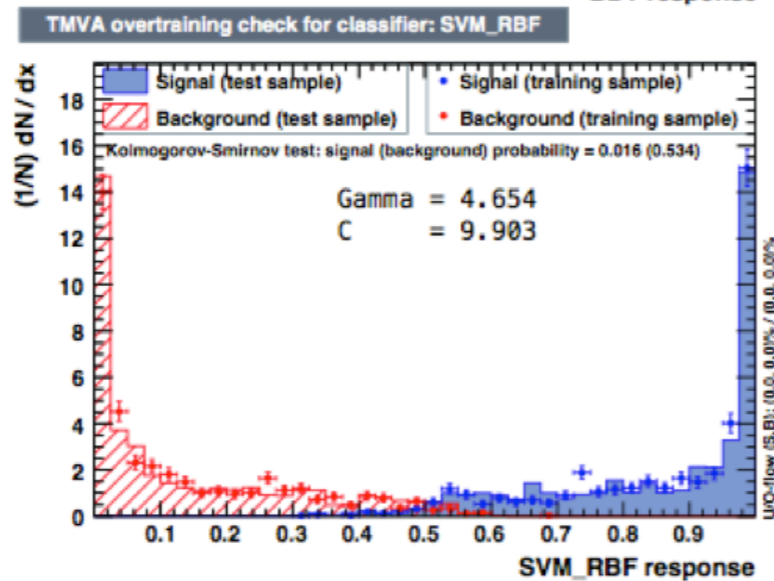
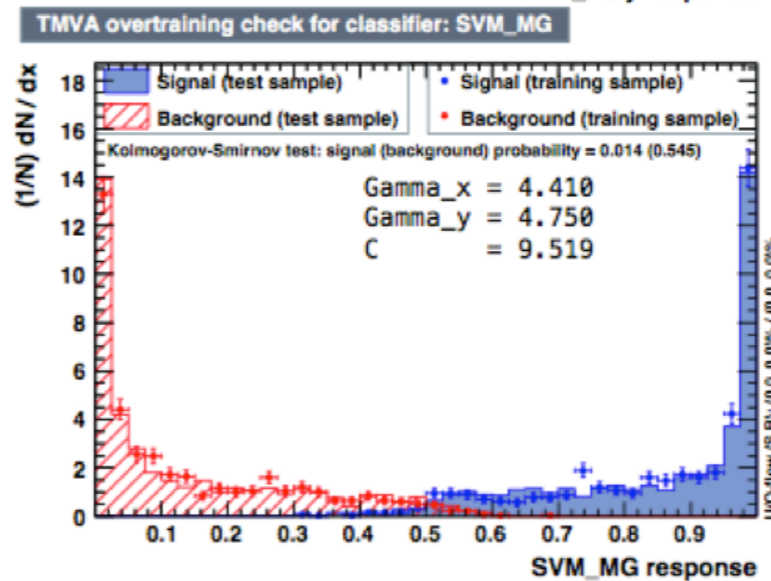


- ▶ This SVM does a good job of separating signal from background.
- ▶ An optimised output would provide a better solution.
- ▶ BDTs and NNs work well with this kind of problem as well.

EXAMPLES: CHECKER BOARD

- ▶ Optimised results for comparison: Very similar responses.

BDT

SVM
Polynomial (1)SVM
RBF (2)SVM
Multi-Gaussian
(3)

Trained using the hold out method of cross validation (what is normally done in TMVA), with optimised hyper-parameters.



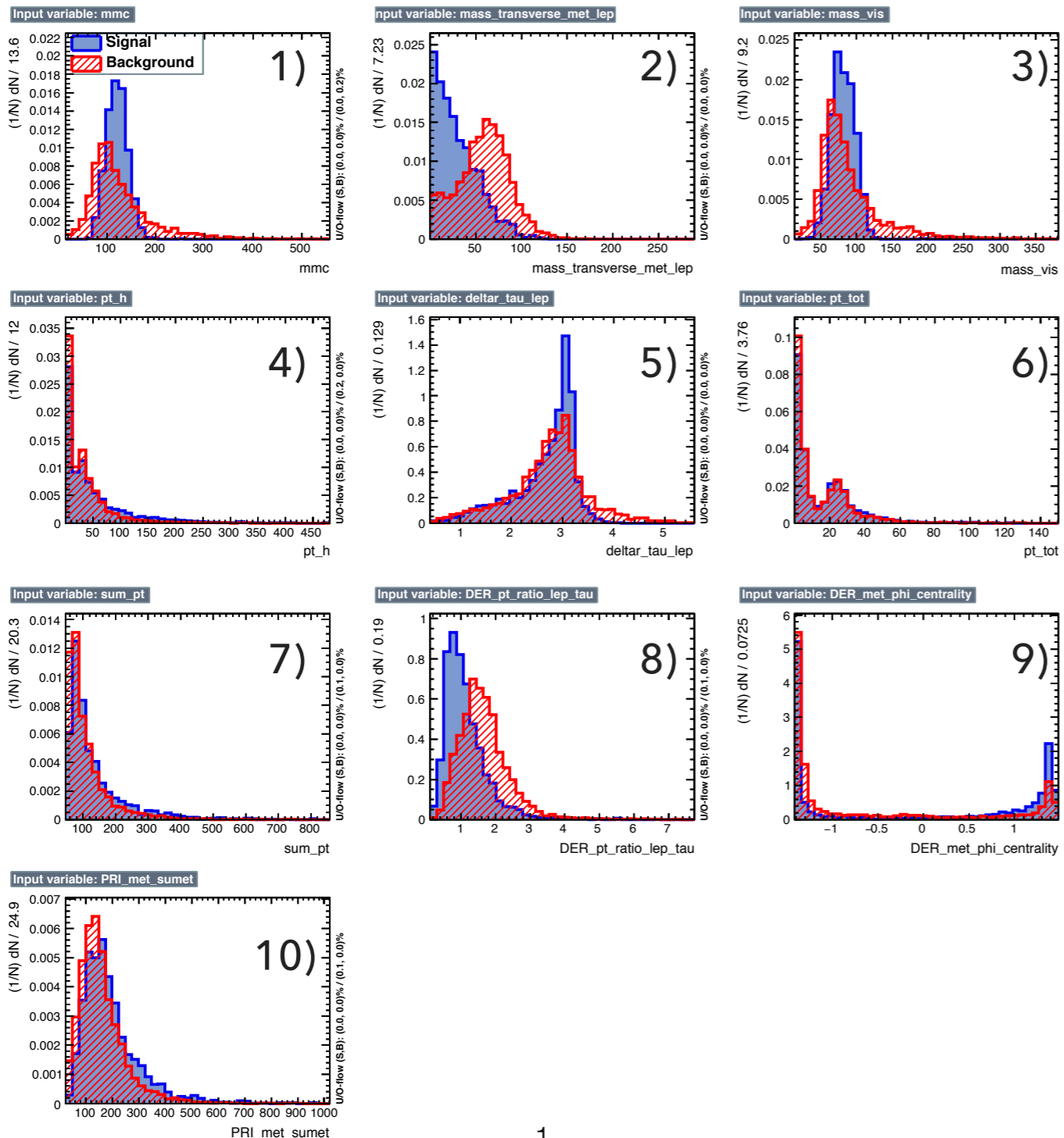
EXAMPLES: $H \rightarrow \tau^+ \tau^-$ (HIGGS KAGGLE DATA CHALLENGE)

- ▶ Use the Kaggle data challenge sample of signal and background events. LHC data (from ATLAS).
- ▶ Packaged up in a convenient format (CSV file).
- ▶ Sufficient description of variables provided for non-HEP users to apply machine learning (ML) techniques to HEP data.
- ▶ Real application to compare performance for different KFs and different MVAs.

<https://www.kaggle.com/c/higgs-boson>

EXAMPLES: $H \rightarrow \tau^+ \tau^-$ (HIGGS KAGGLE DATA CHALLENGE)

► Use 10 variables as inputs; 20K events.



- 1) MMC
- 2) transverse mass between MET and lep
- 3) Visible invariant mass of H
- 4) $p_T(H)$
- 5) R between τ_{had} and lepton
- 6) $p_T(tot)$
- 7) Σp_T
- 8) $p_T(lepton)/p_T(had \tau)$
- 9) MET ϕ centrality
- 10) ET_{total}

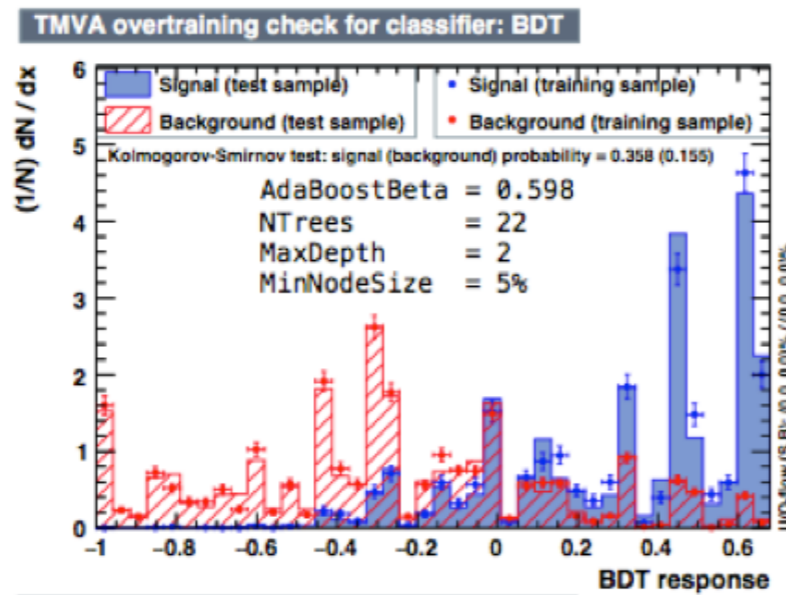
This selection of variables is not optimised, and is selected in order to show a physics example for illustrative purposes.

EXAMPLES: $H \rightarrow \tau^+\tau^-$ (HIGGS KAGGLE DATA CHALLENGE)

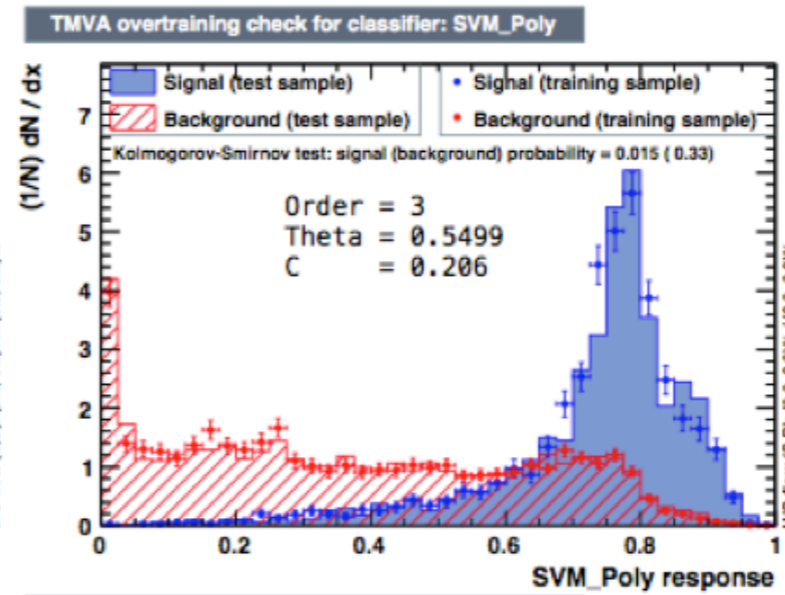
- ▶ NOTE: this is an illustrative example - not a fully optimised analysis of the sample; hyper-parameters are optimised.

Spiky as optimisation chooses a low number of trees.

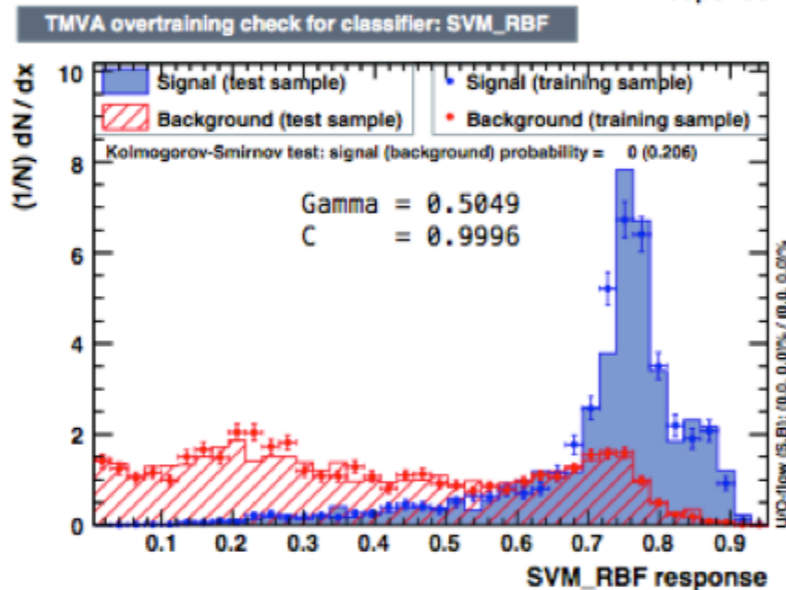
BDT



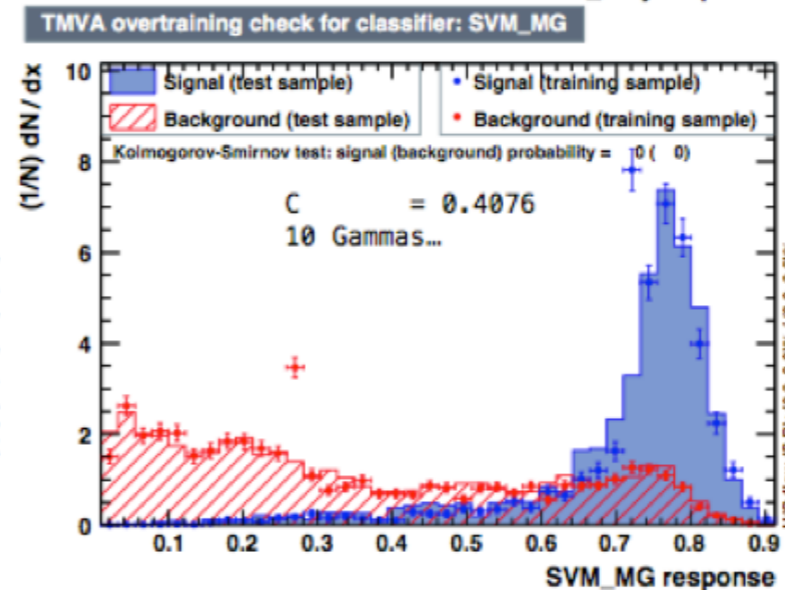
SVM Polynomial (1)



SVM RBF (2)



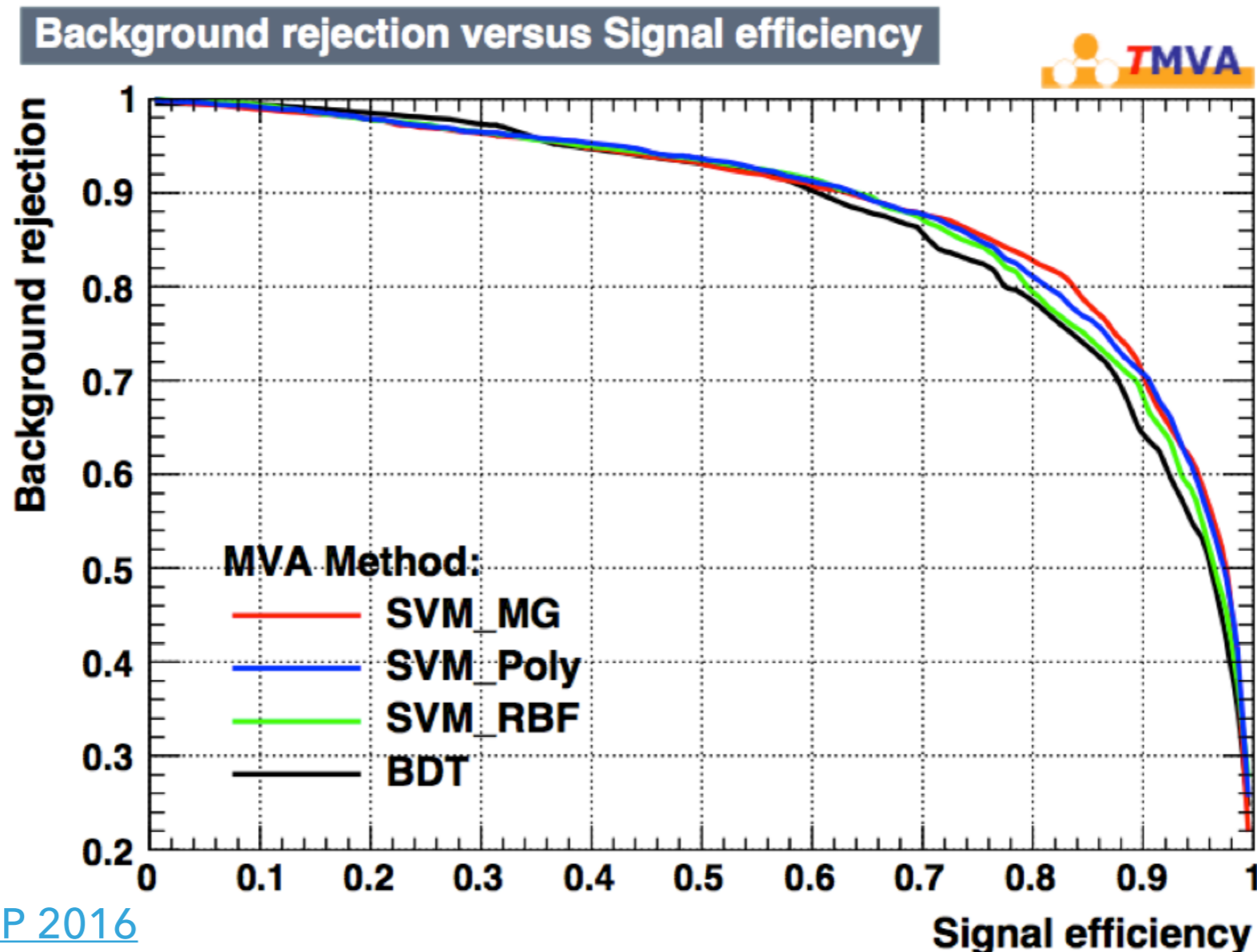
SVM Multi-Gaussian (3)



Trained using the hold out method of cross validation (what is normally done in TMVA), with optimised hyper-parameters.

EXAMPLES: $H \rightarrow \tau^+\tau^-$ (HIGGS KAGGLE DATA CHALLENGE)

- ▶ SVM provides comparable performance to BDT (and neural networks)*.

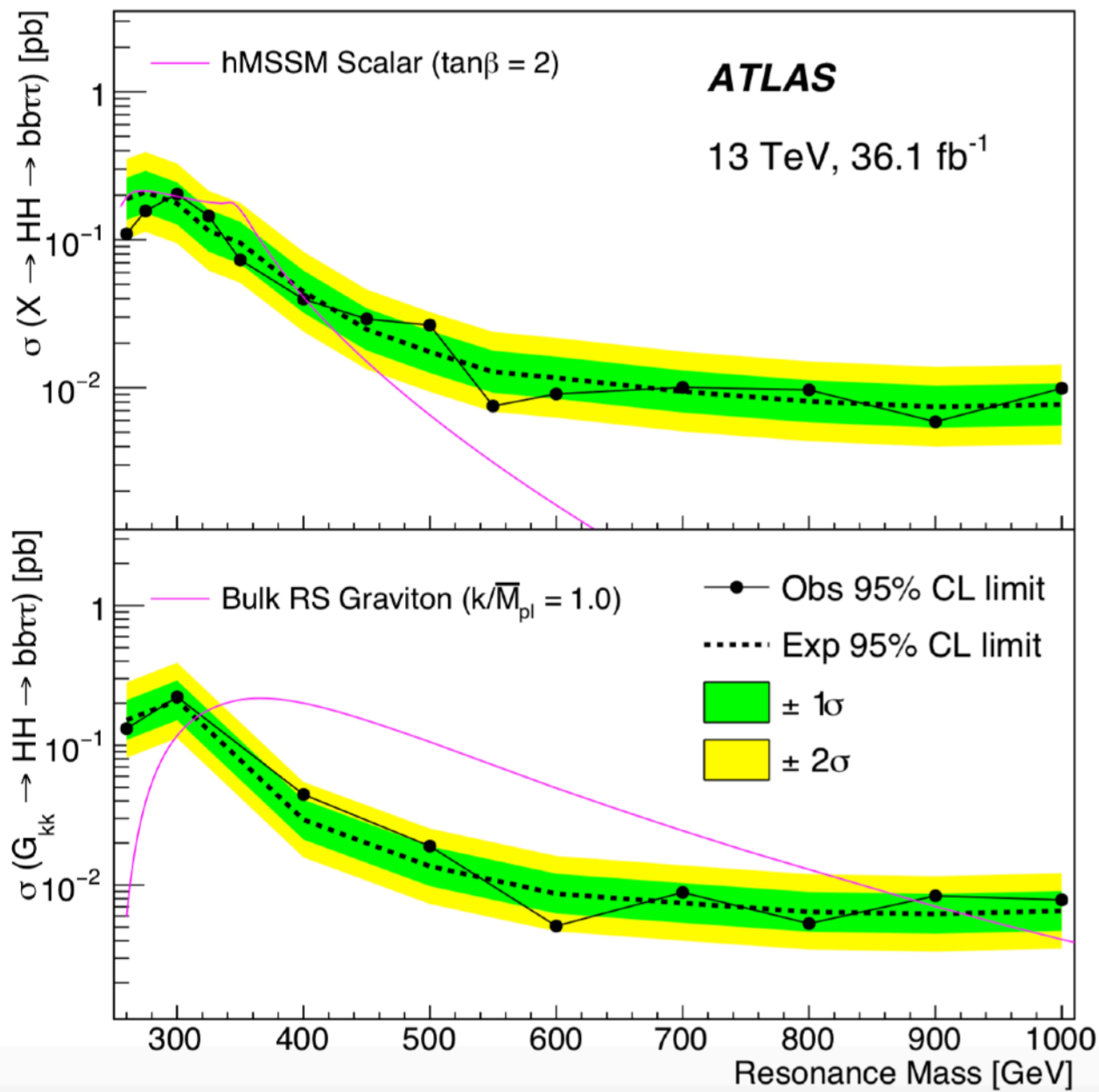


Bevan et al., [proc CHEP 2016](#)

*This general conclusion has been reached in one form or another by people studying BDTs vs SVMs and NNs vs SVMs for HEP problems. The take home message is that SVMs require less data to train in order to obtain a generalised result (follows from the fact there are fewer hyper-parameters to determine for SVMs vs other algorithms).

EXAMPLES: $HH \rightarrow BB\tau^+\tau^-$ (ATLAS - OFFICIAL RESULT)

- ▶ ATLAS recently reported limits on resonant and non-resonant production of HH via $bb\tau^+\tau^-$.

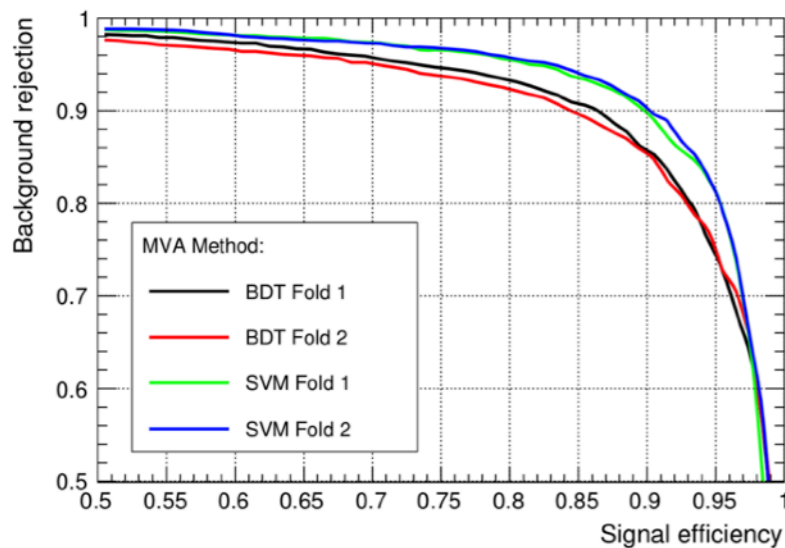


- ▶ The standard analysis shown here uses a BDT for both channels that contribute to the final state:
 - ▶ Two hadronically decaying τ leptons.
 - ▶ One hadronically and one leptonically decaying τ .
- ▶ Results for the SM search are 12.7 times the Standard Model expected sensitivity.

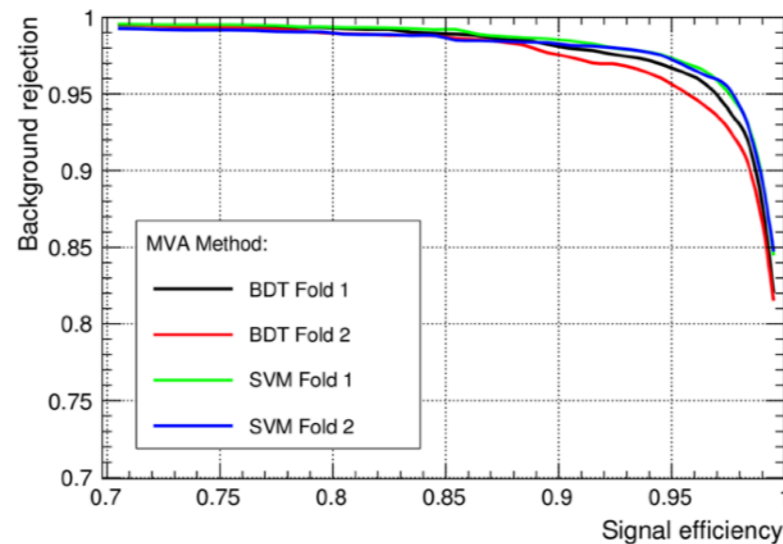
EXAMPLES: $HH \rightarrow BB\tau^+\tau^-$ (ATLAS THESIS)

- ▶ A student working on this mode also looked at using SVMs (instead of BDTs) for the analysis.
- ▶ Similar performance obtained to the official result when using an SVM for both ROC curves and limit plots.

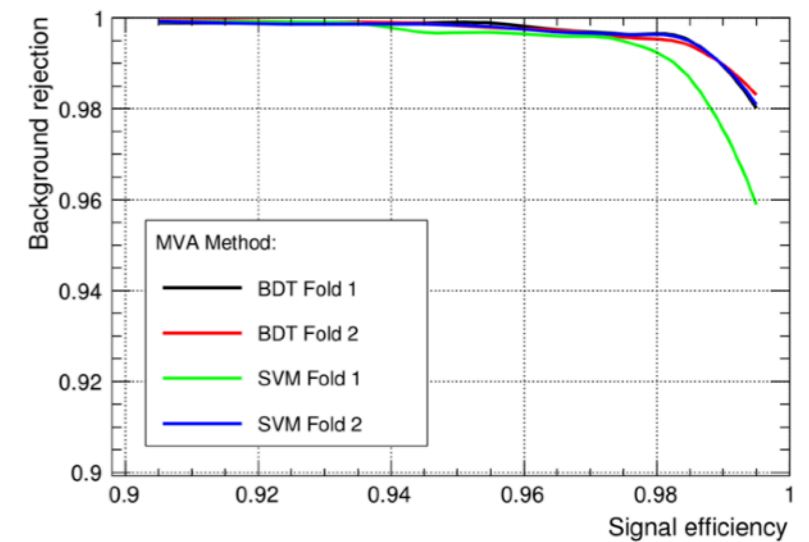
ROC curves for different mass points in the 2HDM search, using one of the trigger lines for the $bb\tau^+\tau^-$ channel.



(a) 2HDM ($m_H = 300$ GeV)



(b) 2HDM ($m_H = 500$ GeV)



(c) 2HDM ($m_H = 800$ GeV)

- ▶ SVMs less susceptible (than BDT) to overtraining for small samples.

EXAMPLES: $HH \rightarrow BB\tau^+\tau^-$ (ATLAS THESIS)

- ▶ A student working on this mode also looked at using SVMs (instead of BDTs) for the analysis.
- ▶ Similar performance obtained to the official result when using an SVM for both ROC curves and limit plots.

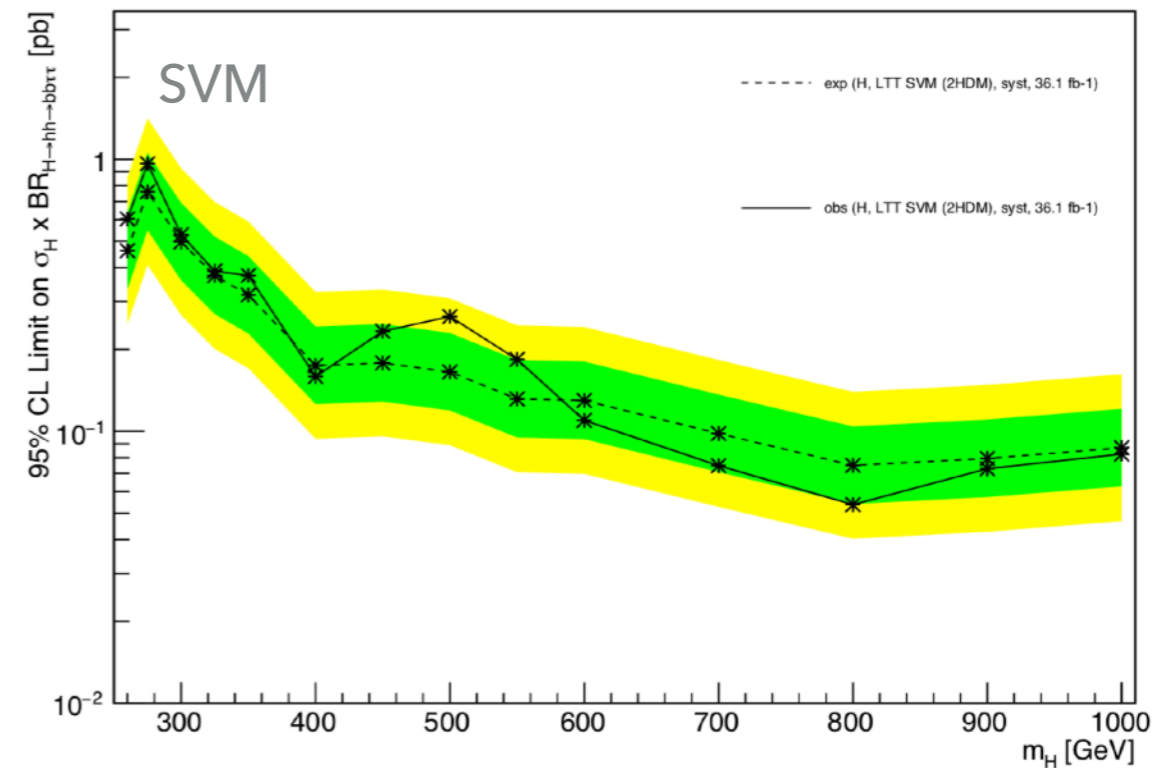
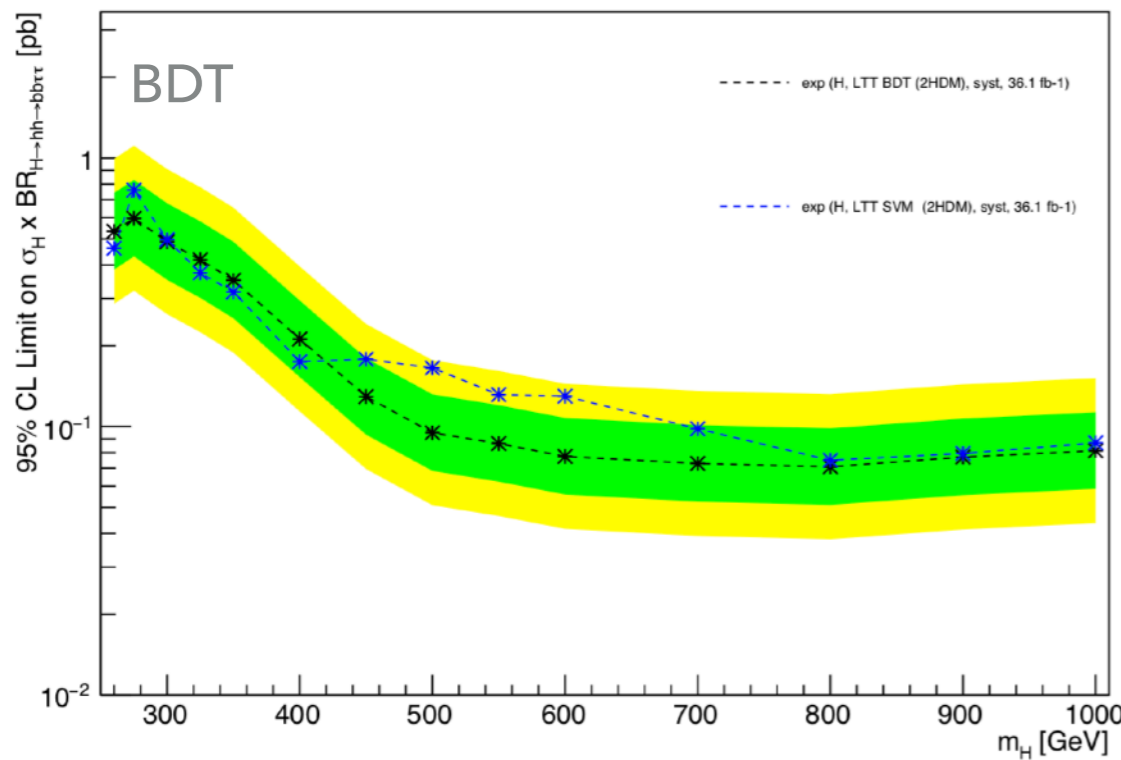


Figure 11.5: Expected limits for the BDT (black) and SVM (blue) at 95% C.L. on the cross-section times branching ratio of the 2HDM heavy scalar Higgs, $H \rightarrow hh \rightarrow bb\tau\tau$, process in the LTT channel.

Figure 11.6: Expected (dashed black) and observed (solid black) limits using SVMs at 95% C.L. on the cross-section times branching ratio of the 2HDM heavy scalar Higgs, $H \rightarrow hh \rightarrow bb\tau\tau$, process in the LTT channel.



EXAMPLES: SVM HINT APPLIED TO CMS DATA

- ▶ Uses libsvm with an RBF kernel function to optimise two parameters: C and Γ .
- ▶ Benchmark example of searching for top squark pair production with stops decaying into the lightest supersymmetric particle (LSP) and a top quark.
 - ▶ Could use the ROC area under the curve (AOC) to optimise on, but this is not directly related to the result being produced.
 - ▶ Instead use the Azimov estimate of the significance of the result as the figure of merit to compare and optimise performance on:

$$Z_A = \left[2 \left((s + b) \ln \left[\frac{(s + b)(b + \sigma_b^2)}{b^2 + (s + b)\sigma_b^2} \right] - \frac{b^2}{\sigma_b^2} \ln \left[1 + \frac{\sigma_b^2 s}{b(b + \sigma_b^2)} \right] \right) \right]^{1/2}$$

This is the median discovery significance from the Poisson form of the signal (s) and background (b), with an uncertainty on the background of σ_b .



EXAMPLES: SVM HINT APPLIED TO CMS DATA

- The variable sets used for the SVM-HINT paper are

	Variable	Set 1	Set 2	Set 3	Set 4
low-level	$p_{T,l}$	•	•		
	η_l	•	•		
	$p_{T,jet(1,2,3,4)}$	•	•		
	$\eta_{jet(1,2,3,4)}$	•	•		
	$p_{T,b\ jet1}$	•	•		
	$\eta_{b\ jet1}$	•	•		
	n_{jet}	•	•		
	$n_{b\ jet}$	•	•		
	E_T	•	•		•
	H_T	•	•		•
high-level	m_T	•		•	•
	m_{T2}^W	•		•	•
	$\Delta\phi(W, l)$	•		•	
	$m(l, b)$	•		•	
	Centrality	•		•	
	Y	•		•	
	H_T -ratio	•		•	
	$\Delta r_{\min}(l, b)$	•		•	
	$\Delta\phi_{\min}(j_{1,2}, E_T)$	•		•	

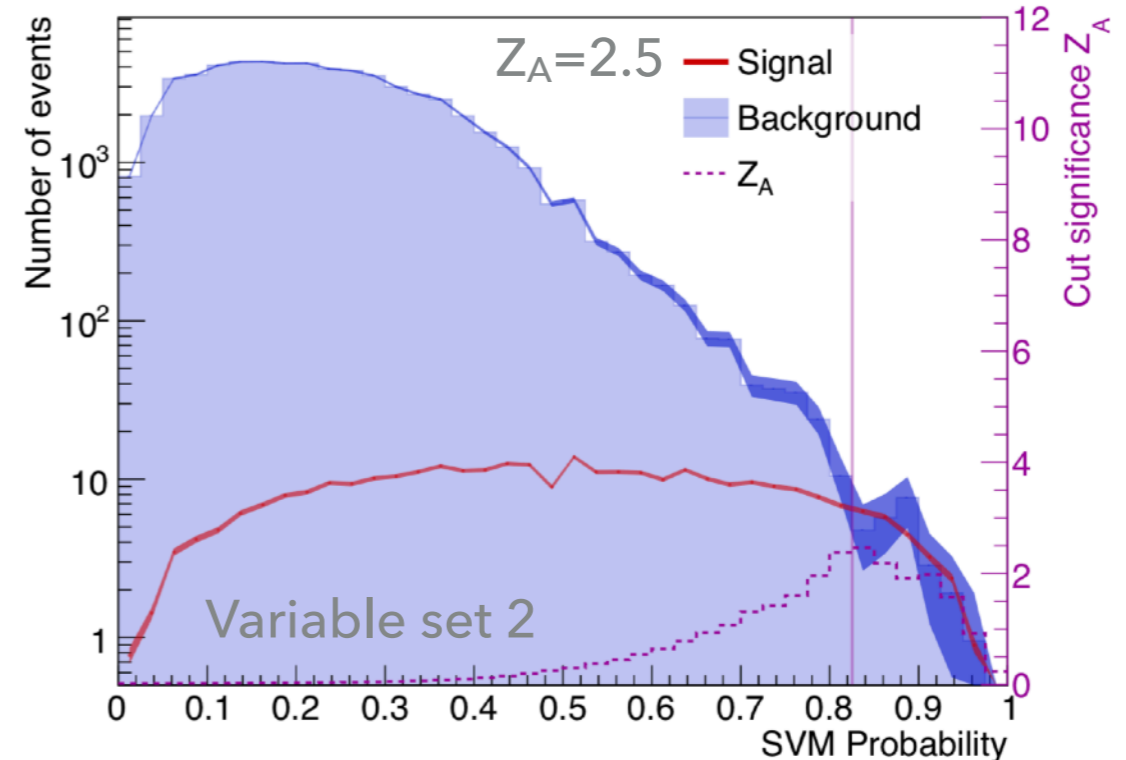
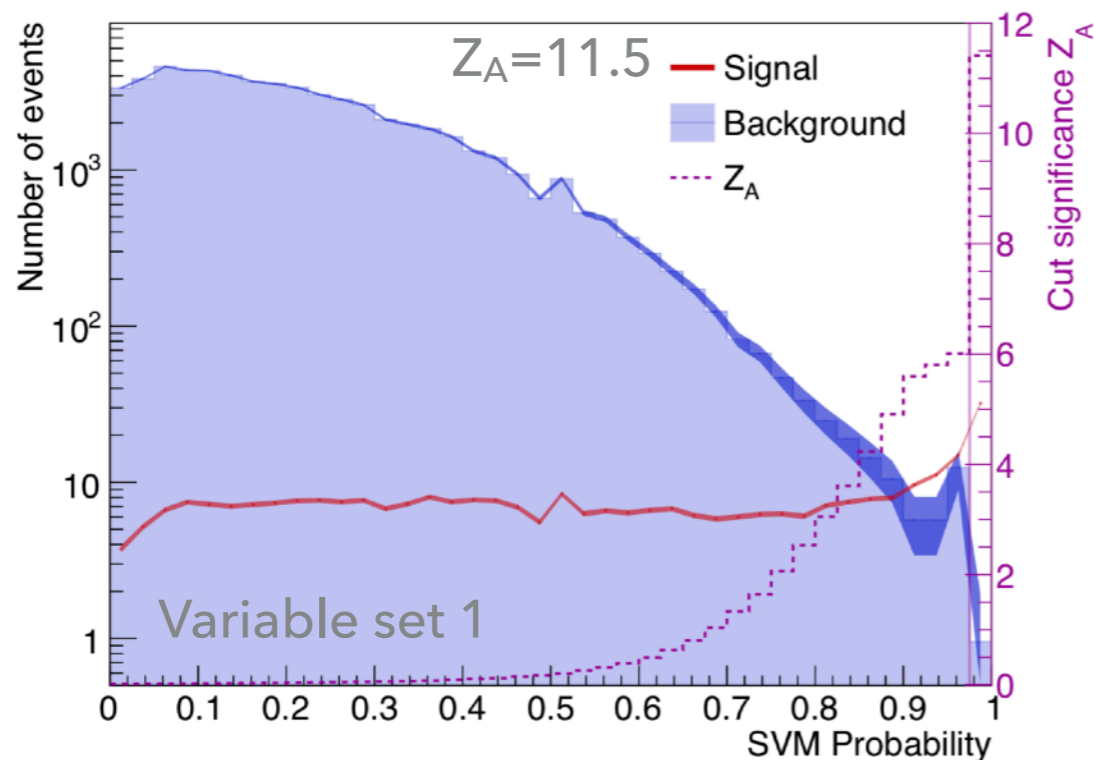
As with other work on using ML methods the expected result that the combination of high level and low level (derived and primitive) features provides better performance than using just one of those sets.

Results on the next two pages illustrate this.

EXAMPLES: SVM HINT APPLIED TO CMS DATA

- ▶ Results are turned into a probabilistic score using a sigmoid function:

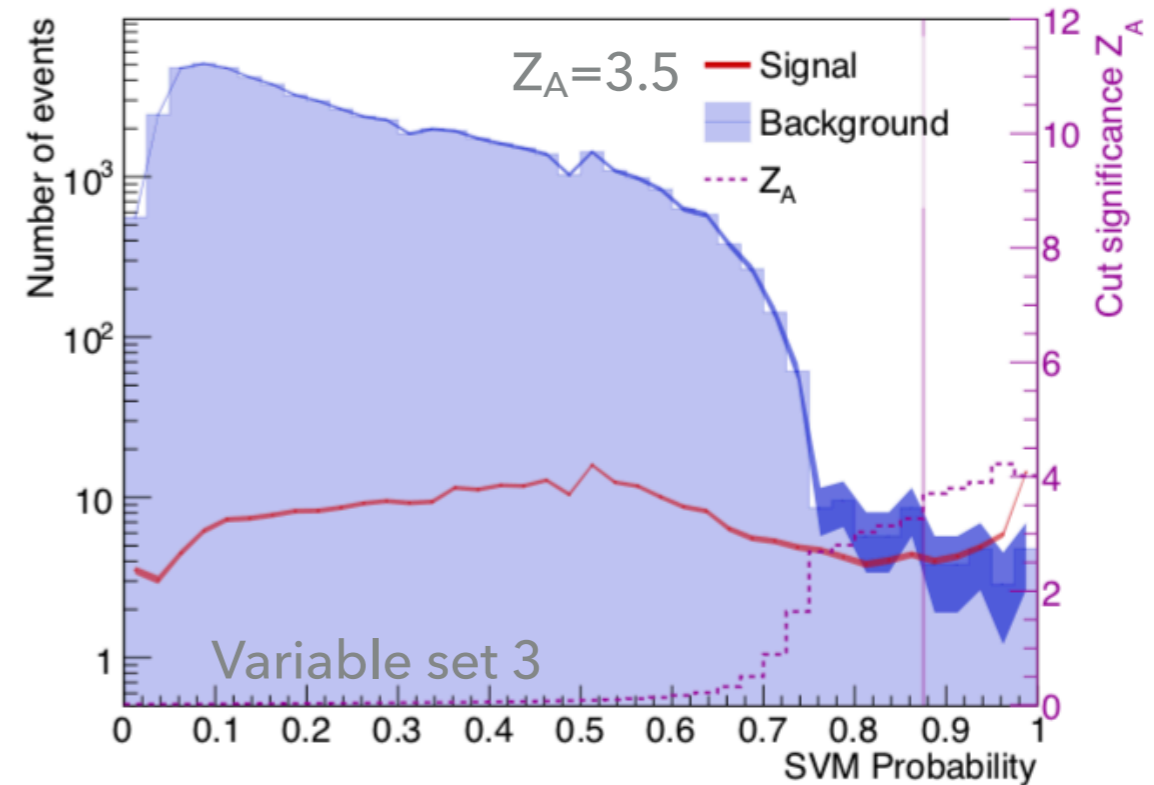
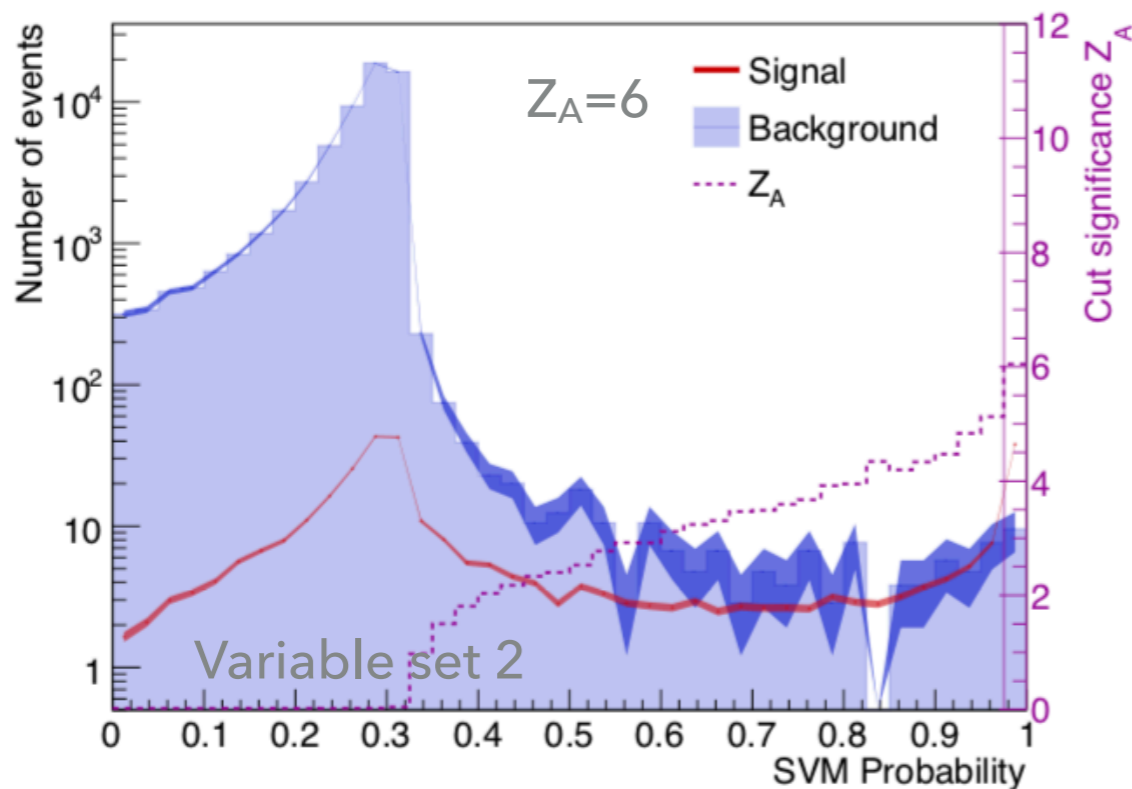
$$P(y = 1|\hat{f}) = \begin{cases} \frac{\exp(-t)}{1+\exp(-t)} & : t \equiv A + B\hat{f} \geq 0 \\ \frac{1}{1+\exp(t)} & : t < 0 \end{cases}$$



EXAMPLES: SVM HINT APPLIED TO CMS DATA

- ▶ Results are turned into a probabilistic score using a sigmoid function:

$$P(y = 1|\hat{f}) = \begin{cases} \frac{\exp(-t)}{1+\exp(-t)} & : t \equiv A + B\hat{f} \geq 0 \\ \frac{1}{1+\exp(t)} & : t < 0 \end{cases}$$





SVMS: SUMMARY AND MISCELLANEOUS NOTES

- ▶ Use SVMs when:
 - ▶ You have small or very small training examples.
 - ▶ If you care about obtaining a generalised result (reproducibility of the output matters even if the data fed to the algorithm changes) and are having difficulty with other algorithms (e.g. BDT, ANN, ...).
 - ▶ Computing time/resource (esp. memory) is not a problem.
- ▶ Do not use an SVM when:
 - ▶ You have a lot of training examples and/or very little computing resource.



SVMS: SUMMARY AND MISCELLANEOUS NOTES

- ▶ We've looked at the hard and soft margin SVMs.
 - ▶ The algorithm stems from the same linear separation problem that is addressed by Rosenblatt's perceptron paper.
 - ▶ However this focusses on how far an example is from the margin defining the separating hyperplane.
 - ▶ Can't understand the mapping from the input feature space to the dual space (but we don't have to).
- ▶ SVMs are widely used outside of HEP.
- ▶ They have been used for a broad range of physics studies in HEP, but the algorithm has not been widely adopted.
- ▶ There are specific reasons why you would or would not want to use the algorithm.
- ▶ Searches where you have limited training examples available (e.g. SUSY or Higgs BSM) are cases where you might want to look at the algorithm.

KNN: K-NEAREST NEIGHBOURS



K-NEAREST NEIGHBOURS (AKA K-MEANS)

- ▶ This is a clustering algorithm, and an example of unsupervised learning.
- ▶ Aim: determine the centroid positions C of K clusters in the data containing N examples using a Euclidean distance from the cluster mean to some data example.
- ▶ Optimisation: The variance of the clusters is minimised in order to determine the corresponding means of the cluster.

K-NEAREST NEIGHBOURS (AKA K-MEANS)

▶ Step 1:

- ▶ Given C compute the total cluster variance and minimise this with respect to the means of the clusters.

$$\min_{c, \{m_k\}_1^K} \sum_{k=1}^K N_k \sum_{C(i)=k} ||x_i - m_k||^2$$

x_i : i^{th} example

N_k : Number of examples in K^{th} cluster

m_k : Centroid of K^{th} cluster

k : Cluster index

- ▶ This gives the current mean positions of the clusters.

▶ Step 2:

- ▶ Given a set of means m , minimise these by assigning elements to the closest current cluster mean. i.e.

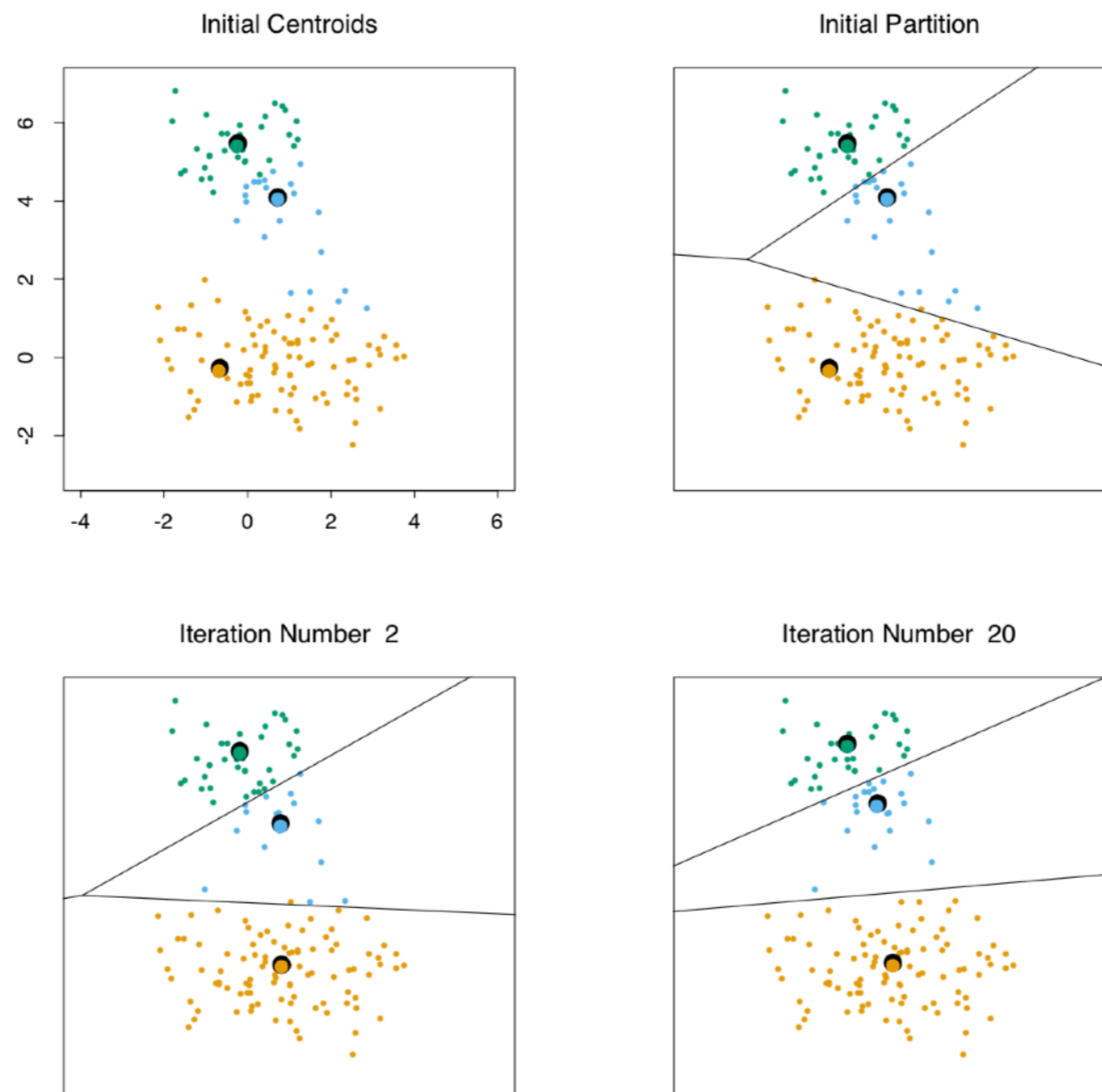
$$C(i) = \operatorname{argmin}_{1 \leq k \leq K} ||x_i - m_k||^2$$

▶ Step 3:

- ▶ Iterate until the assignments stabilise.

K-NEAREST NEIGHBOURS (AKA K-MEANS)

- ▶ This example shows successive iterations of the K-means algorithm to a set of data with $K=3$.



This algorithm has the number of clusters, K , as a parameter.

Clustering results will depend on the choice of K .

Colour indicates example assignment to a given cluster.

EXPLAINABILITY AND INTERPRETABILITY



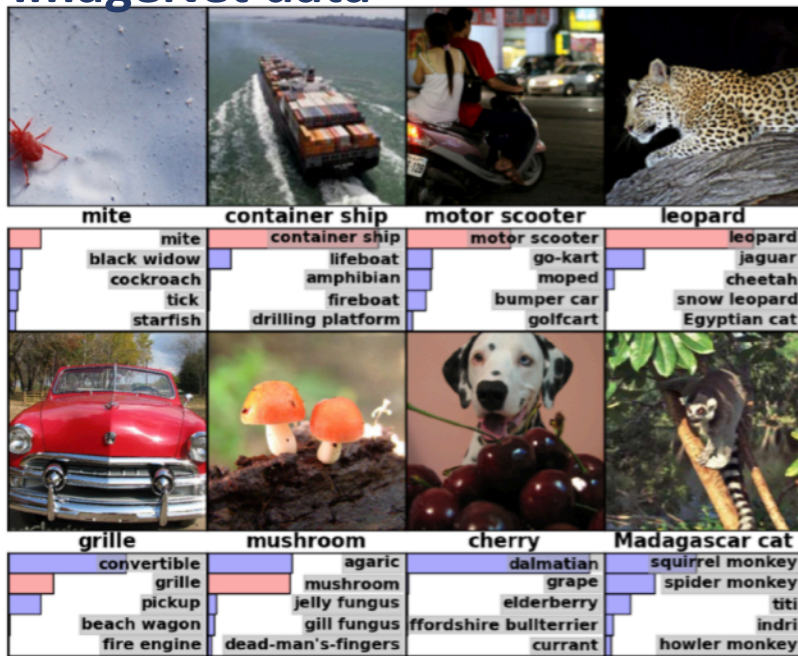
EXPLAINABILITY AND INTERPRETABILITY

- ▶ The issue of how to explain the model, and how to interpret it is challenging.
 - ▶ e.g. why was a given prediction made?
 - ▶ Event classification / decision making
 - ▶ Real value prediction (e.g. signal strength in a score)
- ▶ There is no consensus on how to approach this problem; it is an active research area.
- ▶ Highlight just a few ways we can help to elucidate our models.

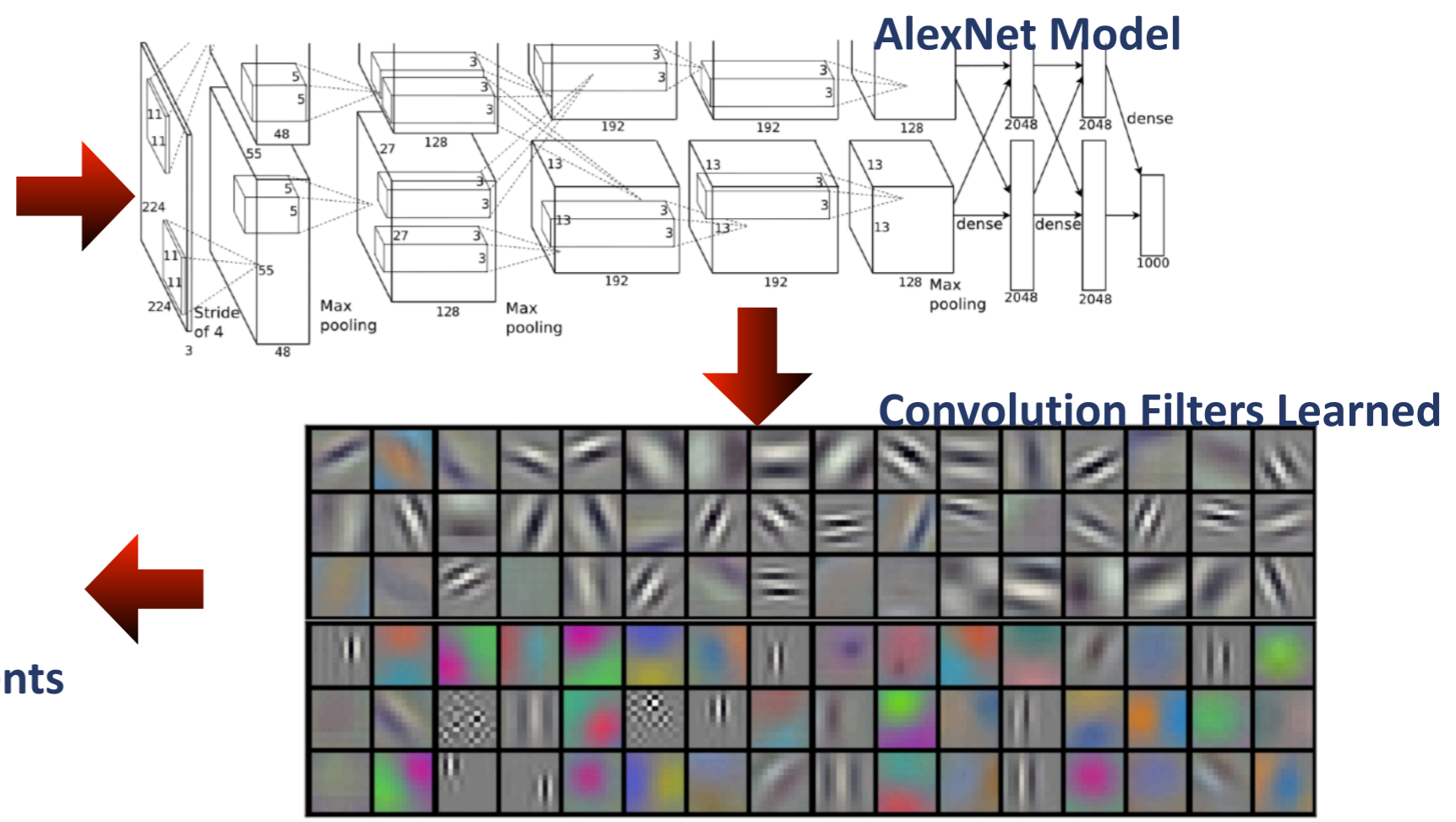
EXPLAINABILITY AND INTERPRETABILITY

- ▶ CNN filter maps provide information about shapes and colour that can be used to interpret how features are identified.

ImageNet data



Label assignments

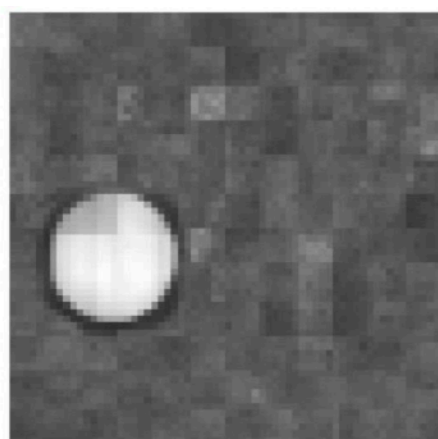


- ▶ Requires effort to "see what is happening in many cases"

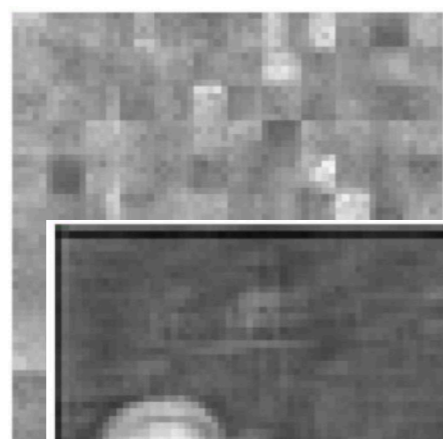


EXPLAINABILITY AND INTERPRETABILITY

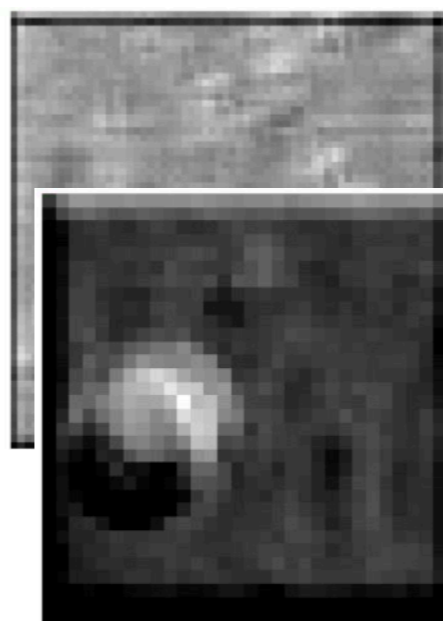
- ▶ Some problems have simpler filter interpretations.



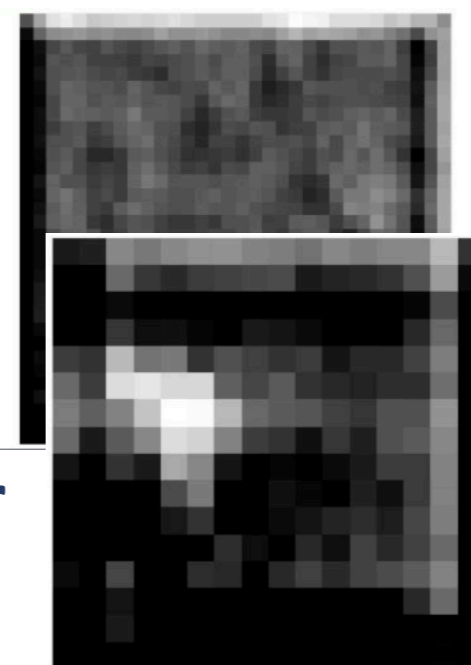
Input images



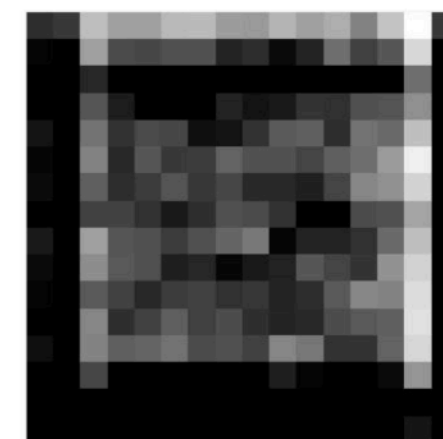
1st Conv layer



2nd Conv layer



3rd Conv layer



Deeper level of abstraction



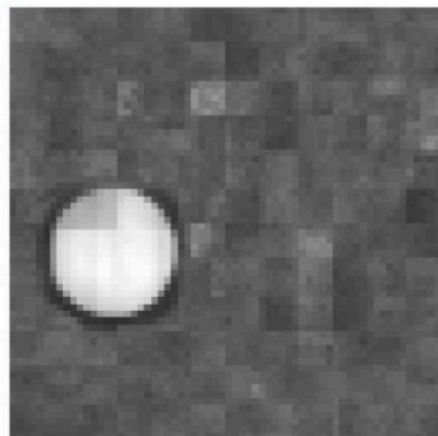
MoEDAL

These images show an alignment pin hole in a MoEDAL NTD sample

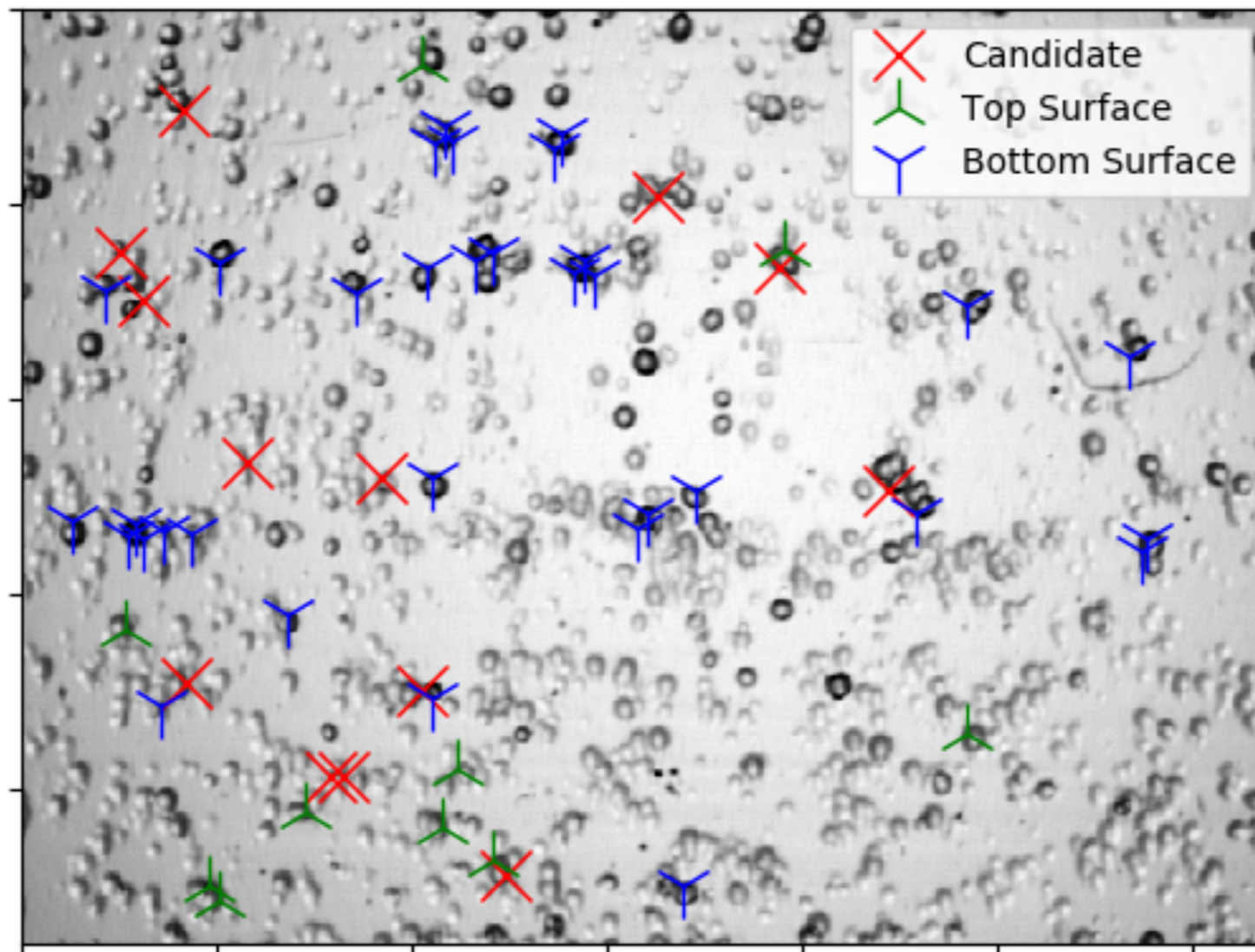


EXPLAINABILITY AND INTERPRETABILITY

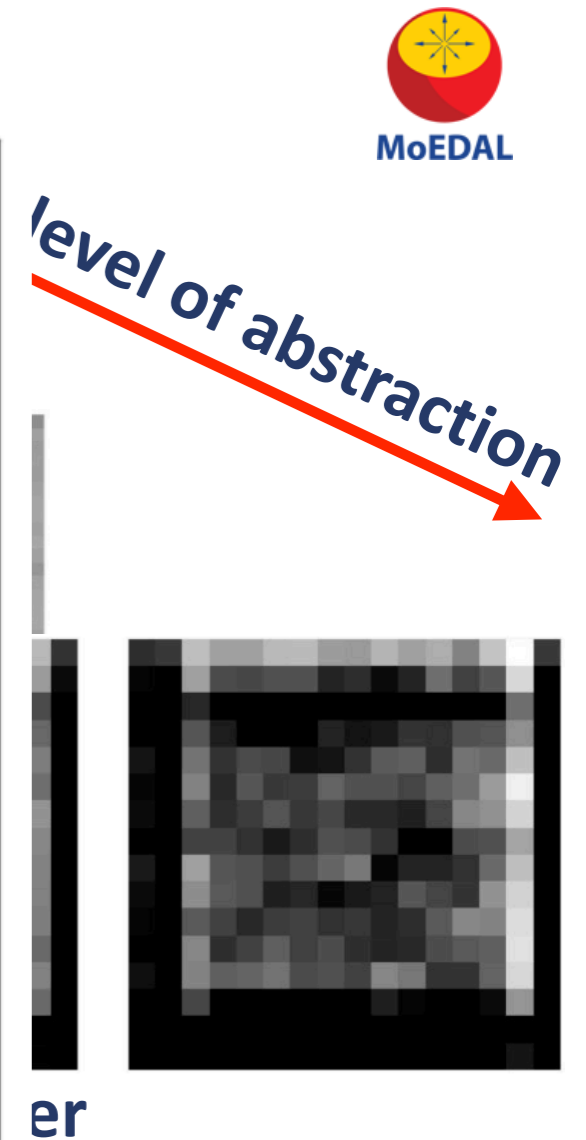
- ▶ Some problems have simpler filter interpretations.



Input images

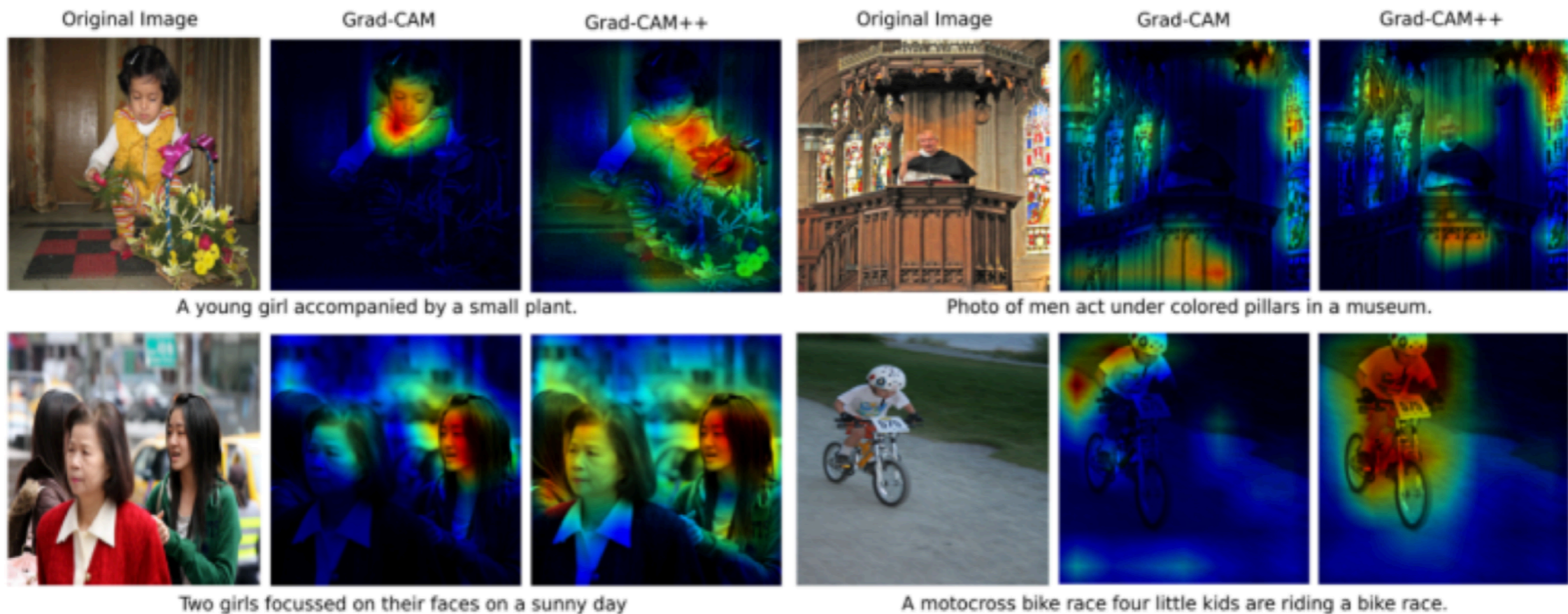


These images show alignment pin hole in MoEDAL NTD sample



EXPLAINABILITY AND INTERPRETABILITY

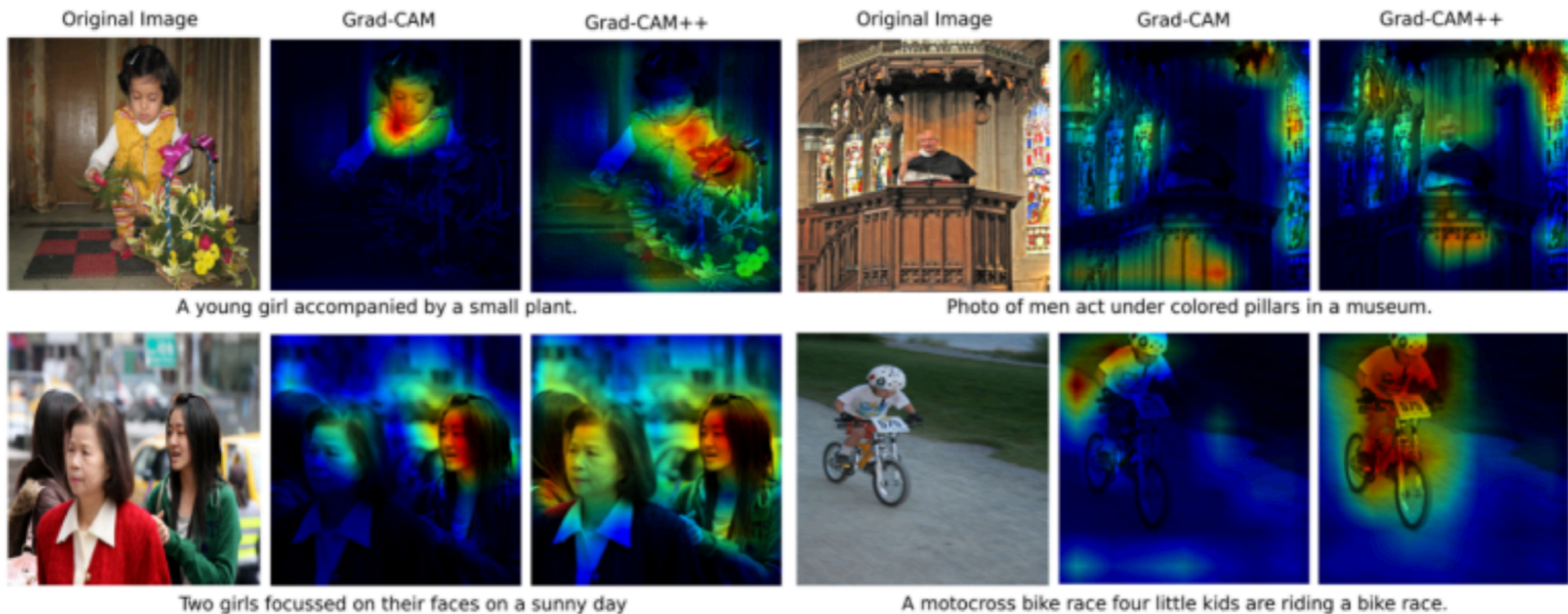
- ▶ There are methods that use gradients and back-propagation to indicate which local regions of an image lead to a particular decision for CNNs: e.g. GradCam, Guided Back Propagation and variants thereof.



- ▶ There are also generalisations for DNNs.

EXPLAINABILITY AND INTERPRETABILITY

- ▶ There are methods that use gradients and back-propagation to indicate which local regions of an image lead to a particular decision for CNNs: e.g. GradCam, Guided Back Propagation and variants thereof.



- ▶ There are also generalisations for DNNs.



EXPLAINABILITY AND INTERPRETABILITY

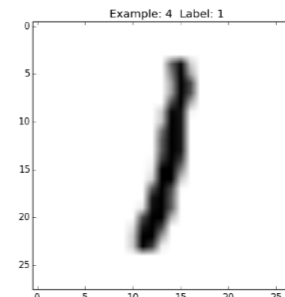
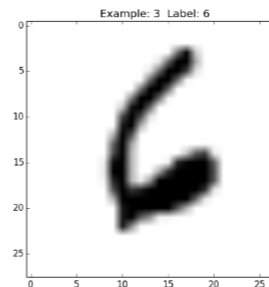
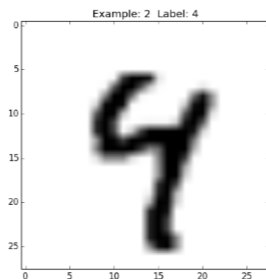
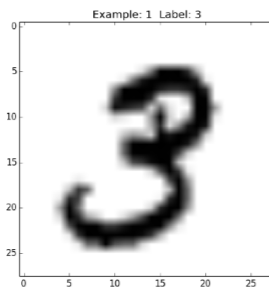
- ▶ Complicated models that rely on function approximation through deep abstractions, or implicit mappings into high dimensional feature spaces can be challenging to understand.
- ▶ Interpretation of their results can be straightforward or challenging.
- ▶ These however are one class of models; other machine learning algorithms can be more transparent (e.g. Decision Trees).
- ▶ Bayesian networks (not discussed here), require causal input in order to construct models, and are by construction easier to interpret than the methods discussed here.

DATA:
MNIST
CFAR-10
CFAR-100
KAGGLE
UCI ML DATA REPOSITORY
TIMIT
RCV1-V2
DEEP LEARNING USING LOW LEVEL FEATURES
CROSS VALIDATION

APPENDIX

APPENDIX: DATA — MNIST

- ▶ MNIST is a standard data set for hand writing pattern recognition. e.g. the numbers 1, 2, 3, ... 9, 0



- ▶ 60000 training examples
- ▶ 10000 test examples
- ▶ These are 8 bit greyscale images (one number required to represent each pixel)
- ▶ Renormalise $[0, 255]$ on to $[0, 1]$ for processing.
- ▶ Each image corresponds to a 28x28 pixel array of data.
- ▶ For an MLP this translates to 784 features.



APPENDIX: DATA — CFAR-10

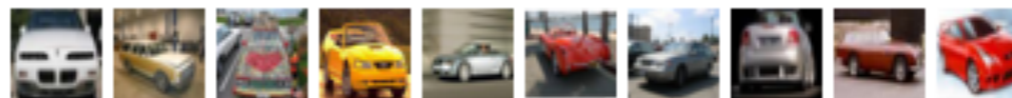
- ▶ 60k 32x32 colour images (so each image is a tensor of dimension 32x32x3).
- ▶ This is a labelled subset of an 80 million image dataset.

- ▶ 10 classes:

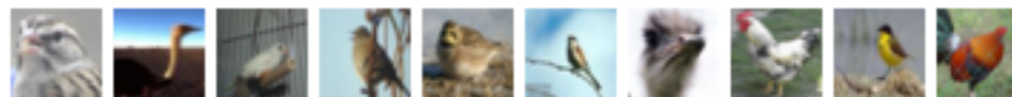
airplane



automobile



bird



cat



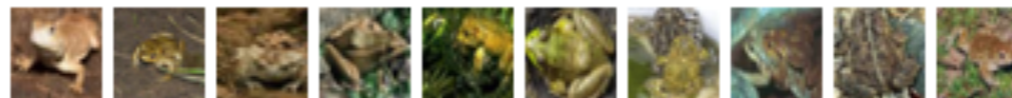
deer



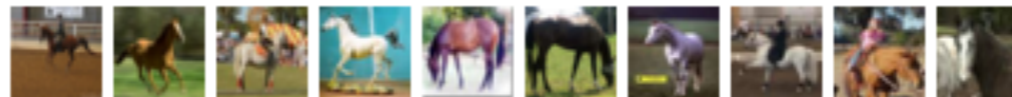
dog



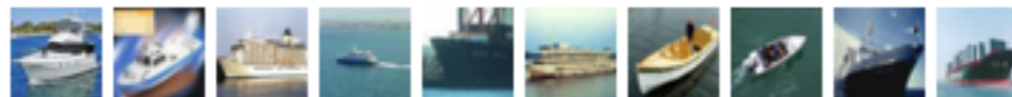
frog



horse



ship



truck





APPENDIX: DATA — CFAR-100

- ▶ 100 class variant on the CFAR10 sample:
- ▶ 32x32 colour images (so each image is a tensor of dimension 32x32x3).

- ▶ 100 classes:

Superclass

aquatic mammals
fish
flowers
food containers
fruit and vegetables
household electrical devices
household furniture
insects
large carnivores
large man-made outdoor things
large natural outdoor scenes
large omnivores and herbivores
medium-sized mammals
non-insect invertebrates
people
reptiles
small mammals
trees
vehicles 1
vehicles 2

Classes

beaver, dolphin, otter, seal, whale
aquarium fish, flatfish, ray, shark, trout
orchids, poppies, roses, sunflowers, tulips
bottles, bowls, cans, cups, plates
apples, mushrooms, oranges, pears, sweet peppers
clock, computer keyboard, lamp, telephone, television
bed, chair, couch, table, wardrobe
bee, beetle, butterfly, caterpillar, cockroach
bear, leopard, lion, tiger, wolf
bridge, castle, house, road, skyscraper
cloud, forest, mountain, plain, sea
camel, cattle, chimpanzee, elephant, kangaroo
fox, porcupine, possum, raccoon, skunk
crab, lobster, snail, spider, worm
baby, boy, girl, man, woman
crocodile, dinosaur, lizard, snake, turtle
hamster, mouse, rabbit, shrew, squirrel
maple, oak, palm, pine, willow
bicycle, bus, motorcycle, pickup truck, train
lawn-mower, rocket, streetcar, tank, tractor




APPENDIX: DATA — KAGGLE

- ▶ Well known website for machine learning competitions; lots of problems and lots of different types of data.
- ▶ Also includes training material at:
 - ▶ <https://www.kaggle.com/learn/overview>
 - ▶ e.g. Intro to machine learning includes a data science problem on [predicting titanic survivors](#) from a limited feature space.
 - ▶ Since the outcome is known, this is a good sample of real world data to try out your data science skills.

Getting Started Prediction Competition

Titanic: Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics

 Kaggle · 11,175 teams · Ongoing

[Overview](#) [Data](#) [Notebooks](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Join Competition](#)



APPENDIX: DATA — UCI ML DATA REPOSITORY



- ▶ Hundreds of data sets covering life sciences, physical sciences, CS / Engineering, Social Sciences, Business, Game and other categories of data.
- ▶ Different types of problem: including Classification, regression and clustering samples.
- ▶ Different types of data: e.g. Multivariate, univariate, time-series etc.
- ▶ <https://archive.ics.uci.edu/ml/datasets.php>



APPENDIX: DATA — TIMIT

- ▶ A corpus of acoustic-phonetic continuous speech data, provided with extensive documentation.
- ▶ Includes audio files and transcripts
- ▶ 630 speakers, each with 10 sentences, corresponding to a corpus of 25200 files (4 files per speaker).
- ▶ Total size is approximately 600Mb.

<https://catalog.ldc.upenn.edu/LDC93S1>



APPENDIX: DATA — RCV1-V2

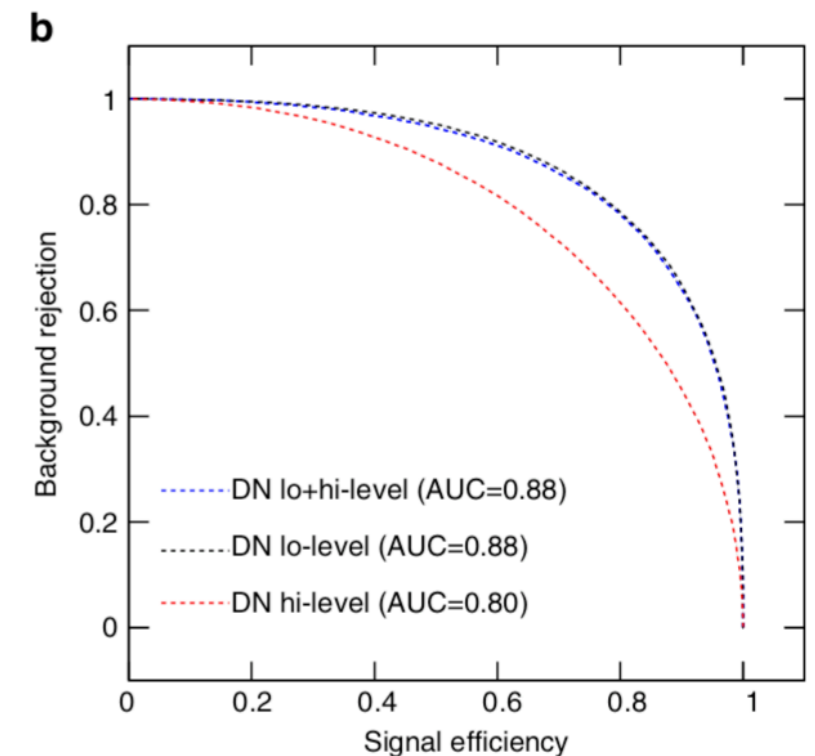
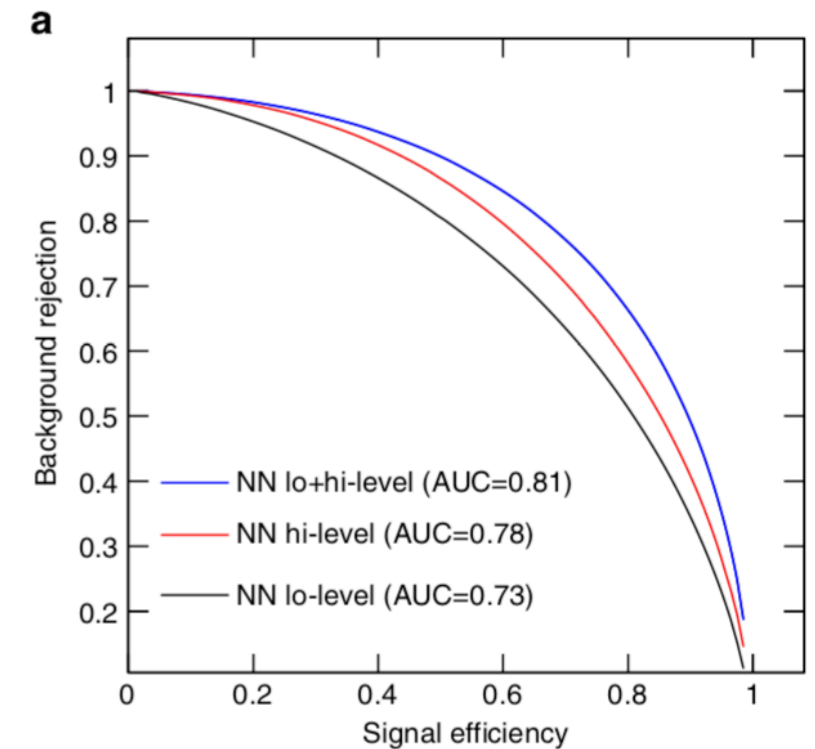
- ▶ RCV1: A New Benchmark Collection for Text Categorization Research
- ▶ A detailed description of this text categorisation data set can be found in: <http://www.jmlr.org/papers/volume5/lewis04a/lewis04a.pdf>

http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm



APPENDIX: DEEP LEARNING USING LOW LEVEL FEATURES

- ▶ Baldi et al. have reported the ability for a deep network to learn additional information from low level features over and above the high level features; doing function approximation from energy and momenta.
 - ▶ 2.6 million (100k) training (validation) examples.
 - ▶ 5 layer network with 300 hidden units in each layer.
 - ▶ learning rate 0.05 and weight decay coef. of 10^{-5} .
 - ▶ Improves discovery significance over and above a NN.
- ▶ Good illustration, is not a realistic scenario as:
 - ▶ No systematics included.
 - ▶ Relies on very large training samples (unrealistic for many LHC scenarios).
 - ▶ FOM optimised is the AUC - we measure limits, cross sections and parameters relating to decay properties or fundamental quantities of the (SM) model.
 - ▶ Anecdotally I've found smart learning (SL) and deep learning (DL) perform equally well in many scenarios with realistic HEP Monte Carlo/ data control sample constraints. SL algs. are less resource hungry than DL ones.





APPENDIX: CROSS VALIDATION

- ▶ In statistics cross validation is used to understand the mean and variance of estimations of model predictions from data.
 - ▶ The bias will be irreducible and mean that the predictions made will have some systematic effect related to the average output value.
 - ▶ The variance will depend on the size of the training sample.
 - ▶ The central limit theorem tells us that:

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

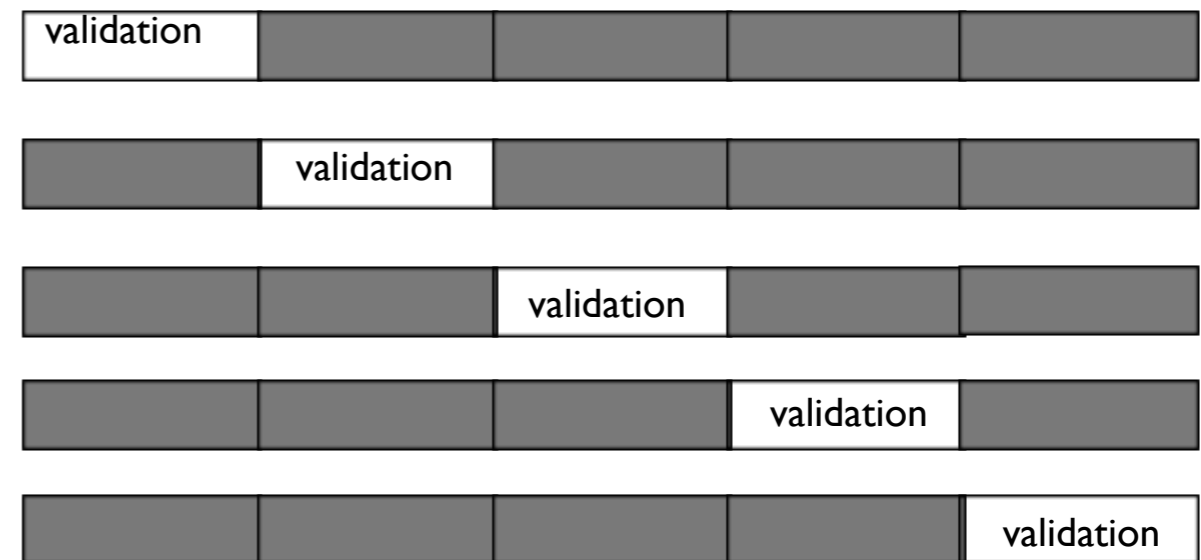
$$M = \sum_{i=1}^N \mu_i$$

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



APPENDIX: CROSS VALIDATION

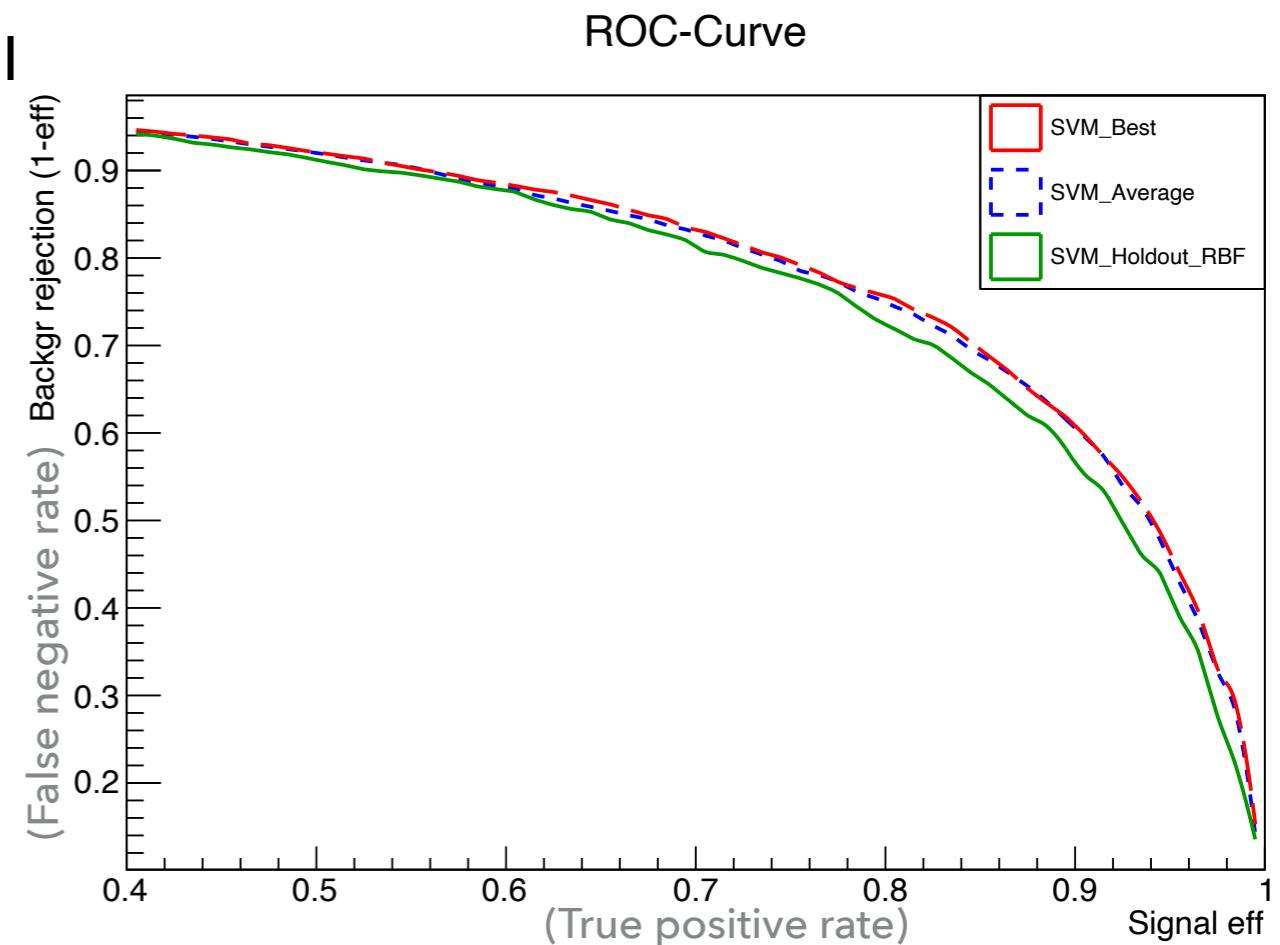
- ▶ Application of this concept to machine learning can be seen via k-fold cross validation and its variants*
- ▶ Divide the data sample for training and validation into k equal sub-samples.
- ▶ From these one can prepare k sets of validation samples and residual training samples.
- ▶ Each set uses all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



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

APPENDIX: CROSS VALIDATION

- ▶ Application of this concept to machine learning can be seen via k-fold cross validation and its variants*
- ▶ The ensemble of response function outputs will vary in analogy with the spread of a Gaussian distribution.
- ▶ This results in family of ROC curves; with a representative performance that is neither the best or worst ROC.
- ▶ The example shown is for a Support Vector Machine, but the principle is the same.
- ▶ It is counter-intuitive, but the robust response comes from the average, not the best performance using the ROC FOM.



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