Queen Mary
University of London

DR ADRIAN BEVAN

MULTIVARIATE ANALYSIS AND ITS USE IN HIGH ENERGY PHYSICS

3) NEURAL NETWORKS

Lectures given at the department of Physics at CINVESTAV, Instituto Politécnico Nacional, Mexico City
28th August - 3rd Sept 2018

LECTURE PLAN

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

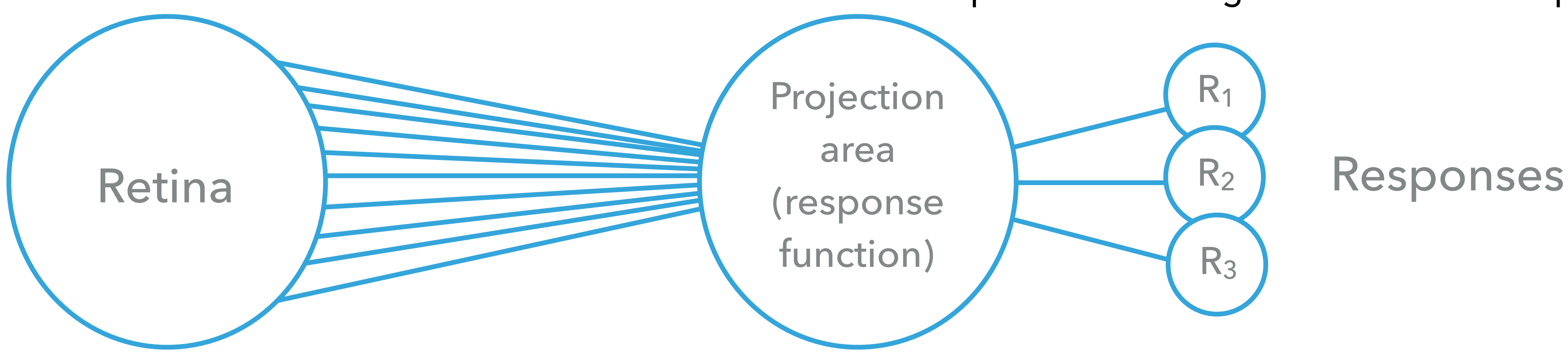
INTRODUCTION

- Neural networks are widely used machine learning algorithms.
- Recent developments in computing have led to “deep learning" applications of neural networks, which I will cover later on once we have gone over the groundwork for more traditional perceptron and multilayer perceptron approaches.
- Some results using these algorithms will be shown at the end of the slides.

PERCEPTRONS

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

Simplified view of Fig 1 from Rosenblatt's paper.



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

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

PERCEPTRONS

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

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

PERCEPTRONS

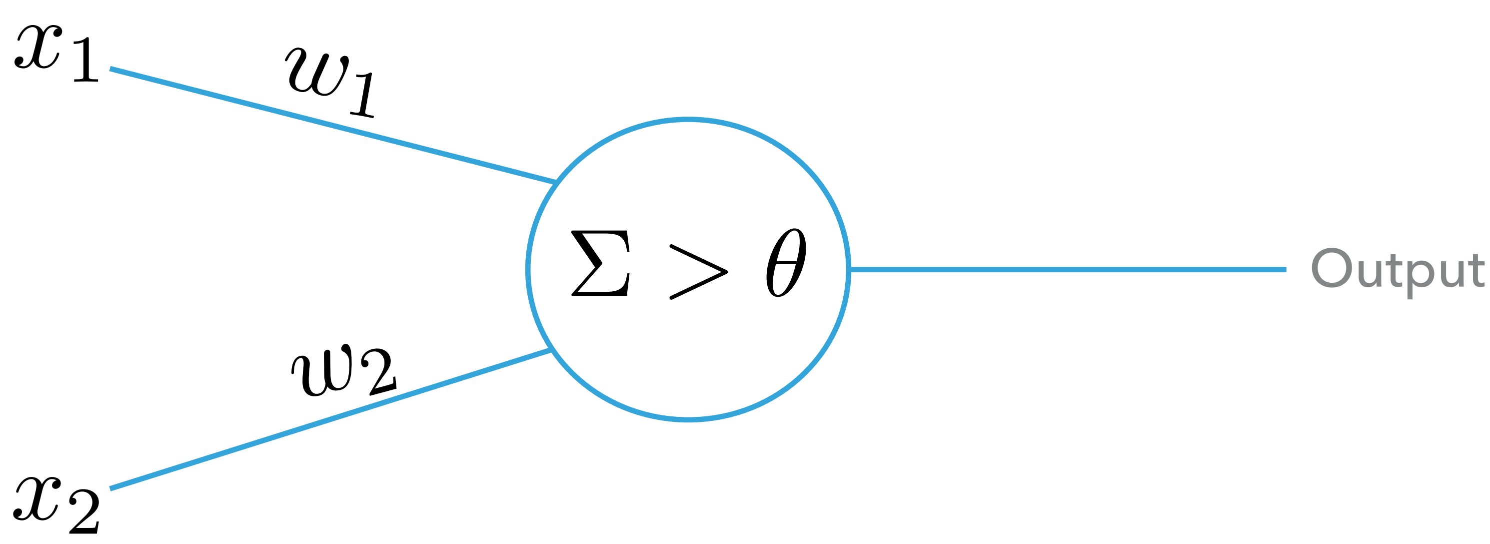
- Illustrative example:
  - Consider a measurement of two quantities x_1, and x_2.
  - Based on these measurements we determine if the perceptron is to give an output (value = 1) or not (value = 0).

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

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

PERCEPTRONS

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



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

PERCEPTRONS

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

If $w_1 x_1 + w_2 x_2 > \theta$

Output = 1

else

Output = 0

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

PERCEPTRONS

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

If $w_1x_1 + w_2x_2 > \theta$

Output = 1

else

Output = 0

This is called a binary activation function or perceptron.

It is another application of the Heavyside function *H(x)*.

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

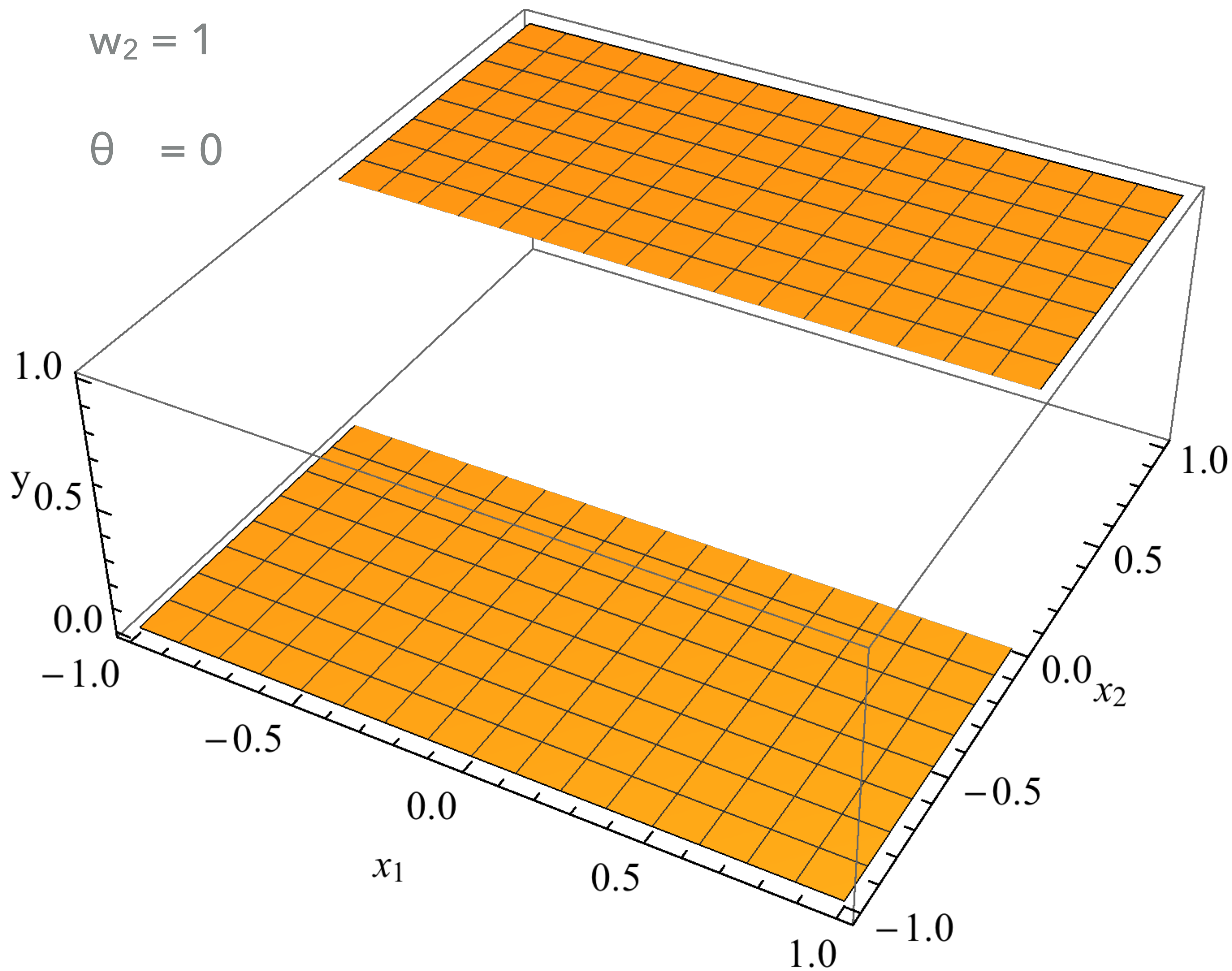
PERCEPTRONS

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

$w_1 = 0$

$w_2 = 1$

$\theta = 0$



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

PERCEPTRONS

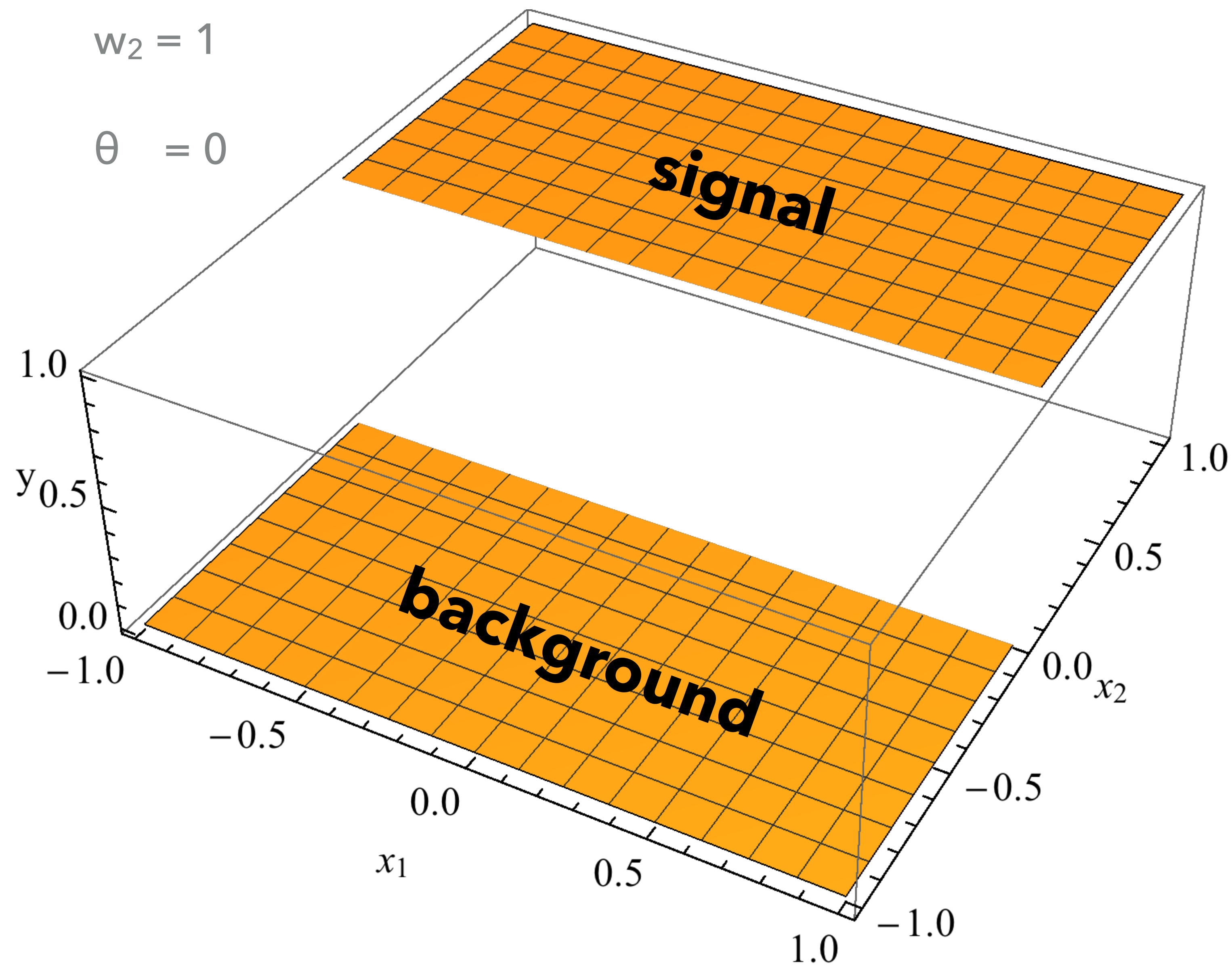
- Illustrative example:
- Decision is made on x_2
- Output value is either 1 or 0 as some $f(x_1, x_2)$ that depends on the values of w_1, w_2 and θ.

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

$w_1 = 0$

$w_2 = 1$

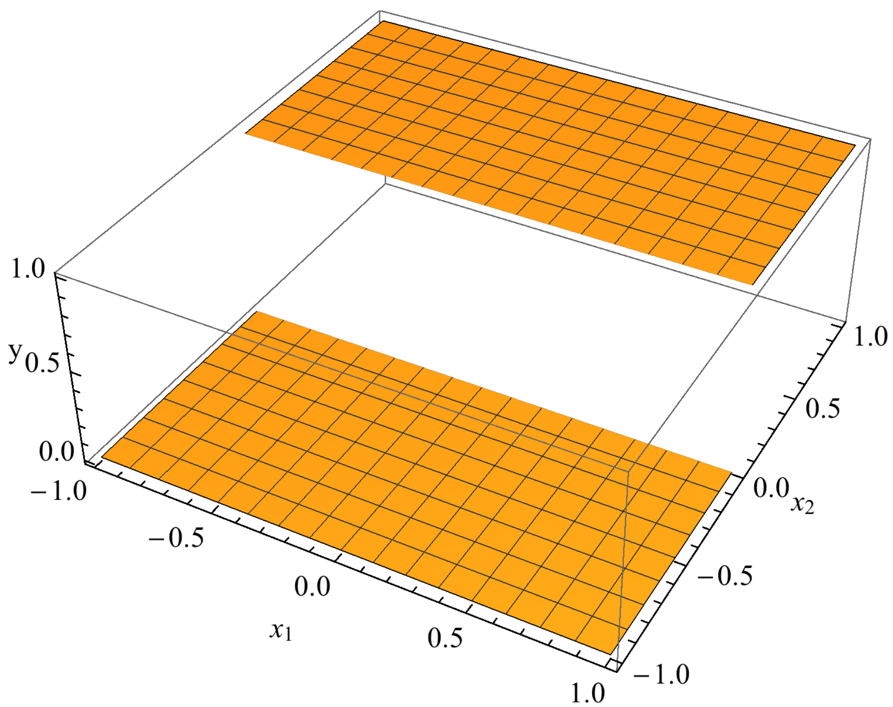
$\theta = 0$



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

PERCEPTRONS

- Illustrative examples:

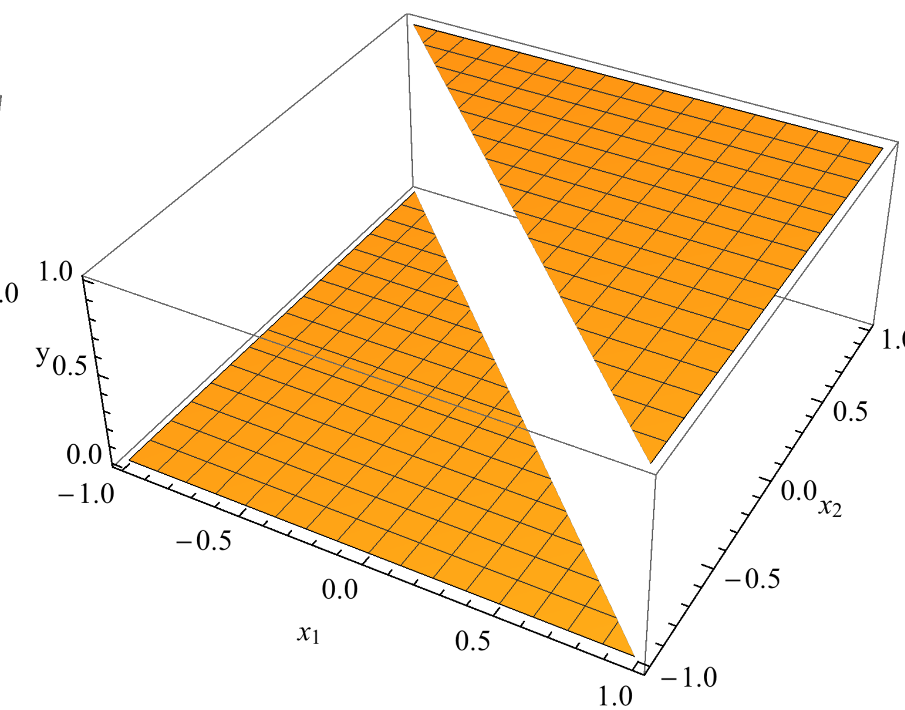


$w_1 = 0$

$w_2 = 1$

$\theta = 0$

Baseline for comparison, decision only on value of x_2

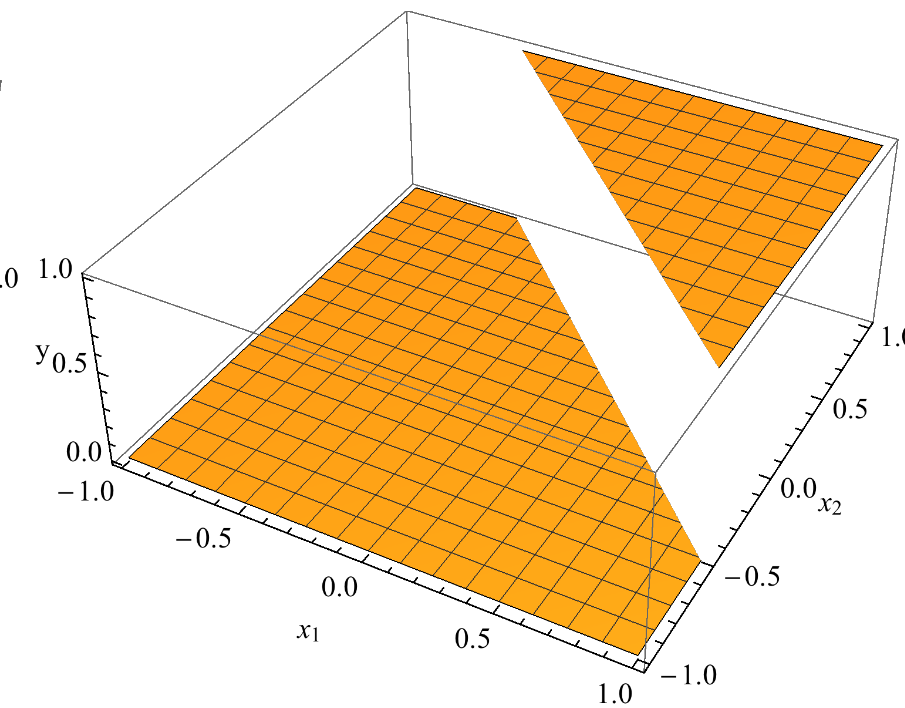


$w_1 = 1$

$w_2 = 1$

$\theta = 0$

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.

PERCEPTRONS

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

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

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

PERCEPTRONS

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

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

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

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

PERCEPTRONS

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

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

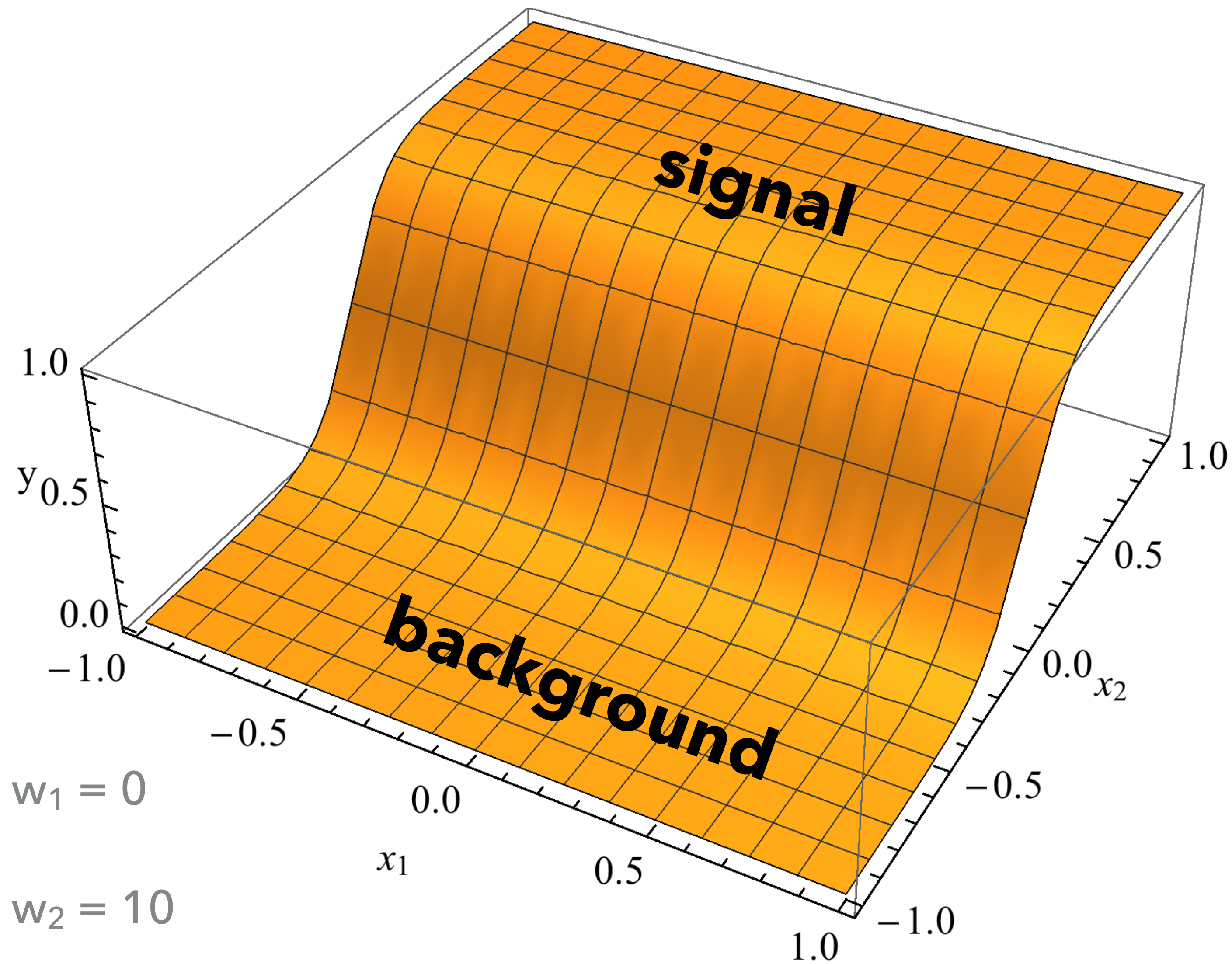
ACTIVATION FUNCTIONS

- The perceptron (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:

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



$w_1 = 0$

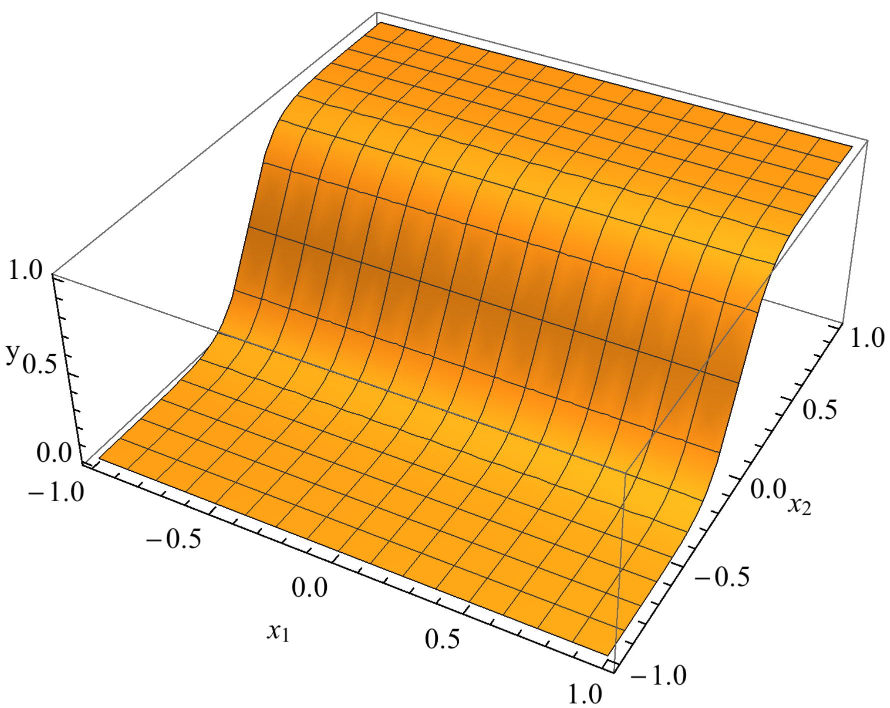
$w_2 = 10$

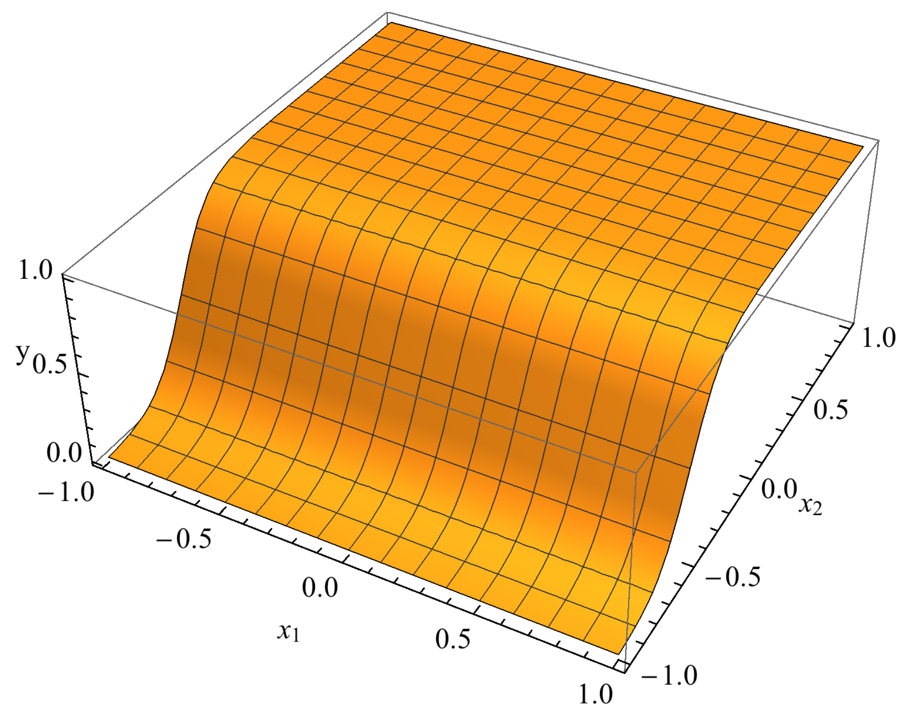
$\theta = 0$

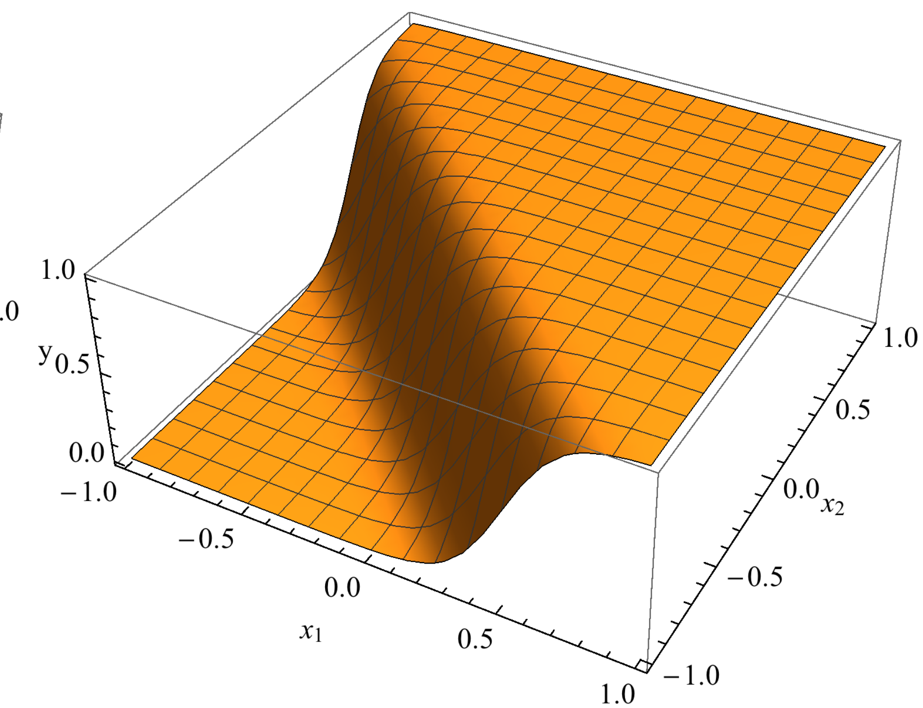
ACTIVATION FUNCTIONS: LOGISTIC (OR SIGMOID)

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

- A common activation function used in neural networks:







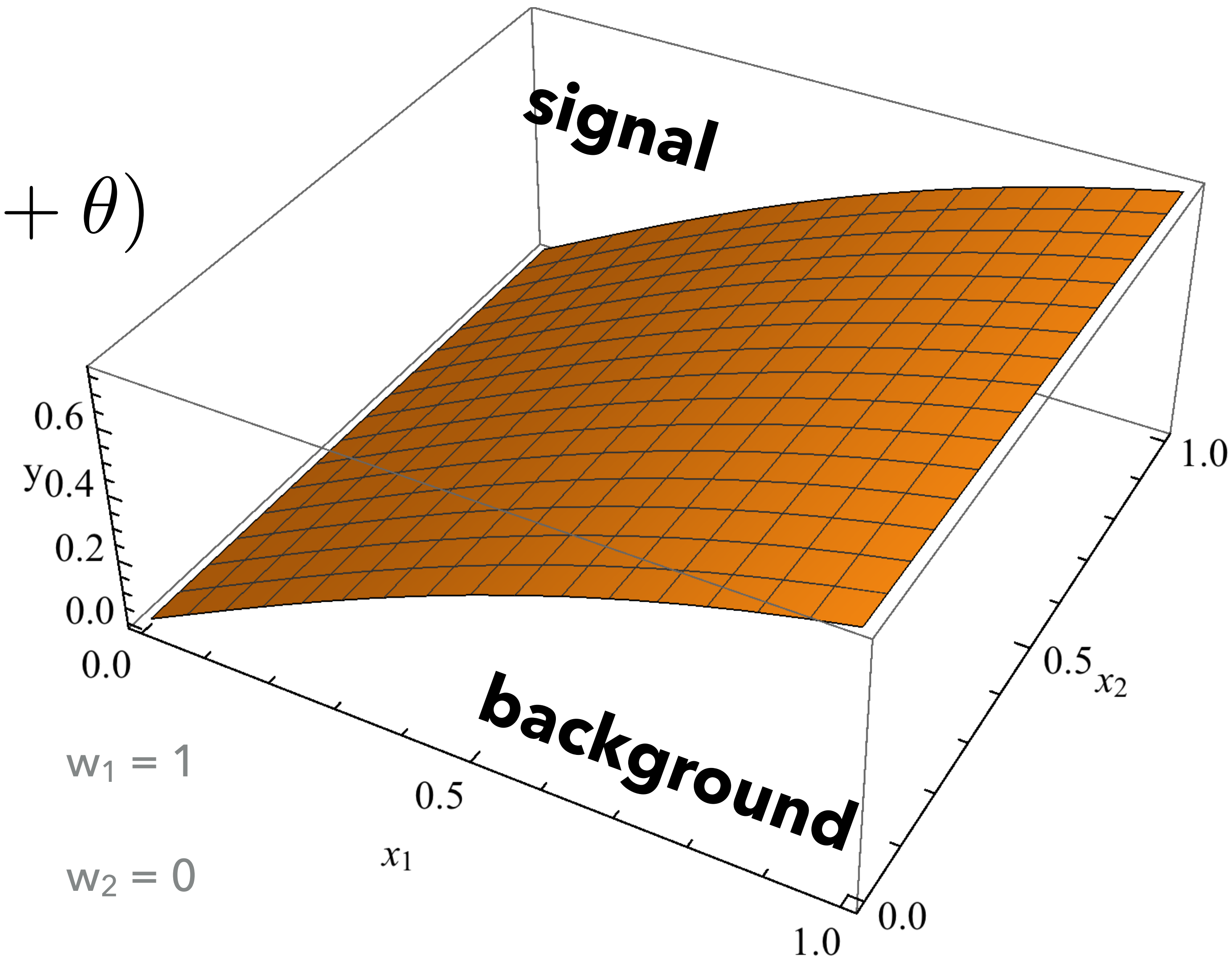
$w_1 = 0$	$w_1 = 0$	$w_1 = 10$
$w_2 = 10$	$w_2 = 10$	$w_2 = 10$
$\theta = 0$	$\theta = -5$	$\theta = -5$
Baseline for comparison, decision only on value of x_2	Offset from zero using θ	rotate "decision boundary" in (x_1, x_2)

ACTIVATION FUNCTIONS: HYPERBOLIC TANGENT

- A common activation function used in neural networks:

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

(Often used with θ = 0)



$w_1 = 1$

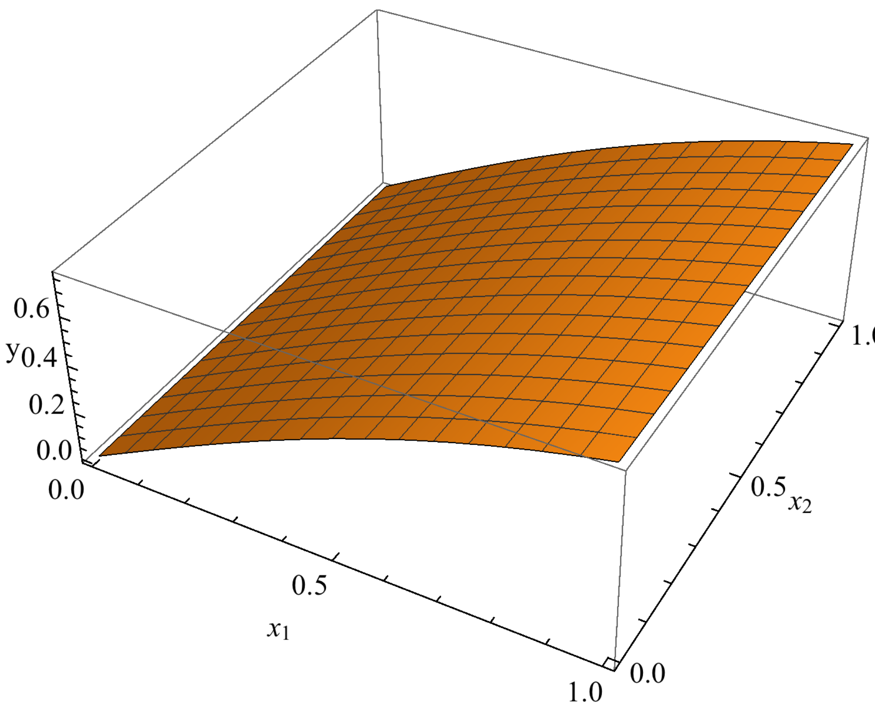
$w_2 = 0$

$\theta = 0$

ACTIVATION FUNCTIONS: HYPERBOLIC TANGENT

$$\tanh(w_1 x_1 + w_2 x_2 + \theta)$$

- A common activation function used in neural networks:

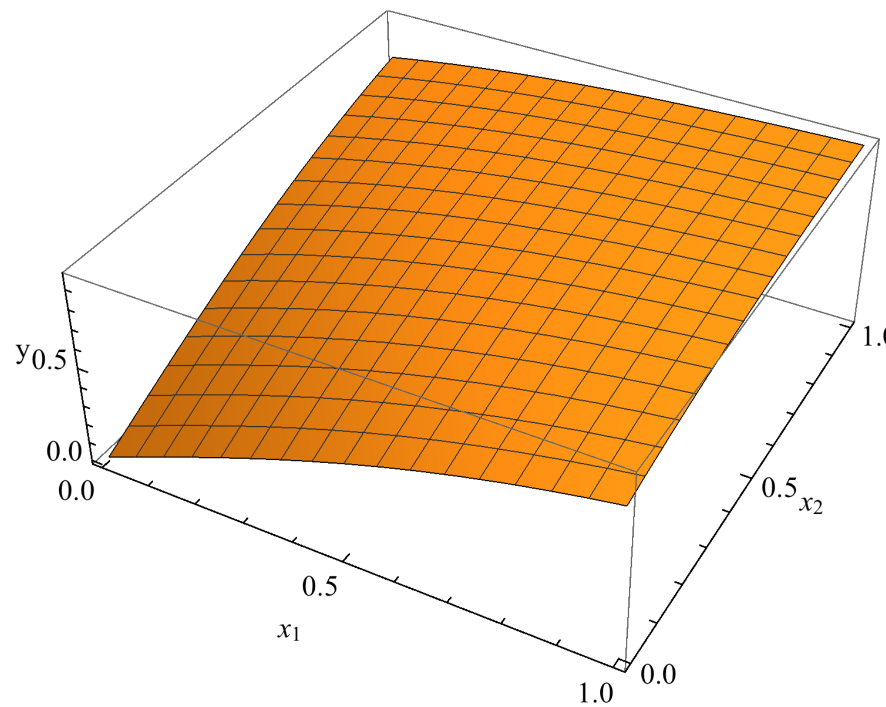


$w_1 = 1$

$w_2 = 0$

$\theta = 0$

Baseline for comparison, decision only on value of x_1

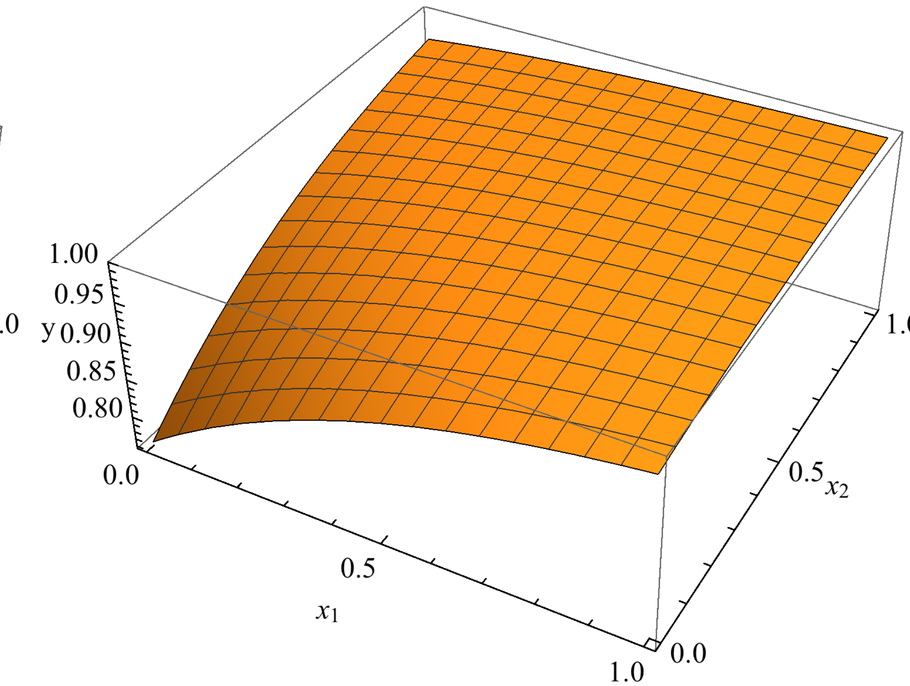


$w_1 = 1$

$w_2 = 1$

$\theta = 0$

rotate "decision boundary" in (x_1, x_2)



$w_1 = 1$

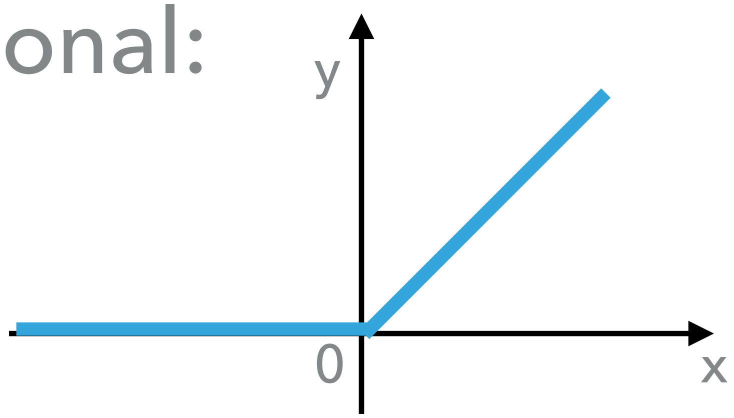
$w_2 = 1$

$\theta = -1$

Offset (vertically) from zero using θ

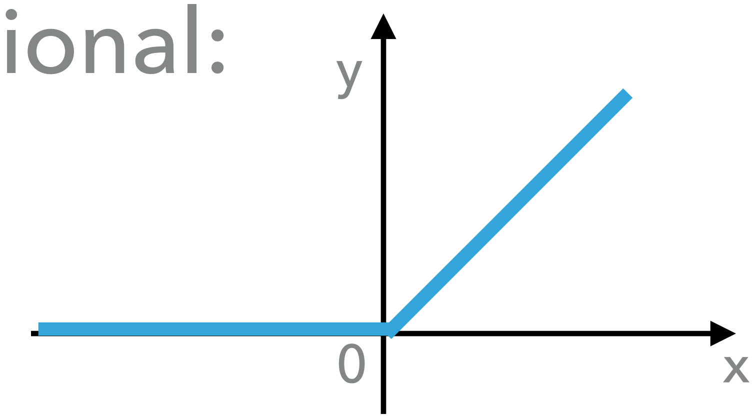
ACTIVATION FUNCTIONS: RELU

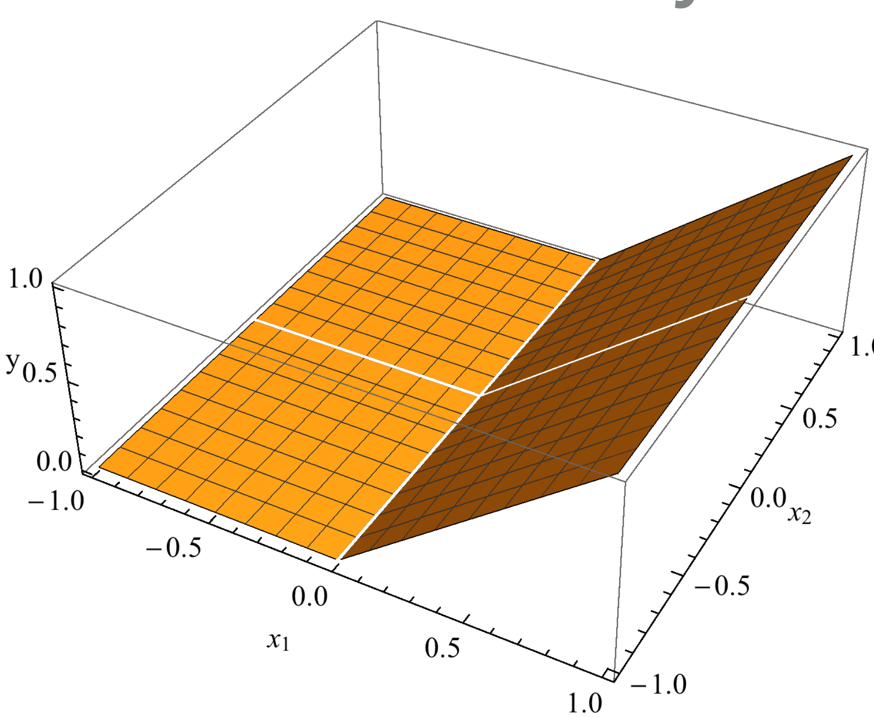
- The **Re**ctified **L**inear **U**nit activation function is commonly used for CNNs. This is given by a conditional:
  - If (x < 0) y = 0
  - otherwise y = x



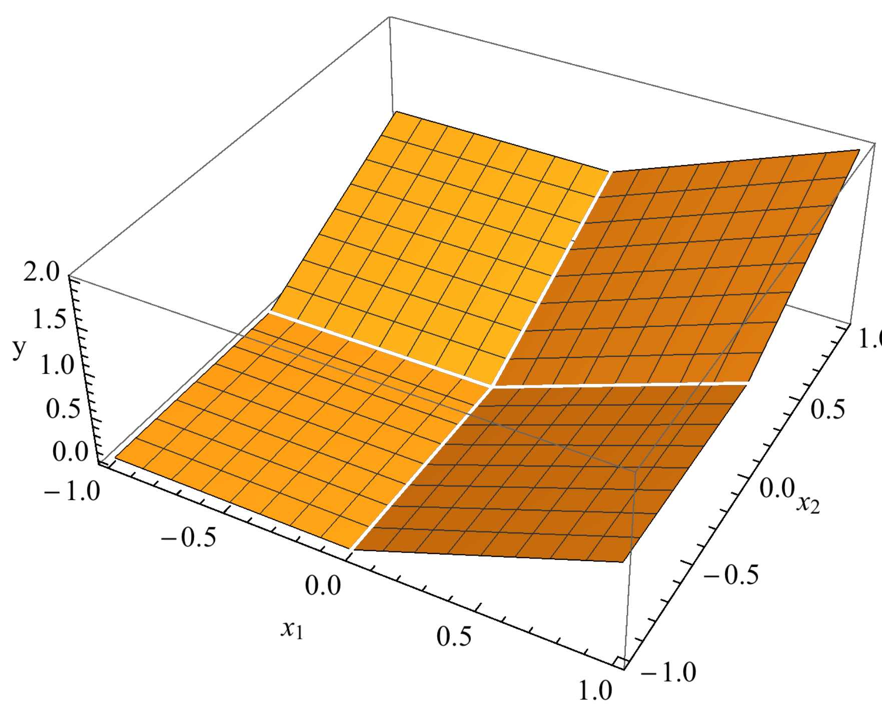
ACTIVATION FUNCTIONS: RELU

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

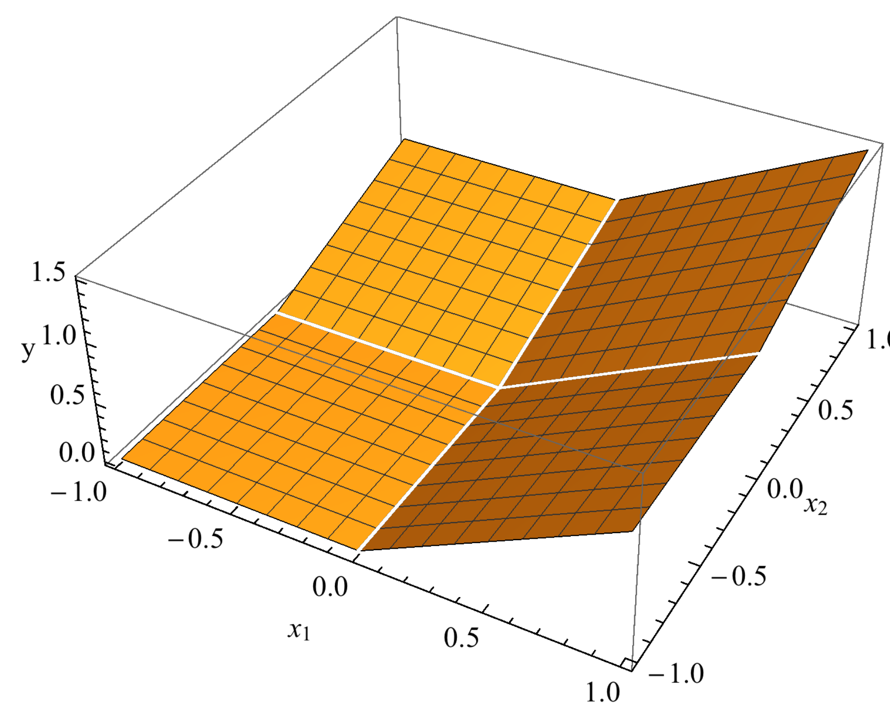




$w_1 = 1, w_2 = 0$



$w_1 = 1, w_2 = 1$

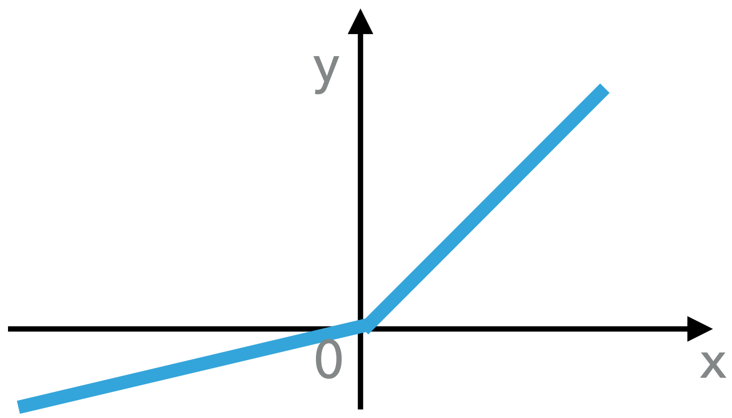


$w_1 = 1, w_2 = 0.5$

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

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 deep convolutional neural networks (**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].

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

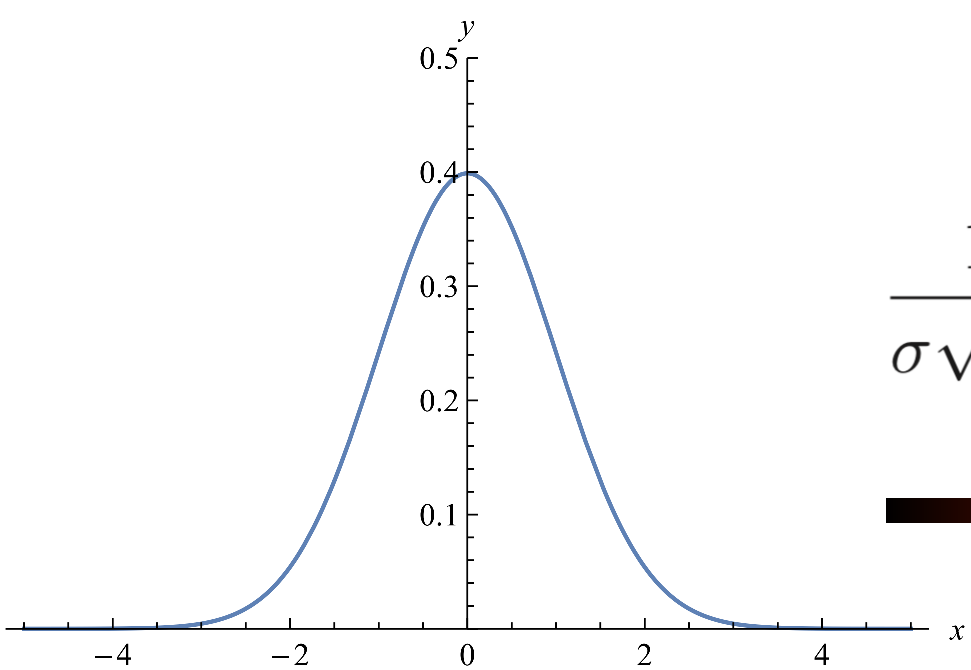
ACTIVATION FUNCTIONS: RELU

- 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].

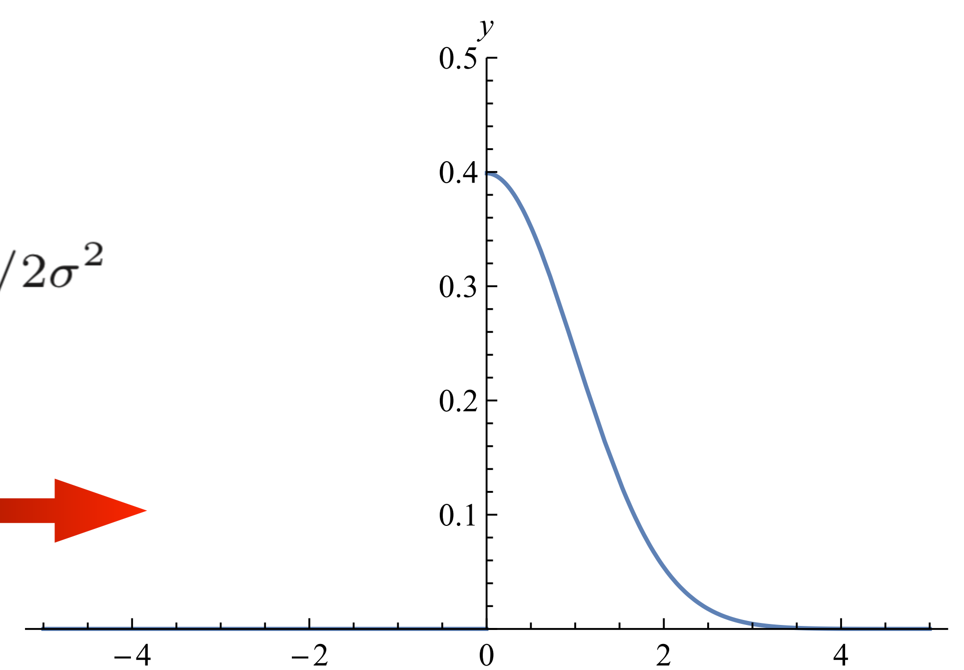
[1] Maas, Hannun, Ng, ICML2013.
[2] He, Zhang, Ren and Sun, arXiv:1502.01852

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∈[-∞, ∞].



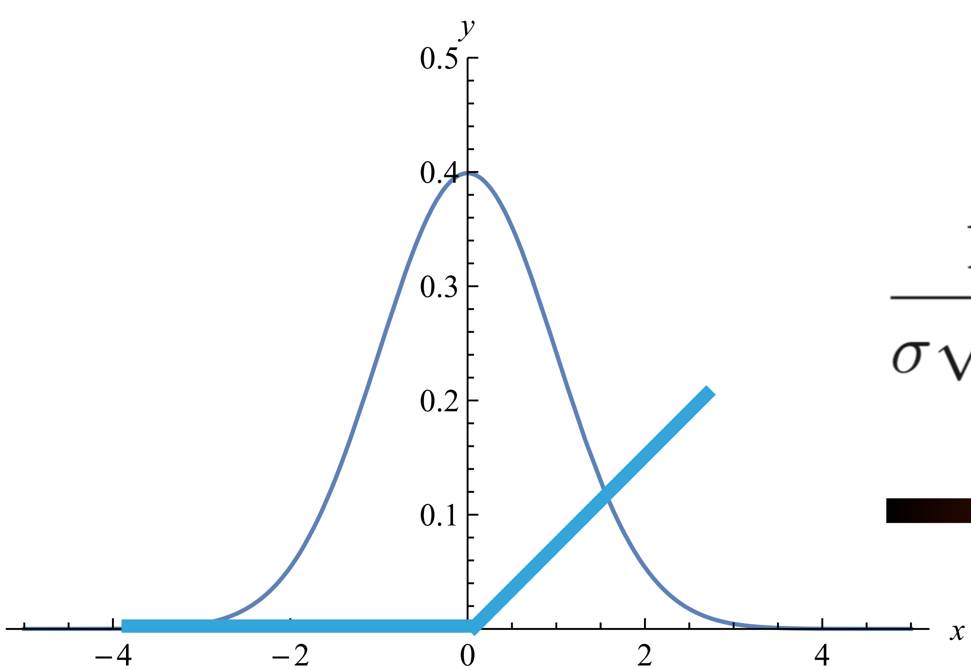
$$\frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/2\sigma^2}$$



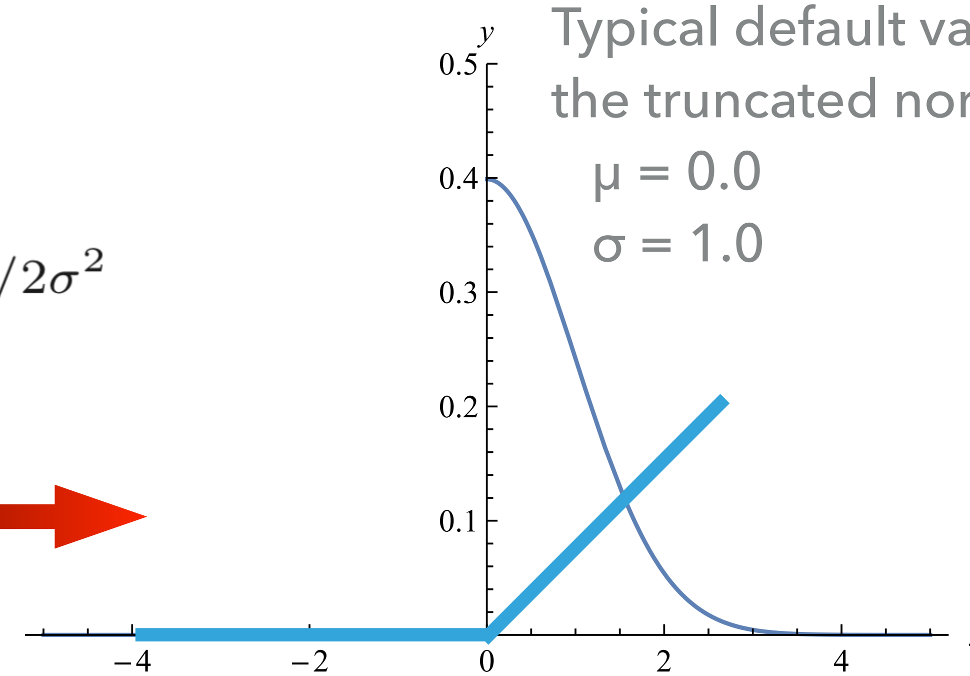
[1] Maas, Hannun, Ng, ICML2013.
[2] He, Zhang, Ren and Sun, arXiv:1502.01852

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∈[-∞, ∞].



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



Typical default values 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

ACTIVATION FUNCTIONS: DATA PREPARATION

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

ACTIVATION FUNCTIONS: DATA PREPARATION

- Input features are arbitrary; whereas activation functions have a standardised input domain of [-1, 1] or [0, 1].
  - We can map our input feature space onto a standardised domain that matches some range that matches that of the activation function.
  - Saves work for the optimiser in determining hyper-parameters.
  - Standardises weights to avoid numerical inaccuracies; and set common starting weights.
  - e.g.
    - having an energy or momentum measured in units of 10^{12} eV, would require weights $O(10^{-12})$ to obtain an $O(1)$ result for $w_i x_i$.
    - Mapping eV $\longmapsto$ TeV would translate 10^{12} eV $\longmapsto$ 1TeV, and allow for $O(1)$ weights leading to an $O(1)$ result for $w_i x_i$.
    - Comparing weights for features that are standardised allows the user to develop an intuition as to what the corresponding activation function will look like.

ACTIVATION FUNCTIONS: DATA PREPARATION

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

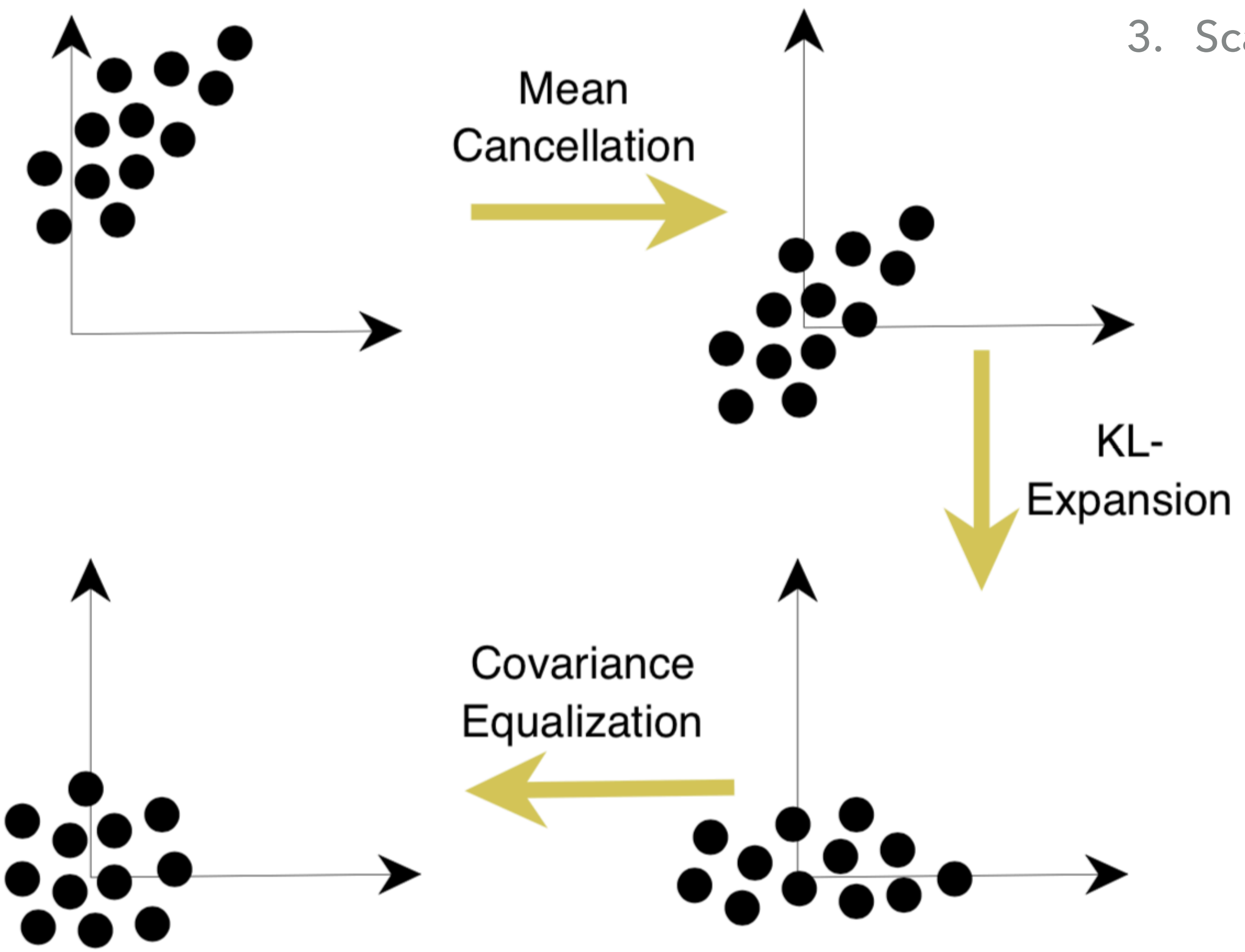
[1] De-correlation of features is not essential assuming a sufficiently general optimisation algorithm is being used. The rationale is that in general if one can de-correlate features then we just have to minimise the cost as a function of weights for one feature at a time, rather than being concerned about the dependence of weights on more than one feature. So this is a choice made to simplify the minimisation process, and in to speed up that process.

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

ACTIVATION FUNCTIONS: DATA PREPARATION

- e.g.

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



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

ACTIVATION FUNCTIONS: SUMMARY

- All of the activation functions can be described as some function:

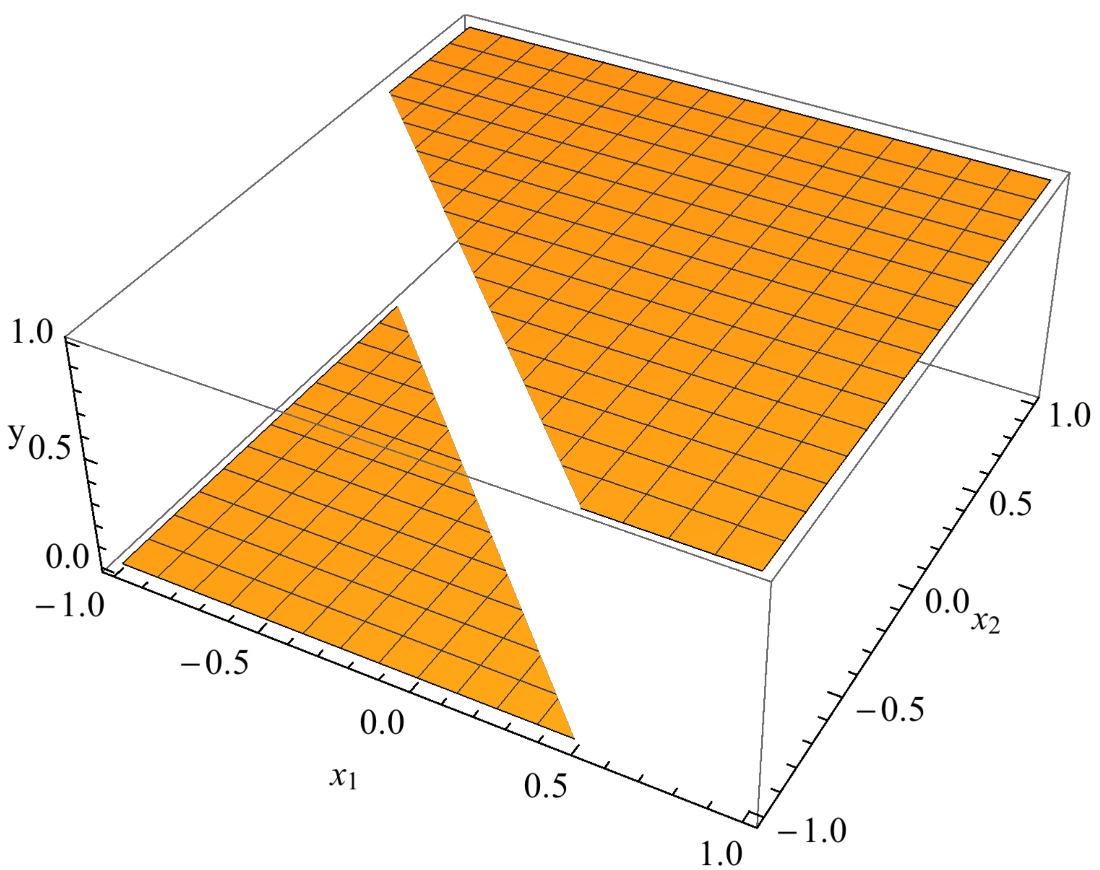
$$y = f(w^T x + b)$$

- e.g. for an N dimensional feature space the argument of the function is just

$$w^T x = (w_1, w_2, \ldots w_N) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

ARTIFICIAL NEURAL NETWORKS (ANNs)

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



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

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

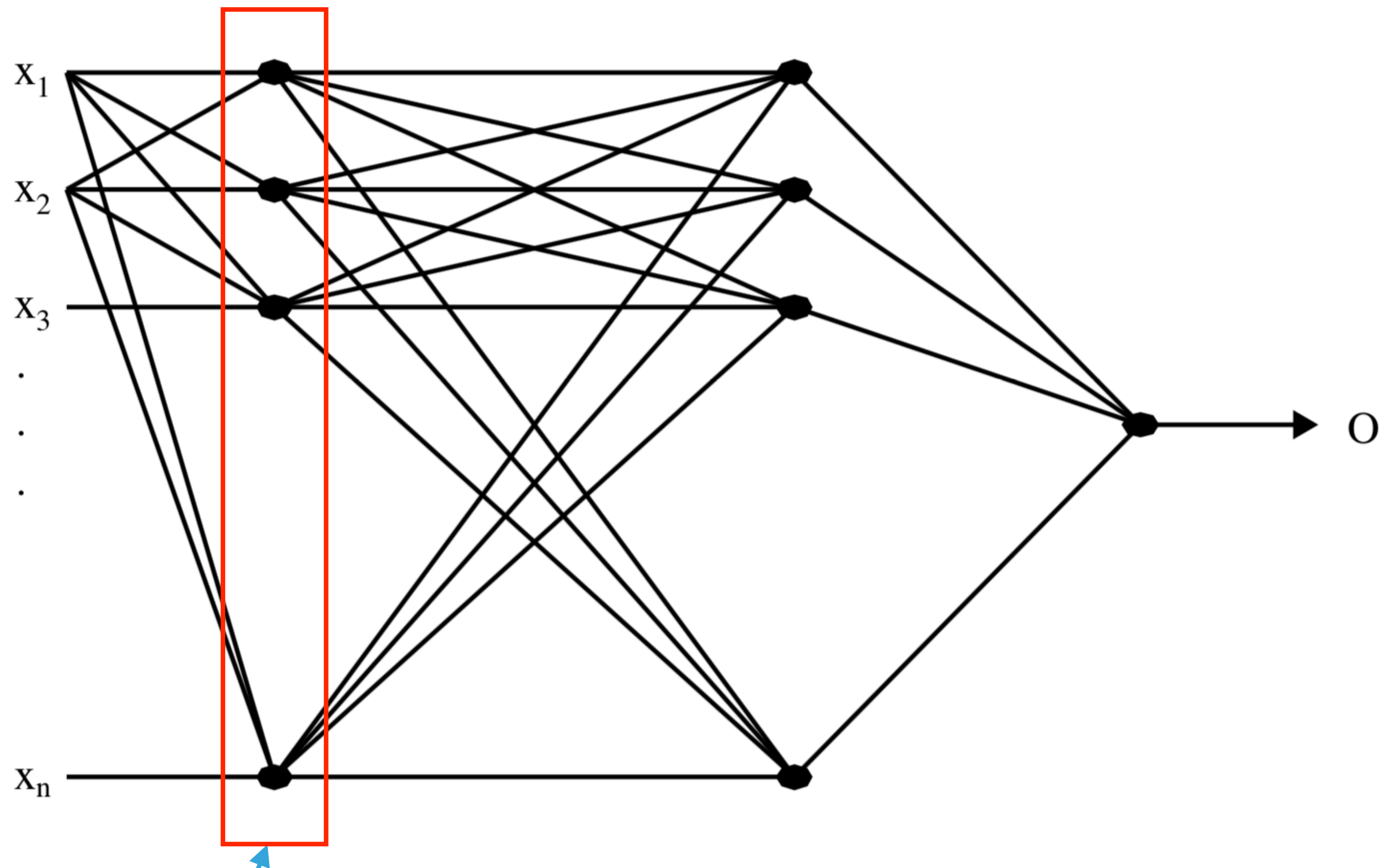
The only real difference is the heuristic used to determine the weights.

ARTIFICIAL NEURAL NETWORKS (ANNs)

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

MULTILAYER PERCEPTRONS

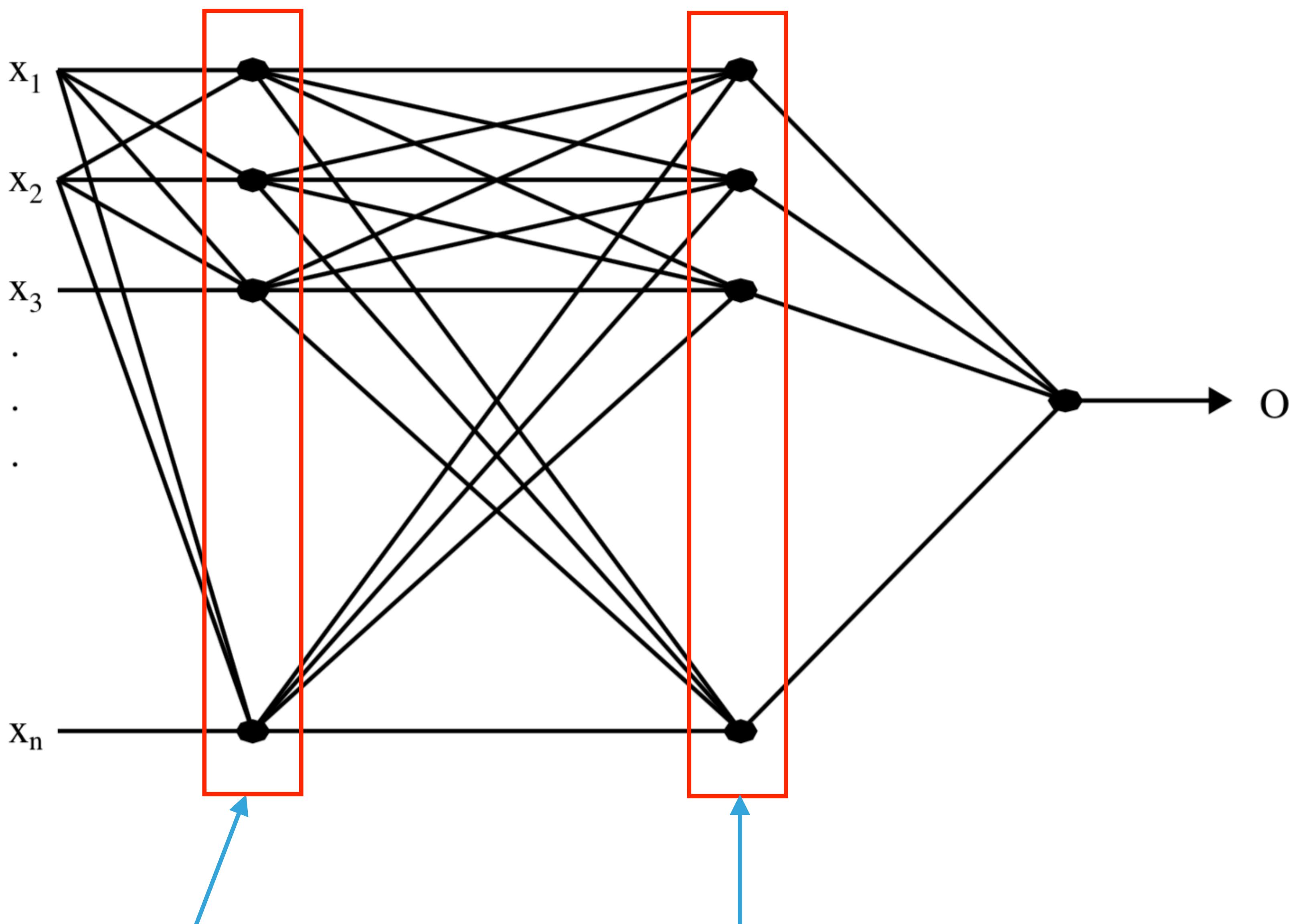
- Illustrative example: Input data example: $x = \{x_1, x_2, x_3, \ldots, x_n\}$



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

MULTILAYER PERCEPTRONS

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

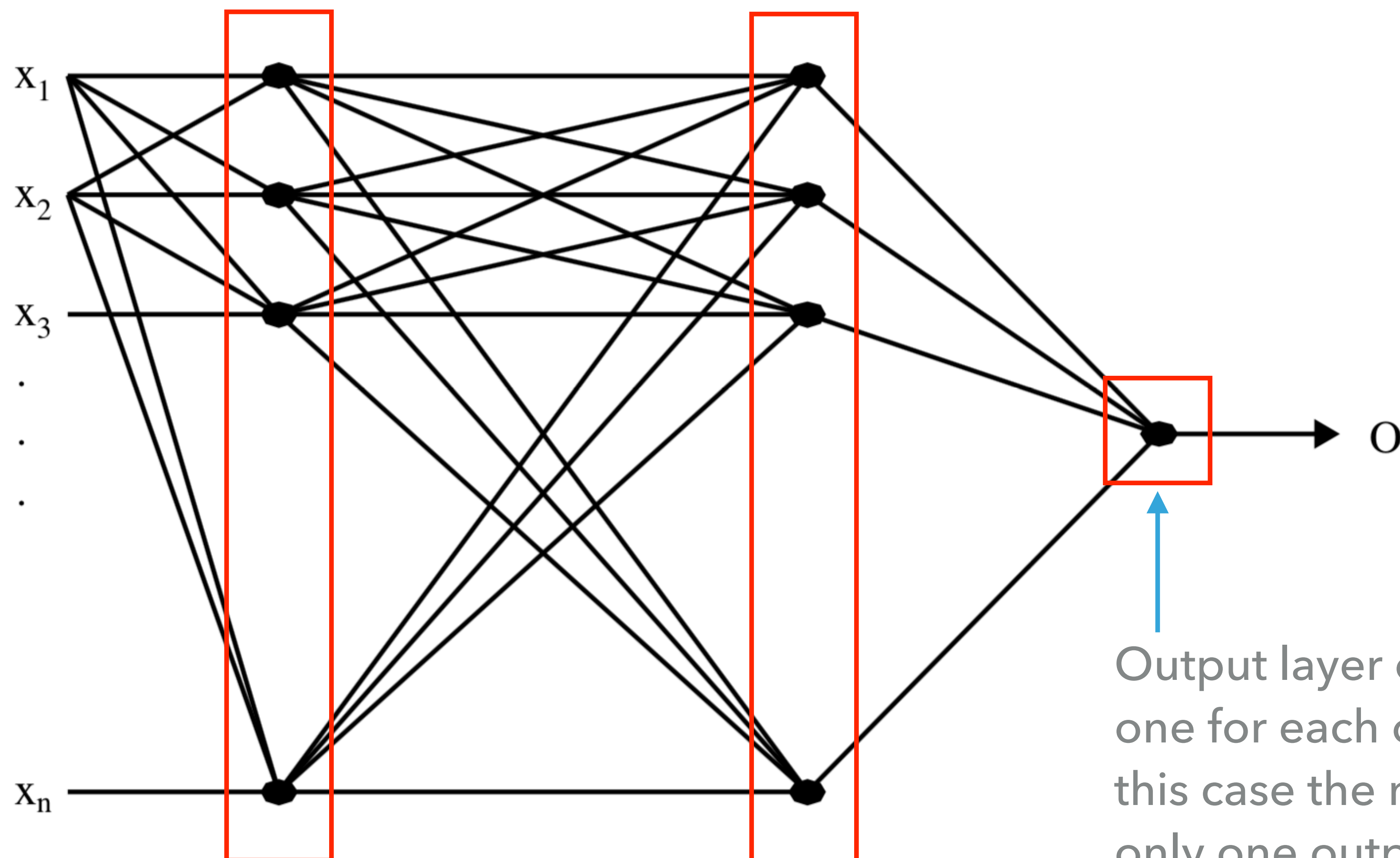


Input layer of n perceptrons; one for each dimension of the input feature space

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

MULTILAYER PERCEPTRONS

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



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

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

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

MULTILAYER PERCEPTRONS

- We can extend the computation of an activation function based on

$$y = f(w^T x + b)$$

- to that of a layer in an MLP. The same equation applies in matrix form (assuming the same activation function is used for each node in the layer).

$$w^T x = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1N} \\ w_{21} & w_{22} & \dots & w_{2N} \\ \vdots & & & \\ w_{N1} & w_{N2} & \dots & w_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

- The result of $w^T x$ is a column matrix, one element for each node in the layer of the MLP. The bias and y are also a column matrices of the same form.

- y becomes the input feature space for the next layer (our output node) in the MLP

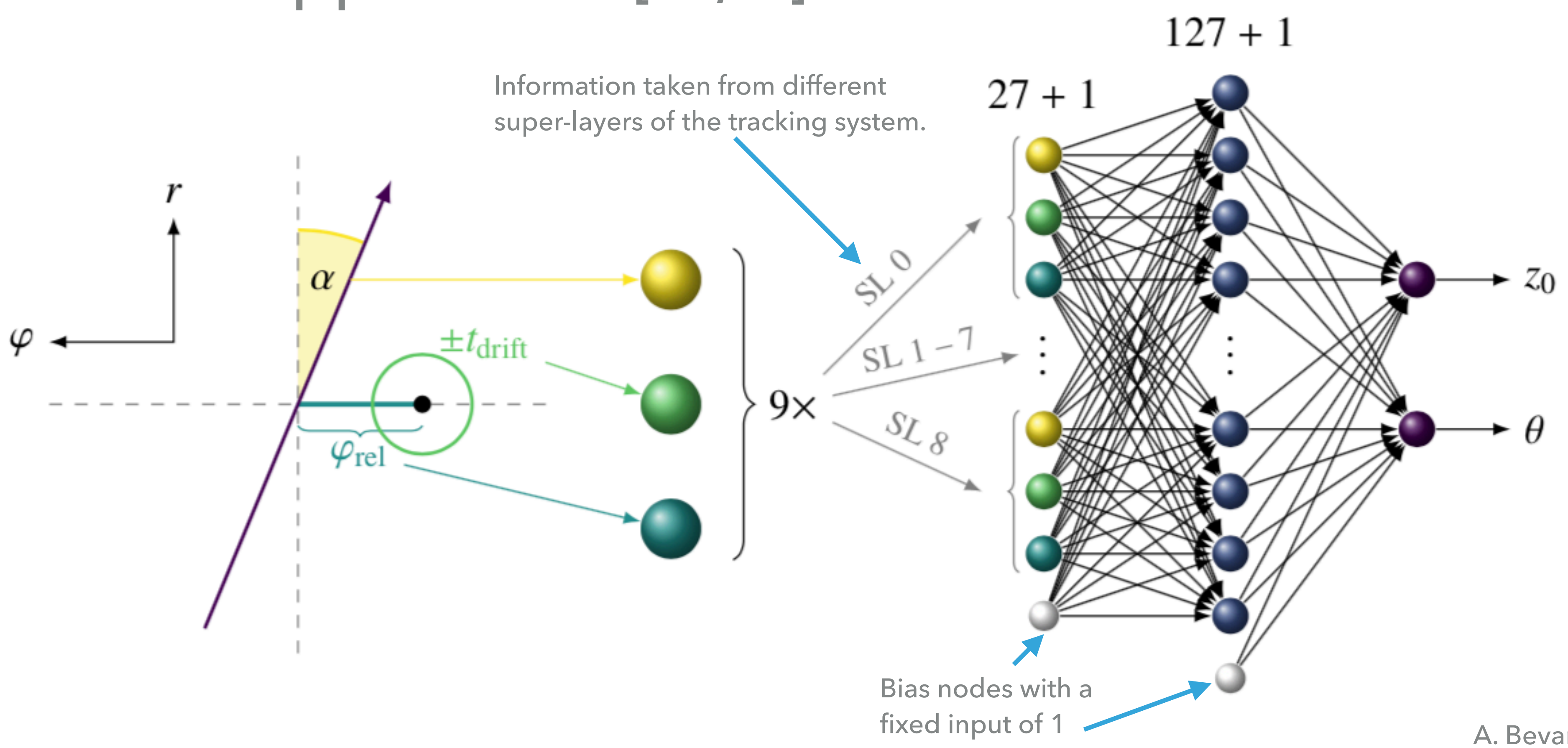
TRAINING

- Parameter tuning is referred to as training or fitting (usually training in the context of NNs).
  - A perceptron of the form $f(w^T x+\theta)$ has $n+1=\dim(x)+1$ hyper-parameters to be tuned.
  - The input layer of perceptrons in an MLP has $n(n+1)$ hyper parameters to be tuned.
  - … and so on.
- We tune parameters based on some metric[1] called the cost or loss function.
- We optimise the hyper-parameters of a network in order to minimise the loss function for an ensemble of data.
- For now we gloss over the details and assume there is an algorithm that takes care of parameter optimisation for us, given some initial guess for the weights.

[1] Also called a figure of merit in the previous lectures.

EXAMPLES: TRIGGER SELECTION FOR BELLE II

- Use neural networks as part of the track trigger.
- 27 dimensional input feature space is fed into a network with 127 nodes using data from the central drift chamber (CDC) of that experiment.
  - Inputs are mapped onto [-1, 1] to use a tanh activation function.



Neuhaus et al., http://inspirehep.net/record/1616030

EXAMPLES: TRIGGER SELECTION FOR BELLE II

- This network is used to approximate the functions of z_0 and θ, the position of the vertex in z and the polar angle, respectively.
- Efficiency computed on Monte Carlo simulated examples for different channels denoted below.

trigger condition		efficiency		
# tracks	# IP tracks	$B\overline{B}$	$\tau^+\tau^-$	$\tau \to \mu\gamma$
≥ 3	≥ 0	93.9 %	18.7 %	12.4 %
2	≥ 0	4.6 %	38.7 %	44.4 %
2	≥ 1	4.4 %	38.1 %	44.1 %
2	2	2.9 %	33.5 %	40.0 %
3 or 2 with ≥ 1 IP		98.3 %	56.8 %	56.5 %

Neuhaus et al., http://inspirehep.net/record/1616030

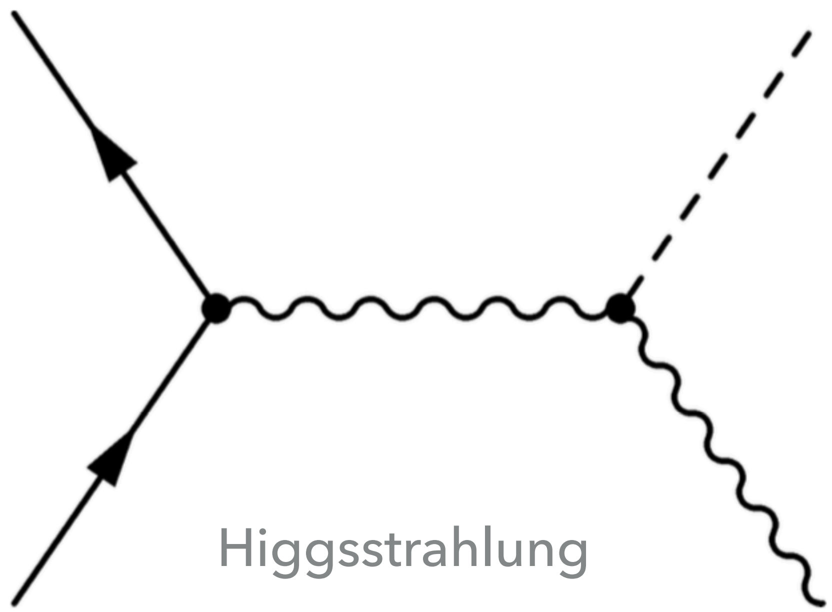
EXAMPLES: HIGGS SEARCH AT ALEPH

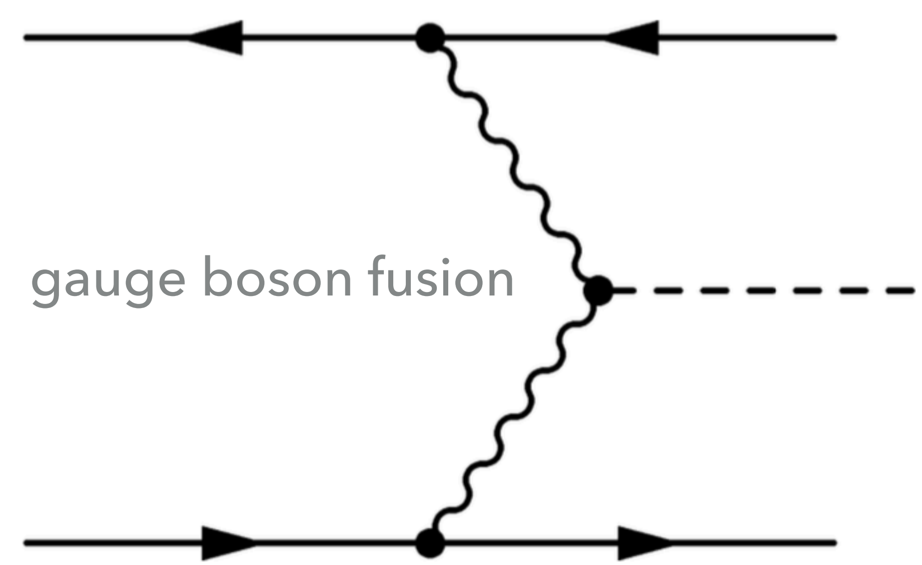
- This LEP experiment presented cut-based and NN-based searches for the Higgs.
- These two approaches had different sensitivities, and the neural network significance presented a hint for a Higgs boson.
  - Single network with three output classes: one for the signal, one for $q\bar{q}$ background and one for W^+W^- background.
- The hint was at 114 GeV/c^2; 11 GeV/c^2 below the discovery point; so ultimately this was a statistical fluctuation.

ALEPH Collaboration, Phys.Lett.B495:1–17,2000

EXAMPLES: HIGGS SEARCH AT ALEPH

- Search for $e^+e^- \to ZH$ produced via





- where the H predominantly decays into two b quarks or two τ leptons, and the associated Z boson decays either into neutrino, quark or lepton pairs.

Footnote: The H→bb decay was discovered by ATLAS and CMS in the summer of 2018 using the this channel (along with several others). Please see the CERN press release for details.

ALEPH Collaboration, Phys.Lett.B495:1-17,2000

EXAMPLES: HIGGS SEARCH AT ALEPH

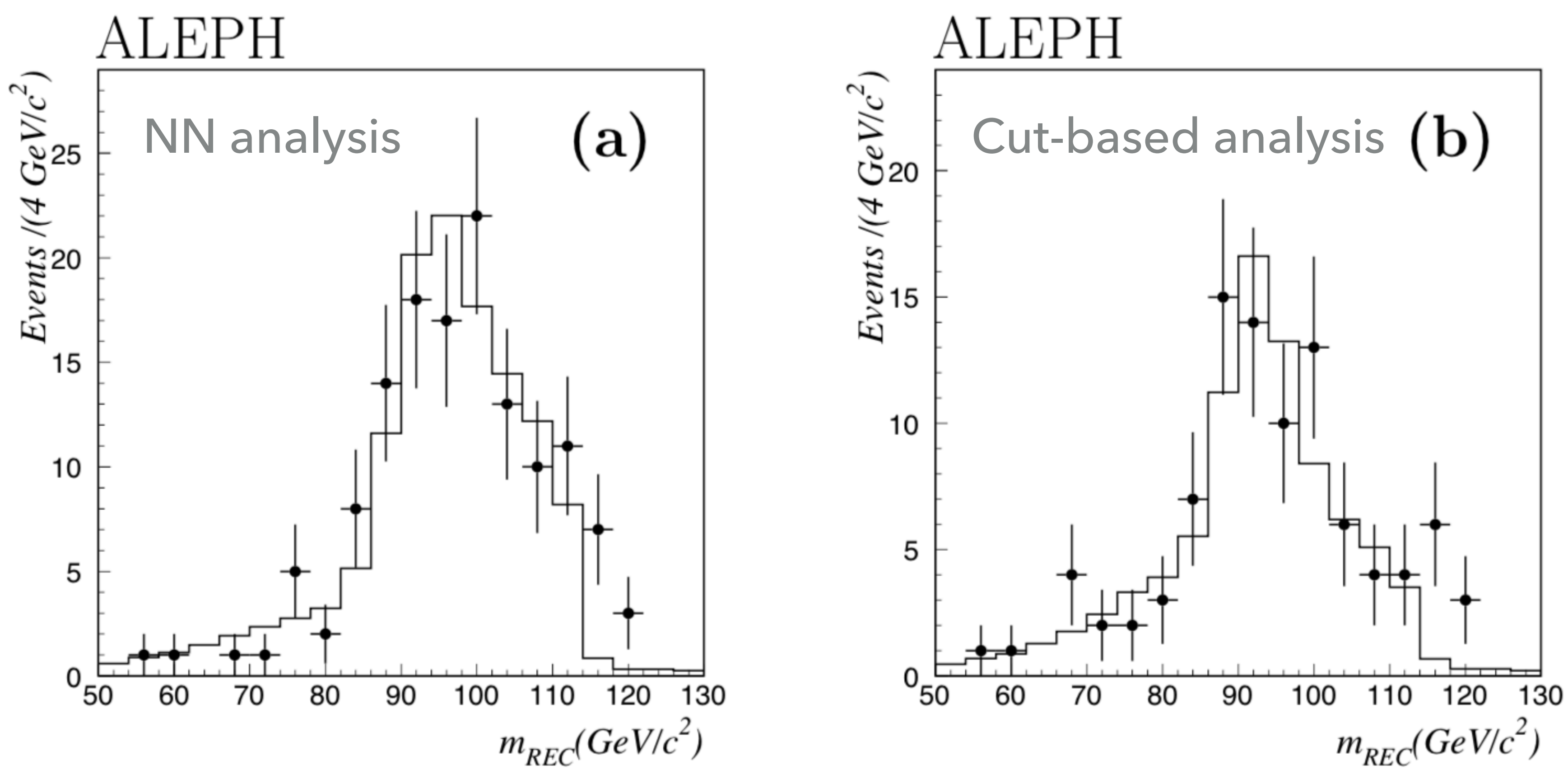
Analysis	Signal Events Expected	Background Events Expected				Events Obs.	Expected Significance (σ)
		ZZ	WW	$f\bar{f}$	Total		
$Hq\bar{q}$ (NN)	4.5	23.0±1.0	8.6±0.6	15.3±1.7	46.9±2.1	52	1.6
$Hq\bar{q}$ (Cut)	2.9	12.6±0.7	3.2±0.2	7.9±0.7	23.7±1.0	31	1.3
$H\nu\bar{\nu}$ (NN)	1.4	13.5±0.7	22.0±1.1	2.0±0.4	37.5±1.4	38	0.8
$H\nu\bar{\nu}$ (Cut)	1.3	9.9±1.1	8.8±1.7	1.0±0.3	19.7±2.0	20	0.7
$H\ell^+\ell^-$	0.7	26.4±0.3	2.4±0.1	1.8±0.3	30.6±0.4	29	0.8
$\tau^+\tau^- q\bar{q}$	0.4	6.4±0.3	6.2±0.3	1.0±0.3	13.6±0.5	15	0.4
NN Total	7.0	69.3±1.3	39.2±1.3	20.1±1.8	128.7±2.6	134	2.1
Cut Total	5.3	55.3±1.4	20.6±1.7	11.7±0.9	87.6±2.4	95	1.8

- The pattern of the NN based analysis consistently out performing the cut-based analysis is to be expected and it is one of the reasons why cut based analyses are being used less often than they used to:
  - In general MVA-based analyses outperform rectangular cut based analyses. If you find that they do not do this for your particular analysis, then you have a problem with your model.

ALEPH Collaboration, Phys.Lett.B495:1-17,2000

EXAMPLES: HIGGS SEARCH AT ALEPH

- The enhanced sensitivity is reflected in the event yields observed in the data:



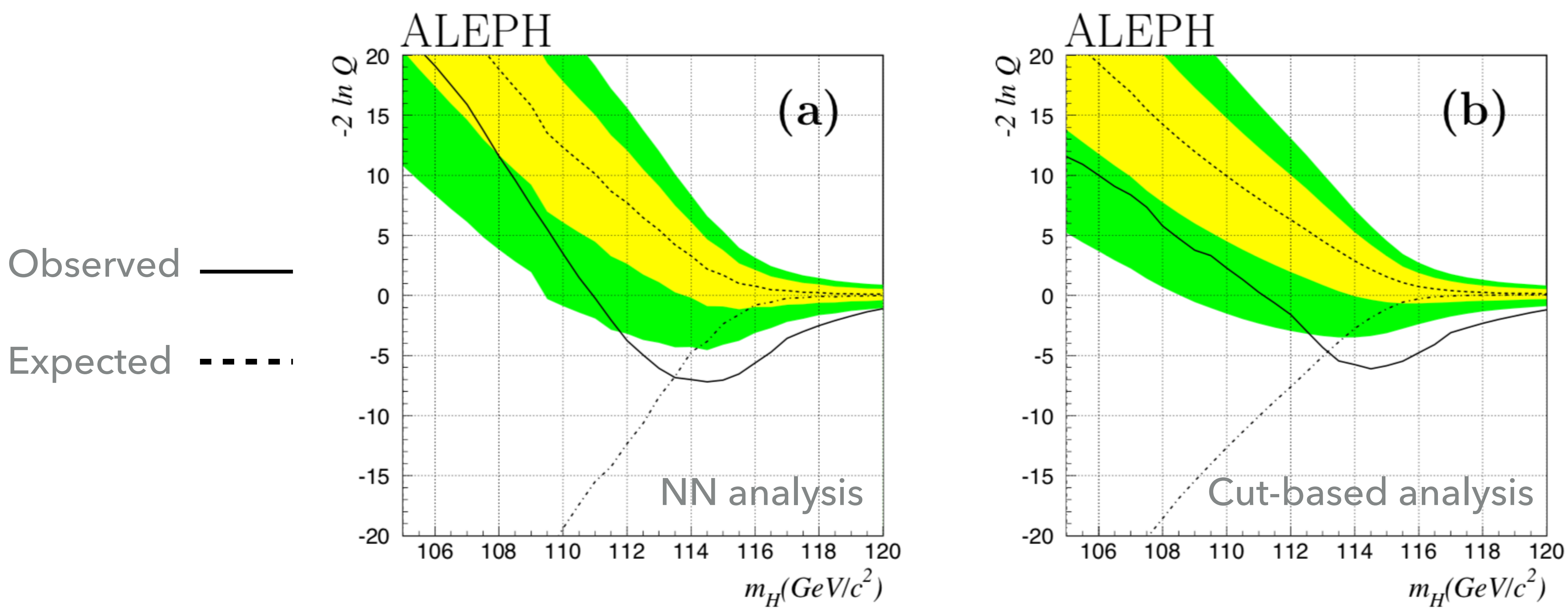
- NN based analysis has 134 events vs 95 for the cut based approach.

ALEPH Collaboration, Phys.Lett.B495:1-17,2000

EXAMPLES: HIGGS SEARCH AT ALEPH

- The presence of a Higgs boson is determined using a likelihood ratio between signal+background and background only hypotheses:

$$Q = \frac{L_{s+b}}{L_b} = \frac{e^{-(s+b)}}{e^{-b}} \prod_{i=1}^{n_{obs}} \frac{s f_s(\vec{X}_i) + b f_b(\vec{X}_i)}{b f_b(\vec{X}_i)}$$



ALEPH Collaboration, Phys.Lett.B495:1-17,2000

SUMMARY

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

SUGGESTED TOOLS

- Various toolkits exist for neural networks. These can be found in commercial packages such as Mathematica and Matlab, in open source toolkits such as CNTK, TensorFlow, Keras, Caffe etc.
- TMVA in ROOT:
  - HEP community developed toolkit is one way to start using neural networks.
  - Several different neural network's available:
    - Clermont-Ferrand neural net (used for Higgs searches on ALEPH and B tagging on BaBar).
    - TMultiLayerPerceptron
    - MLP
    - Bayesian MLP
    - Deep Networks

* see Sahin et al in the references.

SUGGESTED READING (HEP RELATED)

- Numerous examples can be found in the literature of applying neural networks to data.
- Books:
  - I. Narsky and F. Porter, "Statistical Analysis Techniques in Particle Physics: Fits, Density Estimation and Supervised Learning", Wiley (2013).
  - Neural networks are also discussed in the multivariate techniques chapter of A. Bevan et al., "The Physics of the B Factories".

SUGGESTED READING (NON-HEP)

- The suggestions made here are for some of the standard text books on the subject. These require a higher level of math than we use in this course, but may have less emphasis on the practical application of the methods we discuss here as a consequence. Many other books have been written on this subject.
  - MacKay: *Information theory, inference and learning algorithms*
  - C. Bishop: *Neural Networks for Pattern Recognition*
  - C. Bishop: *Pattern Recognition and Machine Learning*
  - T. Hastie, R. Tibshirani, J. Friedman, *Elements of statistical learning*

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

APPENDIX

- We have discussed only 2 output classification types in these slides. The following slides illustrate how to extend models to multiple classification output types.

MULTICLASS CLASSIFICATION

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

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

The null hypothesis provides a specific response/expectation.

The alternative hypothesis is the complement of the null.

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

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

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

MULTICLASS CLASSIFICATION

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

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

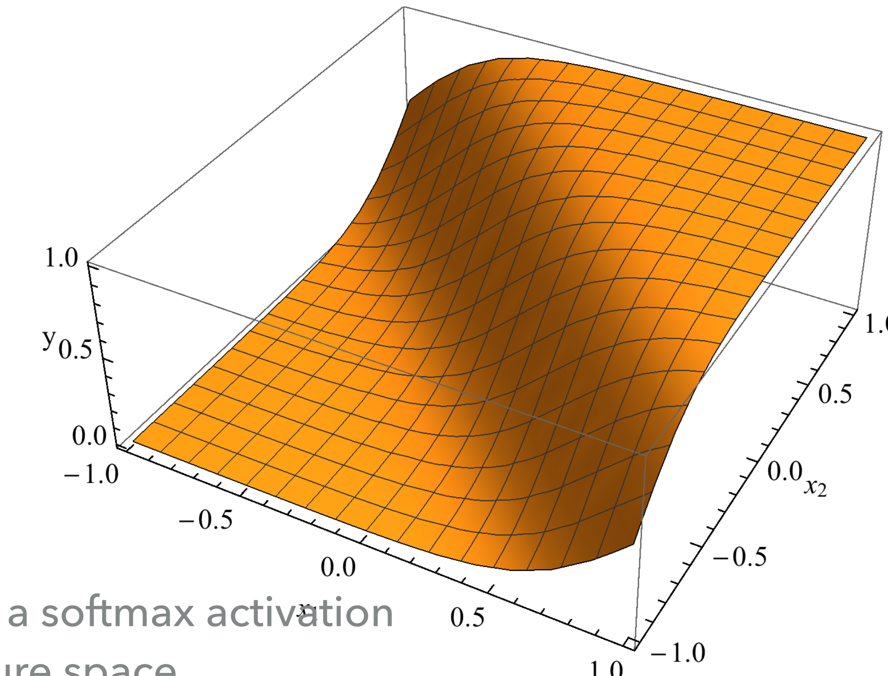
$$f(x) = \frac{1}{\sum_{i=1}^{N} e^{w_i^T x}} \begin{bmatrix} e^{w_1^T x} \\ e^{w_2^T x} \\ \vdots \\ e^{w_N^T x} \end{bmatrix}$$

i is the index for the output classification type

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

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

- Can convert output to {0, 1}.



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

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

MULTICLASS CLASSIFICATION

- Loss functions (see the next lecture) need to be modified to accommodate multiple output classification types.
  - e.g. softmax-cross entropy