ADRIAN BEVAN

INTRODUCTION TO MACHINE LEARNING
PART 2

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

ARTIFICIAL NEURAL NETWORKS
MULTILAYER PERCEPTRONS
AUTO-ENCODERS
CONVOLUTIONAL NEURAL NETWORKS
GENERATIVE ADVERSARIAL NETWORKS

NEURAL NETWORKS

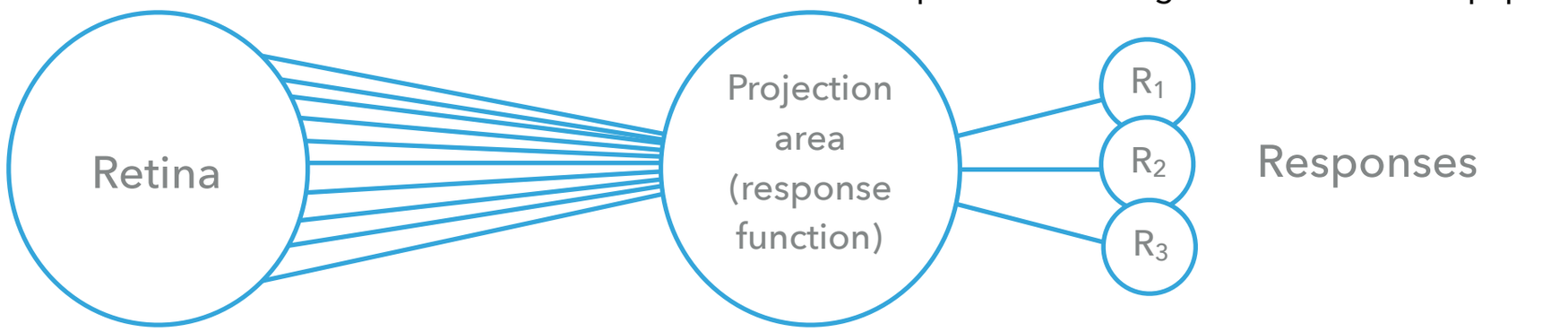
ARTIFICIAL NEURAL NETWORKS

institute of CODING
A. Bevan Queen Mary University of London

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.

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.

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.

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

ROSENBLATT'S PERCEPTRON

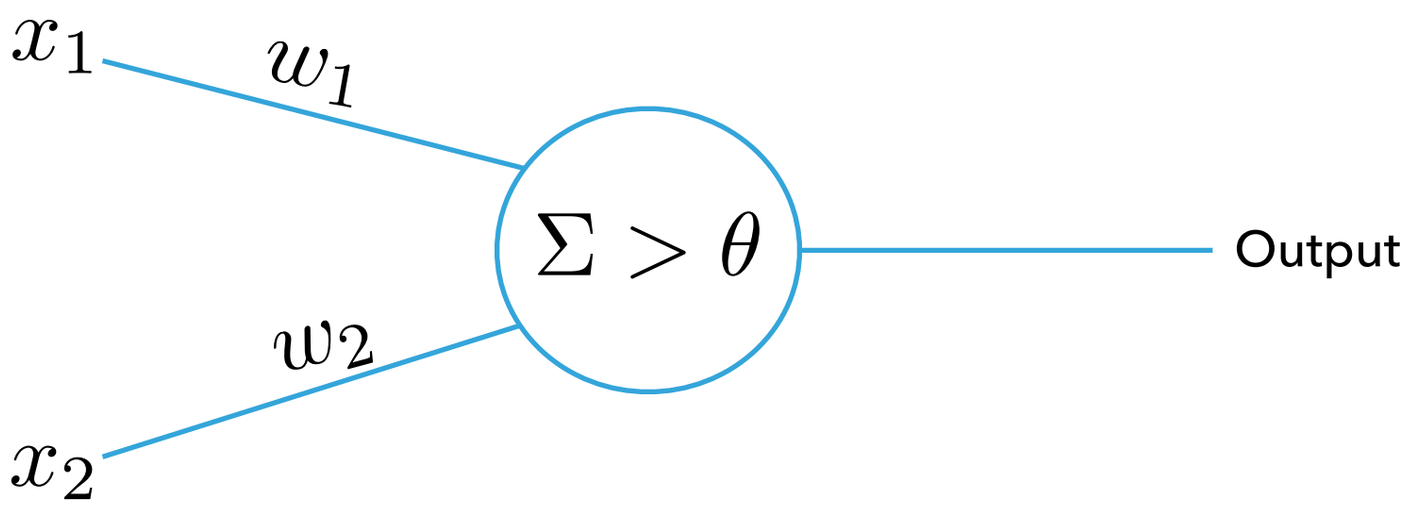
- Illustrative example:
  - Consider a measurement of two quantities x_1, and x_2.
  - Based on these measurements we determine if the perceptron is to give an output (value = 1) or not (value = 0).

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

[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).



[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).

If $w_1 x_1 + w_2 x_2 > \theta$

Output = 1

else

Output = 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).

If $w_1 x_1 + w_2 x_2 > \theta$

Output = 1

else

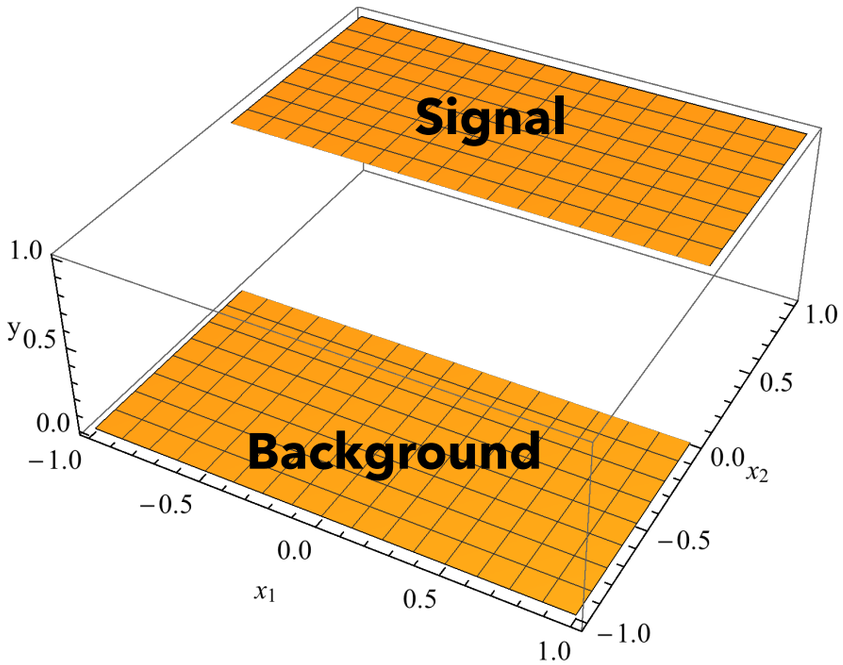
Output = 0

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

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

ROSENBLATT'S PERCEPTRON

- Illustrative examples:

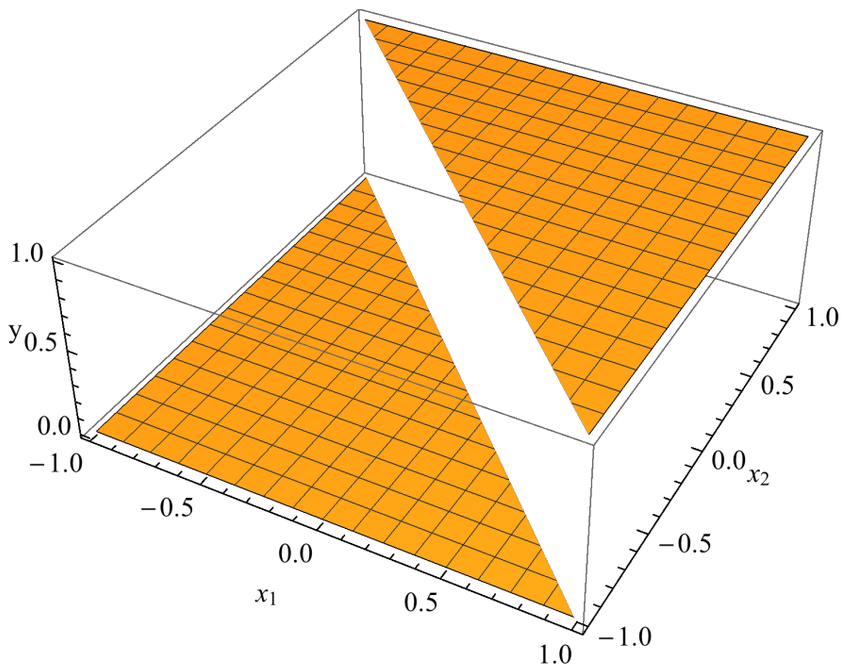


$w_1 = 0$

$w_2 = 1$

$\theta = 0$

Baseline for comparison, decision only on value of x_2

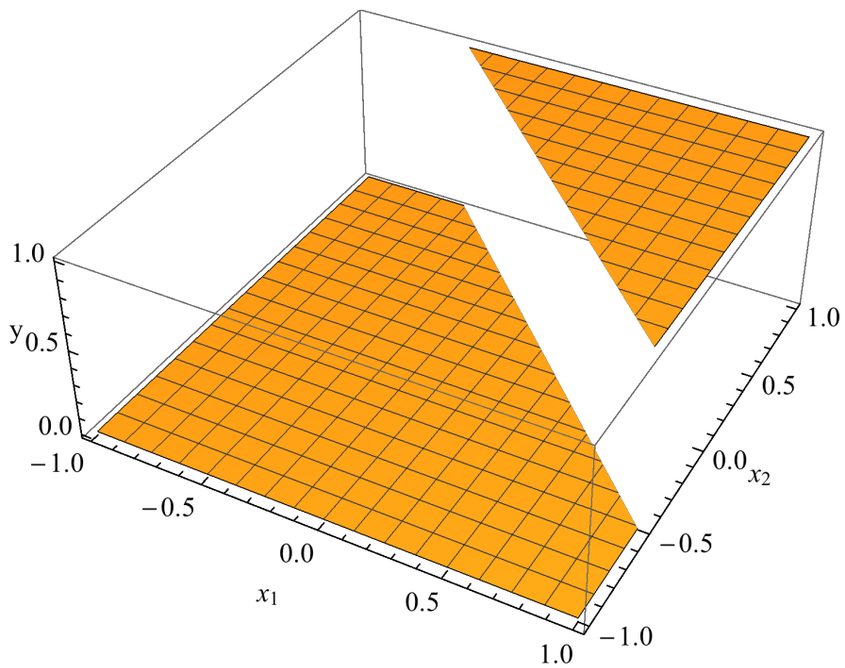


$w_1 = 1$

$w_2 = 1$

$\theta = 0$

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.

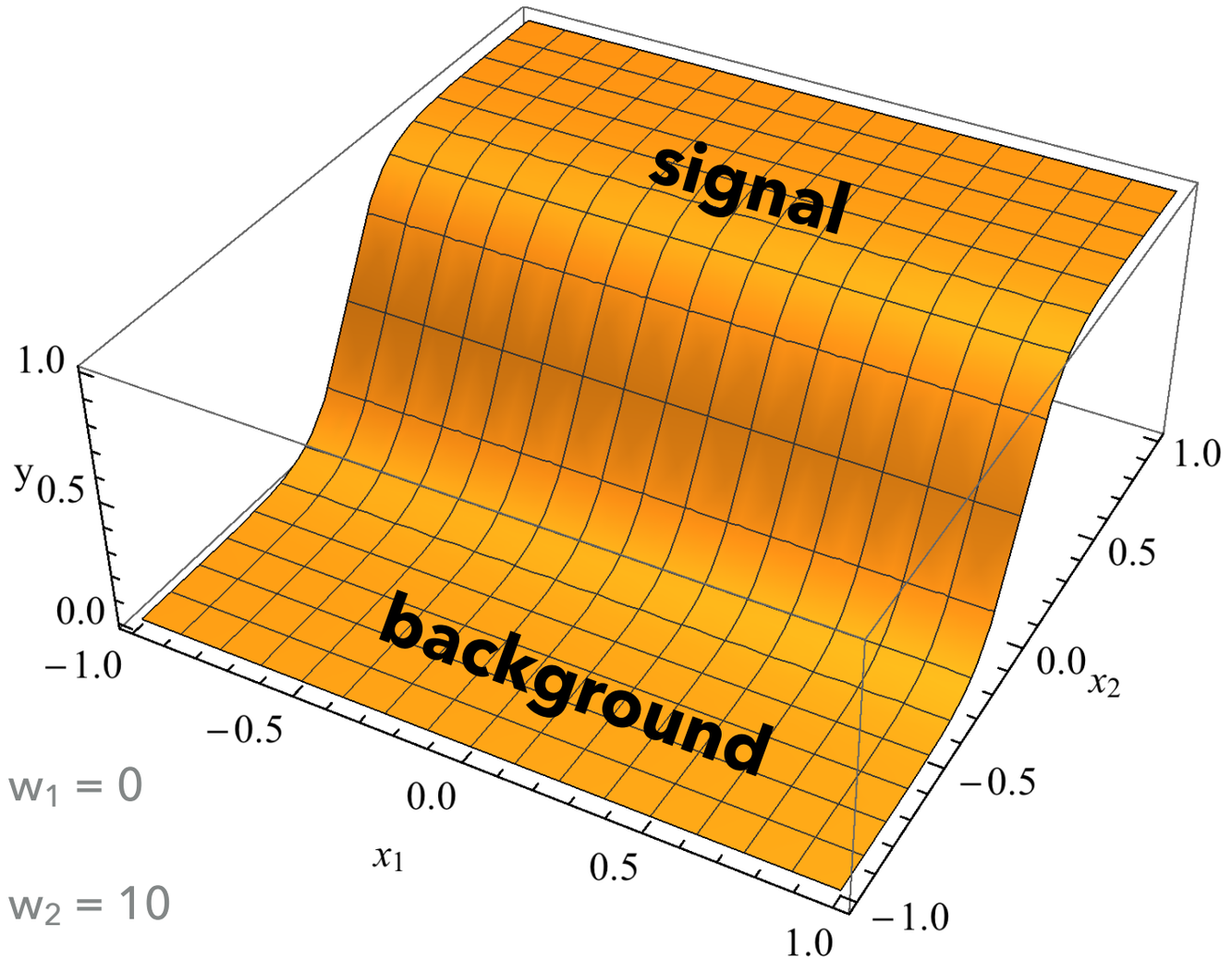
- It can be useful to provide an output that is continuous between these two extremes.
  - For that we require additional forms of activation function.

ACTIVATION FUNCTIONS: LOGISTIC (OR SIGMOID)

- A common activation function used in neural networks:

$$y = \frac{1}{1 + e^{w^T x + \theta}}$$

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



$w_1 = 0$

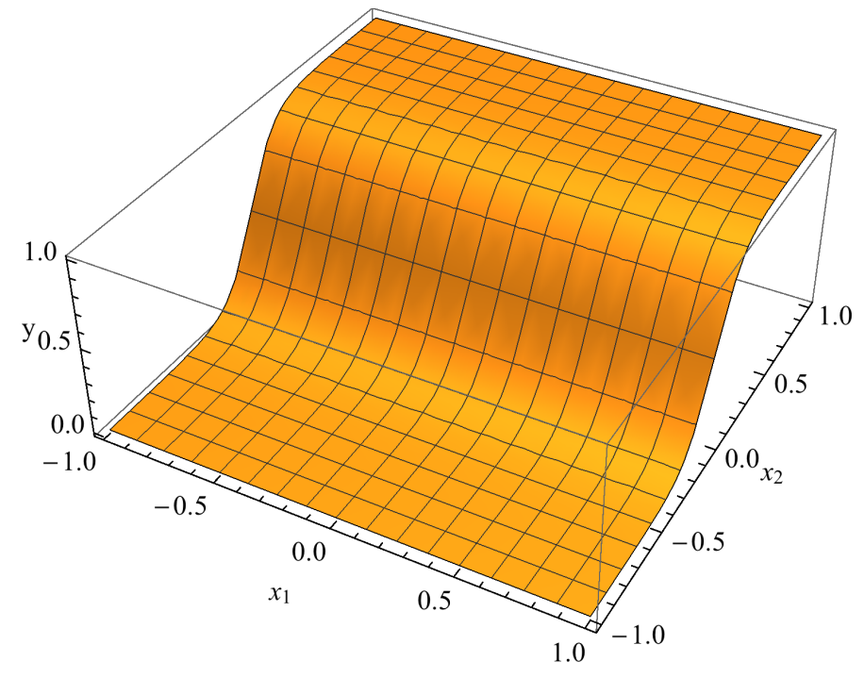
$w_2 = 10$

$\theta = 0$

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:

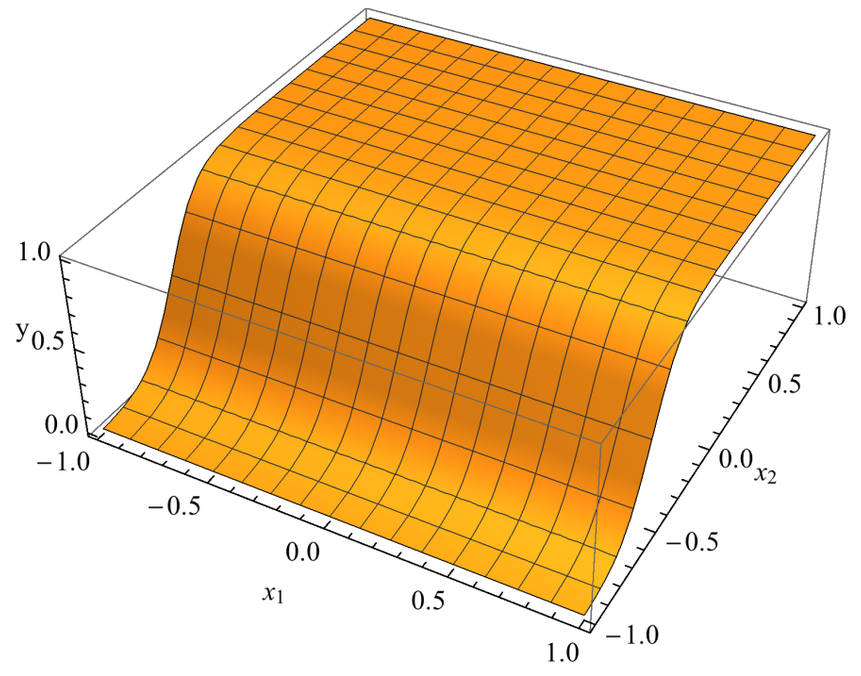


$w_1 = 0$

$w_2 = 10$

$\theta = 0$

Baseline for comparison, decision only on value of x_2

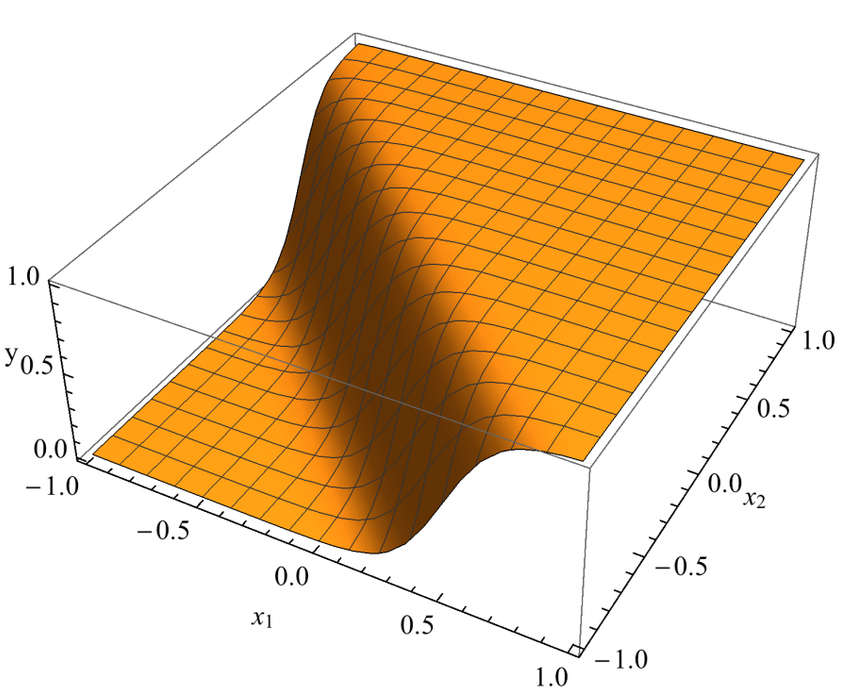


$w_1 = 0$

$w_2 = 10$

$\theta = -5$

Offset from zero using θ



$w_1 = 10$

$w_2 = 10$

$\theta = -5$

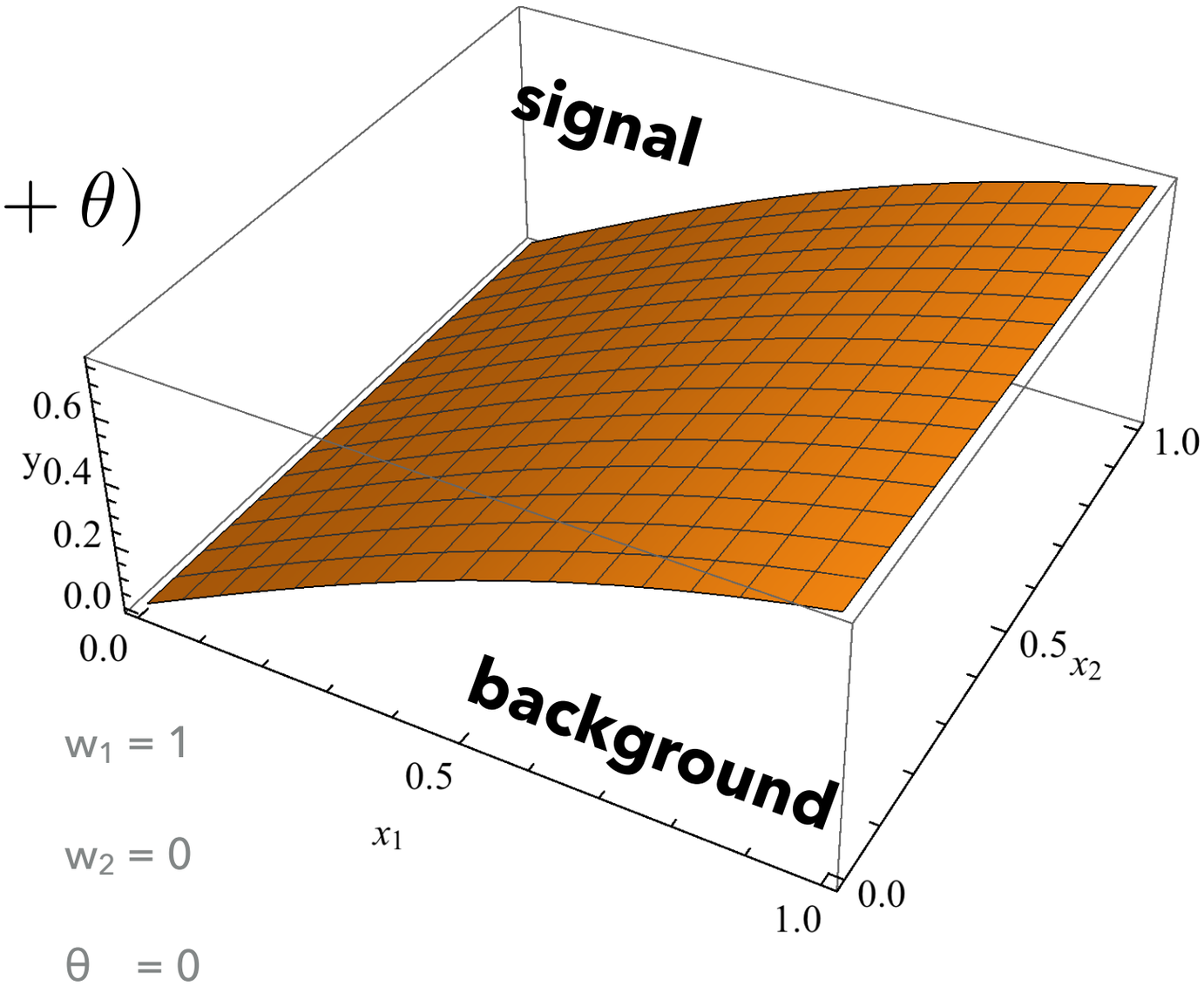
rotate "decision boundary" in (x_1, x_2)

ACTIVATION FUNCTIONS: HYPERBOLIC TANGENT

- A common activation function used in neural networks:

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

(Often used with θ = 0)



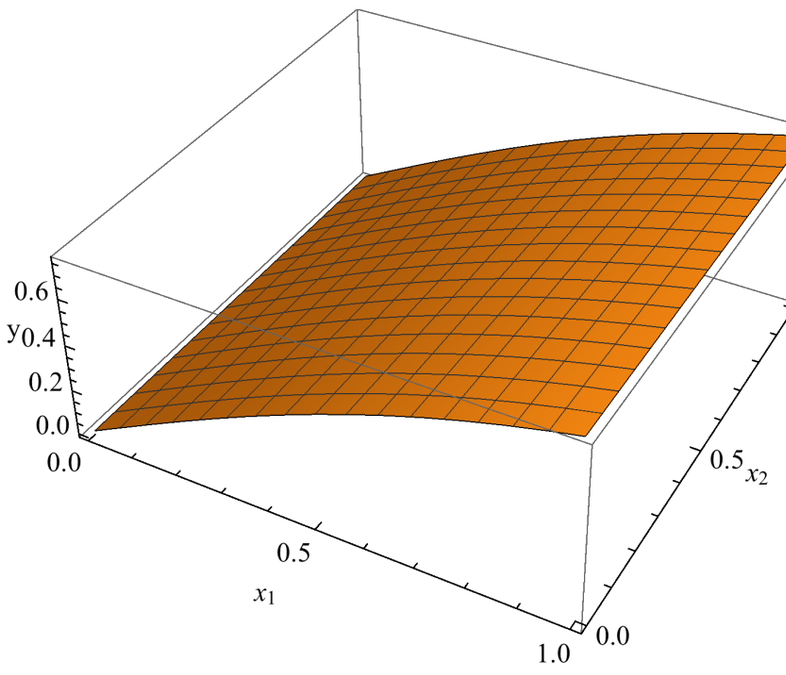
$w_1 = 1$

$w_2 = 0$

$\theta = 0$

ACTIVATION FUNCTIONS: HYPERBOLIC TANGENT $\tanh(w_1 x_1 + w_2 x_2 + \theta)$

- A common activation function used in neural networks:

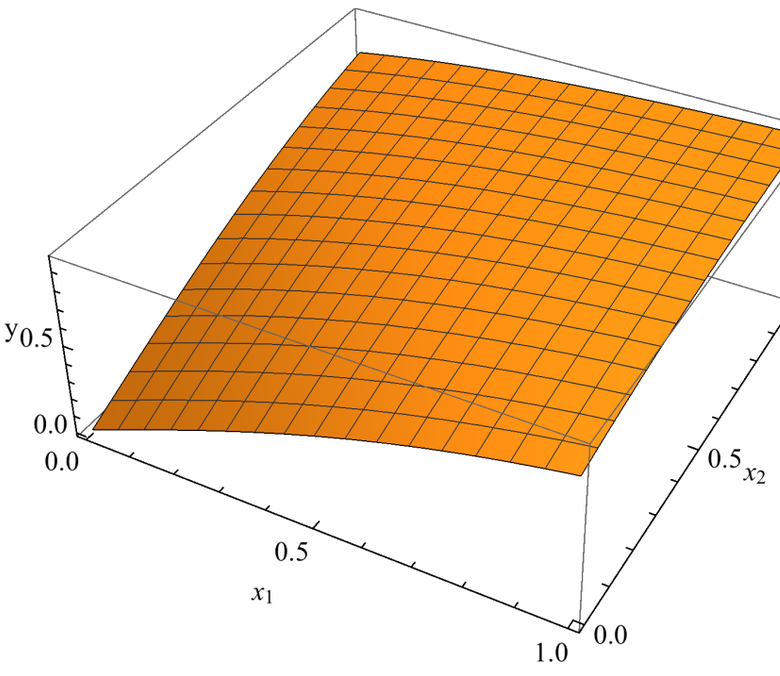


$w_1 = 1$

$w_2 = 0$

$\theta = 0$

Baseline for comparison, decision only on value of x_1

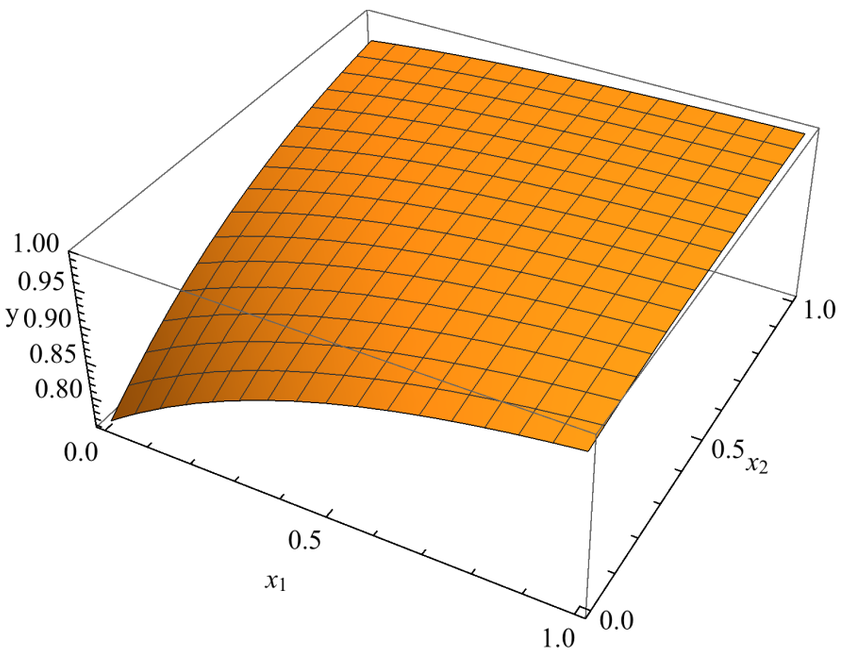


$w_1 = 1$

$w_2 = 1$

$\theta = 0$

rotate "decision boundary" in (x_1, x_2)



$w_1 = 1$

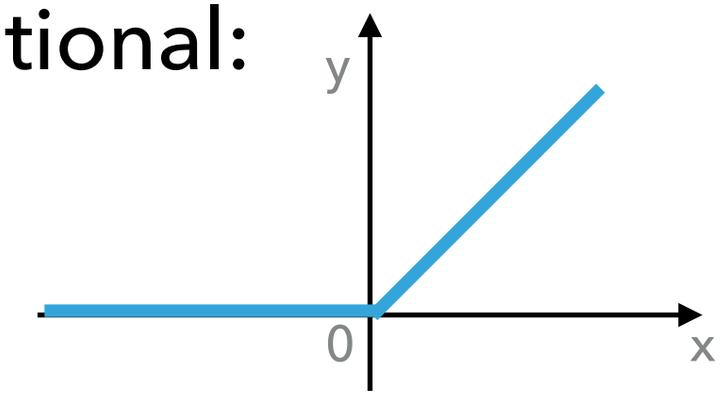
$w_2 = 1$

$\theta = -1$

Offset (vertically) from zero using θ

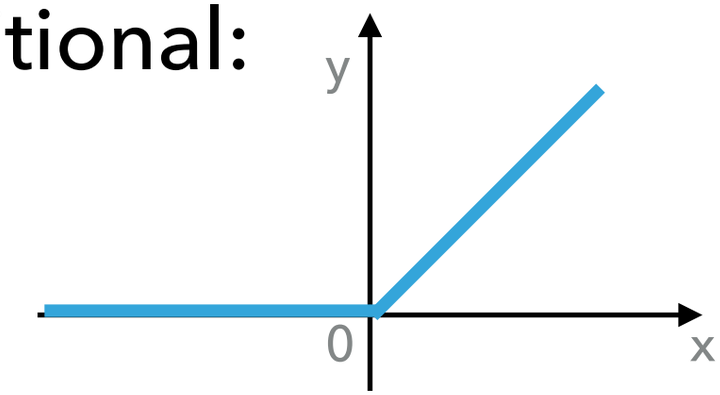
ACTIVATION FUNCTIONS: RELU

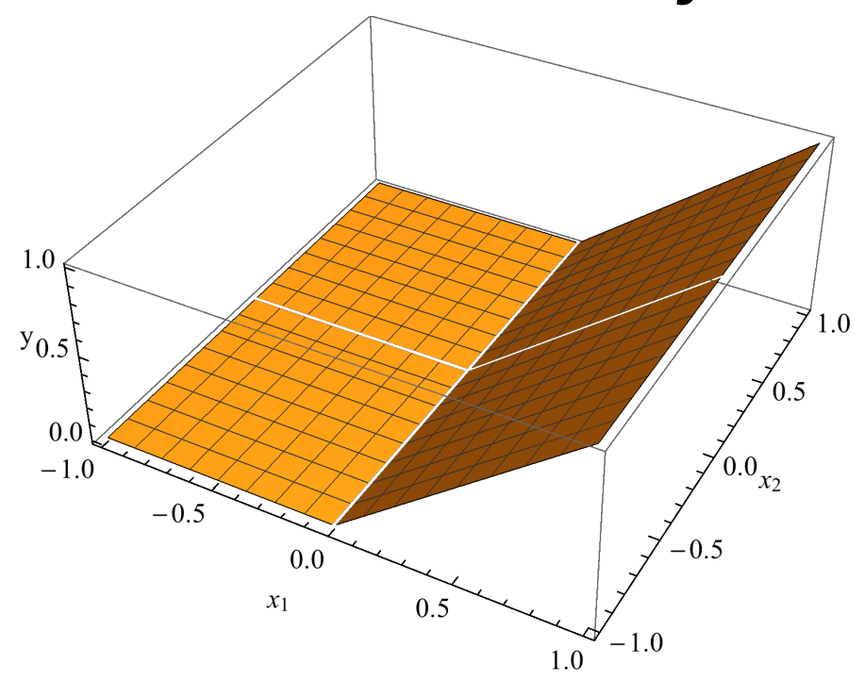
- The **Re**ctified **L**inear **U**nit activation function is commonly used for CNNs. This is given by a conditional:
  - If ($x < 0$) $y = 0$
  - otherwise $y = x$



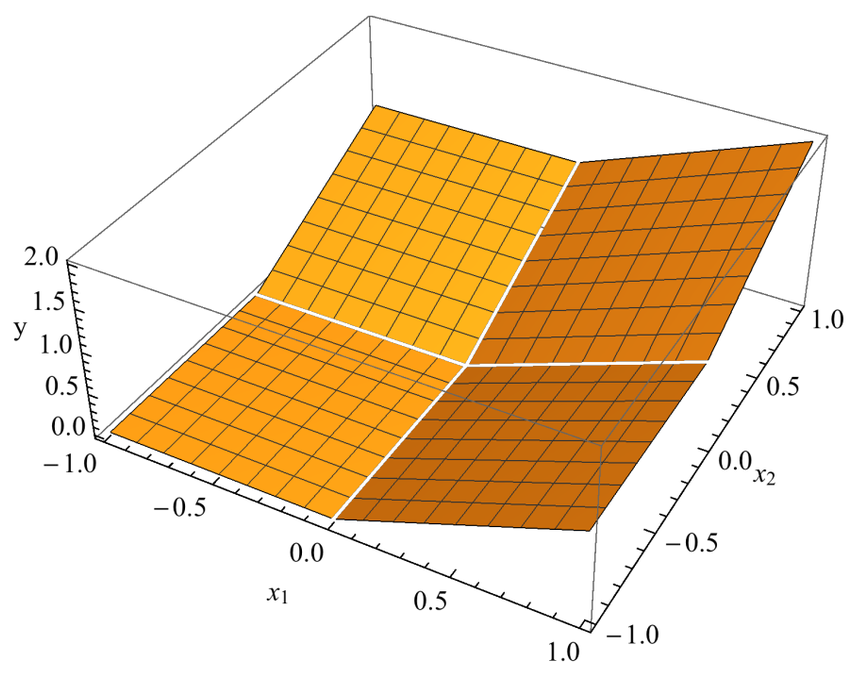
ACTIVATION FUNCTIONS: RELU

- 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 $y = x$

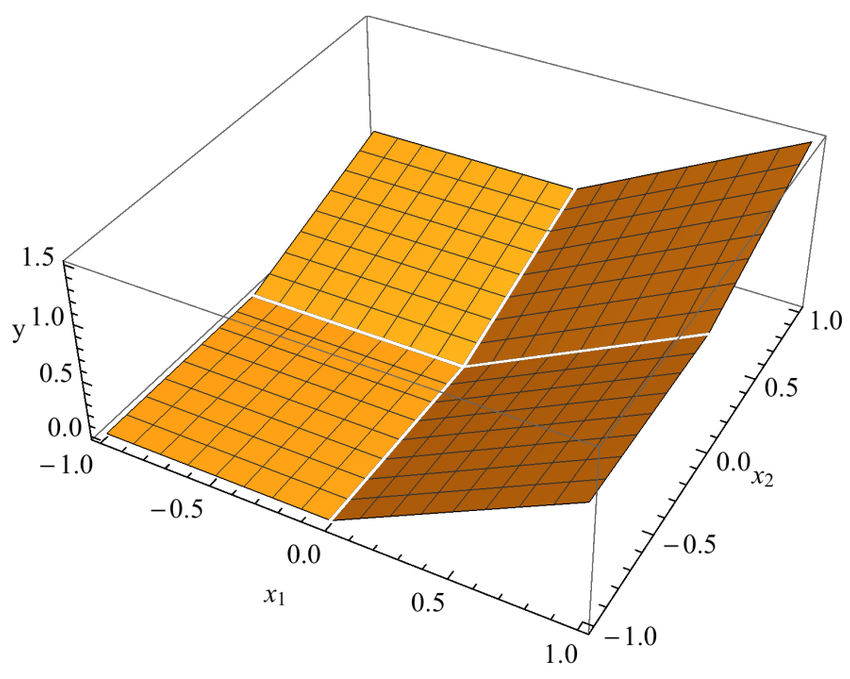




$w_1 = 1, w_2 = 0$



$w_1 = 1, w_2 = 1$

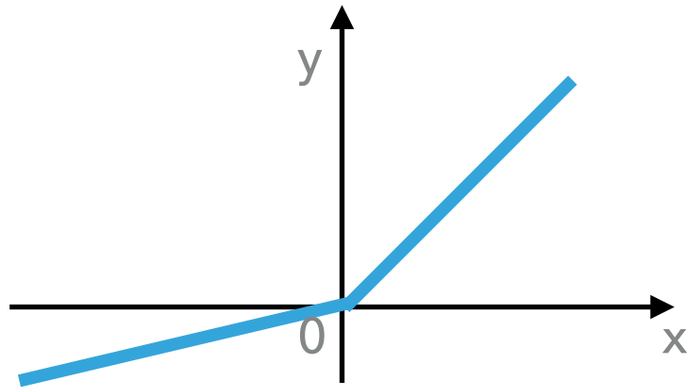


$w_1 = 1, w_2 = 0.5$

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 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

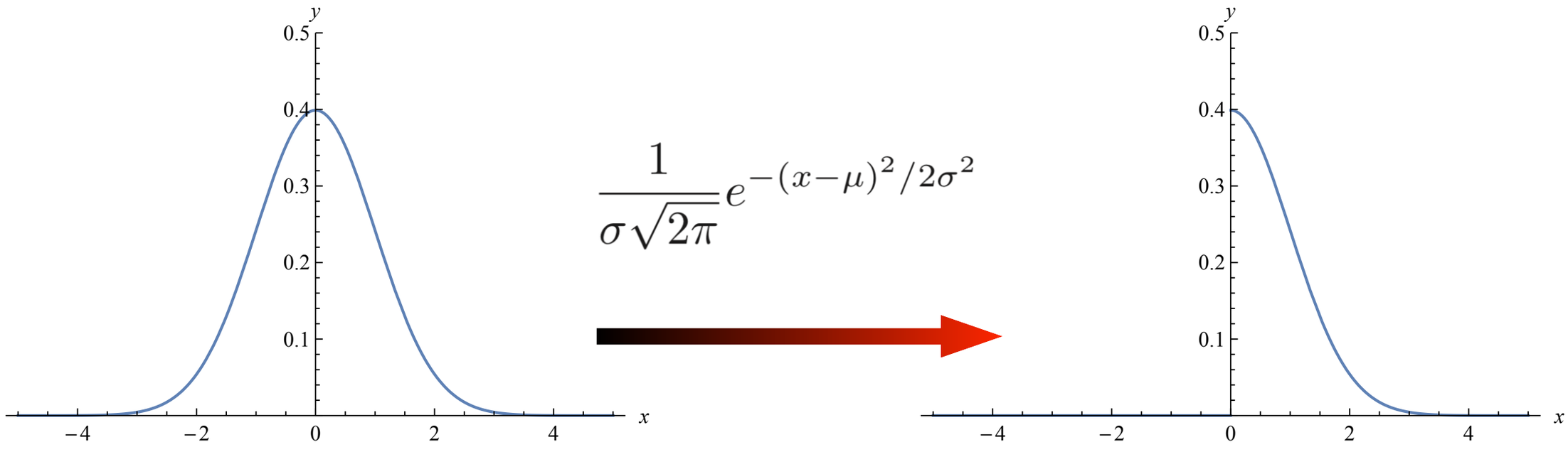
ACTIVATION FUNCTIONS: RELU

- 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

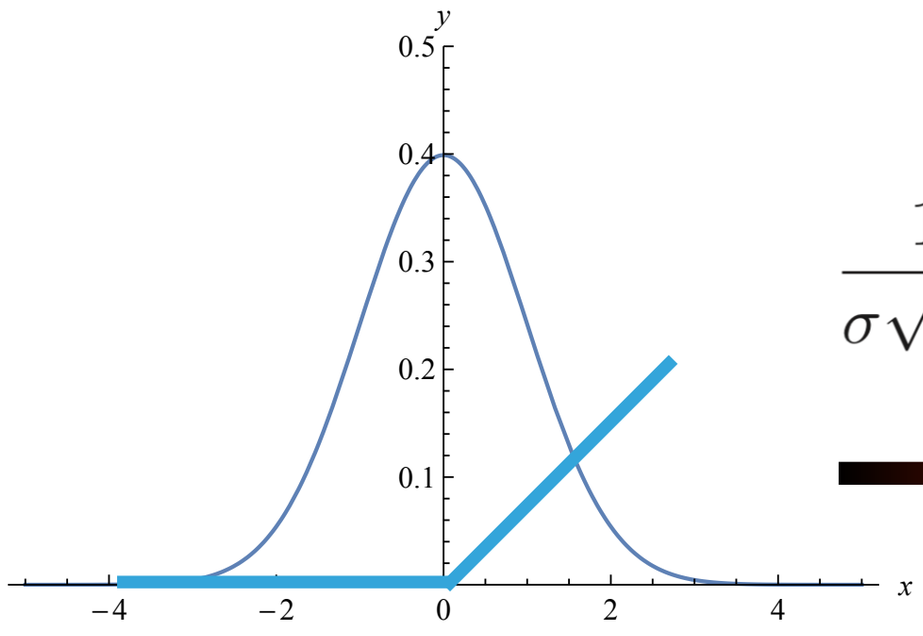
- N.B. Gradient descent optimisation algorithms will not change the weights for an activation function if the initial weight is set to zero.
  - This is why a truncated normal is used for initialisation, rather than a Gaussian that has $x \in [-\infty, +\infty]$.



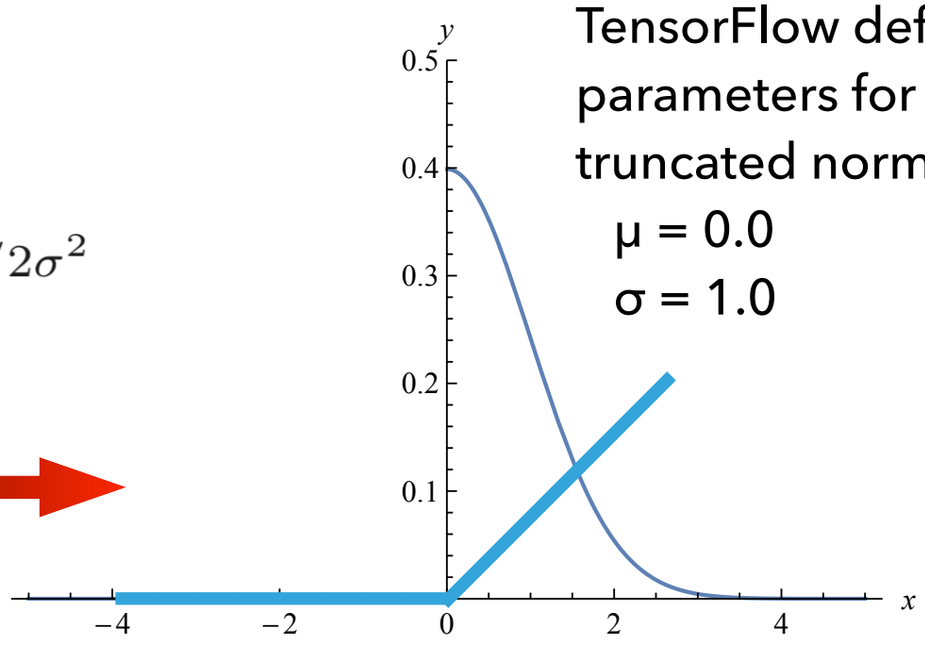
[1] Maas, Hannun, Ng, ICML2013.
[2] He, Zhang, Ren and Sun, arXiv:1502.01852

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:

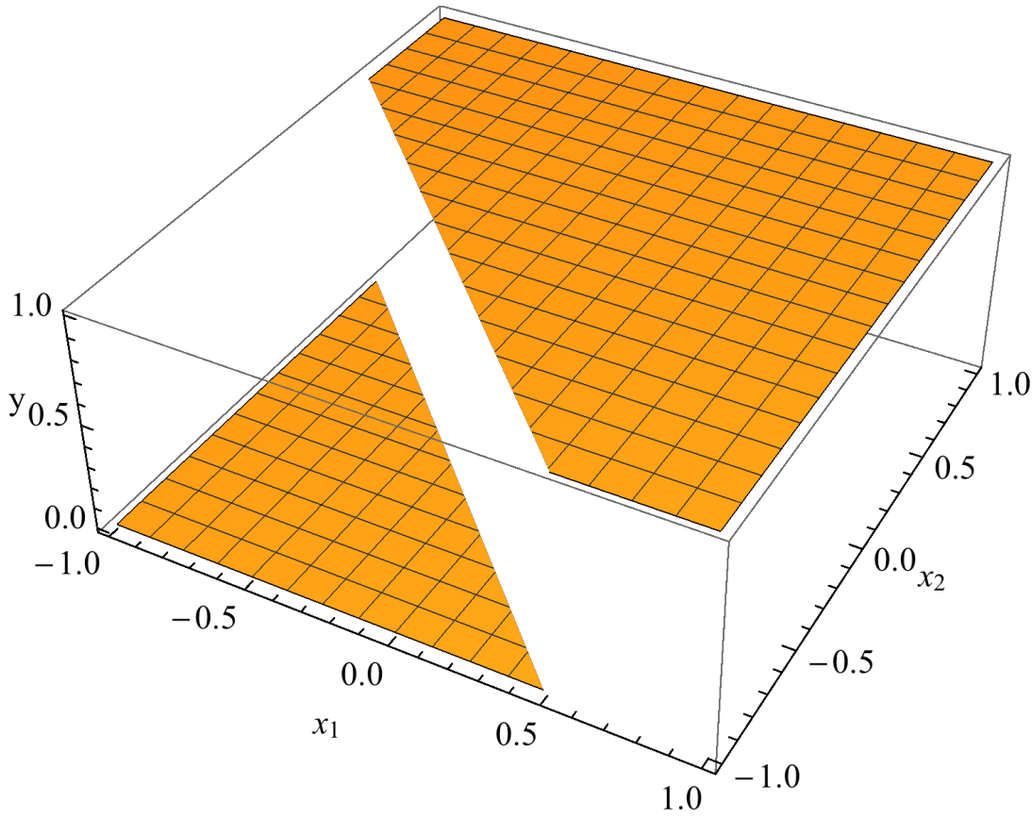
μ = 0.0

σ = 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.

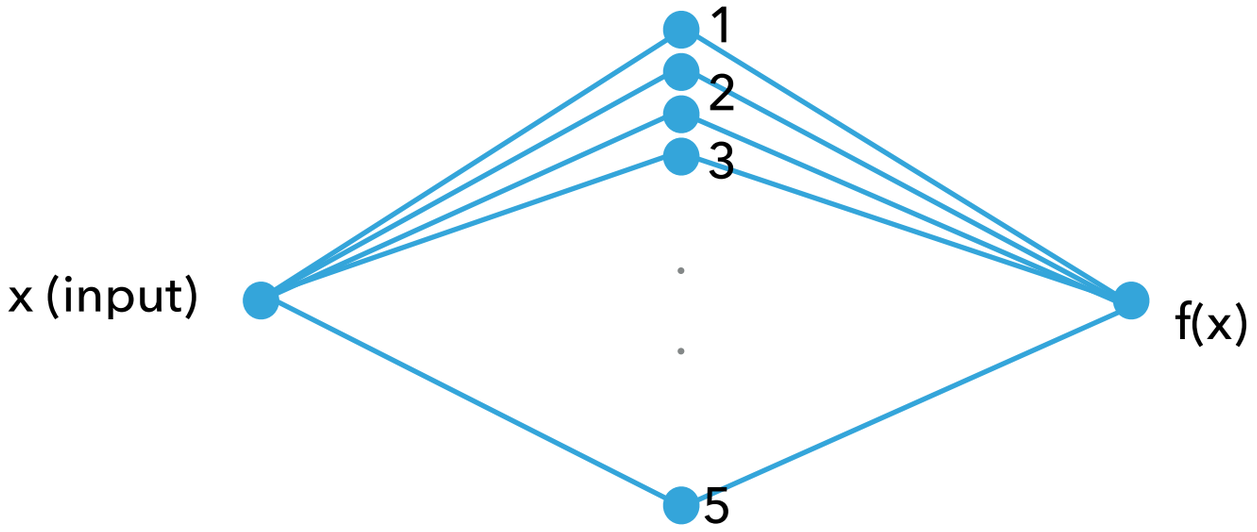
The only real difference is the heuristic used to determine the weights.

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

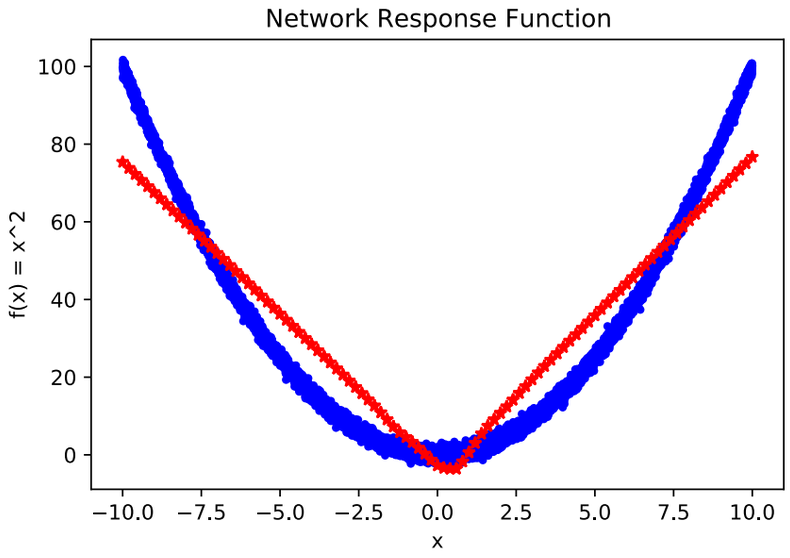
- When performing stochastic training, the optimisation converges faster when presenting unexpected examples.
- A pragmatic way to do this is to alternate the presentation of examples from different classes of event to the optimisation.
- Ensuring equal numbers of different classes are presented to the 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.

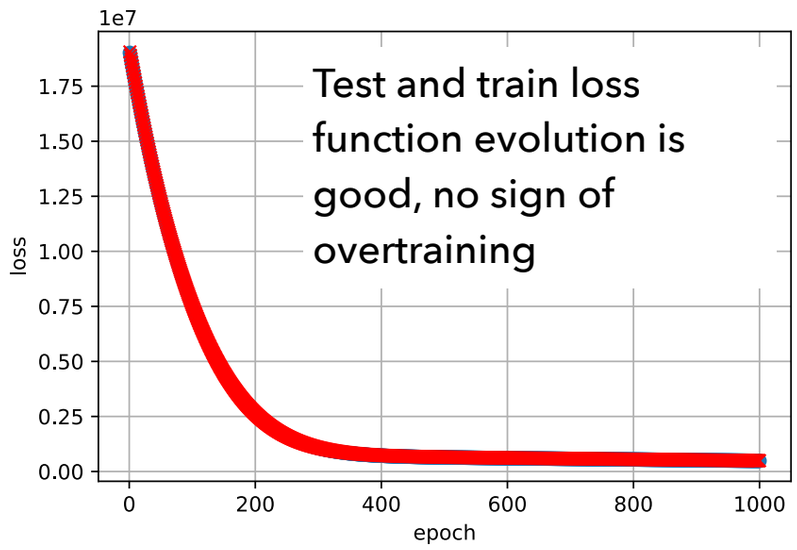
EXAMPLE: FUNCTION APPROXIMATION

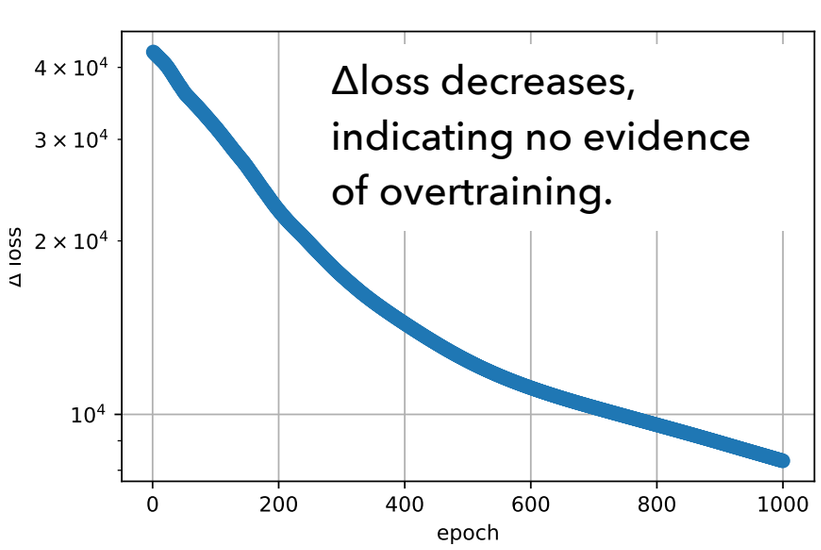
- Consider the model $y = x^2$
- This can be learned using an ANN, with a single input, x, and a single output y for each example.
- Model:
  - ADAM optimiser
  - 1 hidden layer with 50 nodes
  - ReLU activation function for nodes in the hidden layer
  - $w^T x + \beta$ for the output node (to give an 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.



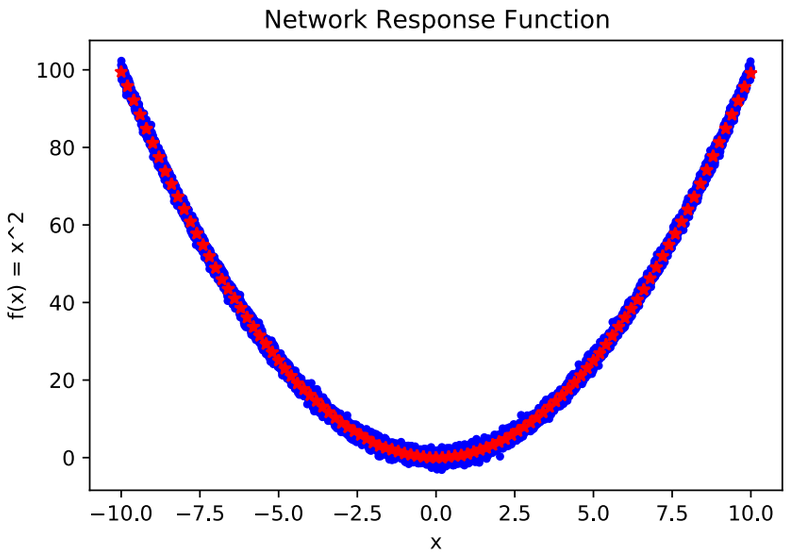


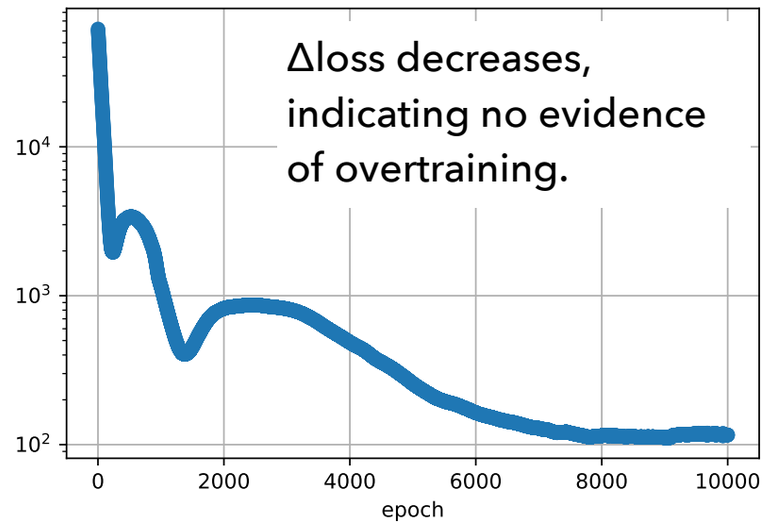


- Increasing or removing the Gaussian noise will not lead to improvement.
- Increasing the number of epochs or number of training data can yield an improvement.
- Changing the network architecture can lead to an improvement.

EXAMPLE: FUNCTION APPROXIMATION

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



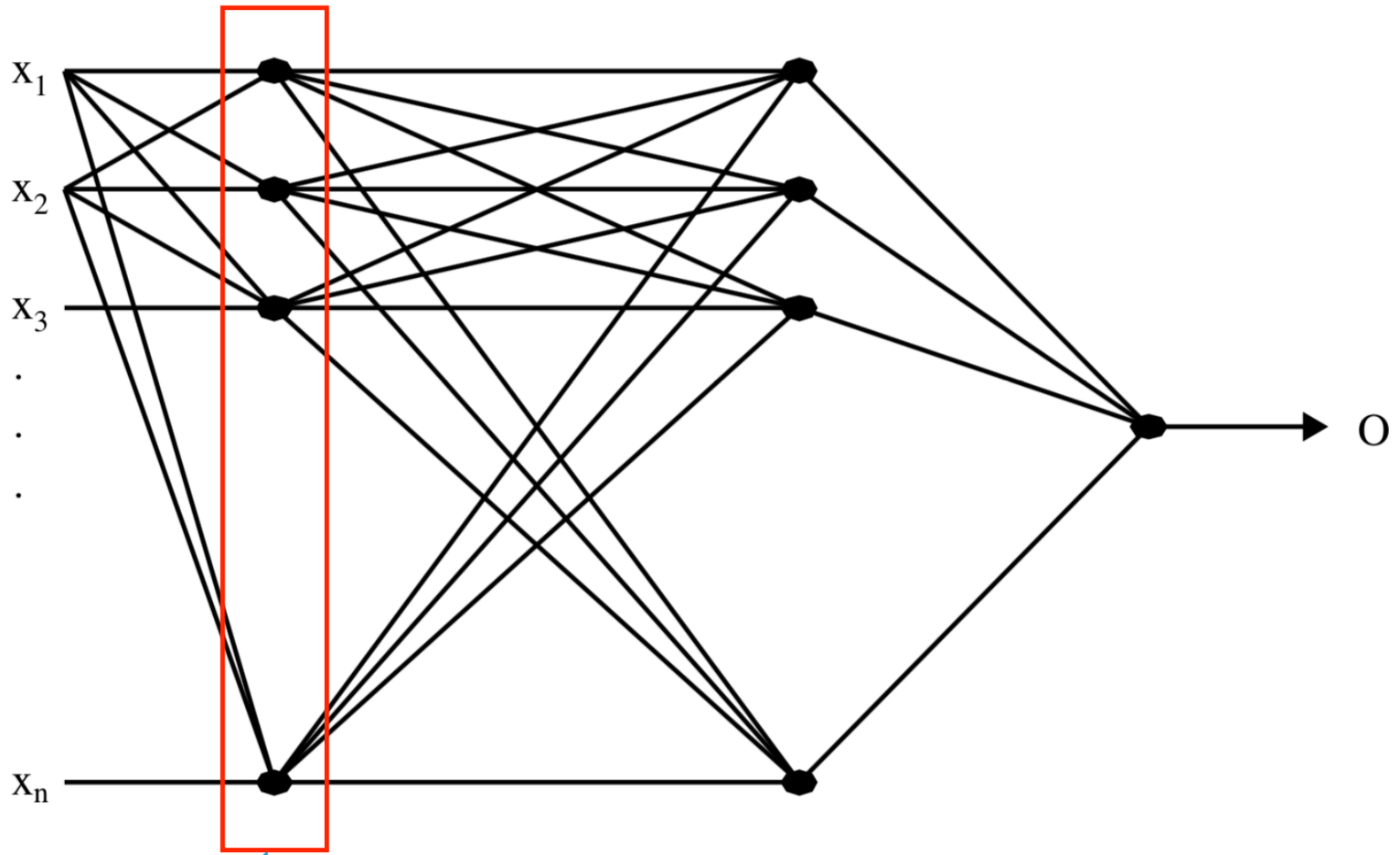


- Able to learn a good model to approximate the target function.
- Can increase the learning rate to learn faster.
- ADAM has built in learning rate decay parameters, so a learning rate of 0.1 will yield a good training with only 1000 epochs; c.f. the previous page with $\alpha = 0.001$.

NEURAL NETWORKS
MULTI LAYER PERCEPTRONS

MULTI LAYER PERCEPTRONS

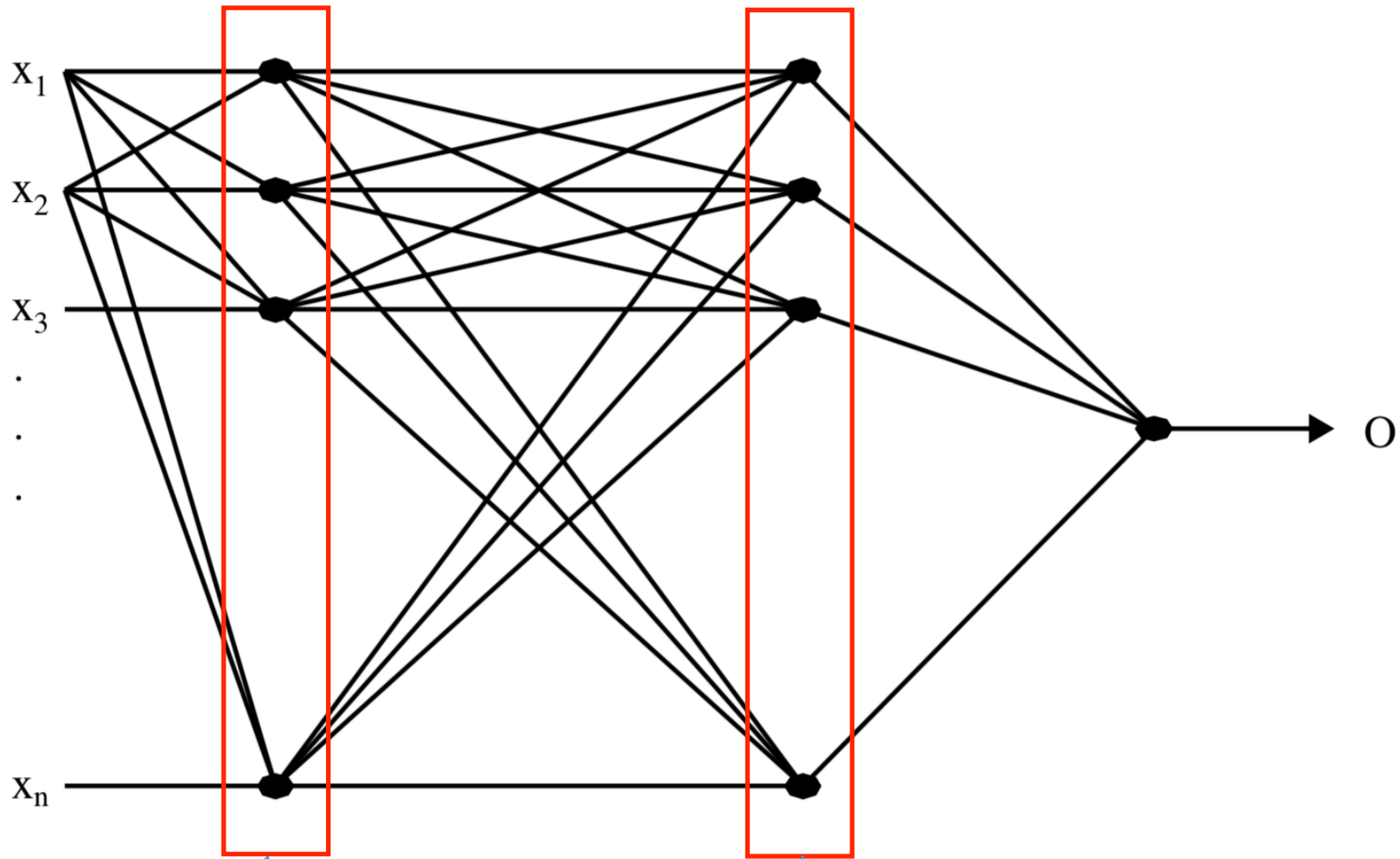
- Illustrative example: Input data example: $x = \{x_1, x_2, x_3, \ldots, x_n\}$



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

MULTI LAYER PERCEPTRONS

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

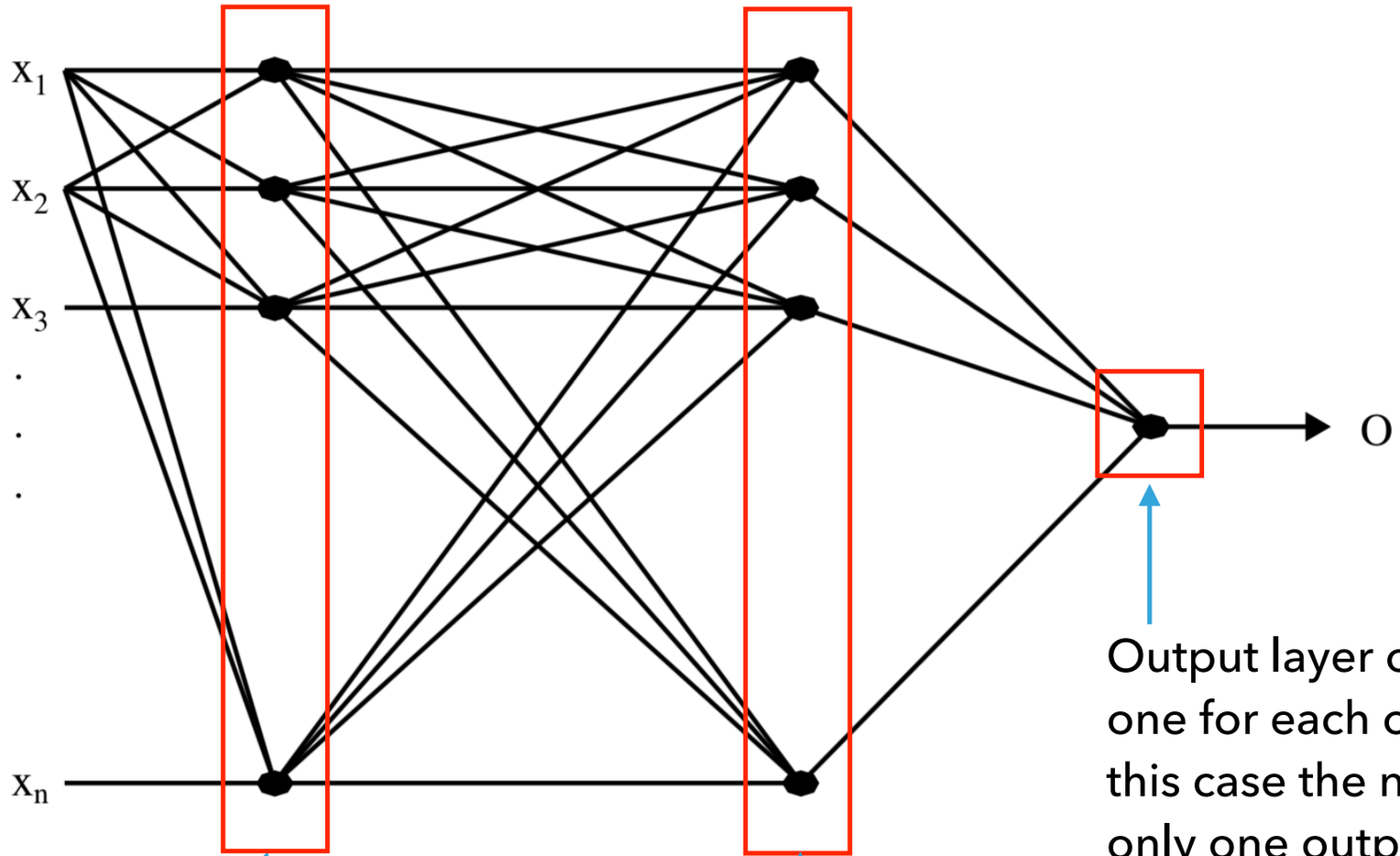


Input layer of n perceptrons; one for each dimension of the input feature space

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

MULTI LAYER 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.

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

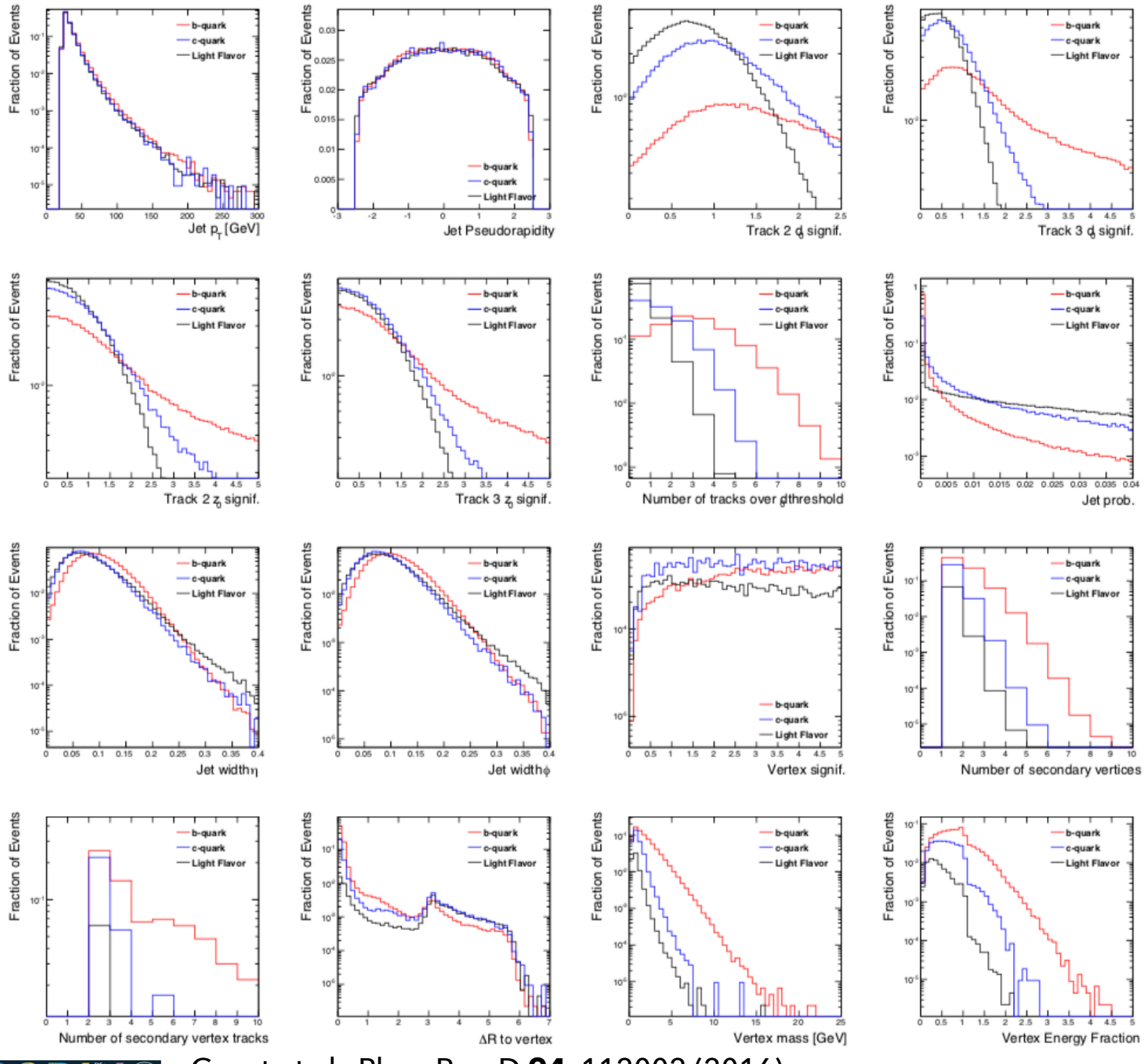
- Quark colour confinement leads to the creation of jets as pairs of quarks and anti-quarks are pulled apart.
  - As a result we don't see bare quarks.
  - However we do see many hadrons eliminating from some underlying quark.
- These hadrons form objects called jets that are reconstructed in our detectors.
- 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.

Guest et al., Phys. Rev. D **94**, 112002 (2016)

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

$$\left(\frac{\sum_i p_{\mathrm{T}i}\Delta\eta_i^2}{\sum_i p_{\mathrm{T}}}\right)^{1/2}$$

  and analogously for ϕ.
- The combined vertex significance,

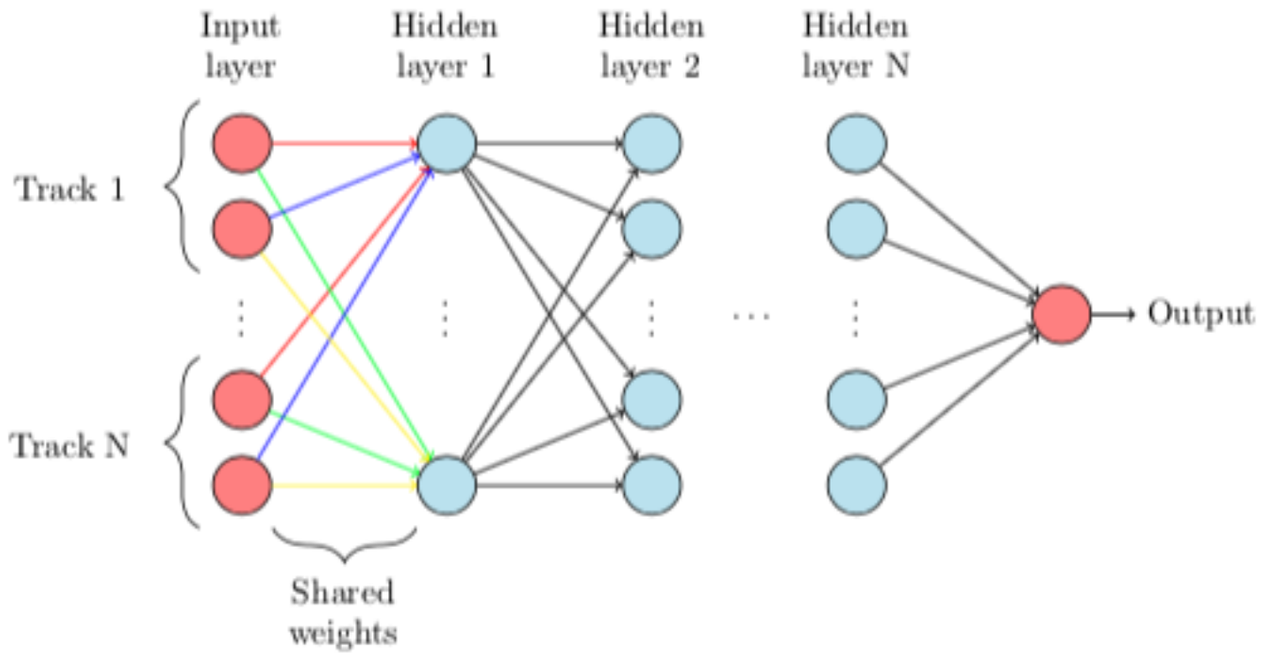
$$\frac{\sum_i d_i/\sigma_i^2}{\sqrt{\sum_i 1/\sigma_i^2}}$$

  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]

Guest et al., Phys. Rev. D **94**, 112002 (2016)

EXAMPLE: JET FLAVOUR CLASSIFICATION

- Several different algorithms are used including MLPs



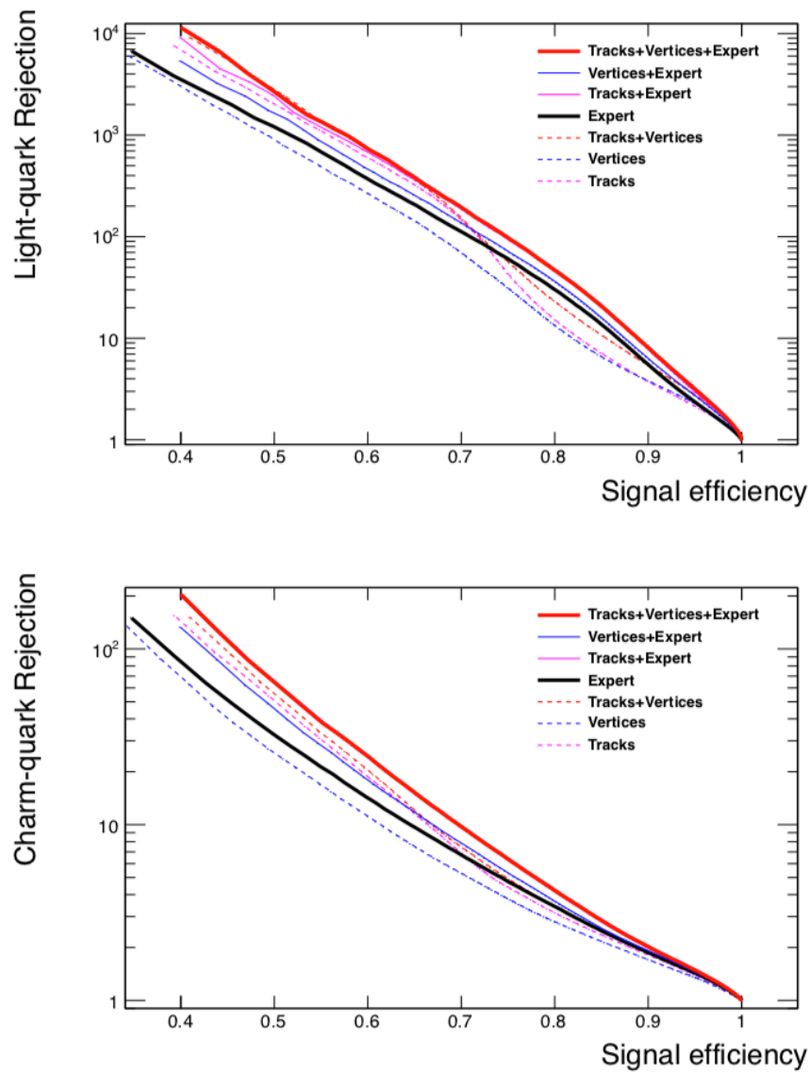
- "Experts" are networks that are trained to address a specific issue. This study constructs "Experts" that are used as inputs to a final network.

- This is an example of a committee machine.

EXAMPLE: JET FLAVOUR CLASSIFICATION

- Several different algorithms are used including MLPs

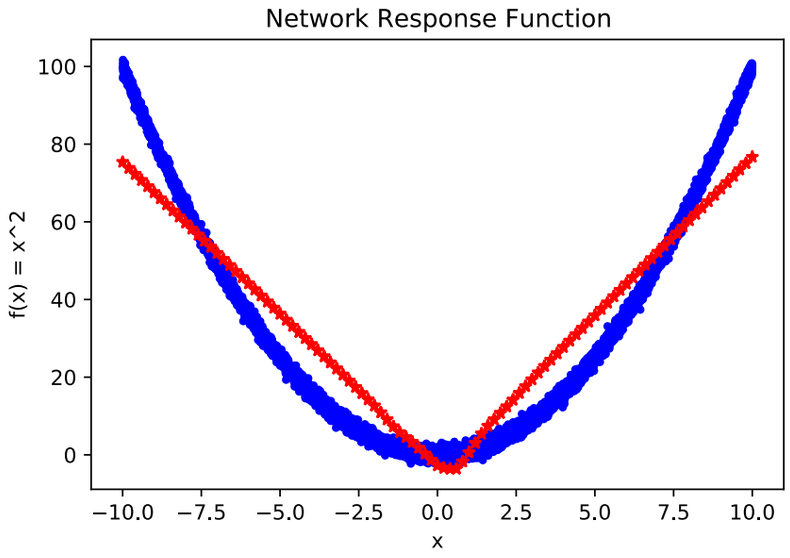
Inputs			Technique	AUC
Tracks	Vertices	Expert		
✓			Feedforward	0.916
✓			LSTM	0.917
✓			Outer	0.915
	✓		Feedforward	0.912
	✓		LSTM	0.911
	✓		Outer	0.911
✓	✓		Feedforward	0.929
✓	✓		LSTM	0.929
✓	✓		Outer	0.928
		✓	Feedforward	0.924
		✓	LSTM	0.925
		✓	Outer	0.924
✓		✓	Feedforward	0.937
✓		✓	LSTM	0.937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

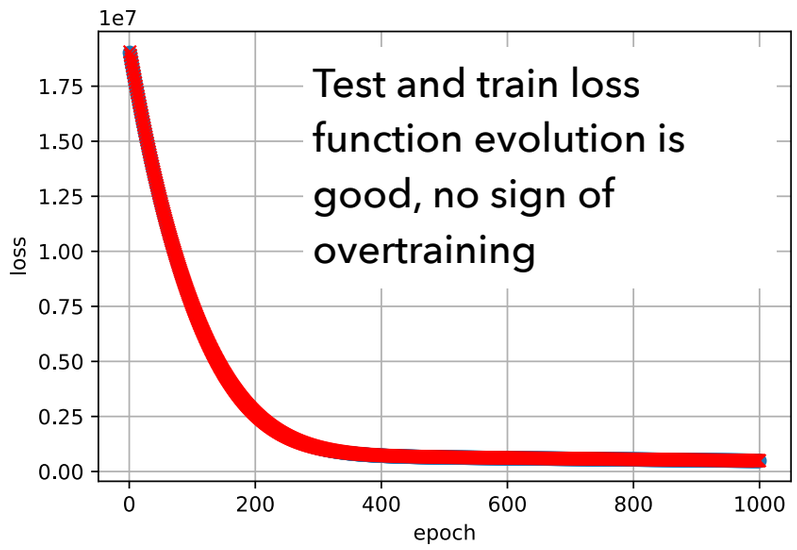


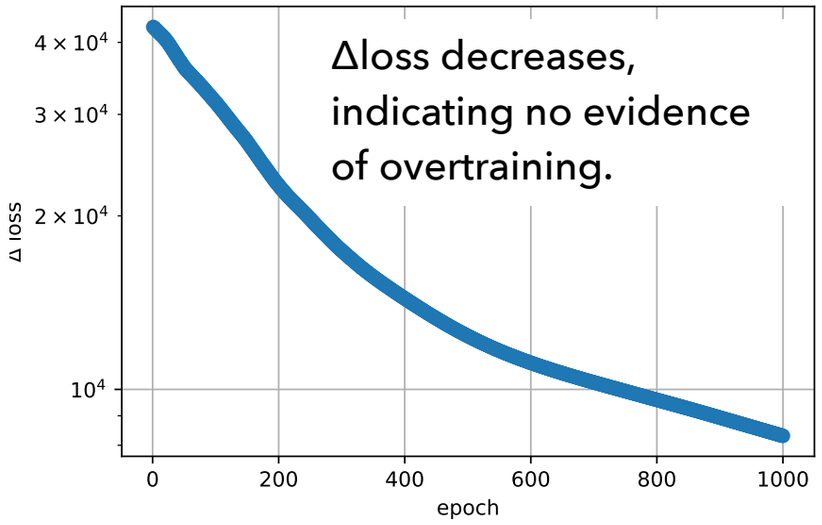
- They give similar performance.
- CMS has followed a deep network approach to jet tagging in their latest work (e.g. H→bb) [see A.M. Sirunyan et al 2018 JINST 13 P05011, CMS PAS HIG-18-016].

EXAMPLE: FUNCTION APPROXIMATION (RECAP)

- The model obtained does not give a good prediction.



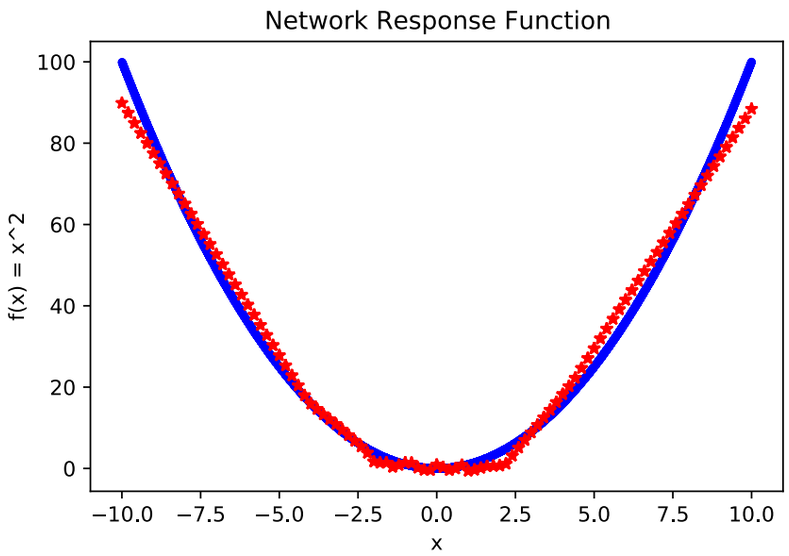


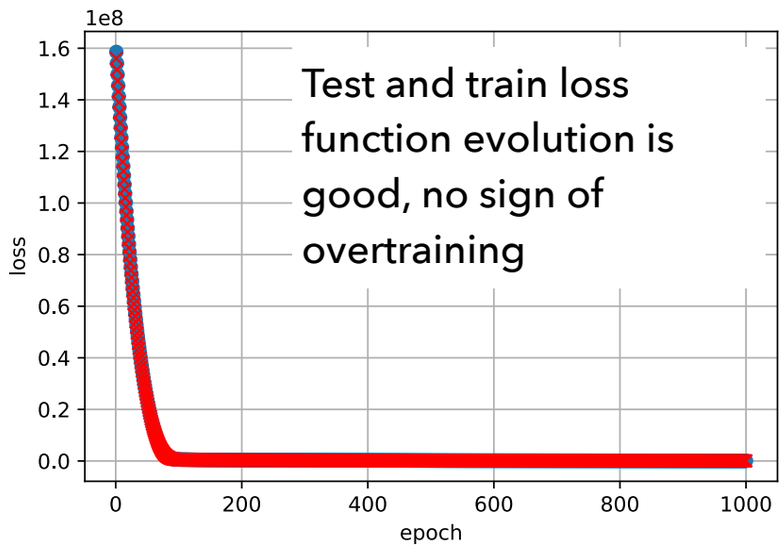


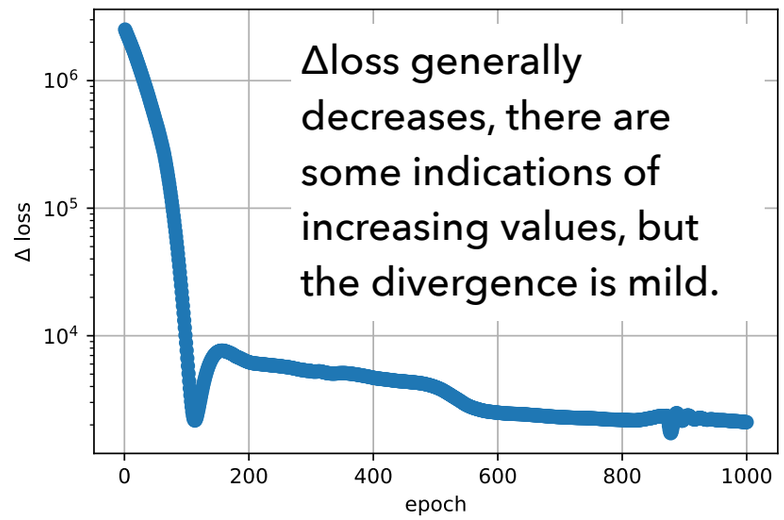
- Increasing or removing the Gaussian noise will not lead to improvement.
- Increasing the number of epochs or number of training data can yield an improvement.
- Changing the network architecture can lead to an improvement.

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-compete 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.

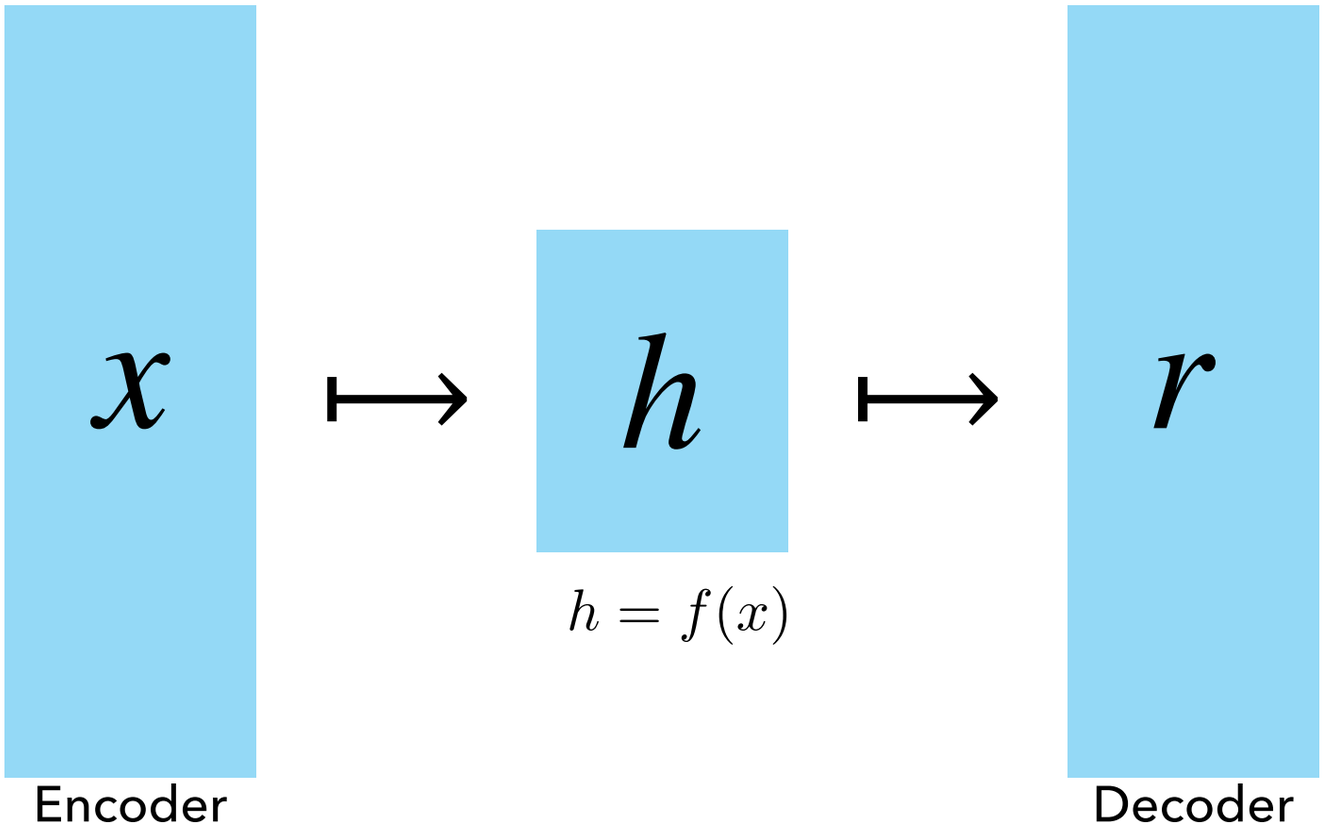
USING AUTO-ENCODERS

$$x \quad \mapsto \quad h \quad \mapsto \quad r$$

x: input feature space

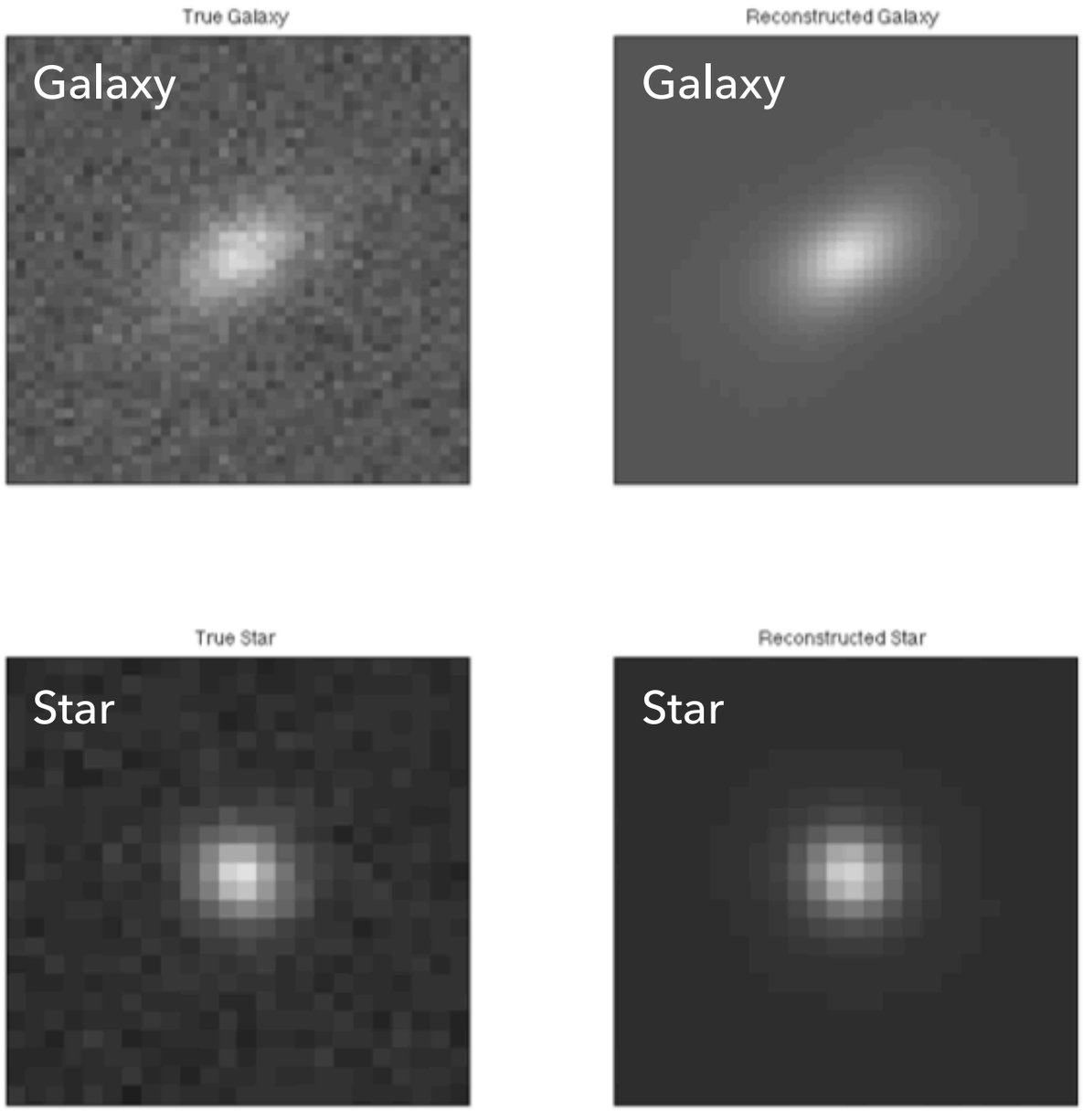
h: hidden layer giving an alternate representation of the data

r: reconstruction of x computed by the auto-encoder



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).
  - Starts 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.



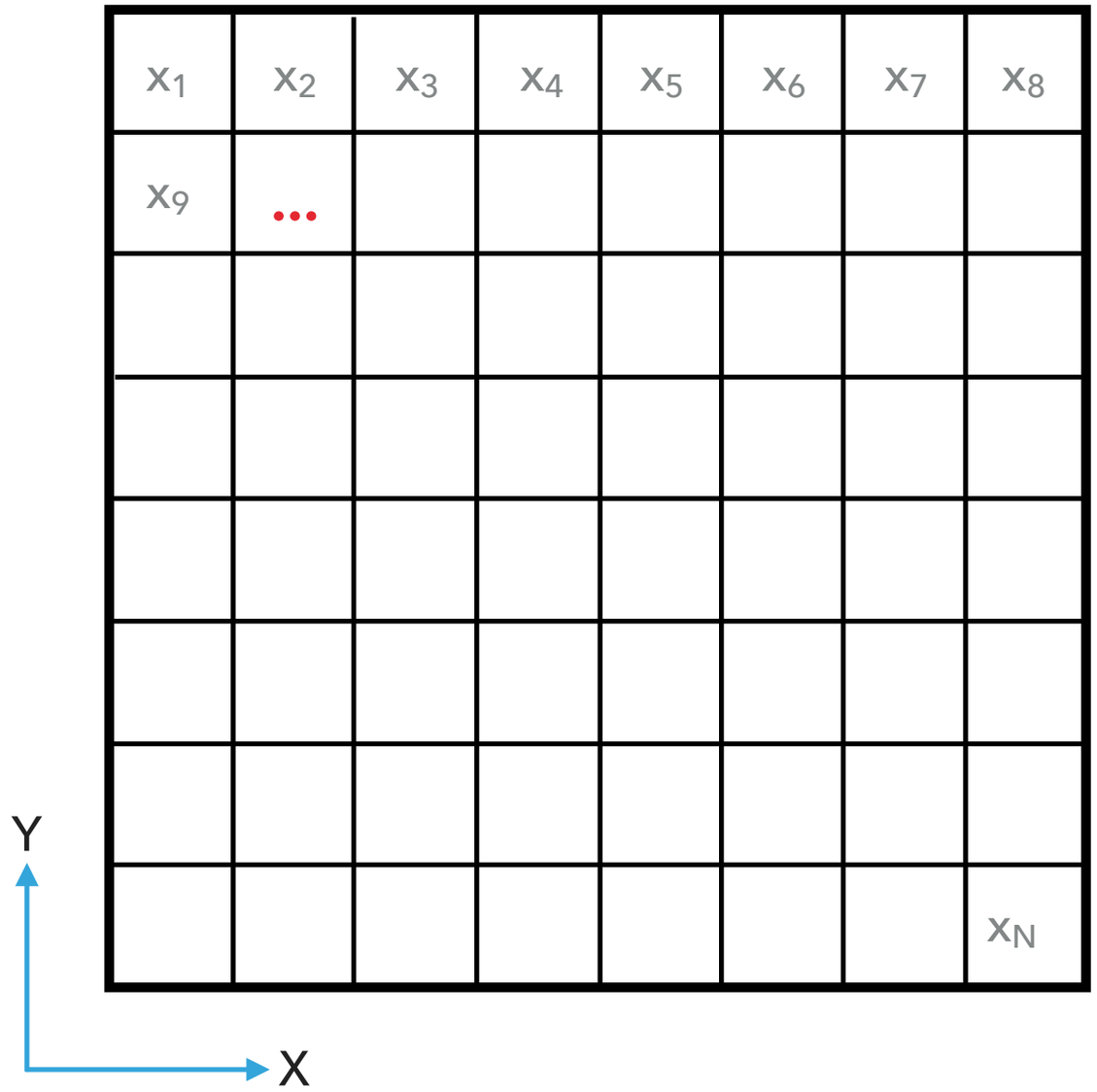
NEURAL NETWORKS
CONVOLUTIONAL NEURAL NETWORKS

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_{pixels} x m_{pixels}.[1]
- Colour images have a depth of 3 (R, G, B); so the dimensionality of the feature space is 3 x n_{pixels} x m_{pixels}.[1]

[1] K Fukushima, Bio. Cybernetics **36** p193-202, 1980.
[1]Typically CNNs are applied to square images.

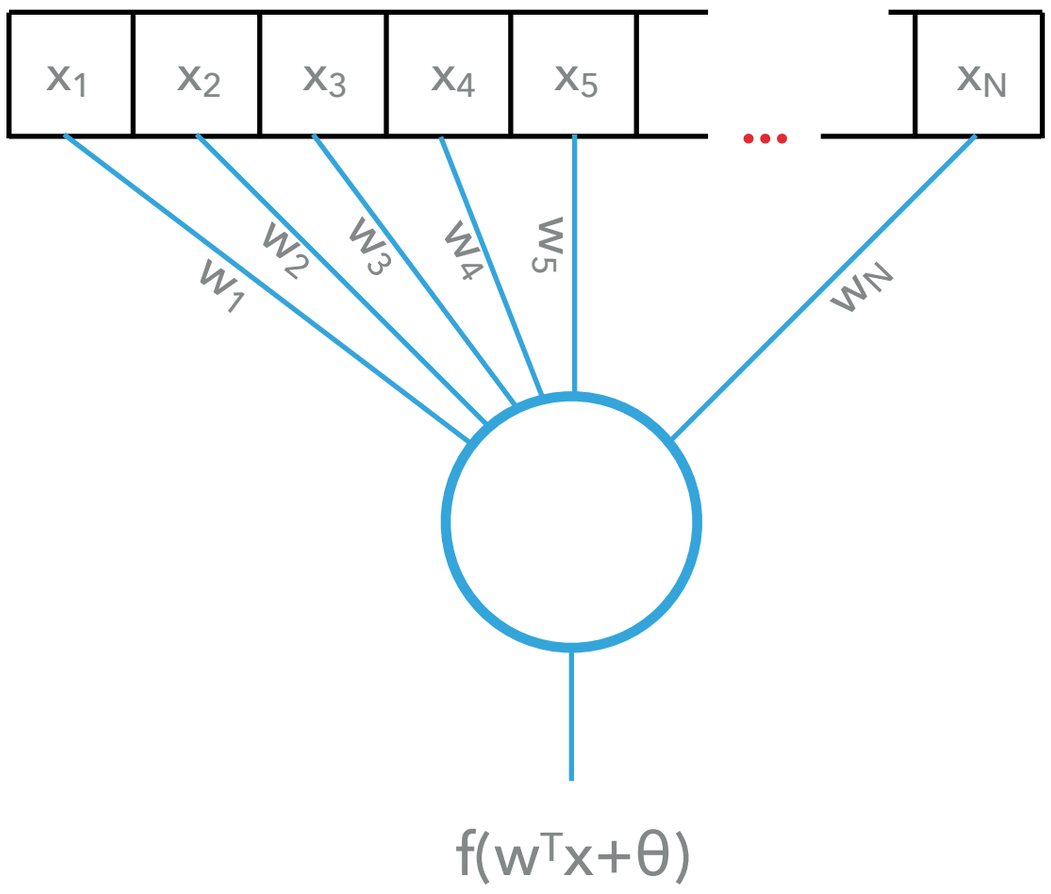
INPUT DATA



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 loose the spatial correlations between information in the image.
- The image can be represented by a a line of features.
- Doing this removes the spatial correlations and would naturally lend itself to being processing by a perceptron; i.e. $f(w^T x+\theta)$.

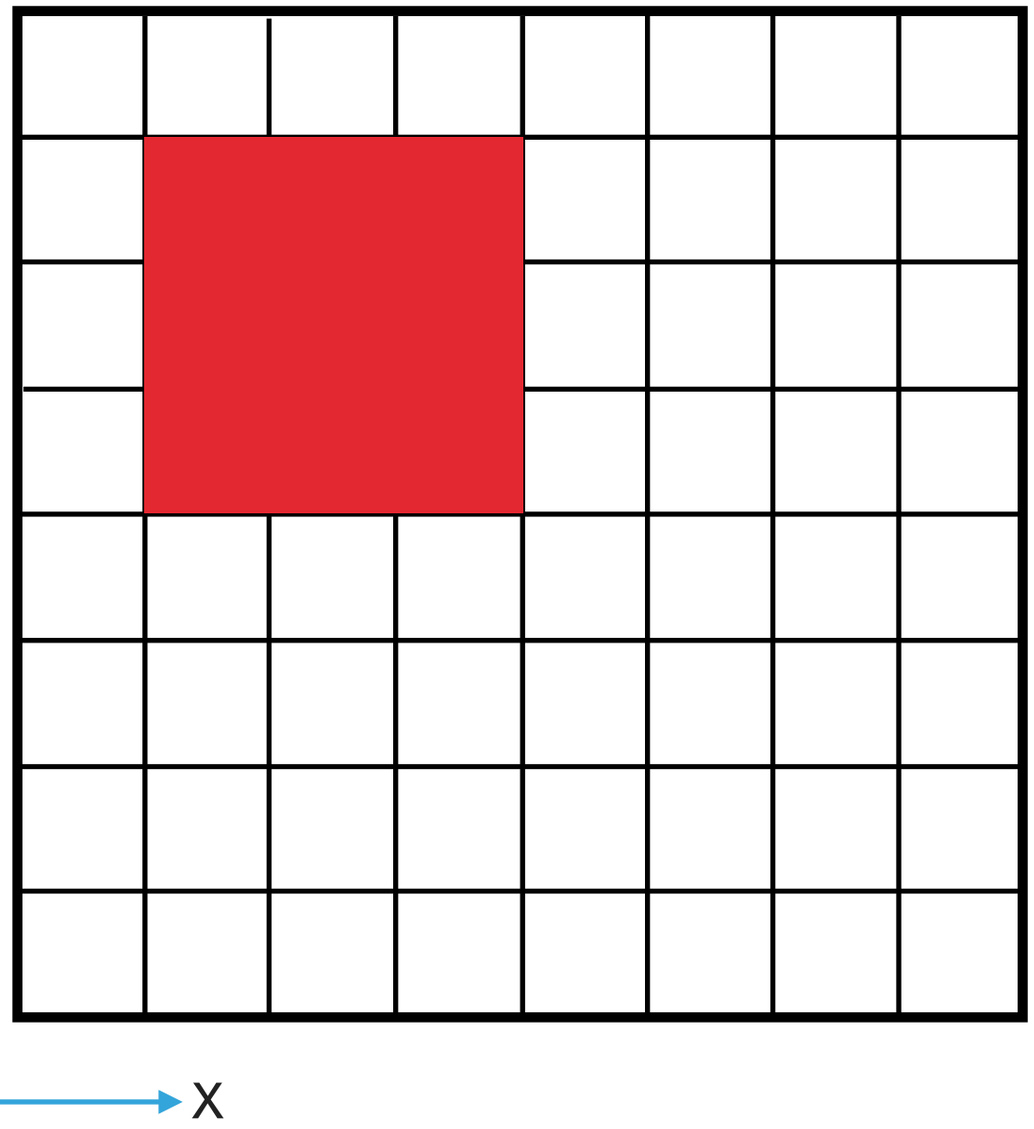
INPUT DATA



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 loose the spatial correlations between information in the image.
- The image can be represented by a a line of features.
- But doing this removes the spatial correlations and would naturally lend itself to being processing by a perceptron; i.e. $f(w^T x+\theta)$.

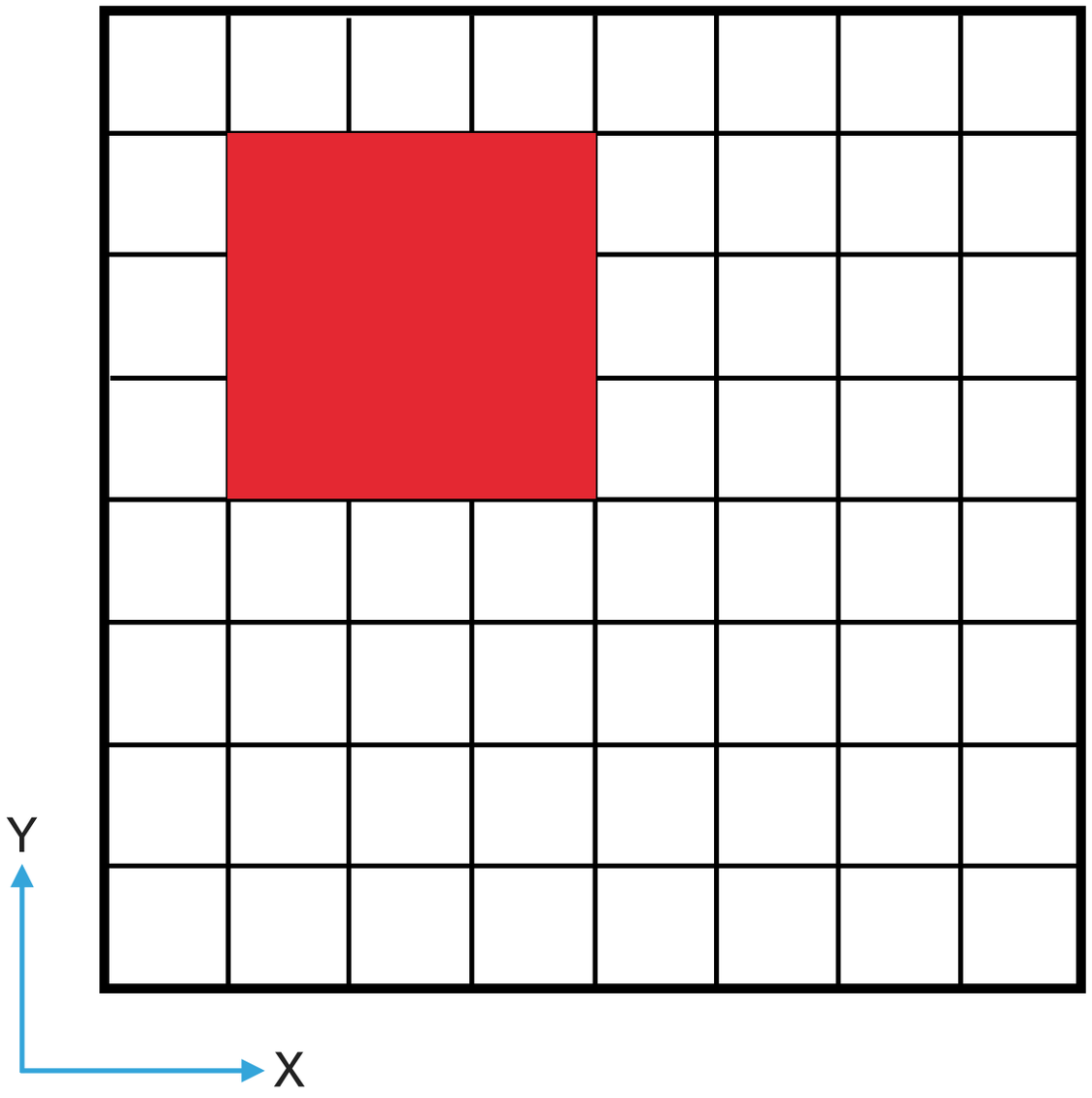
CONVOLUTION LAYERS



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

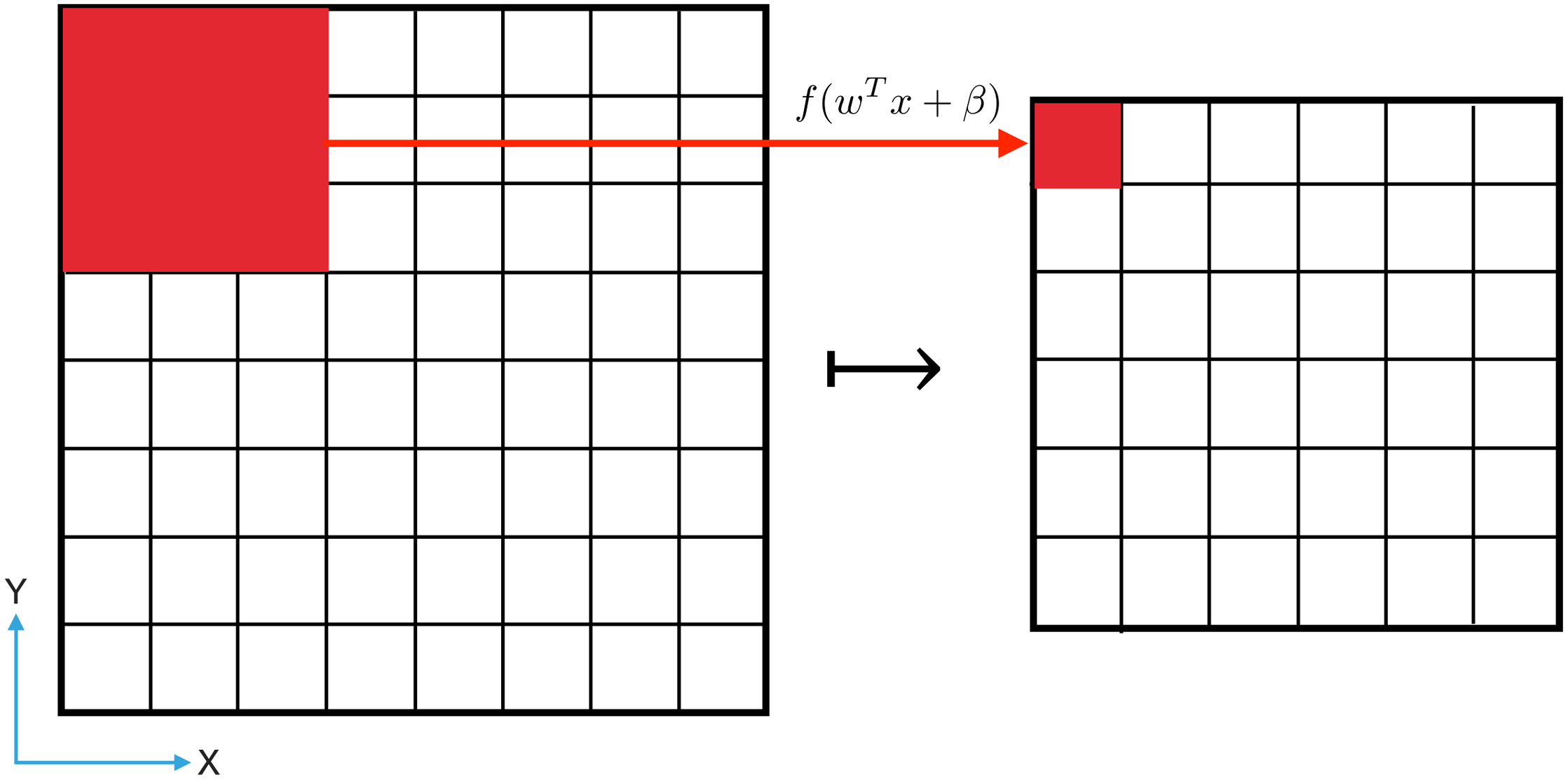


This is an 8 by 8 array of pixels, that corresponds to a 64 dimensional feature space.

- 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”.

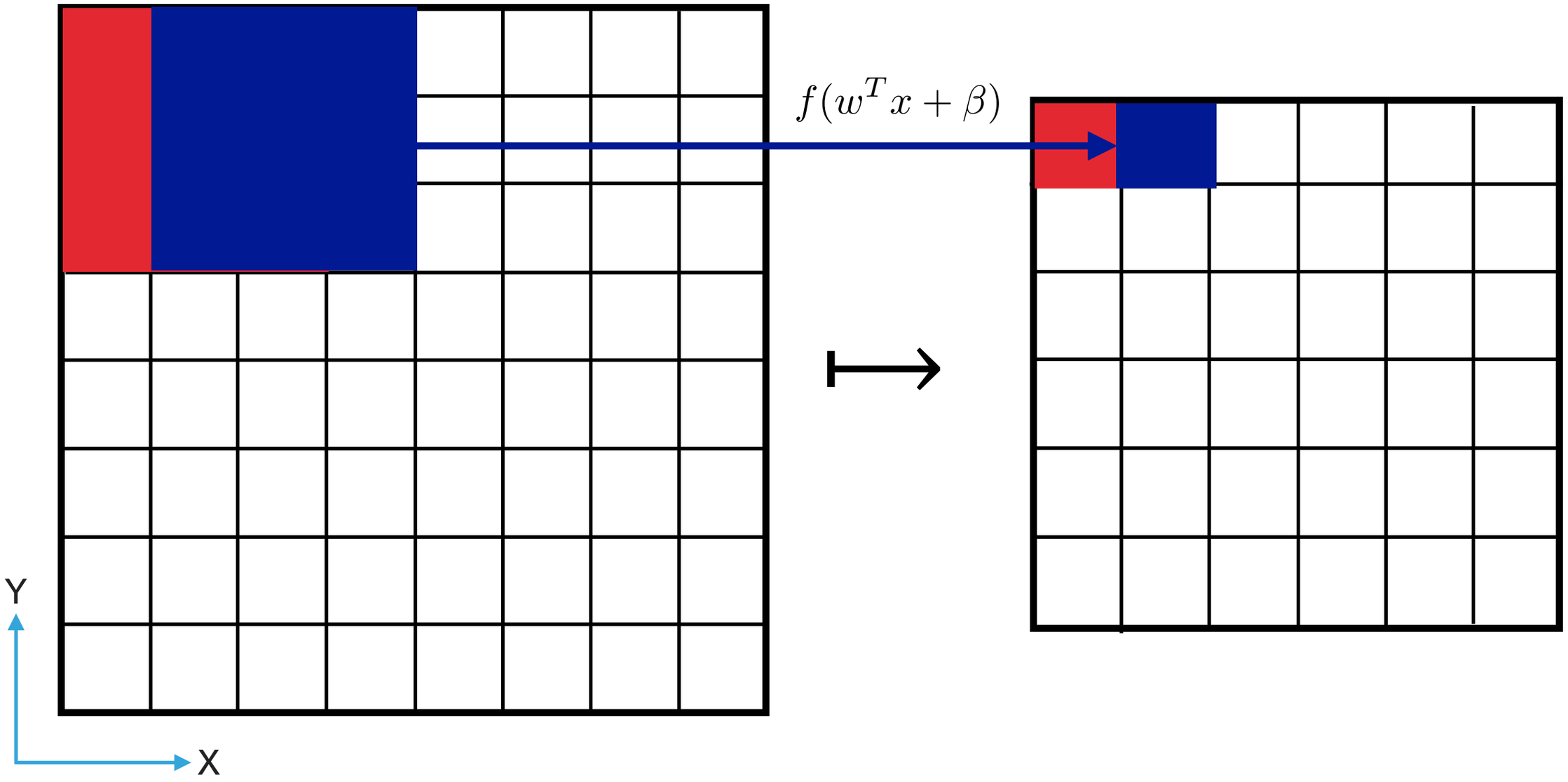
CONVOLUTION LAYERS

- If you use an M x M filter on an N x N image, the convolved image is smaller than the original.



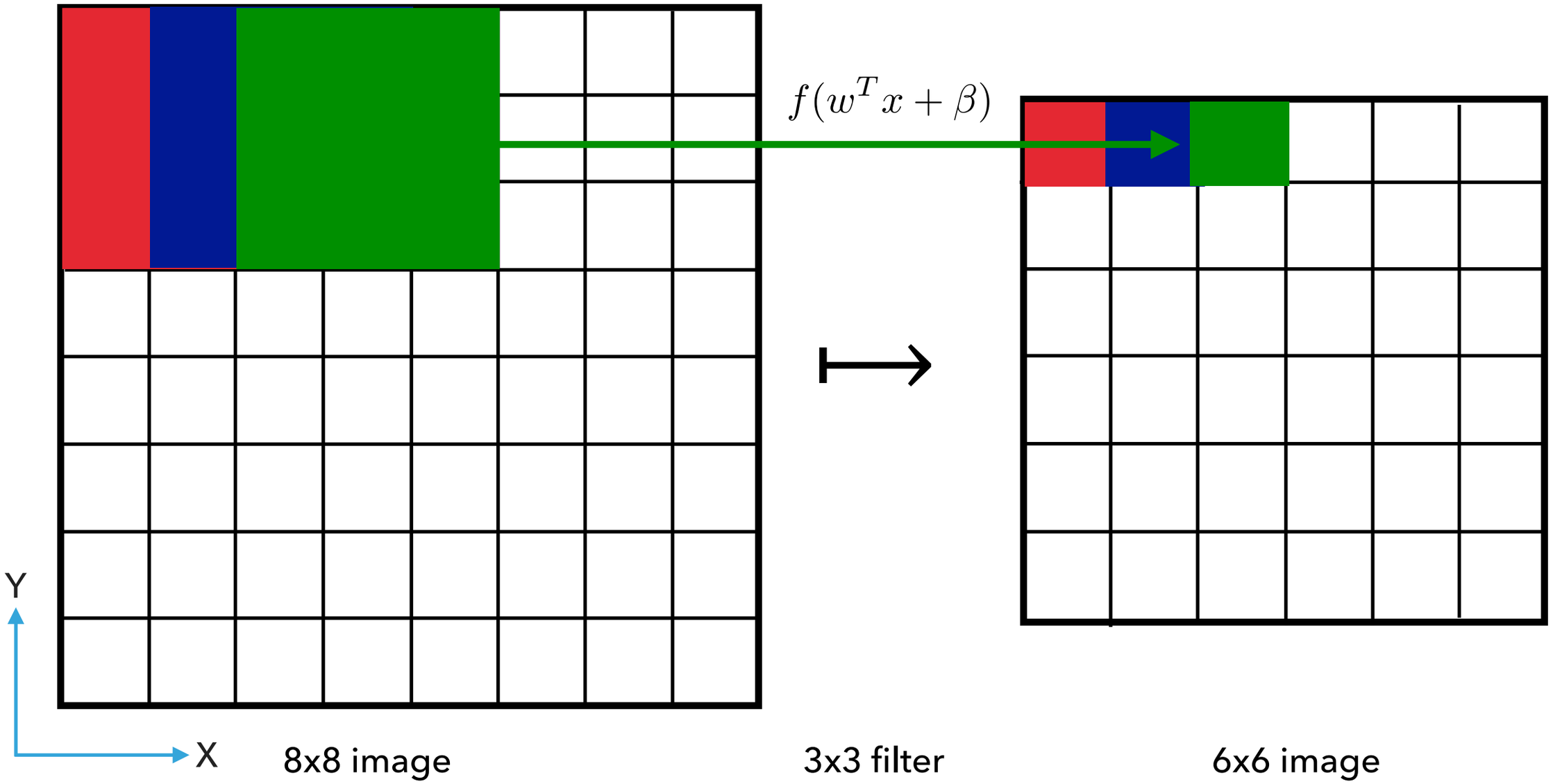
CONVOLUTION LAYERS

- If you use an M x M filter on an N x N image, the convolved image is smaller than the original.



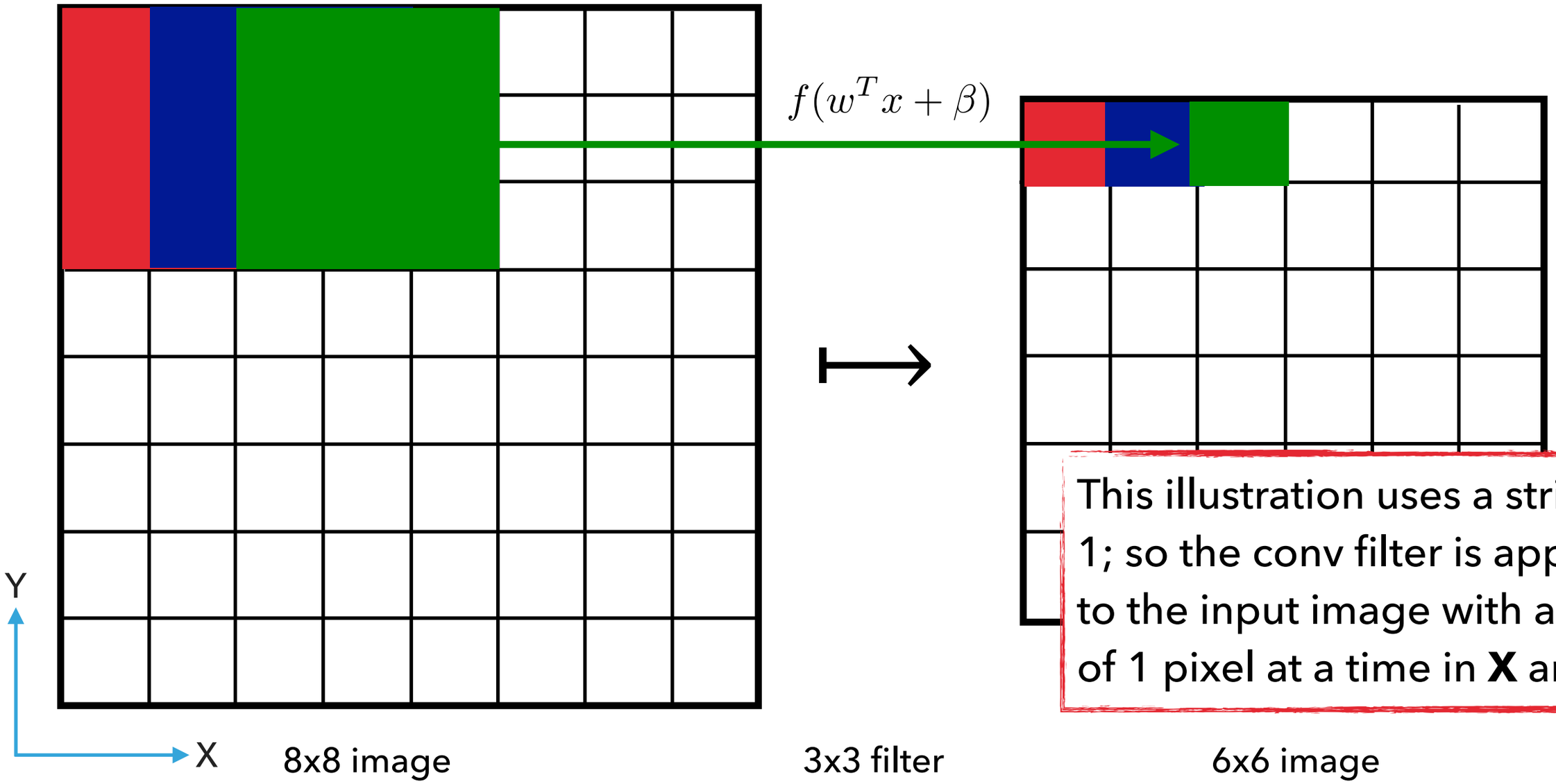
CONVOLUTION LAYERS

- If you use an M x M filter on an N x N image, the convolved image is smaller than the original.



CONVOLUTION LAYERS

- If you use an M x M filter on an N x N image, the convolved image is smaller than the original.



This illustration uses a stride of 1; so the conv filter is applied to the input image with a shift of 1 pixel at a time in **X** and **Y**.

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 NxN image becomes a (N-M+1) x (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.

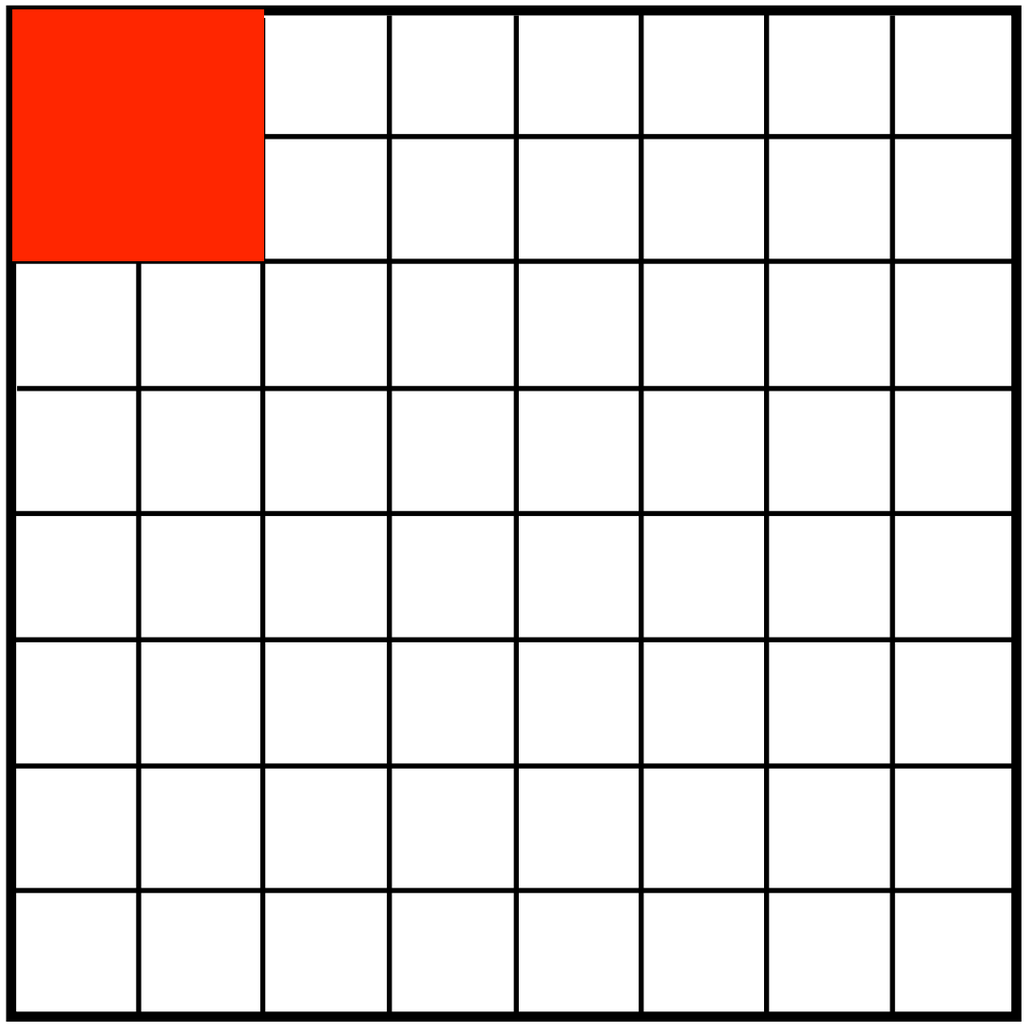
CONVOLUTION LAYERS: POOLING

- 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.

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

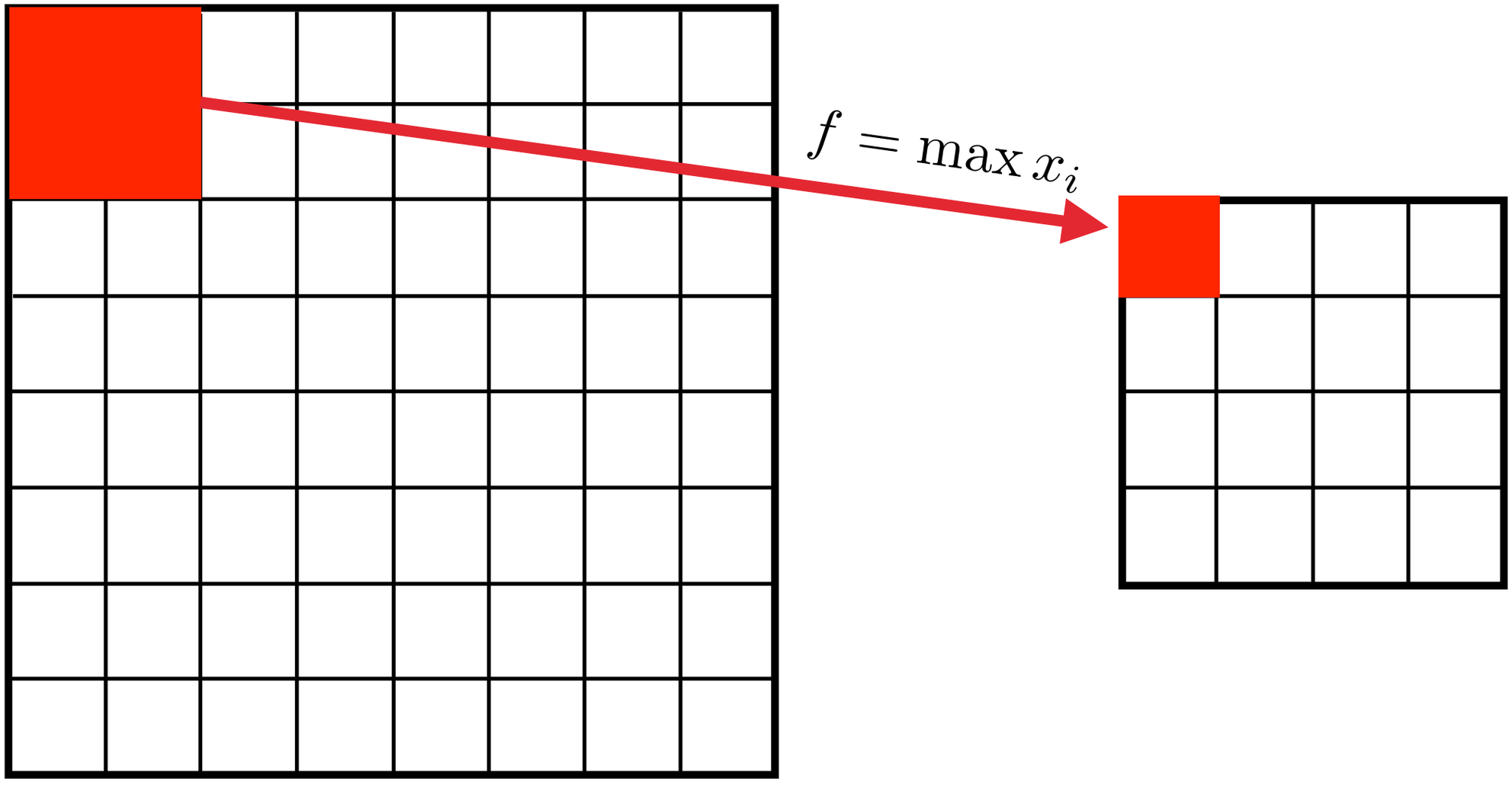
Max pooling is equivalent but taking

$$f = \max x_i$$

The output is a smaller image.

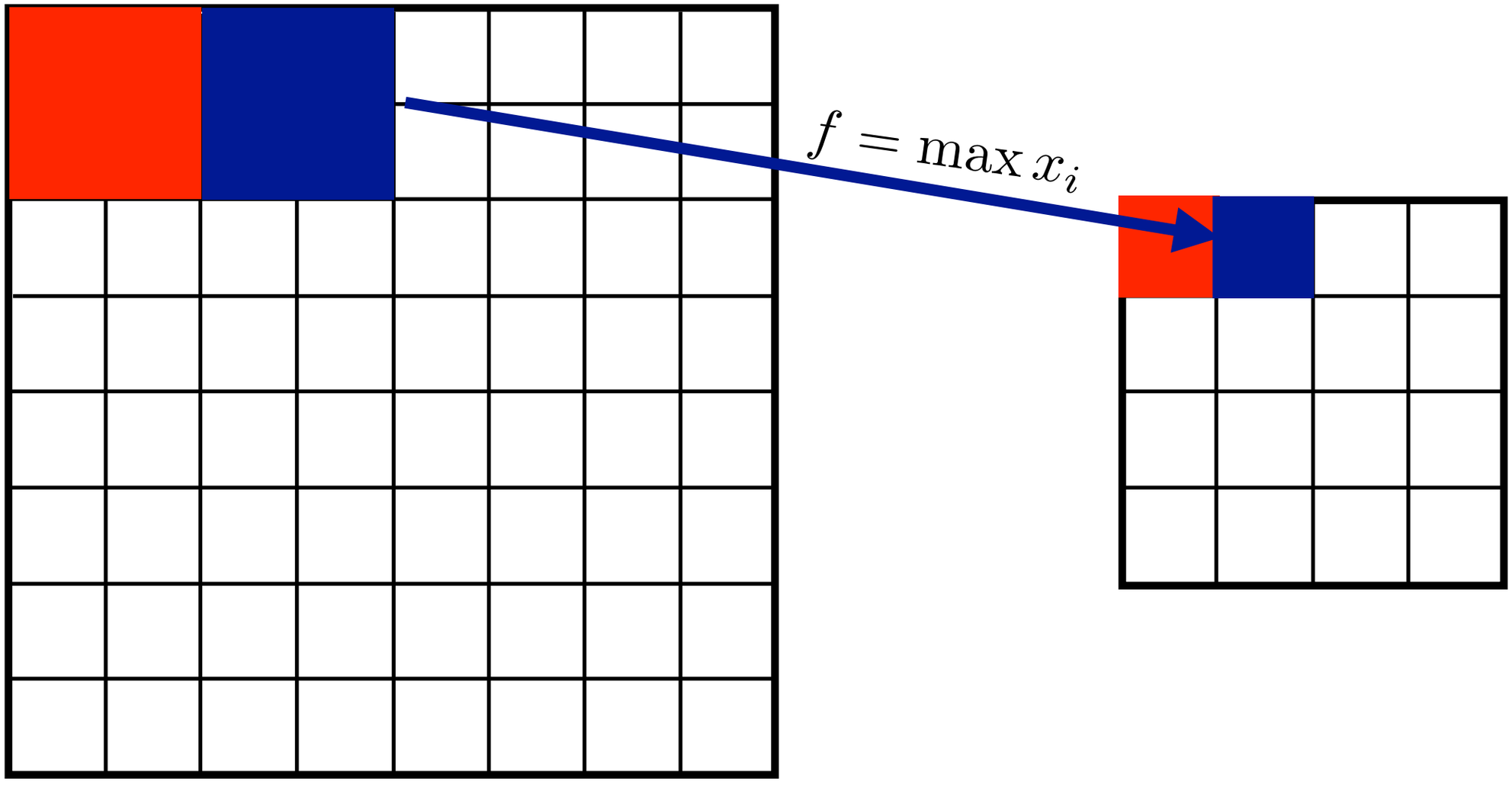
CONVOLUTION LAYERS: POOLING

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



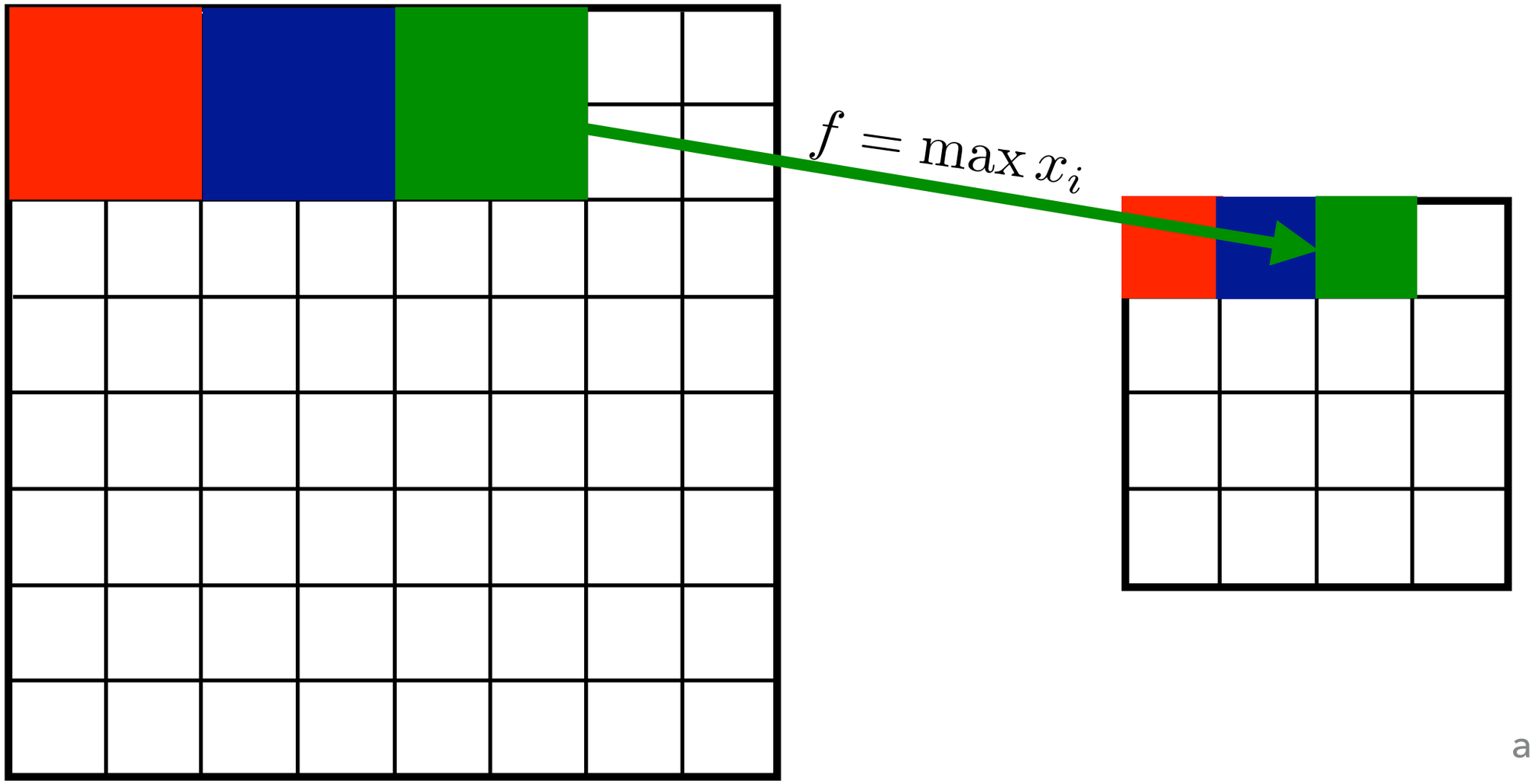
CONVOLUTION LAYERS: POOLING

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



CONVOLUTION LAYERS: POOLING

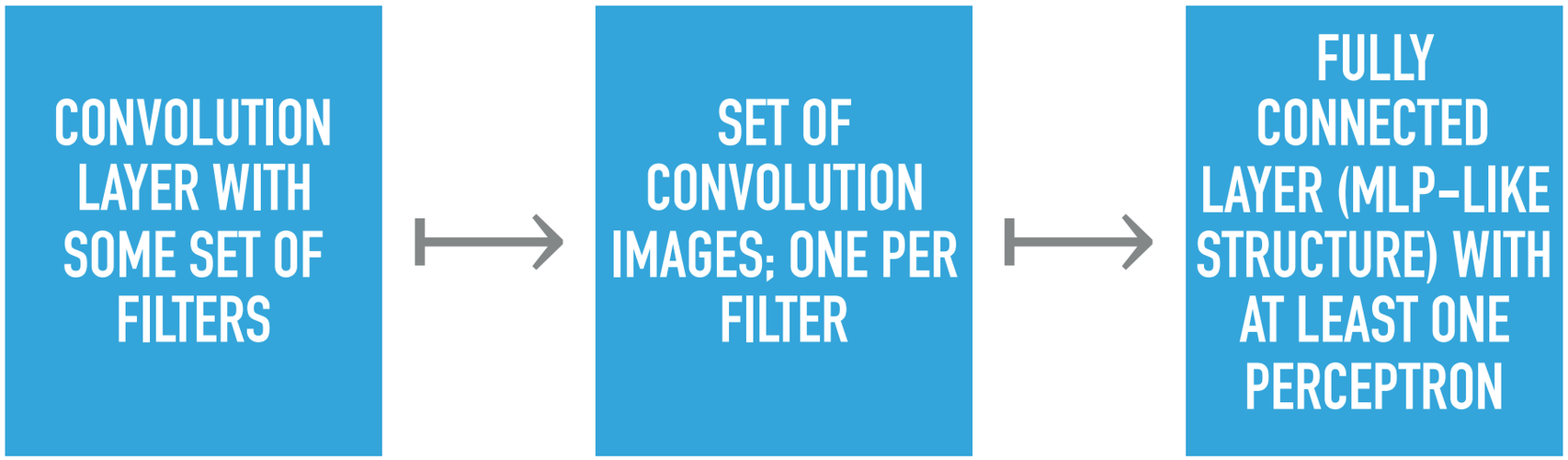
- The pooling process is applied to each individual part of the image (i.e. dimensional reduction)



and so on...

CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES

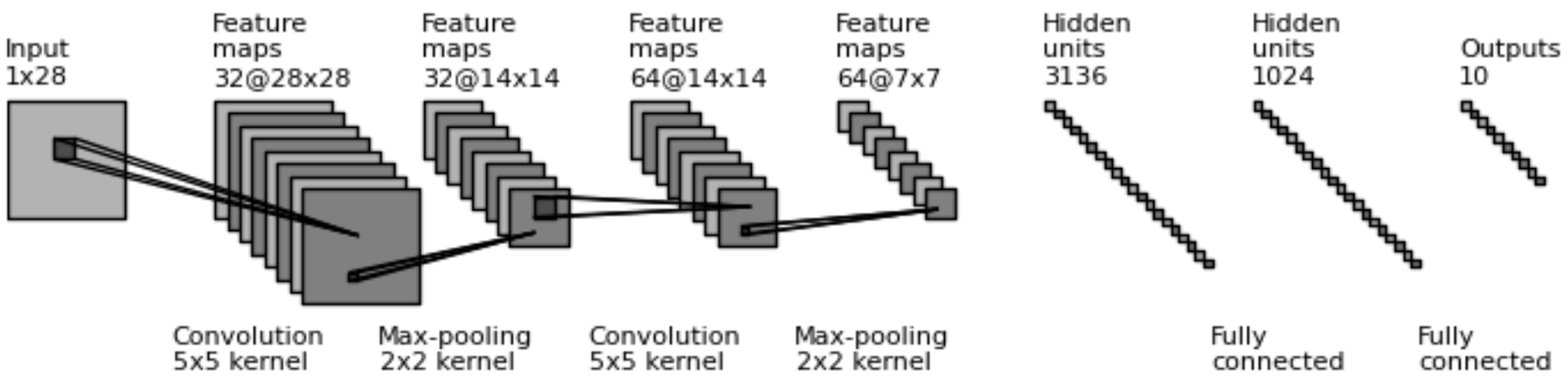
- The simplest convolutional neural network (CNN) architecture is:



- 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

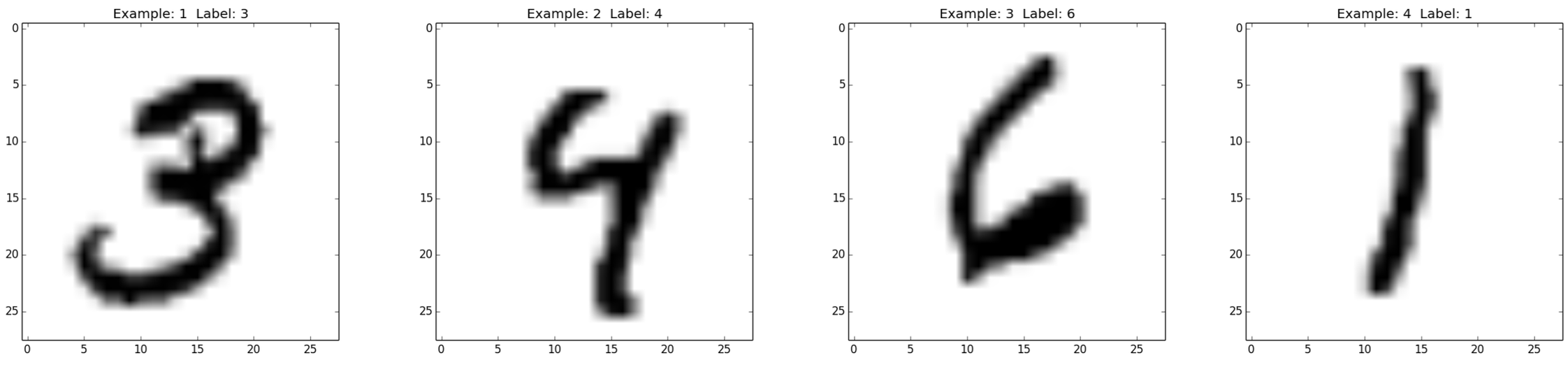
- We can include multiple convolution layers layers that may add to the information extracted from the image.
- We can add pooling layers to reduce the dimensionality.
- e.g. MNIST: handwritten numbers from 0 to 9.



- 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



[1] Neural Computation, Volume 22, Number 12, December 2010
http://yann.lecun.com/exdb/mnist/

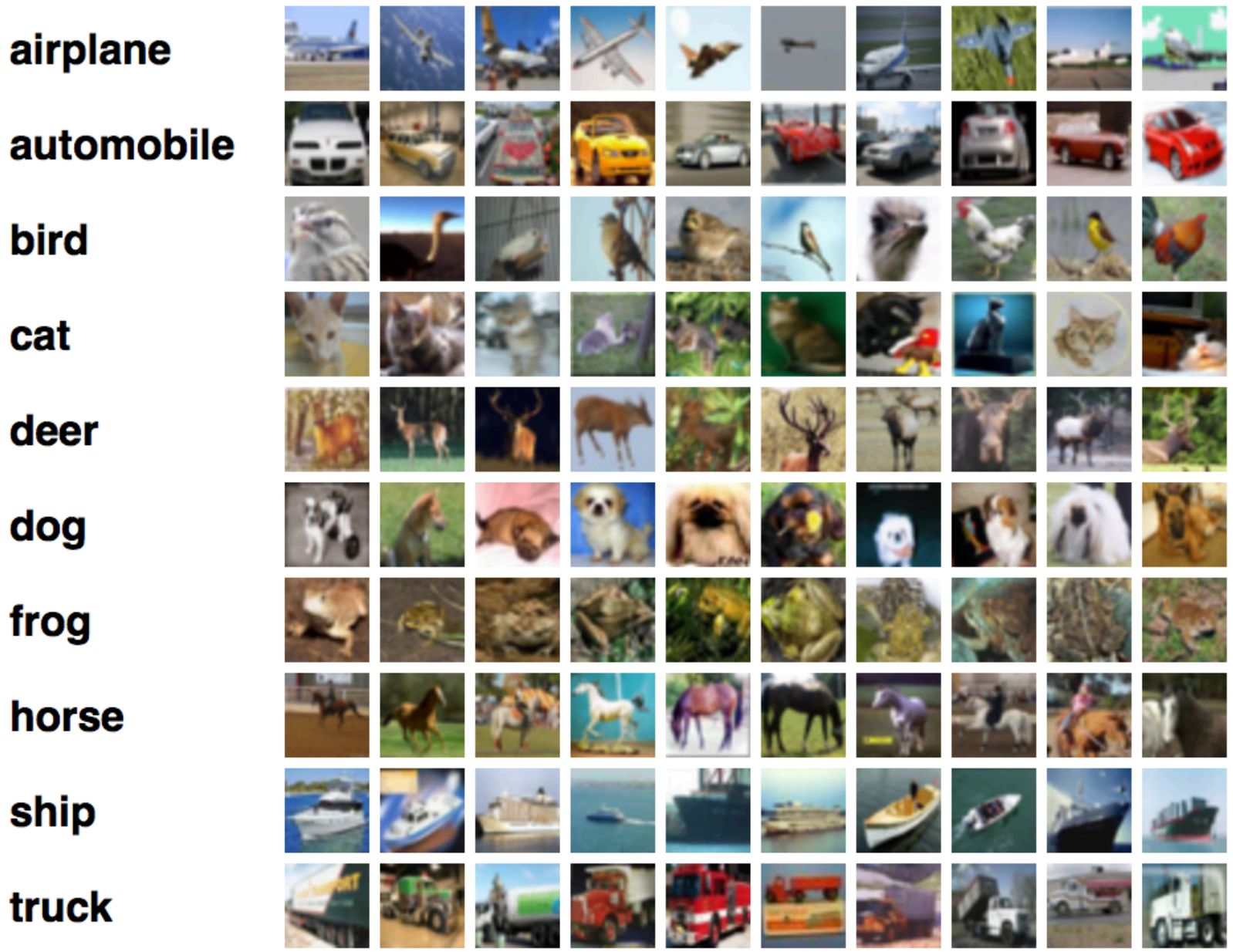
CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES & INPUT DATA

- 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. CFAR10 benchmark training set[1].

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

CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES & INPUT DATA

- CFAR 10 examples:

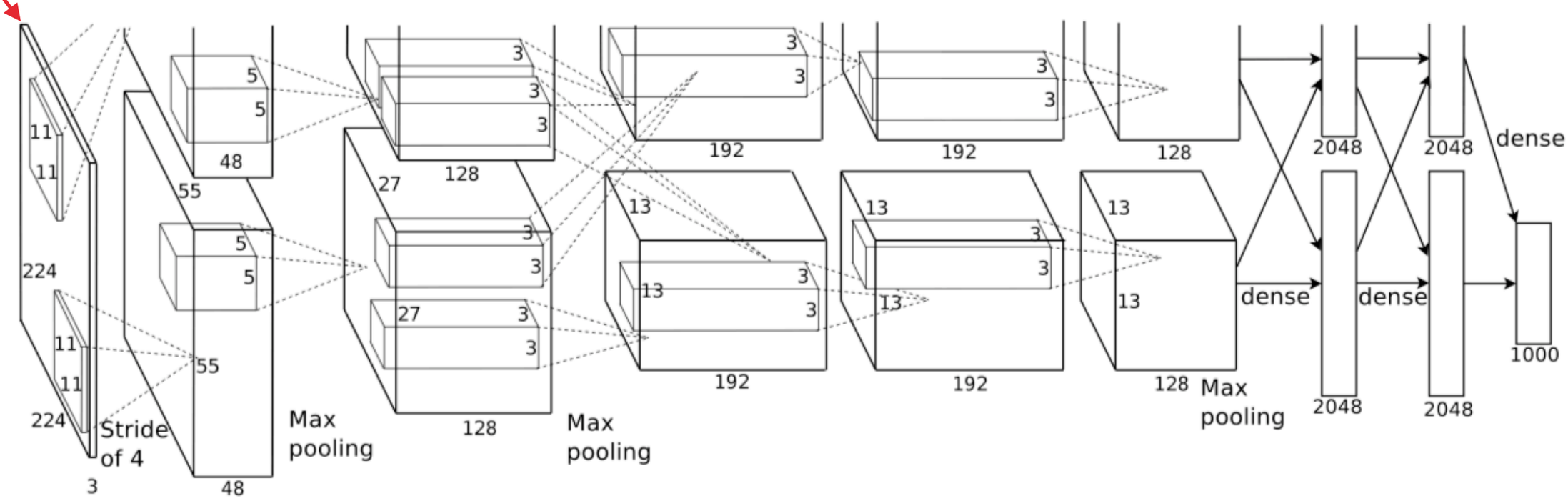


[1] https://www.cs.toronto.edu/~kriz/cifar.html

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.

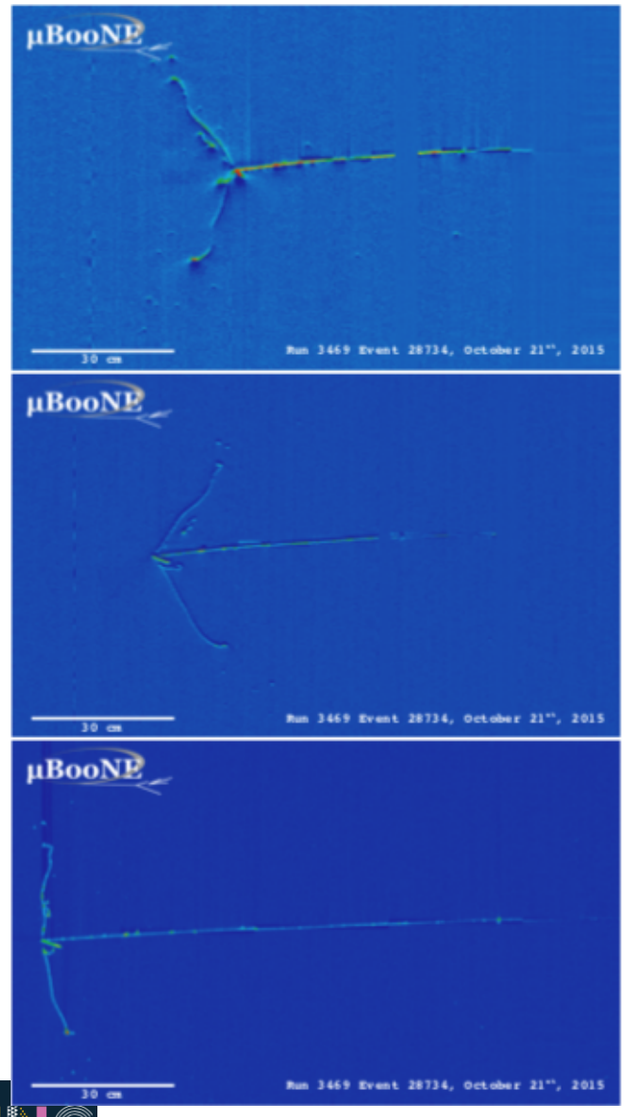
A. Krizhevsky, I. Sutskever, G. Hinton, (2017-05-24), Communications of the ACM. **60** (6): 84–90
https://dl.acm.org/doi/10.1145/3065386

CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURES & INPUT DATA

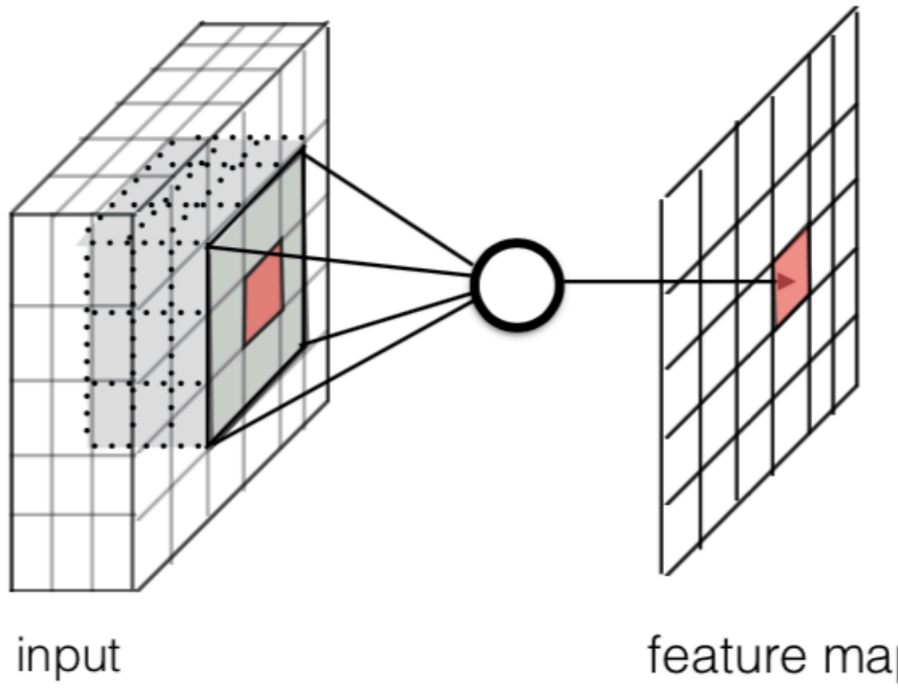
- More abstract problems can be addressed in the same way.
- Some examples are given below:
  - Transient searches can be addressed by stacking images together to form an image of depth D.
  - Tracking problems can be addressed by stacking measurement data from successive layers.
  - More arbitrary problems 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:



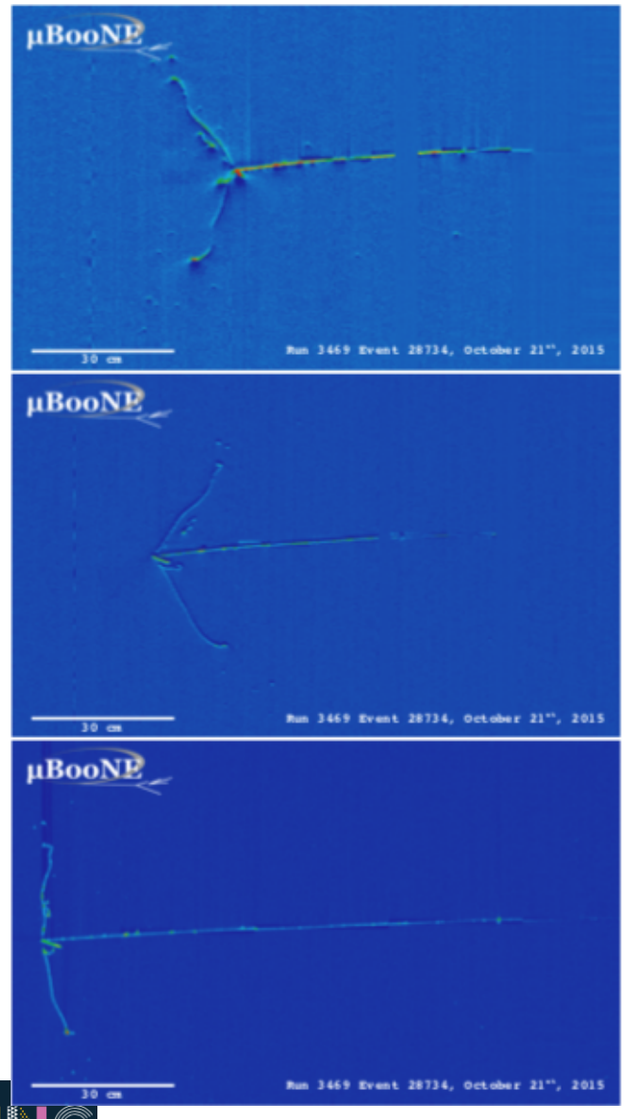
  - 3 different views of the same ν interaction.
  - Use existing software available from the web with many of the techniques discussed in these lectures.



MicroBooNE Collaboration, JINST **12** (2017) no.03, P03011

EXAMPLE: PARTICLE IDENTIFICATION AT MICROBOONE

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



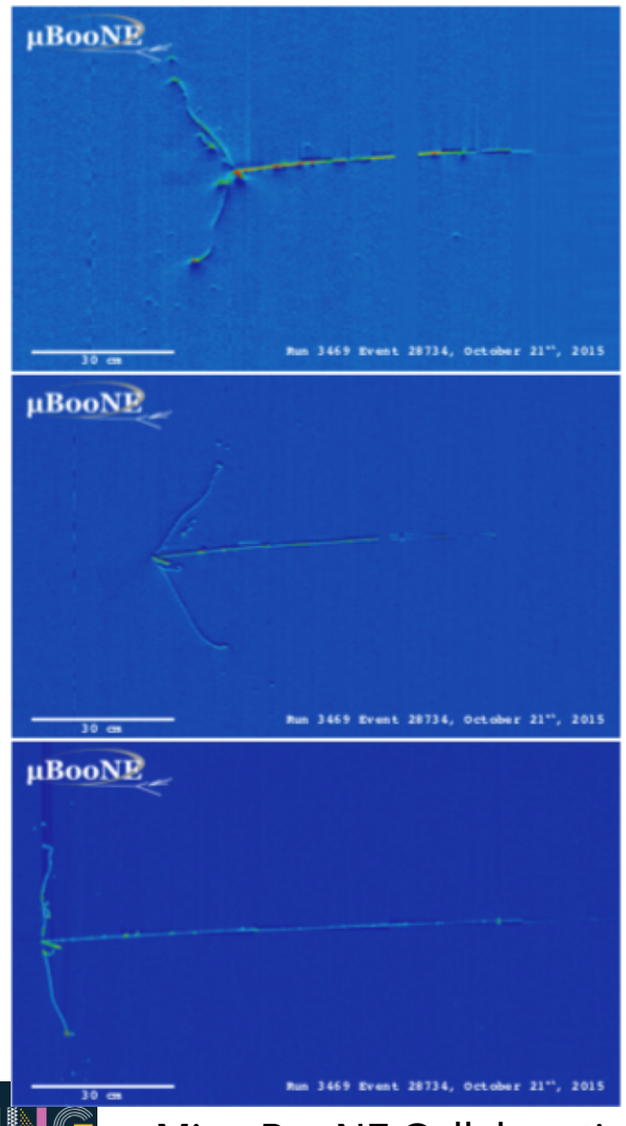
  - 3 different views of the same ν interaction.
  - Use existing software available from the web with many of the techniques discussed in these lectures.

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

MicroBooNE Collaboration, JINST **12** (2017) no.03, P03011

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.

	Classified Particle Type				
Image, Network	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

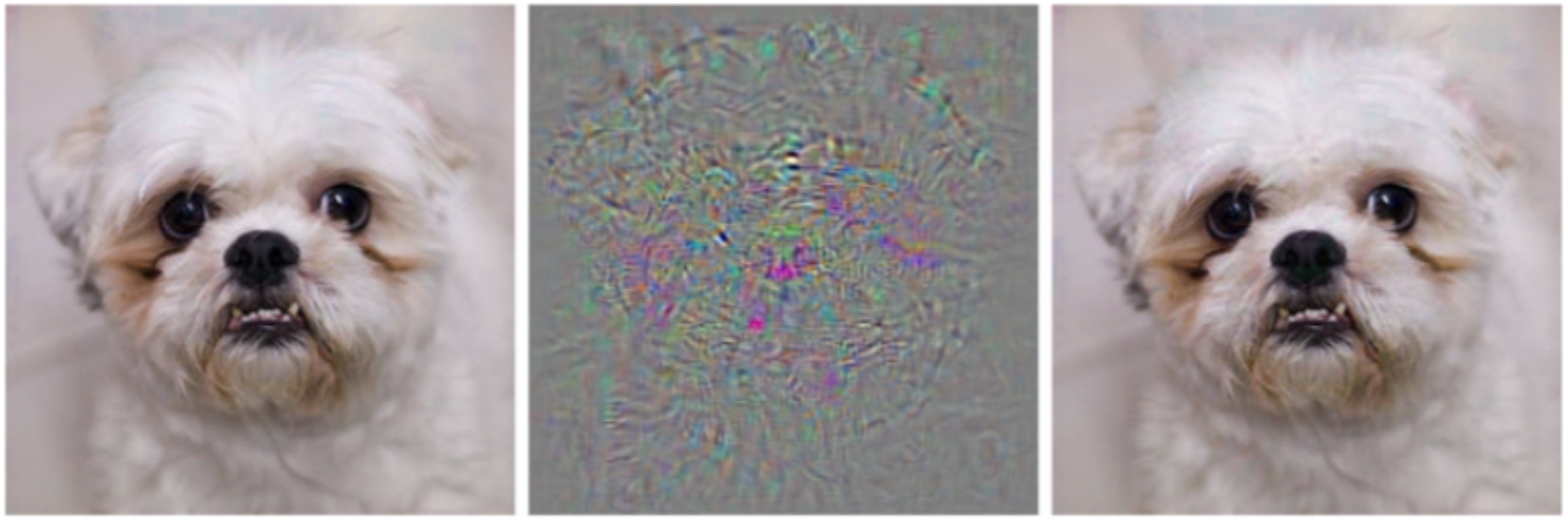
Table 2. Five particle classification performances. The very left column describes the image type and network where *HiRes* refers to a standard 576 by 576 pixel image while *LowRes* refers to a downsized image of 288 by 288 pixels. The five remaining columns denote the classification performance per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

MicroBooNE Collaboration, JINST **12** (2017) no.03, P03011

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.



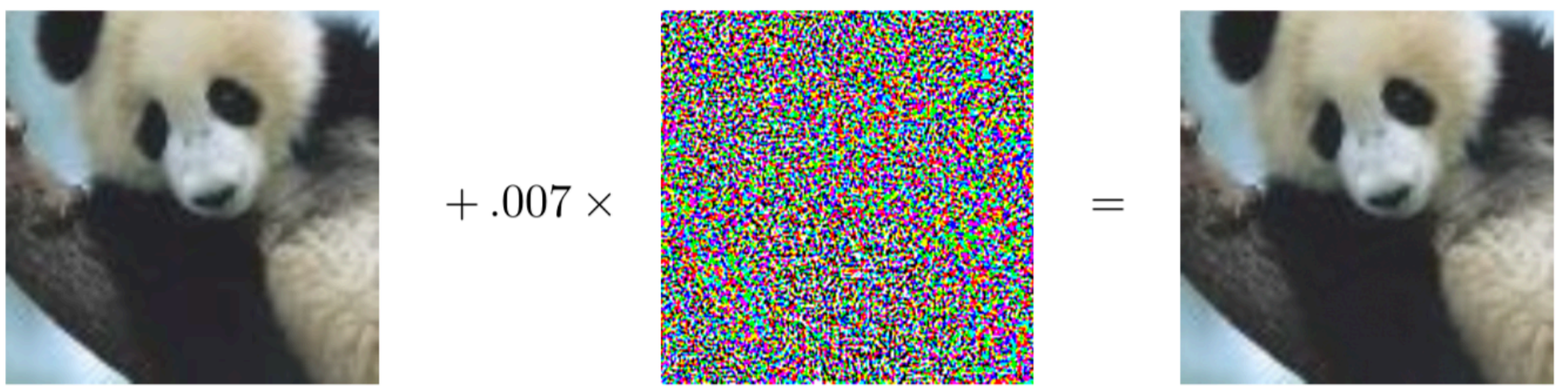
From Szegedy et al,

Correctly classified image | Perturbation of image | Incorrectly classified resultant image

Szegedy et al, ICLR, abs/1312.6199
Goodfellow et al, CoRR, abs/1412.6572, arXiv:1406.2661

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.



From Goodfellow et al,

Szegedy et al, ICLR, abs/1312.6199
Goodfellow et al, CoRR, abs/1412.6572, arXiv:1406.2661

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”.

Szegedy et al, ICLR, abs/1312.6199
Goodfellow et al, CoRR, abs/1412.6572, arXiv:1406.2661

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.

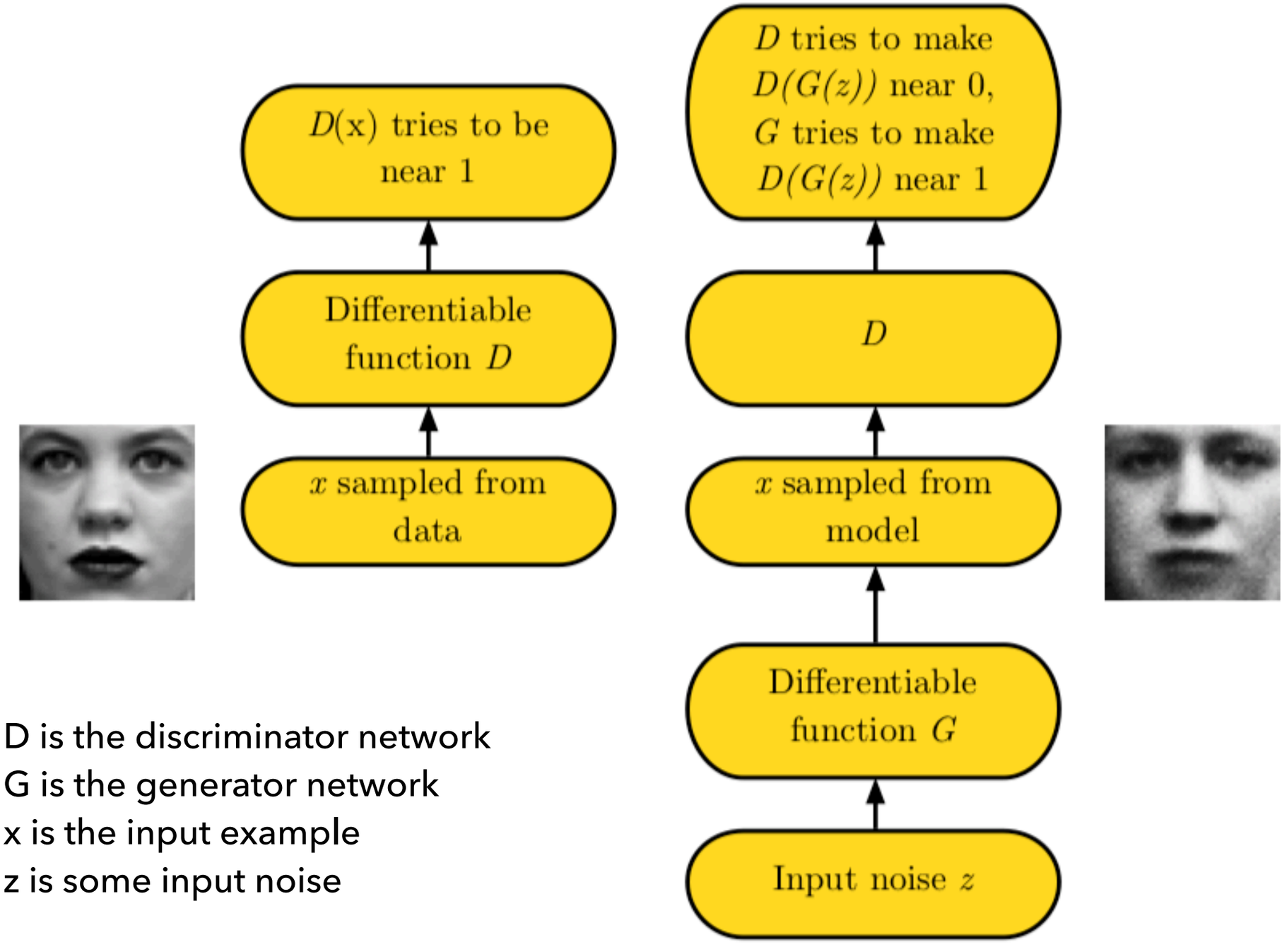
- Train two models simultaneously:
  - **G: a generative model** (the model used to generate adversarial examples for training)
  - **D: a discriminative model** (the model used to make a prediction that an example is either data or from the generative model)
  - Train D to maximise the rate of correct 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
Goodfellow et al, CoRR, abs/1412.6572, arXiv:1406.2661 also see Goodfellow's Neural Information Processing Systems proceedings on Generative Adversarial Networks: arxiv: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.

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

- 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).

Goodfellow's Neural Information Processing Systems proceedings: arxiv:1701.00160.

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:

Karras et al., ICLR 2018 arXiv:1710.10196