Text Mining and Language Analytics

Lecture 5

Feed-forward Neural Networks & Convolutional Neural Networks

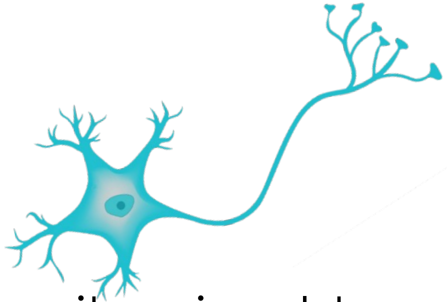
Dr Stamos Katsigiannis

2023-24

Durham University

Artificial Neural Networks (ANN)

- Simply called **Neural Networks (NN)**
- **Algorithms that vaguely mimic the operations of a biological brain to recognise relationships between data**
- Based on a collection of connected units (nodes) called artificial neurons
- Artificial neurons loosely model the neurons in a biological brain

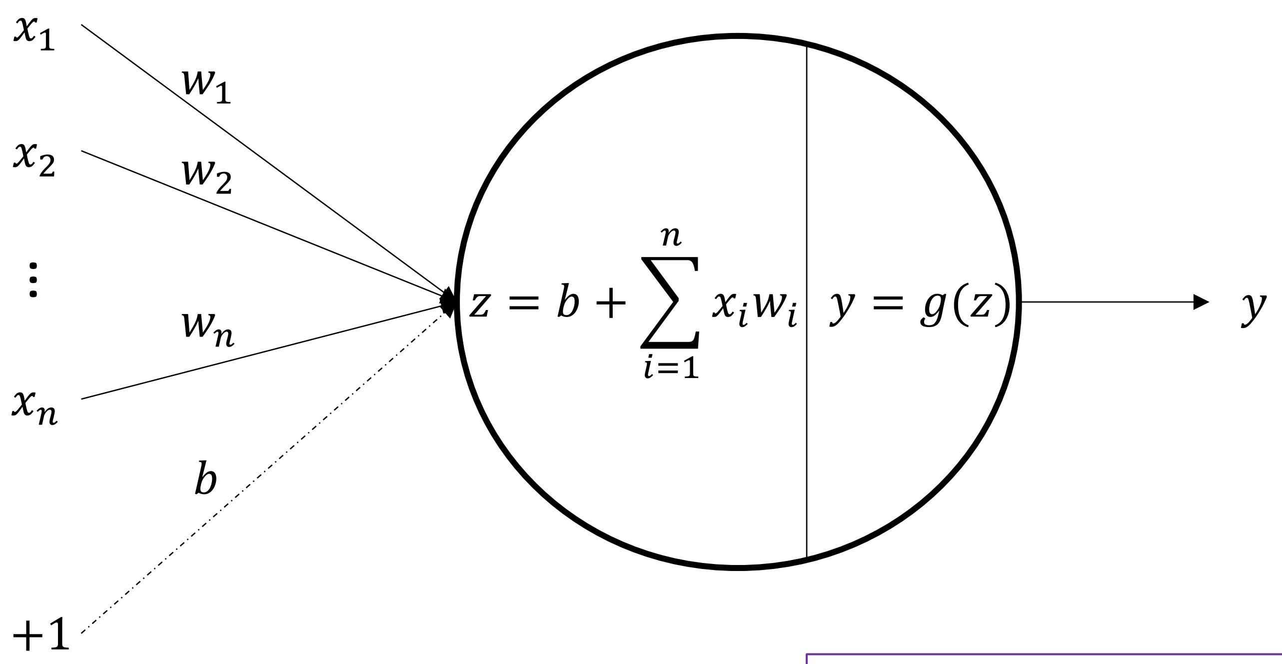


- Neural network connections, like the synapses in a biological brain, can transmit a signal to other neurons
- Each neuron takes some numbers as input and outputs the result of a function of the sum of the inputs

Durham University

Perceptron: The building block of NNs

Input **Weights** **Output**



The bias term is optional

$$y = g\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

Activation functions $g(z)$

$$sigmoid(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

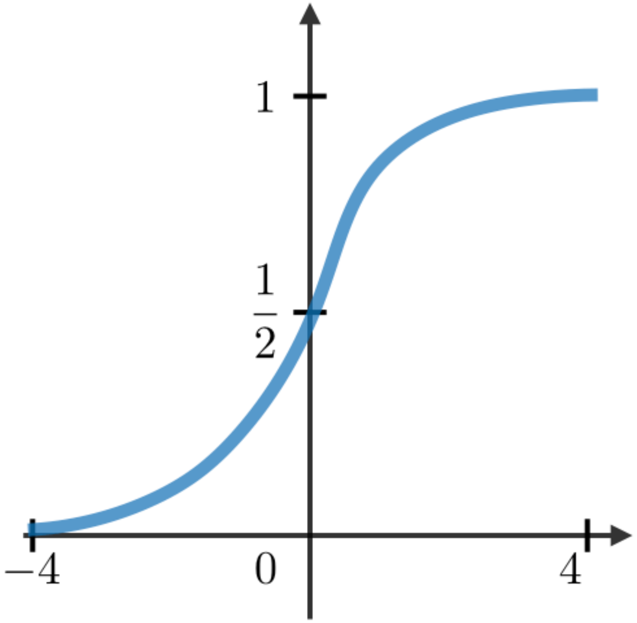
$$ReLU(z) = \max(0, z)$$

$$Leaky\ ReLU(z) = \begin{cases} z & z > 0 \\ 0.01z & \text{otherwise} \end{cases}$$

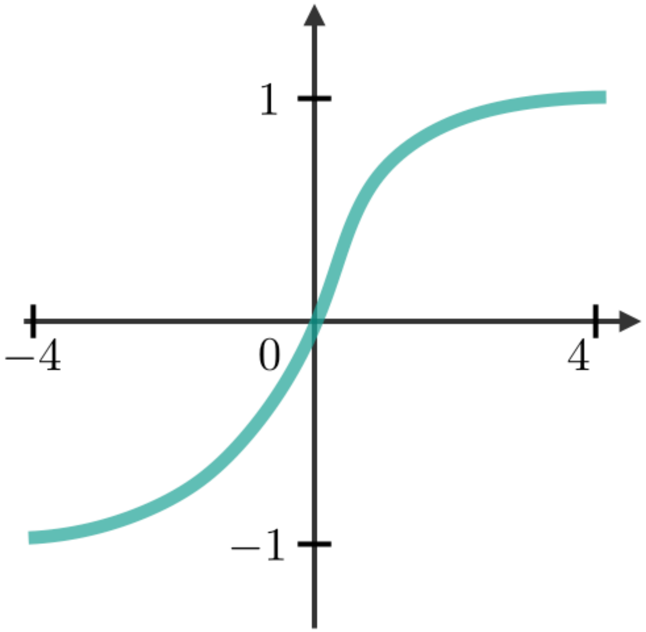
…

Activation functions

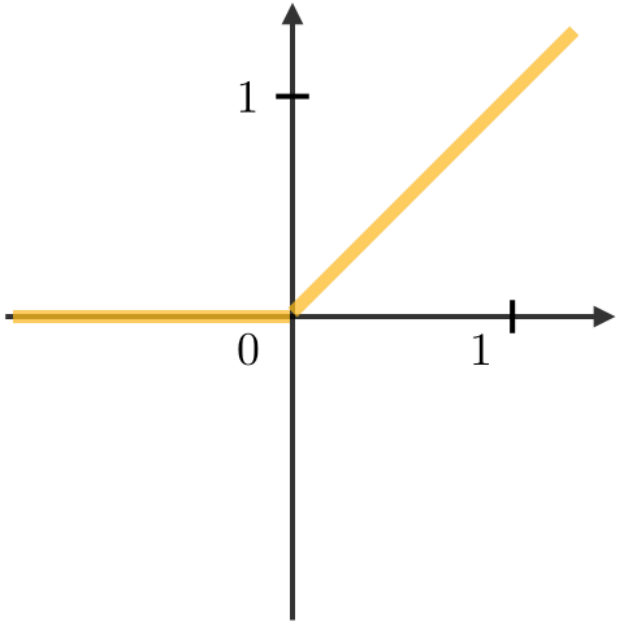
Sigmoid



tanh

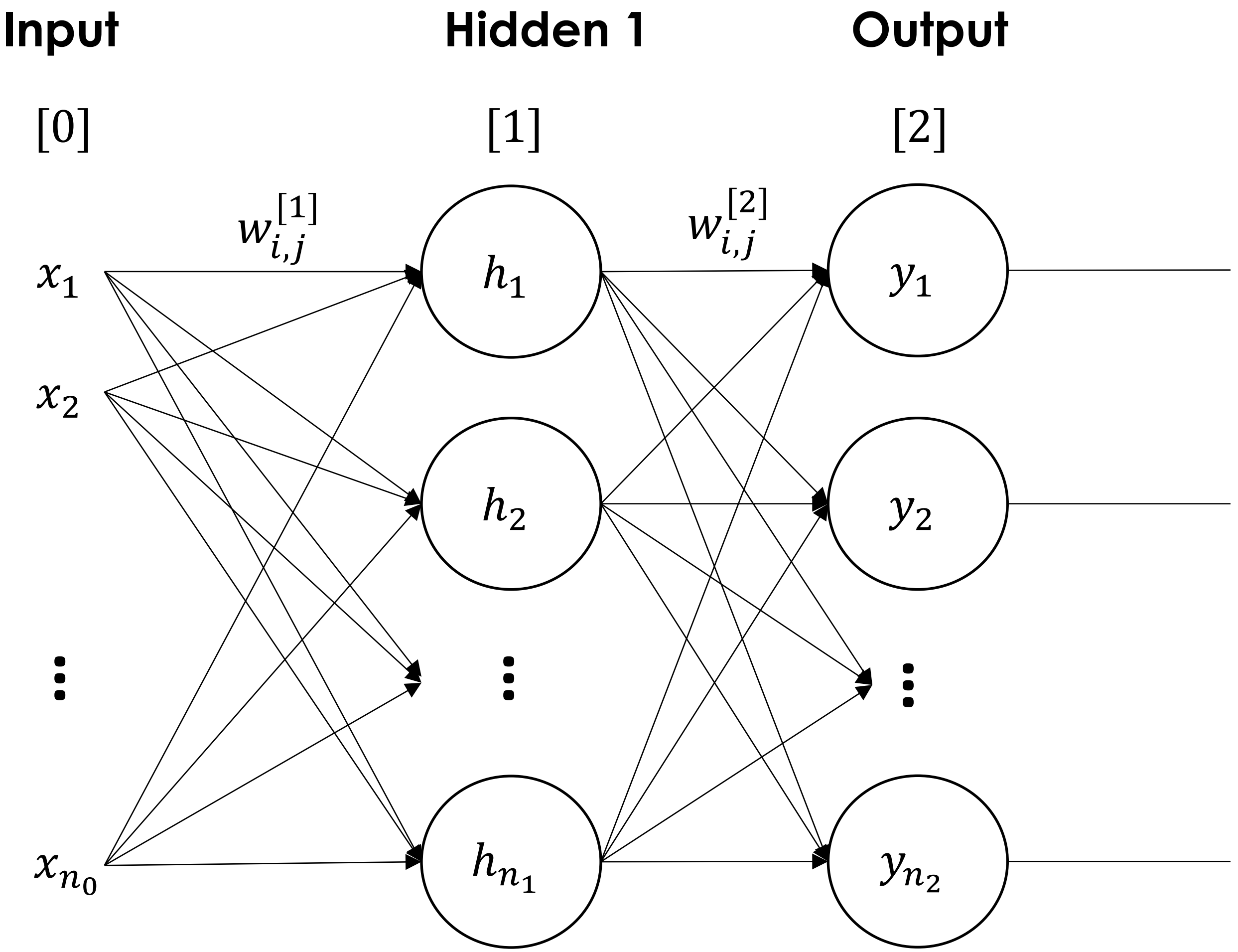


ReLU



Durham University

Feed-Forward Neural Networks (I)



$$z_j^{[1]} = \sum_{i=1}^{n_0} x_i w_{i,j}^{[1]}$$

$$h_j = g(z_j^{[1]})$$

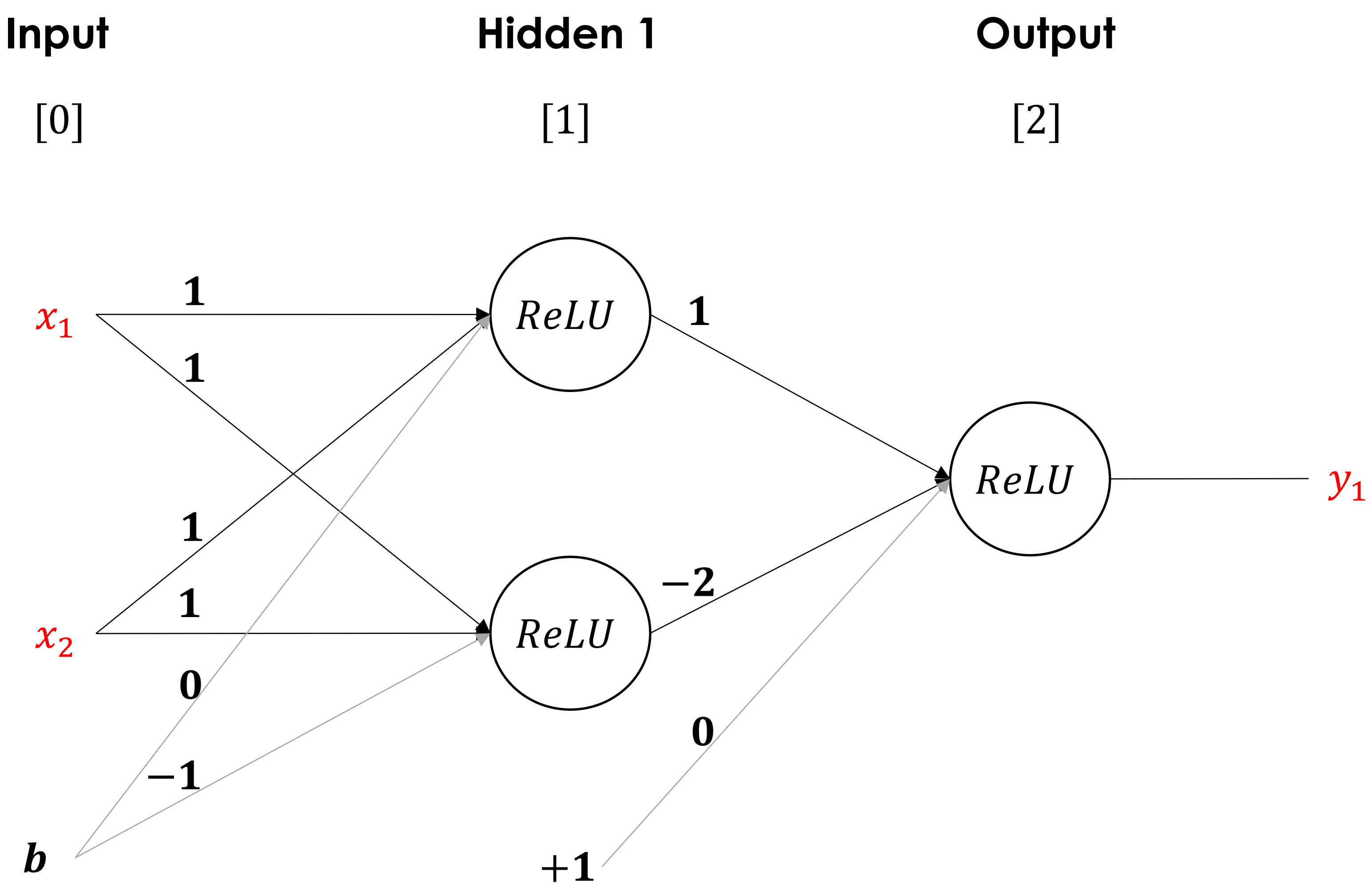
$$z_j^{[2]} = \sum_{i=1}^{n_1} h_i w_{i,j}^{[2]}$$

$$y_j = g(z_j^{[2]})$$

Use *softmax* to convert output to class probabilities

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}}$$

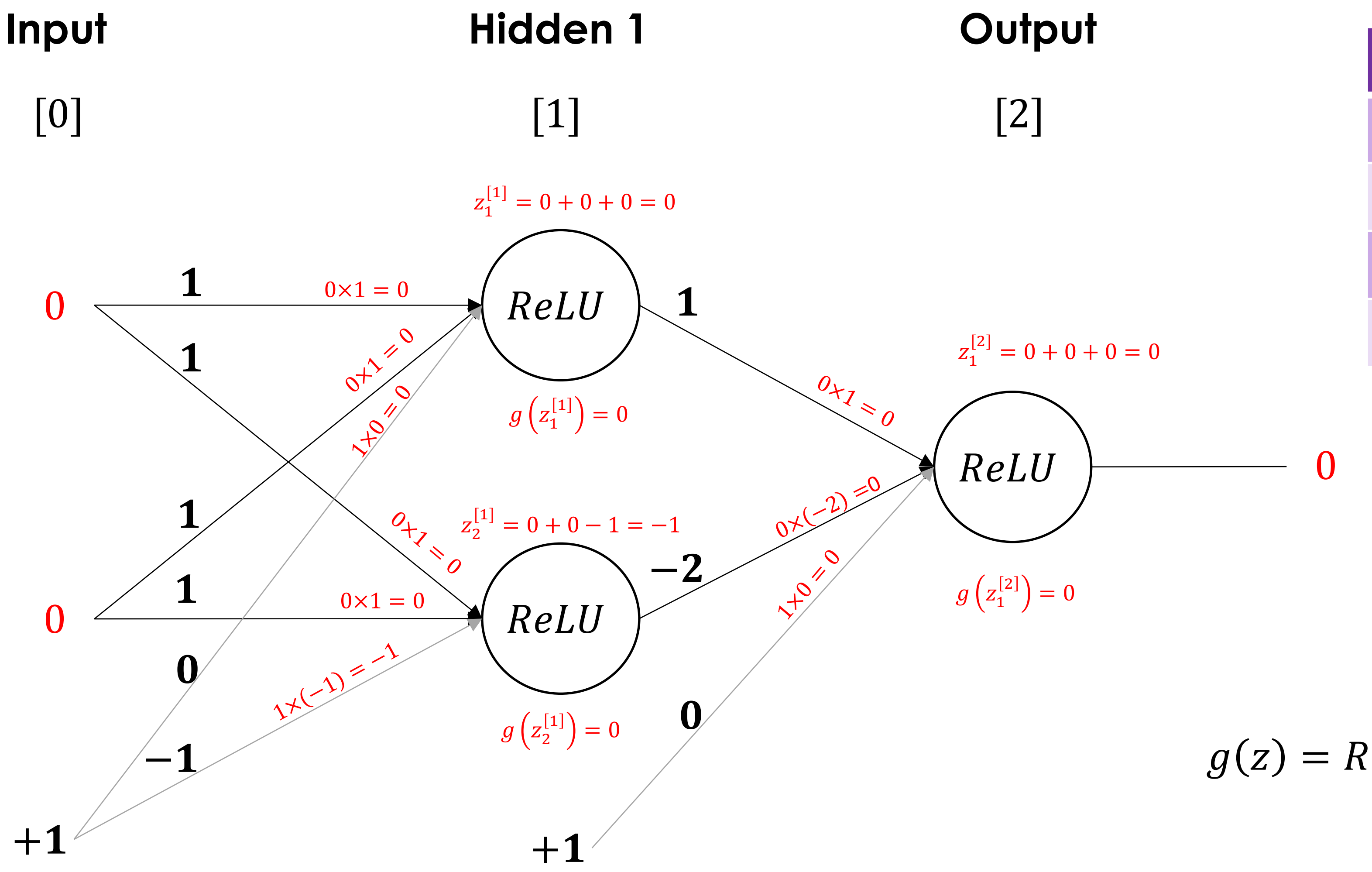
Feed-Forward Neural Networks (II)



x_1	x_2	y_1

Durham University

Feed-Forward Neural Networks (II)

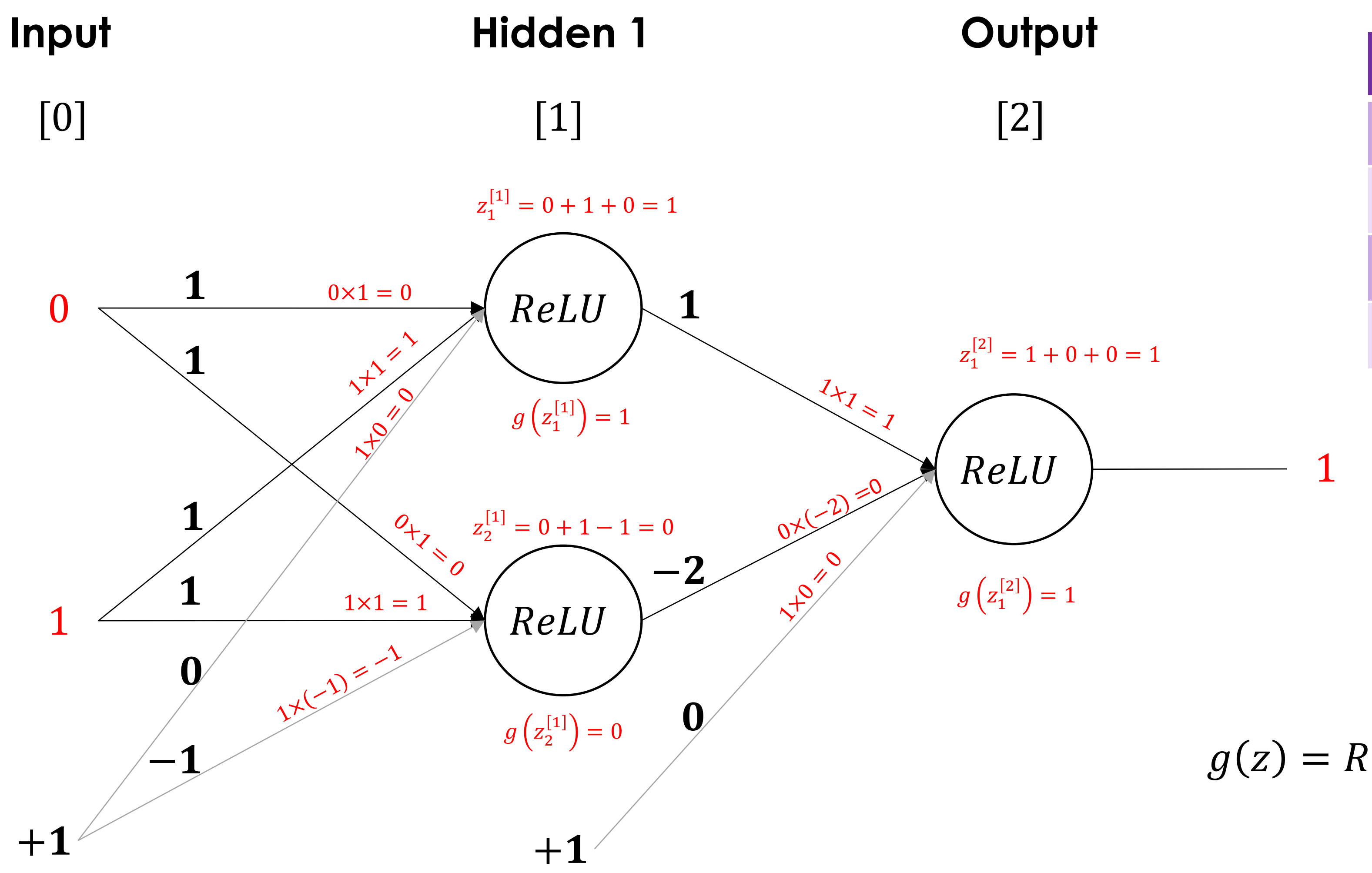


x_1	x_2	y_1
0	0	**0**

$$g(z) = ReLU(z) = \max(0, z)$$

Durham University

Feed-Forward Neural Networks (II)

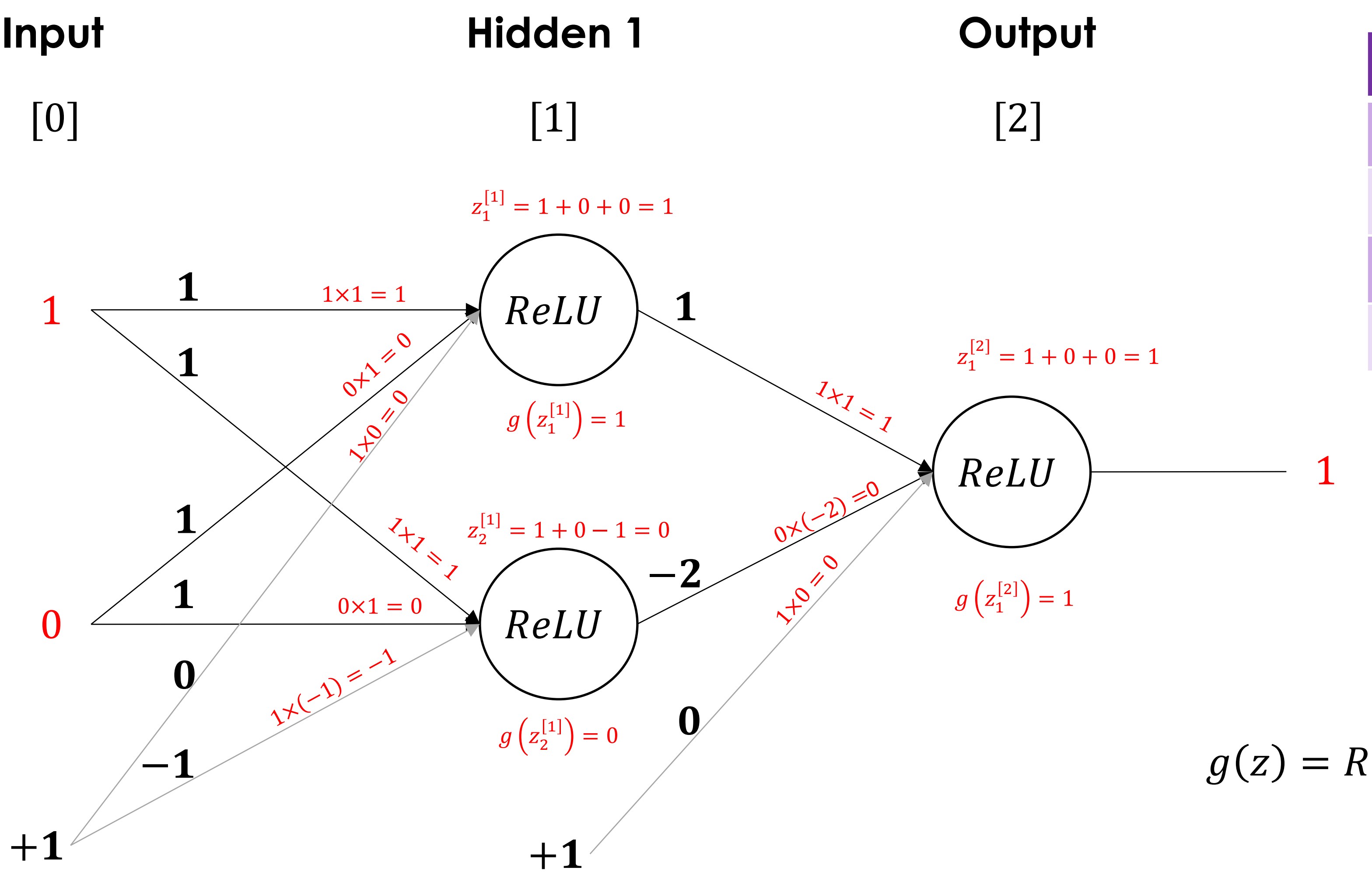


x_1	x_2	y_1
0	0	**0**
0	1	**1**

$$g(z) = ReLU(z) = \max(0, z)$$

Durham University

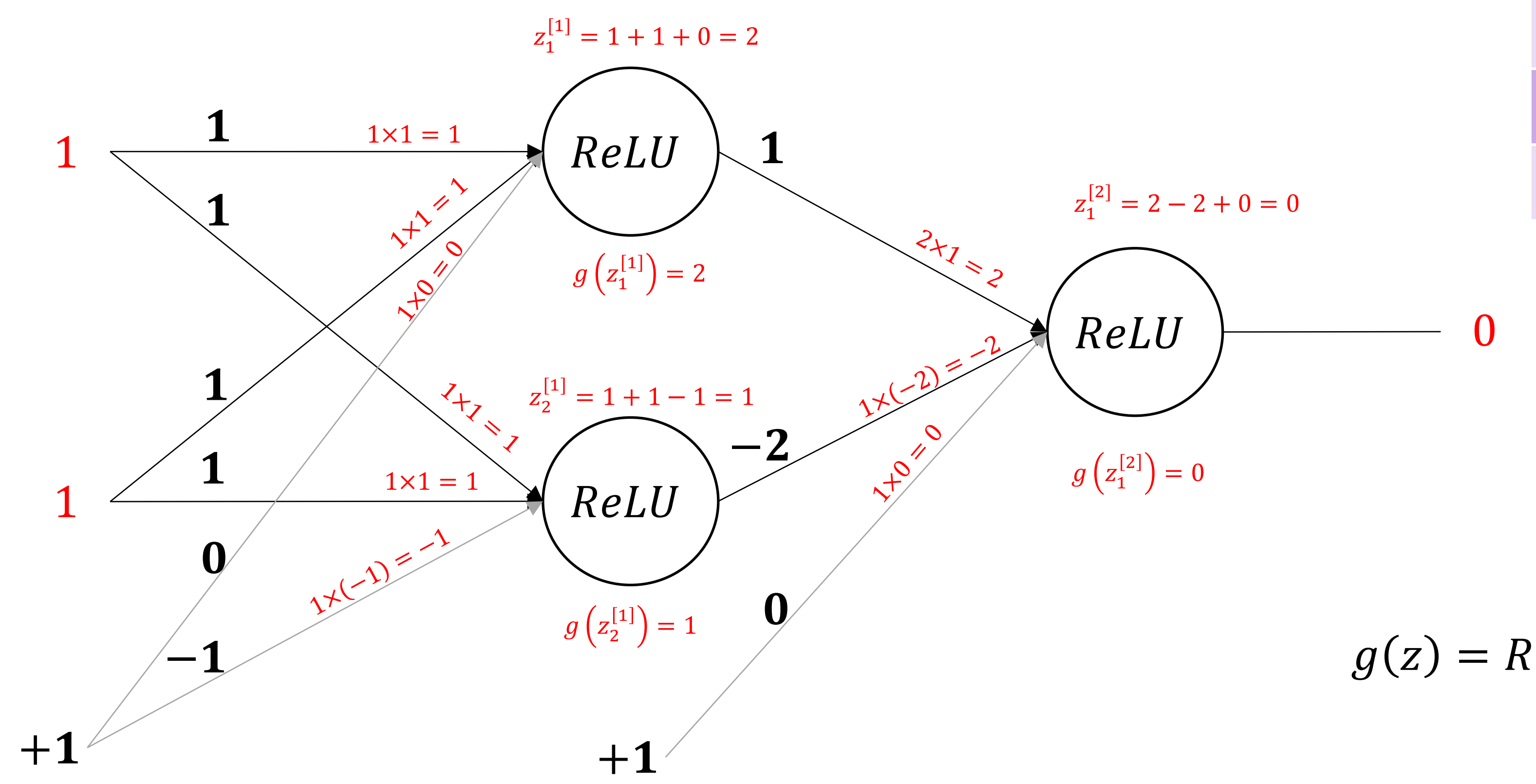
Feed-Forward Neural Networks (II)



x_1	x_2	y_1
0	0	**0**
0	1	**1**
1	0	**1**

$$g(z) = ReLU(z) = \max(0, z)$$

Feed-Forward Neural Networks (II)

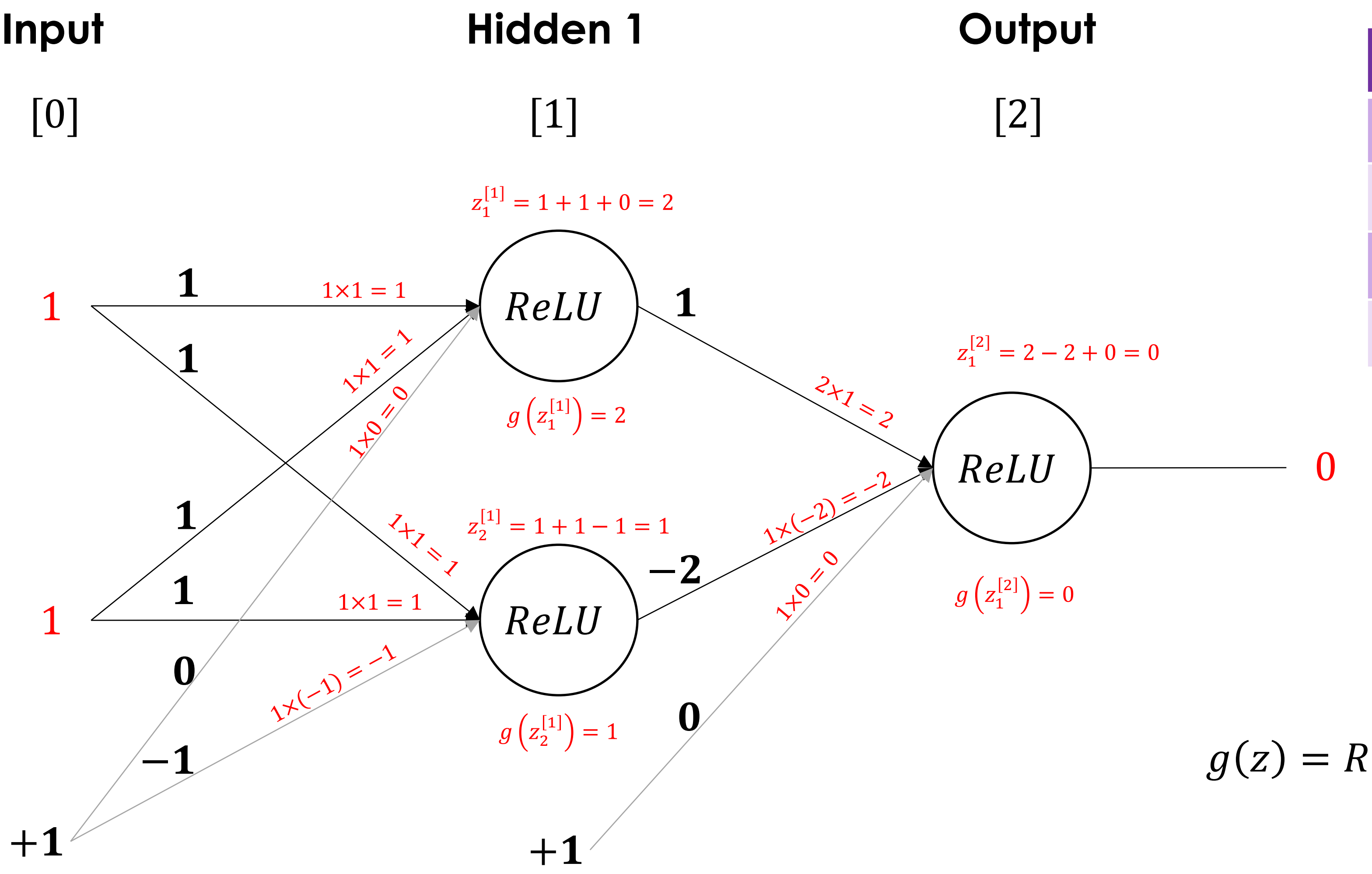


x_1	x_2	y_1
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

$$g(z) = ReLU(z) = \max(0, z)$$

Durham University

Feed-Forward Neural Networks (II)



x_1	x_2	y_1
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

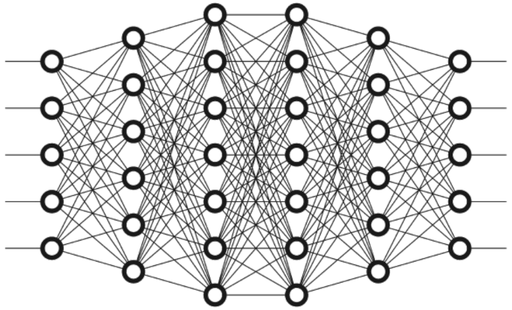
It's a XOR gate!

$$g(z) = ReLU(z) = \max(0, z)$$

Durham University

Feed-Forward Neural Networks (III)

- The number of neurons in the output layer is typically equal to the number of classes (classification)

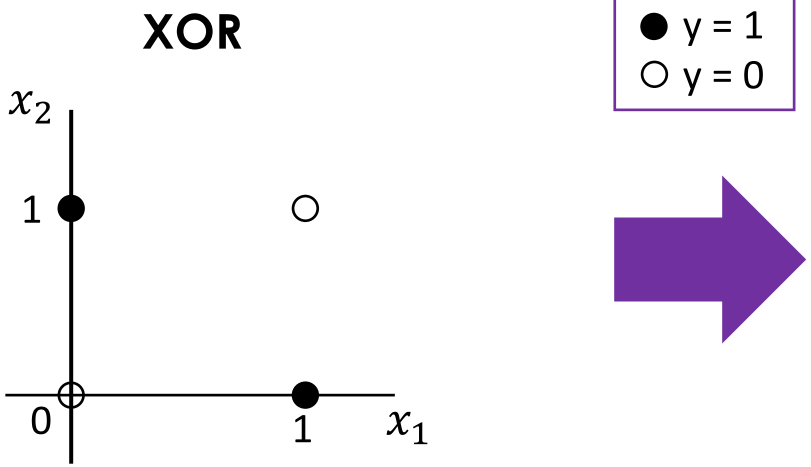


- Can have multiple hidden layers of any size (trial and error?)
- The more layers a NN has, the deeper it is considered
- **Intuition behind NN neurons and layers**
  - At the first hidden layer, each neuron will create a linear decision boundary between classes
  - At the next layer, the combination of the linear decision boundaries by a neuron will create a non-linear decision boundary
  - The non-linear decision boundaries will be combined in the next layer and create even more complex decision boundaries
- Each layer creates a new representation of its input
  - Problem may be linearly separable using the new representation

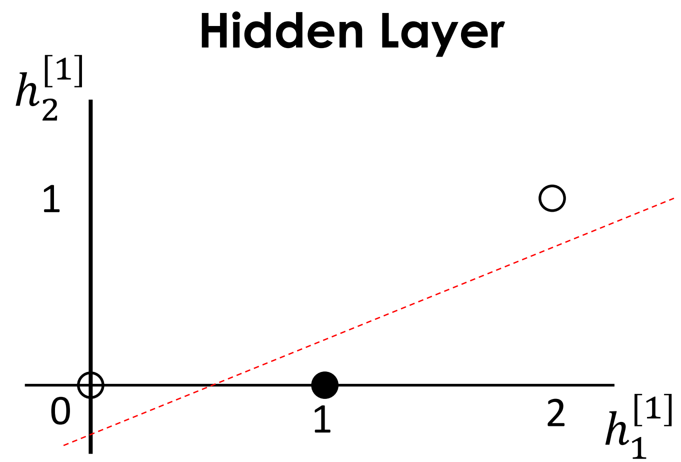
Durham University

Feed-Forward Neural Networks (IV)

The **XOR** example:



Cannot separate class samples using a line!

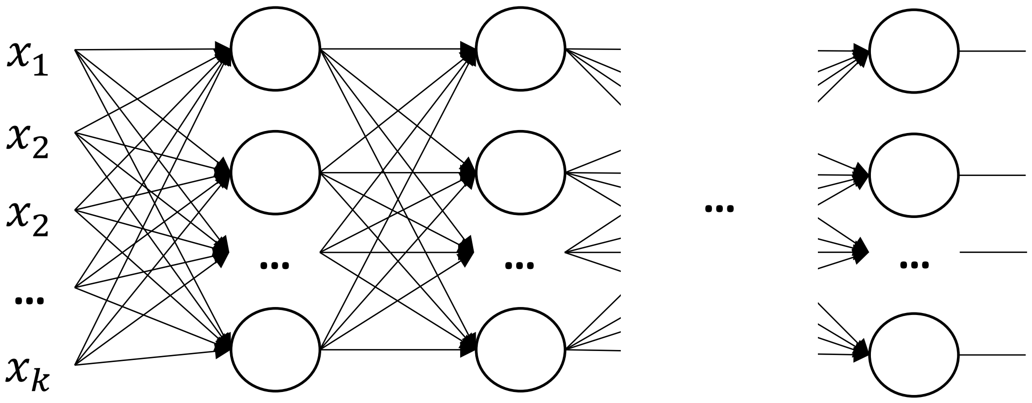


New representation is linearly separable!

Feed data into an ANN

- **How to use a feature vector as the input to an ANN?**
  - e.g. A word or a document embedding
- Define the size of the input layer equal to the number of elements in the feature vector
  - An input layer of size *k* is needed for an embedding with *k* elements
- As the input layer's size increases and the number of additional layers and their size increases, the computational complexity of training the ANN increases

Feature vector:
$(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \dots, \boldsymbol{x_k})$



Loss computation

- The output $\hat{y}$ of a NN is an estimate of a true value y

- We would like $\hat{y}$ to be as close as possible to y

- **How can we measure the error (loss) of an ANN in estimating y?**
- **We need a loss function!**

- Cross-entropy loss typically used in classification

$$L_{CE} = -\sum_{i=0}^{n} y_i \log \hat{y}_i$$

$\hat{y}_i$: predicted probability that sample belongs to class i
y_i: 1 or 0 if sample actually belongs to class i or not
n: number of classes

- **ANN training:** Given an error (loss) function $E \rightarrow$ Minimise E

Durham University

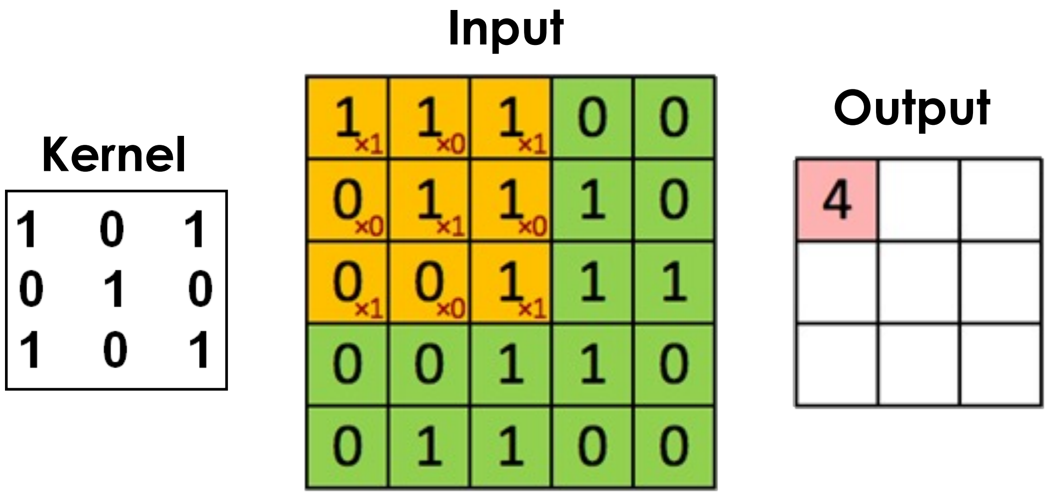
Convolutional Neural Networks (CNN)

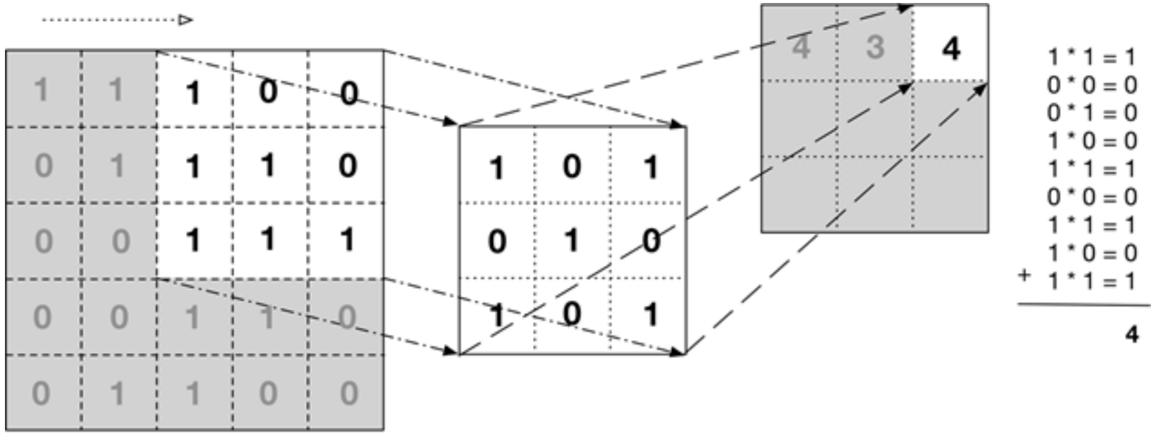
- Neural Networks that contain Convolutional layers
- Convolutional layers apply filters on the input using the convolution operation
- **Local information calculated from parts of the input**
- **Captures local neighbourhood relationships**
- Convolution operator is a sliding window function applied to the input data
- A filter kernel must be defined and moved (slid) across the data

CNNs are commonly used for image classification

Durham University

Convolutional layer





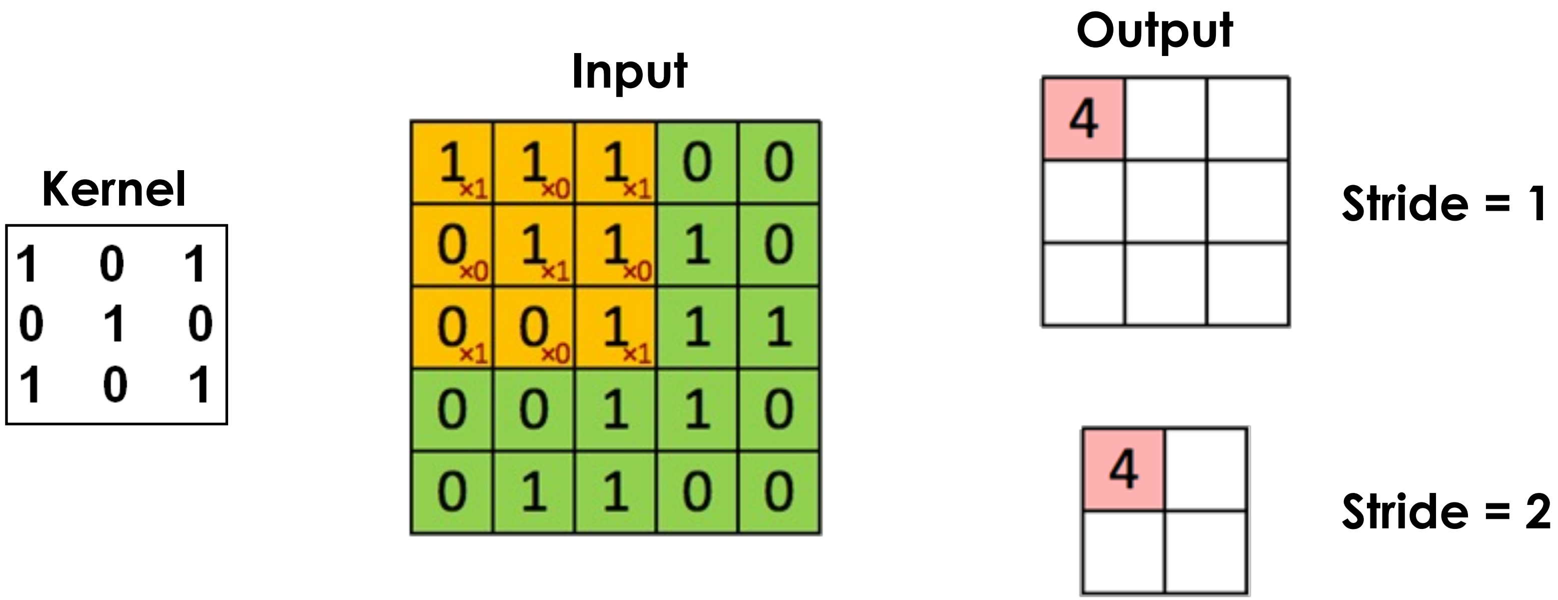
$$(1\times1 + 0\times1 + 1\times1 + 0\times0 + 1\times1 + 0\times1 + 1\times0 + 0\times0 + 1\times1)$$
$$= (1 + 0 + 1 + 0 + 1 + 0 + 0 + 0 + 1) = 4$$

Slide kernel 1 element at a time and compute result

Convolutional layer: Stride

- **Stride** → **How much to slide the kernel at each step**

- Larger stride leads to smaller result



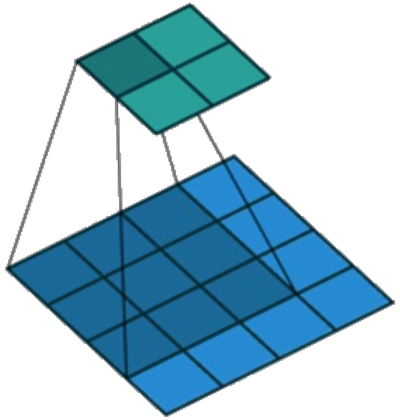
Convolutional layer: Padding

- Applying a filter within the matrix works fine
  - But leads to smaller output than the input
- **What about the edges?**
- How to apply the filter to the first element of a matrix since it doesn't have neighbours on the left and above?
  - Similar for other elements on the edges
- Use **zero-padding!** → **Add zero valued elements**

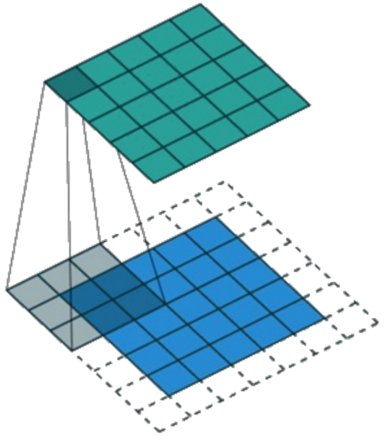
$x_{0,0}$	$x_{0,1}$
$x_{1,0}$	$x_{1,1}$

0	0	0	0
0	$x_{0,0}$	$x_{0,1}$	0
0	$x_{1,0}$	$x_{1,1}$	0
0	0	0	0

Valid padding



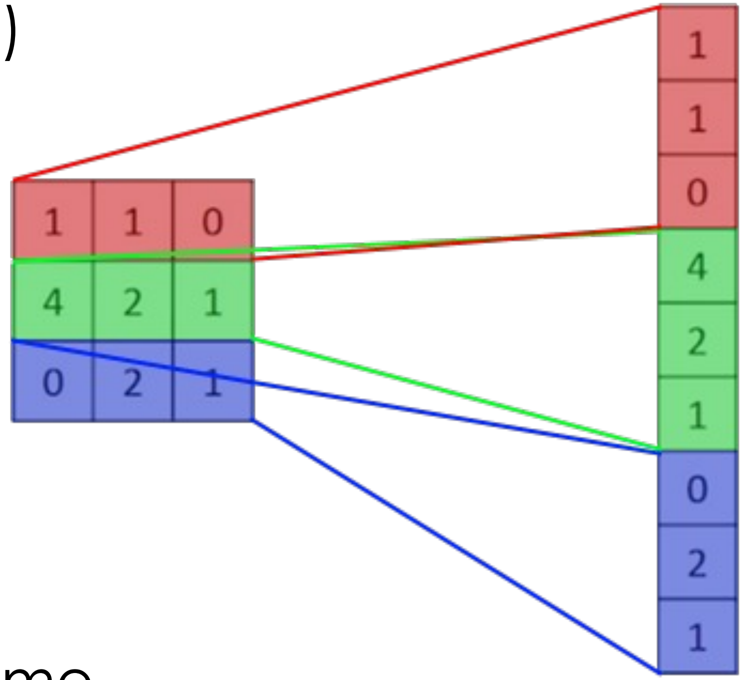
Same padding



Durham University

Flatten layer (I)

- **What if we have input vectors with more than 1 dimension?**
  - e.g. A greyscale image (2-D) (x, y)
  - e.g. An RGB (3-channel) image (3-D) $(x, y, channel)$
  - e.g. A colour video (4-D) $(x, y, channel, time)$
  - e.g. Output of a convolutional layer
  - …

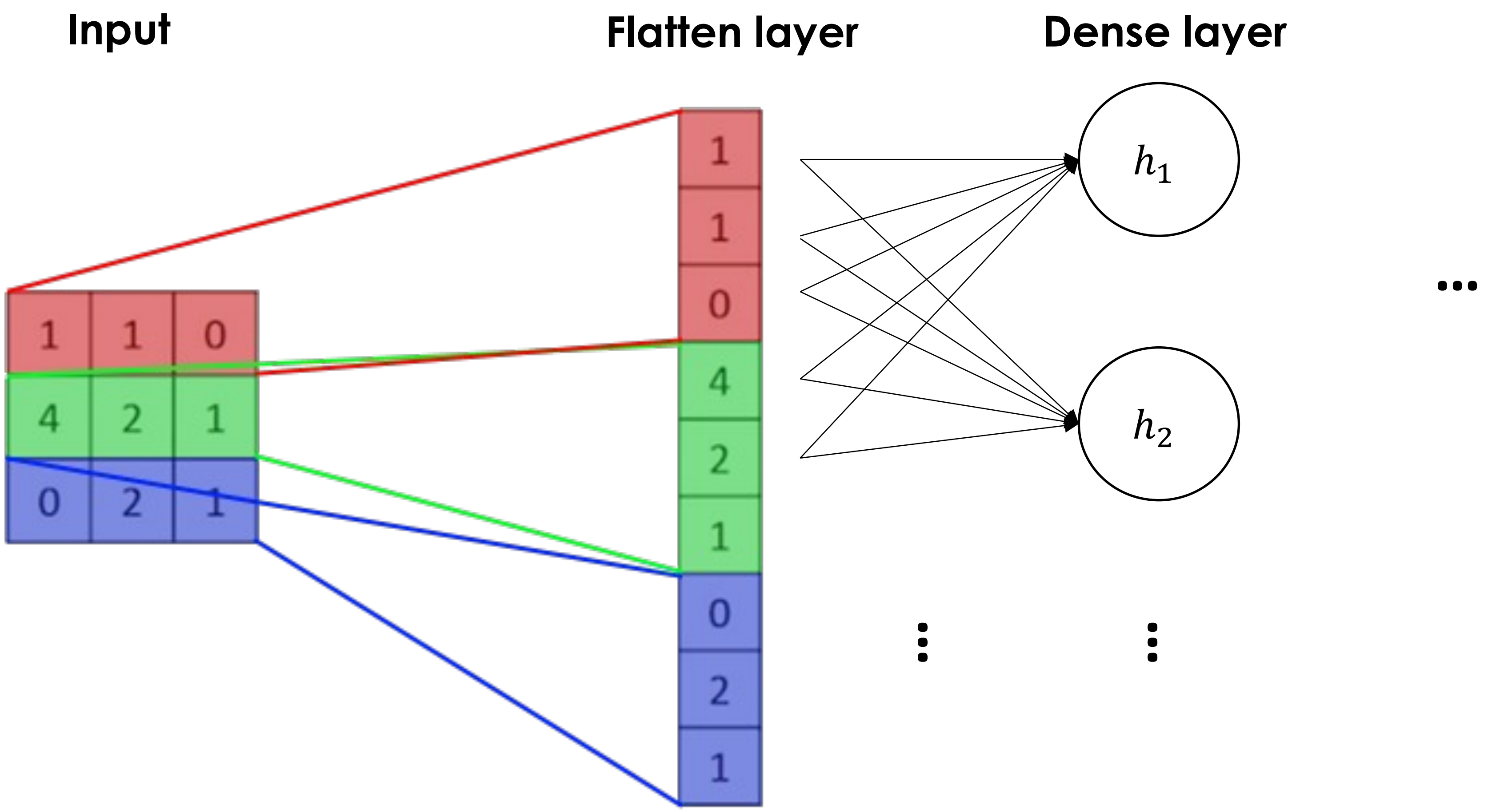


- Typical dense layers only accept 1-D input

Dense / Fully connected layer
Consists of artificial neurons

- **Flatten layer** → **Converts N-D data to 1-D**
  - Concatenates data to a 1-D array, one row at a time

Flatten layer (II)



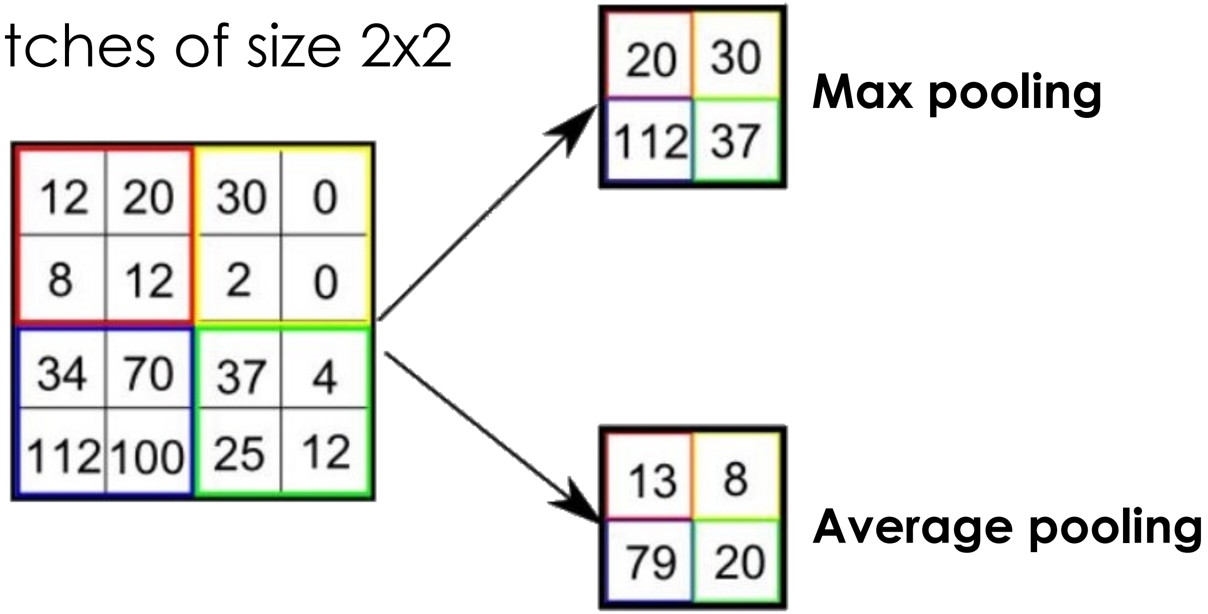
Durham University

Pooling layer (Max/Average)

- Reduces the dimensionality of its input

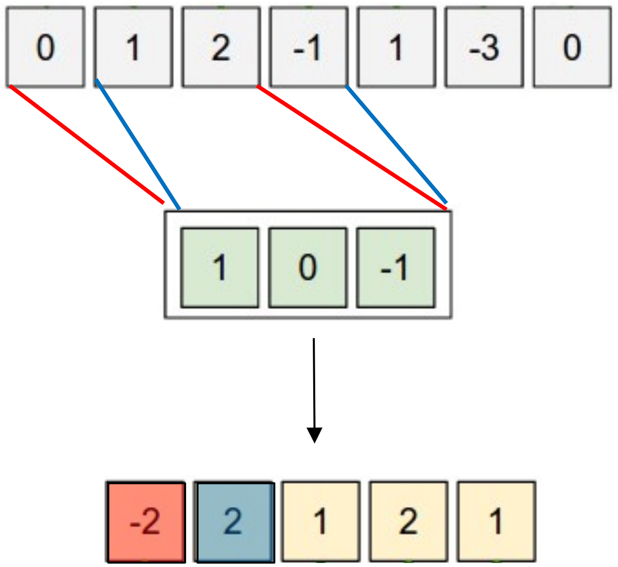
- Reduces the size of its input by pooling together the values of an input patch
  - **Max pooling:** Replaces the patch with the maximum value in the patch
  - **Average pooling:** Replaces the patch with the average value of the patch

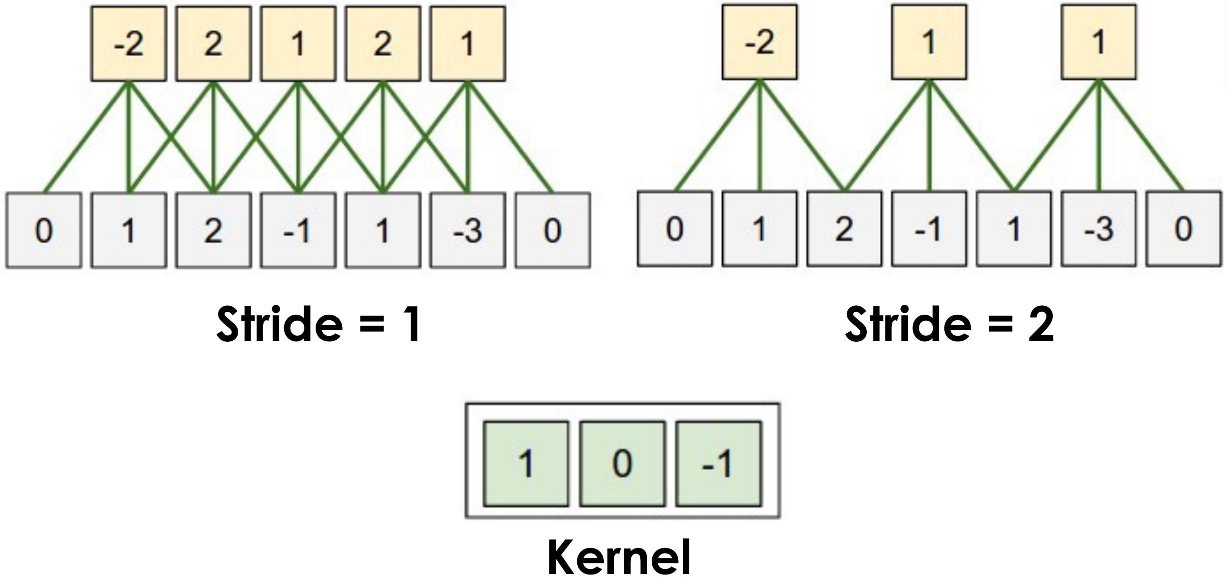
- Typically applied on patches of size 2x2



1-D Convolution

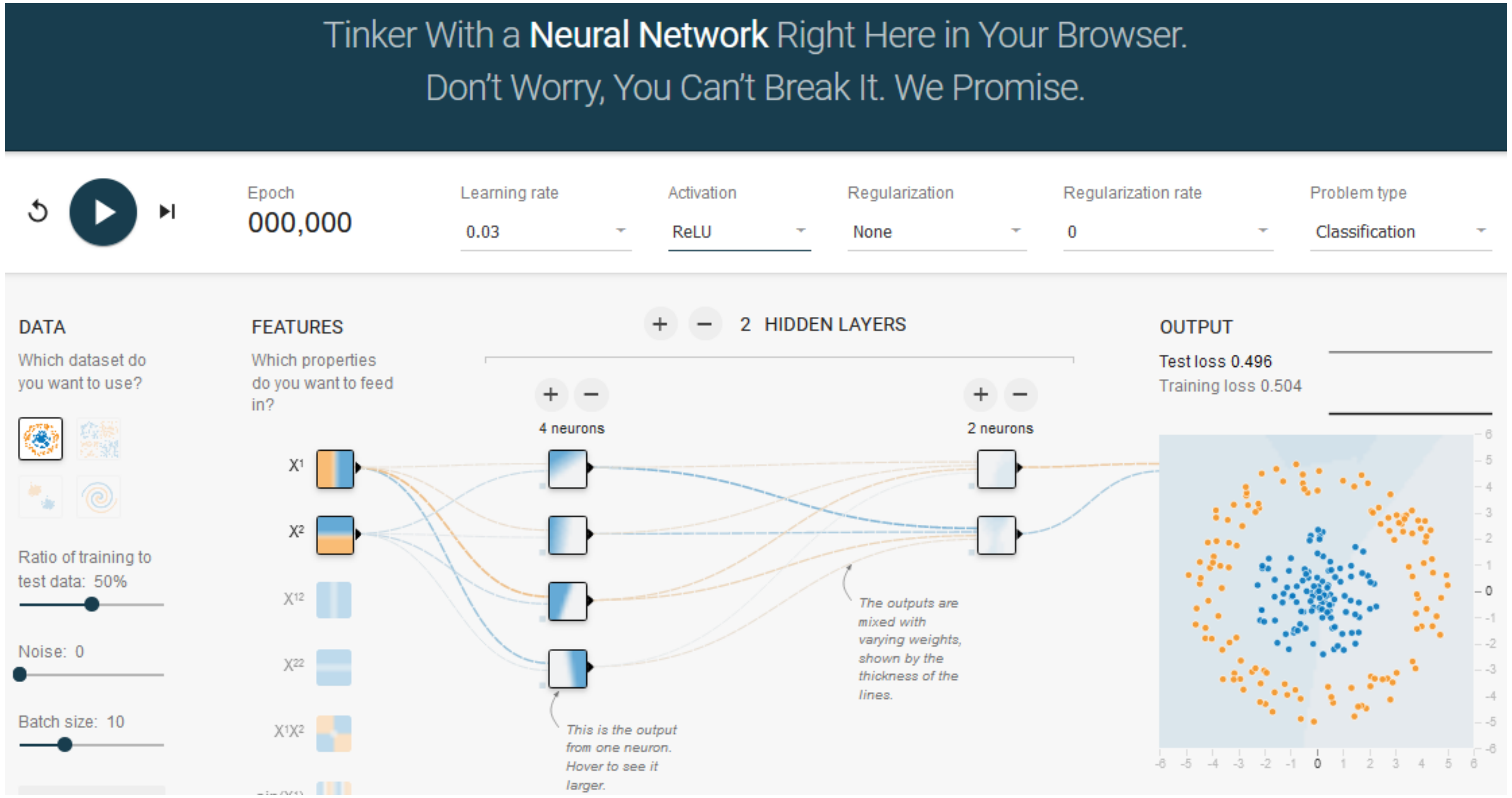
- The convolution operation can be applied to input of any number of dimensions
- **Text is usually 1-D** → Use 1-D convolution





Durham University

Explore ANN training and testing



https://playground.tensorflow.org

Questions?

Durham University

Appendix: Back Propagation

- **ANN training:** Given an error (loss) function $E \rightarrow$ Minimise E

- The Back Propagation algorithm:
  - Start with random weights
  - Compute the output of the network for all available inputs
  - For each input and each weight $w_{i,j}^{[k]}$, compute $\frac{\partial E}{\partial w_{i,j}^{[k]}}$, the partial derivative of E with respect to $w_{i,j}^{[k]}$
  - For each weight $w_{i,j}^{[k]}$, compute the average partial derivative of E across all inputs: $\frac{\partial E_{total}}{\partial w_{i,j}^{[k]}} = \frac{1}{N} \sum \frac{\partial E}{\partial w_{i,j}^{[k]}}$
  - Update each weight as follows: $w'^{[k]}_{i,j} = w_{i,j}^{[k]} - \eta \cdot \frac{\partial E_{total}}{\partial w_{i,j}^{[k]}}$
  - n is the learning rate, typically $0 < \eta < 1$
  - Repeat until a maximum number of iterations (epochs) is reached or a stopping criterion has been met