OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

CT4101
Machine Learning

Dr. Frank Glavin
Room 404, IT Building
Frank.Glavin@University*of*Galway.ie

School of Computer Science

University
*of*Galway.ie

Outline

- What is Deep Learning?
- Artificial neurons
- Activation functions
- Artificial neural networks
- Handling categorial features
- Brief overview of deep learning libraries

What is deep learning?

Deep learning is an approach to machine learning that is inspired by how the brain is structured and operates
Deep learning is a relatively new term that describes research on modern artificial neural networks (ANNs)

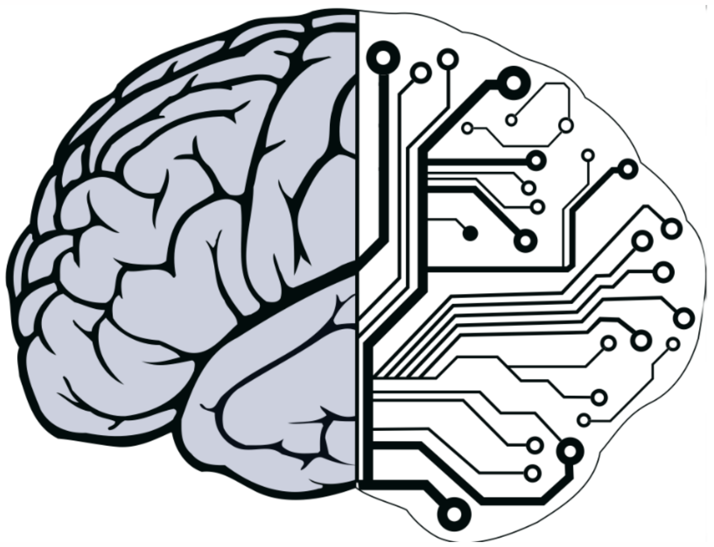
Artificial neural network models are composed of large numbers of simple processing units, called neurons, that typically are arranged into layers and are highly interconnected.
Artificial neural networks are some of the most powerful machine learning models, able to learn complex non-linear mappings from inputs to outputs.

ANNs generally work well in domains in which there are large numbers of input features (such as image, speech, or language processing), and for which there are very large datasets available for training

What is deep learning?

- The history of ANNs dates to the 1940s
- The term deep learning became prominent in the mid 2000s
- The term deep learning emphasizes that modern networks are deeper (in terms of **number of layers** of neurons) than previous networks
- This extra depth enables the networks to learn more complex input-output mappings

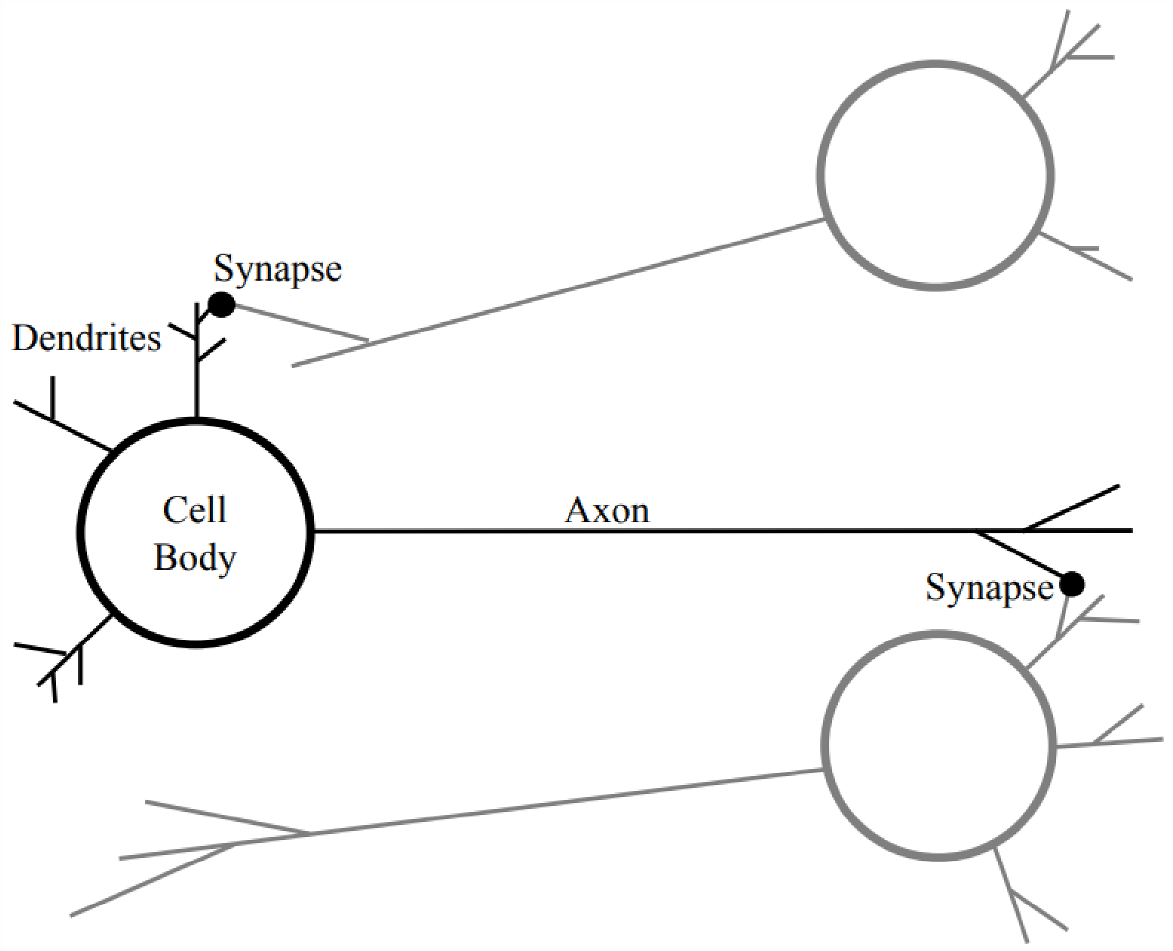
The human brain as inspiration



The human brain is an incredibly powerful learning system.
Thanks to neuroscience we now know quite a bit about the structure of the brain.
For example, we know that the brain works by propagating electrical signals through a massive network of interconnected cells, known as neurons.
It is estimated that the human brain contains around 100 billion interconnected neurons

Neurons in the brain

This figure illustrates three interconnected neurons; the middle neuron is highlighted in black, and the major structural components of this neuron are labelled cell body, dendrites, and axon.
Also marked are the synapses connecting the axon of one neuron and the dendrite of another, which allow signals to pass between the neurons.
These synapses allow electrical signals to pass from the axon of one neuron to a dendrite of another
Dendrites are the neuron’s input channels, and the axon is the output channel



OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Relationship to deep learning

The big idea in deep learning is to develop computational models that are inspired by the structure and operations of the human brain.

The human brain is, of course, much more complex and sophisticated than even the most advanced deep learning models.

Because deep learning models are inspired by the human brain they are known as artificial neural networks ANNs are designed (at least at a very abstract level) to mirror the structure of the brain, and the adoption of a learning mechanism based on adjusting the connections between neurons can be understood as mimicking Hebb's theory of how the brain learns.

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Neural Networks: Artificial neurons

University *of*Galway.ie

Artificial neurons

The fundamental building block of a neural network is a computational model known as an **artificial neuron**

First defined by McCulloch and Pitts (1943) - trying to develop a model of the activity in the human brain based on propositional logic

They recognised that propositional logic using a Boolean representation (TRUE/FALSE or 1/0) and neurons in the brain are similar, as they have an all-or-none character (i.e., they act as a switch that responds to a set of inputs by outputting either a high activation or no activation)

They designed a computational model of the neuron that would take in multiple inputs and then output either a high signal (1), or a low signal (0).

McCulloch and Pitts model

The McCulloch and Pitts model has a two-part structure:

1. Calculation the **result of a weighted sum** (we refer to the result as z)
2. Passing the result of the weighted sum through a **threshold activation function**

In the first stage of the McCulloch and Pitts model, each input is multiplied by a weight, and the results of these multiplications are then added together.

This calculation is known as a weighted sum because it involves summing the weighted inputs to the neuron.

Weighted sum calculation

$\boldsymbol{d}$ is a vector of size $m+1$ inputs/descriptive features ($\boldsymbol{d}[0]$ is a dummy feature always = 1)

$\boldsymbol{w}$ is a vector of $m+1$ weight, one weight for each feature ($\boldsymbol{w}[0]$ is the weight for the dummy feature)

Note similarity to linear regression equation, although (networks of) artificial neurons can do much more than just tackle regression tasks

$$z = \underbrace{\mathbf{w}[0] \times \mathbf{d}[0] + \mathbf{w}[1] \times \mathbf{d}[1] + \cdots + \mathbf{w}[m] \times \mathbf{d}[m]}_{weighted\ sum}$$

$$= \sum_{j=0}^{m} \mathbf{w}[j] \times \mathbf{d}[j]$$

$$= \underbrace{\mathbf{w} \cdot \mathbf{d}}_{dot\ product} = \underbrace{\mathbf{w}^T \mathbf{d}}_{matrix\ product} = [w_0, w_1, \ldots, w_m] \begin{bmatrix} d_0 \\ d_2 \\ \vdots \\ d_m \end{bmatrix}$$

Weights

The weights can either be **excitatory** (having a **positive value**, which increases the probability of the neuron activating) or **inhibitory** (having a **negative value**, which decreases the probability of a neuron firing)
$\boldsymbol{w}[0]$ is the equivalent of the y-intercept in the equation of the line from secondary school geometry; in that the neuron this captures constant effects
$\boldsymbol{d}[0]$ is a dummy feature used for notational convenience and is always equal to 1
This $\boldsymbol{w}[0]$ term is often referred to as the **bias** parameter because in the absence of any other input, the output of the weighted sum is biased to be the value of $\boldsymbol{w}[0]$

Technically, the inclusion of the bias parameter as an extra weight in this operation changes the function from a linear function on the inputs to an **affine function**.
An affine function is composed of a linear function followed by a translation (i.e., the inclusion of the bias term means that the straight line would not pass through the origin)

Threshold activation function

In the second stage of the McCulloch and Pitts model, the result of the weighted sum calculation, z, is then converted into a high or a low activation by passing z through an **activation** function.

McCulloch and Pitts used a **threshold activation function**: if z is greater than or equal to the threshold, the artificial neuron outputs a 1 (high activation), and otherwise it outputs a 0 (low activation).
This threshold is a number that is selected by the designer of the artificial neuron.

Using the symbol θ to denote the threshold, the second stage of processing in the McCulloch and Pitts model can be defined. We see that the neuron 'fires' when $z \geq \theta$

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & otherwise \end{cases}$$

Ollscoil na Gaillimhe
University of Galway

Artificial neurons

φ (Greek lowercase letter ‘phi’) is the **activation function** of the neuron
$\boldsymbol{d}$ is a vector of $m + 1$ descriptive features ($\boldsymbol{d}[0]$ is a dummy feature always = 1)
$\boldsymbol{w}$ is a vector of $m + 1$ weights ($\boldsymbol{w}[0]$ is the weight for the dummy feature, also referred to as the **bias**)

$$\begin{aligned}
\mathbb{M}_{\mathbf{w}}(\mathbf{d}) &= \varphi\left(\mathbf{w}[0] \times \mathbf{d}[0] + \mathbf{w}[1] \times \mathbf{d}[1] + \cdots + \mathbf{w}[m] \times \mathbf{d}[m]\right) \\
&= \varphi\left(\sum_{i=0}^{m} w_i \times d_i\right) = \varphi\left(\underbrace{\mathbf{w} \cdot \mathbf{d}}_{dot\ product}\right) \\
&= \varphi\left(\underbrace{\mathbf{w}^T \mathbf{d}}_{matrix\ product}\right) = \varphi\left(\left[w_0, w_1, \ldots, w_m\right] \begin{bmatrix} d_0 \\ d_2 \\ \vdots \\ d_m \end{bmatrix}\right)
\end{aligned}$$

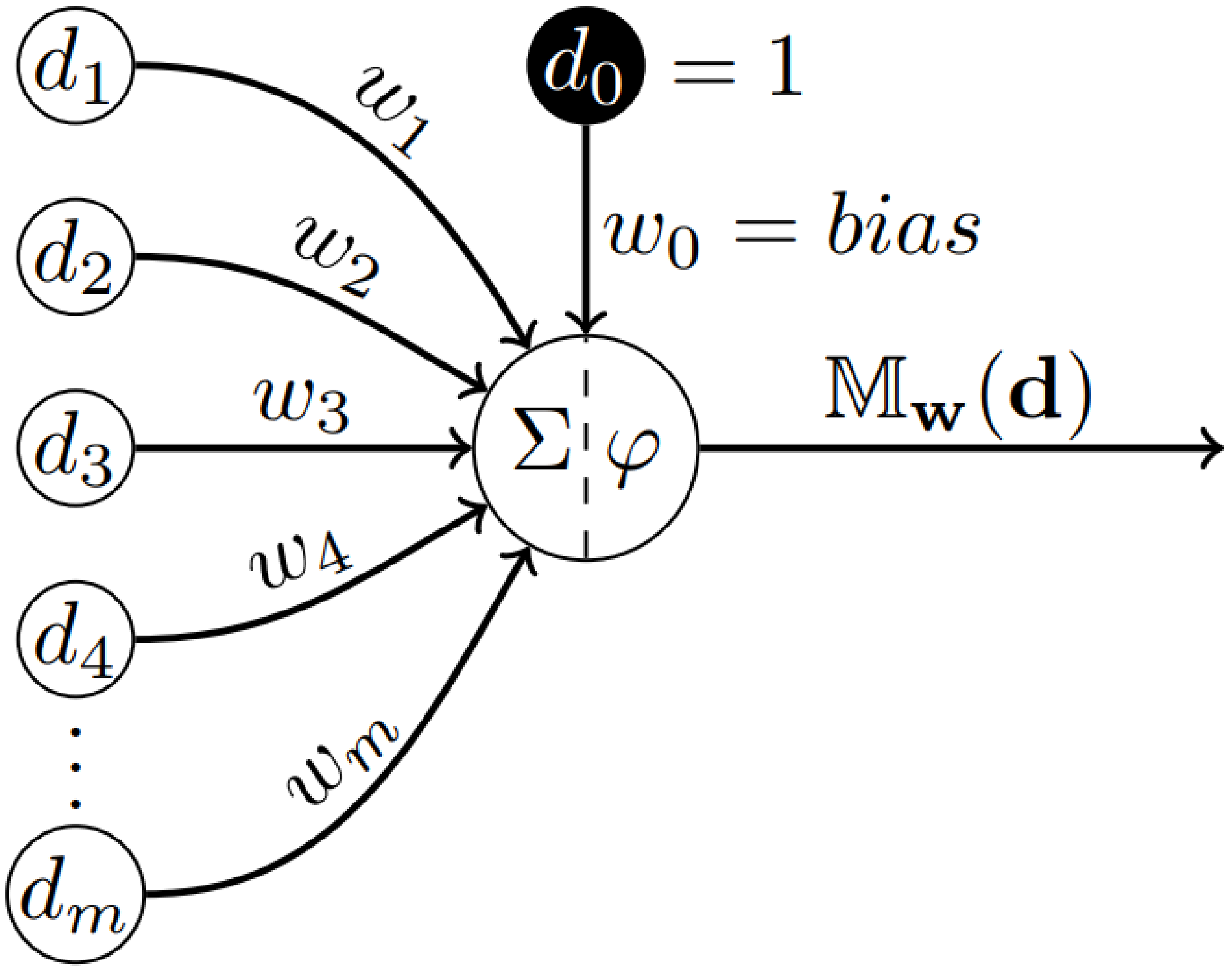
Artificial neuron schematic

Arrows carry activations in the direction the arrow is pointing

The weight label on each arrow represents the weight that will be applied to the input carried along the arrow

φ (Greek lowercase letter 'phi') is the **activation function** of the neuron

Also referred to as a **perceptron**



McCulloch and Pitts vs. Modern Artificial Neurons

The two-part structure of the McCulloch and Pitts model is the basic blueprint for modern artificial neurons

Key difference: how to set the weights?

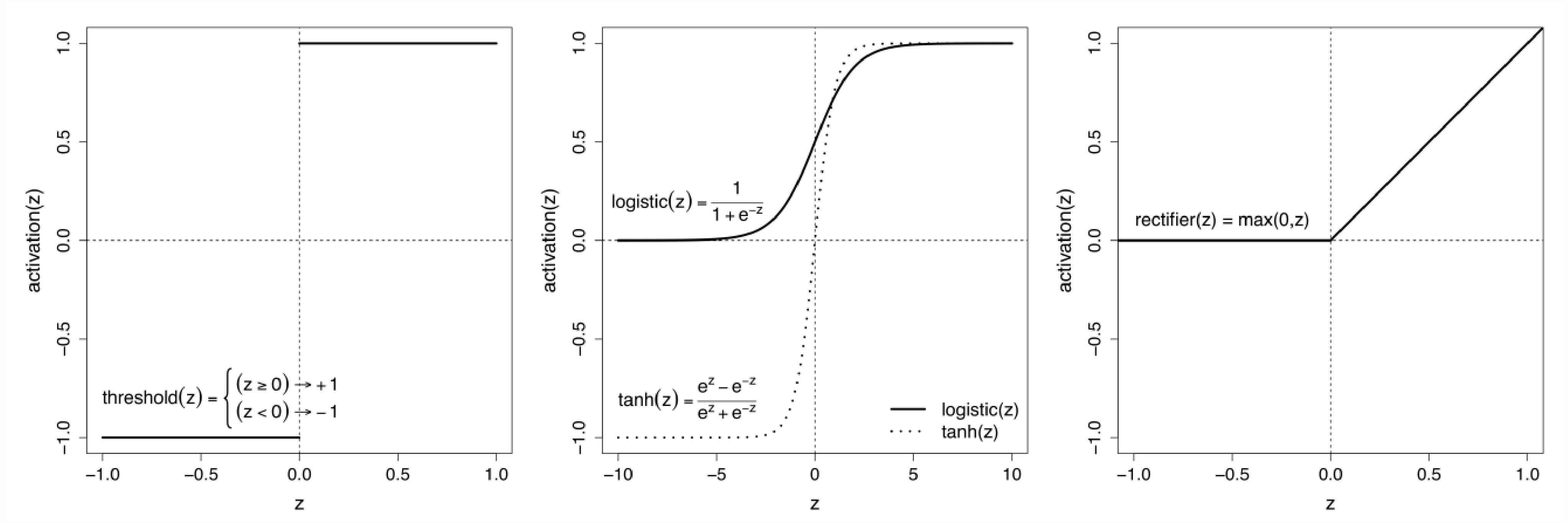
- In the McCulloch and Pitts model the weights were manually set (very difficult to do!)
- In modern ML we **learn the weights using data**

Key difference: activation functions

- McCulloch and Pitts considered the threshold activation function only
- Modern artificial neurons use one of a **range of different activation functions**

Example activation functions

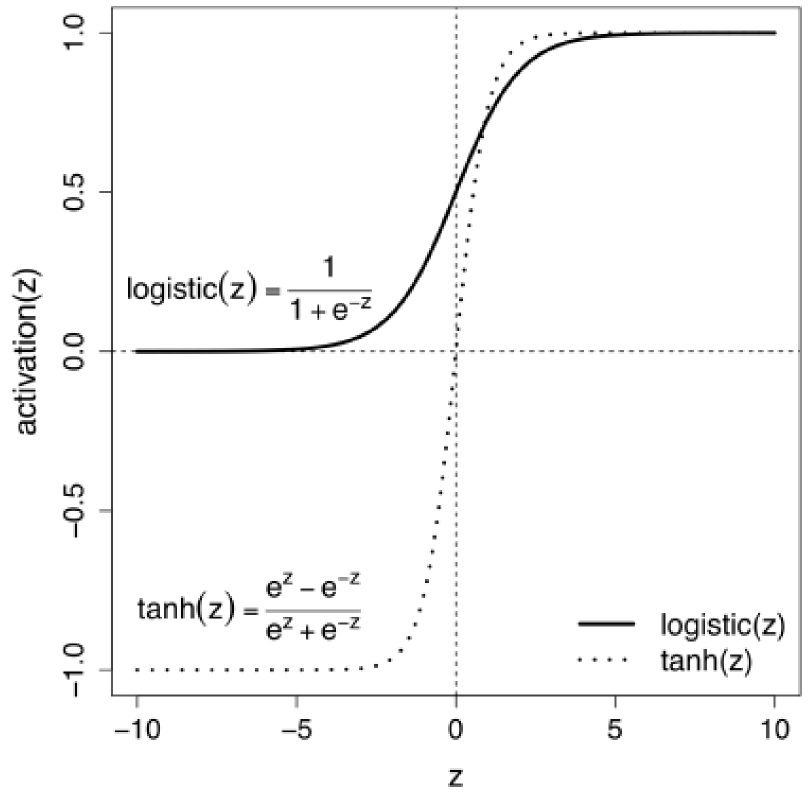
Note that all of these are **non-linear** functions (we will discuss the implications of this later)



Logistic activation function

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

- Historically the **logistic activation function** was among the most commonly-used choices of activation functions
- Note that neurons are often referred to as units, and they are distinguished by the type of activation function they use.
- Hence a neuron that uses a logistic activation function is referred to as a **logistic unit**
- Logistic activation functions map any real number input to an output between 0 and 1



OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

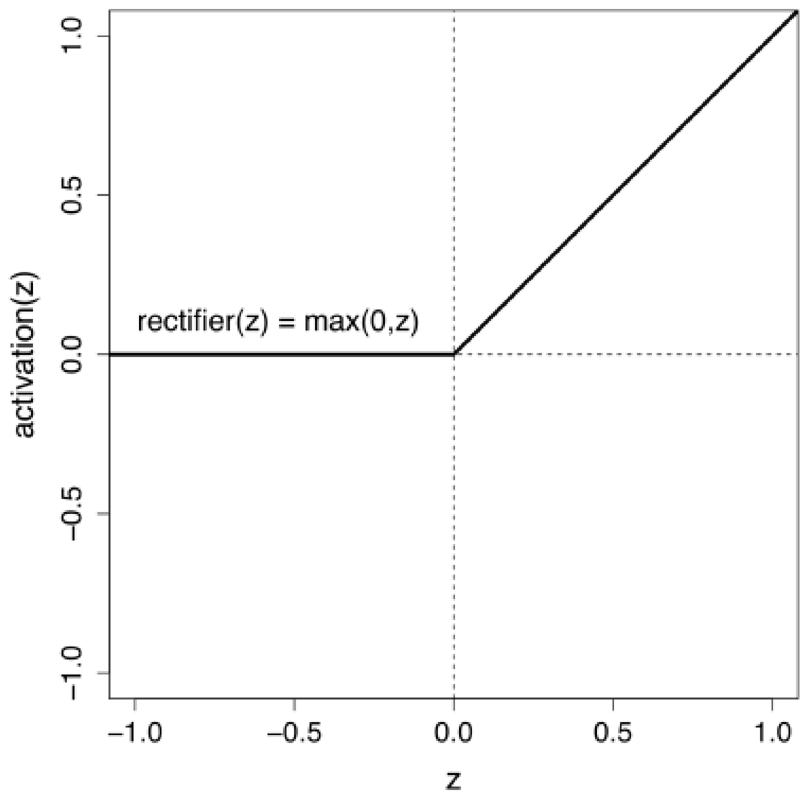
Rectified linear activation function

$$rectifier(z) = max(0, z)$$

Today the most popular choice of function for an activation function is the **rectified linear activation** function or **rectifier**

A unit that uses the rectifier function is known as a **rectified linear unit** or **ReLU**

The rectified linear function does not saturate (i.e., there is no maximum output value constraint)



OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Neural Networks: Artificial neural networks

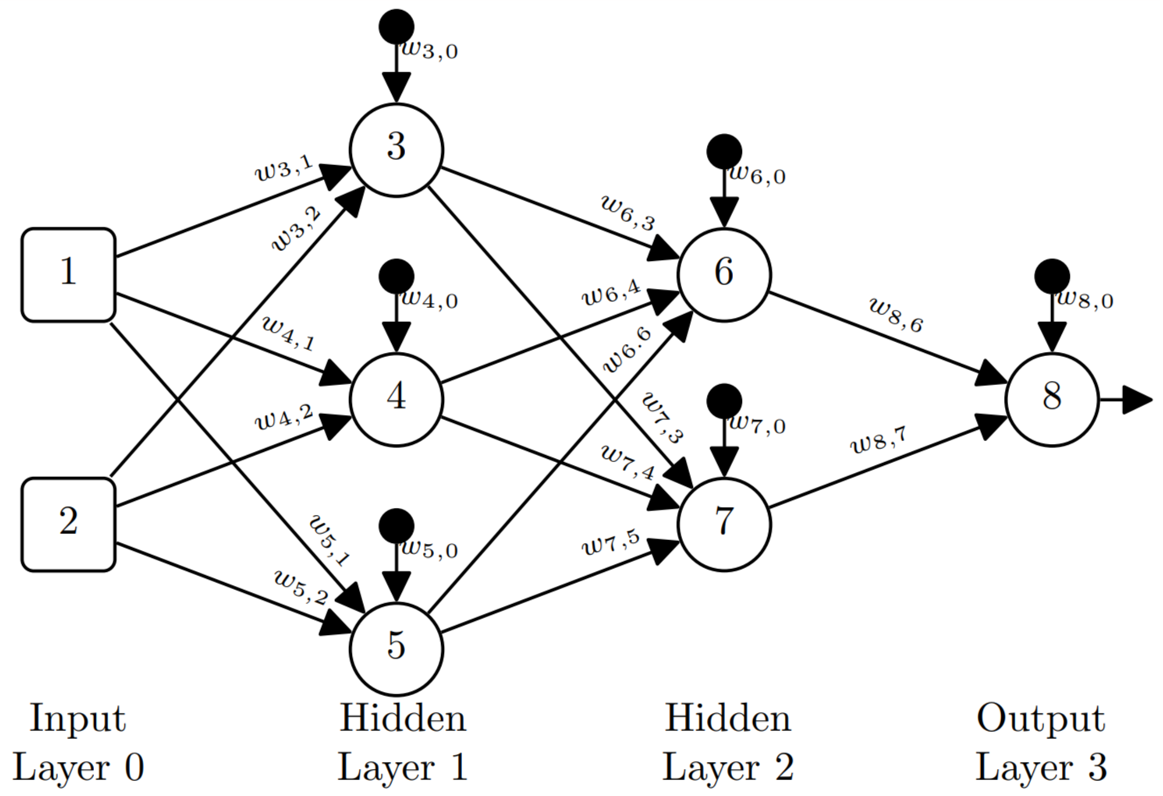
University ofGalway.ie

Artificial neural networks

An artificial neural network (also called a multi-layer perceptron or MLP) consists of a network of interconnected artificial neurons.

The neurons in the **feedforward neural network** to the right are organized into a sequence of **layers**
An artificial neural network can have any structure, but a layer-based organization of neurons is common

There are two types of neurons in this network: sensing neurons and processing neurons.

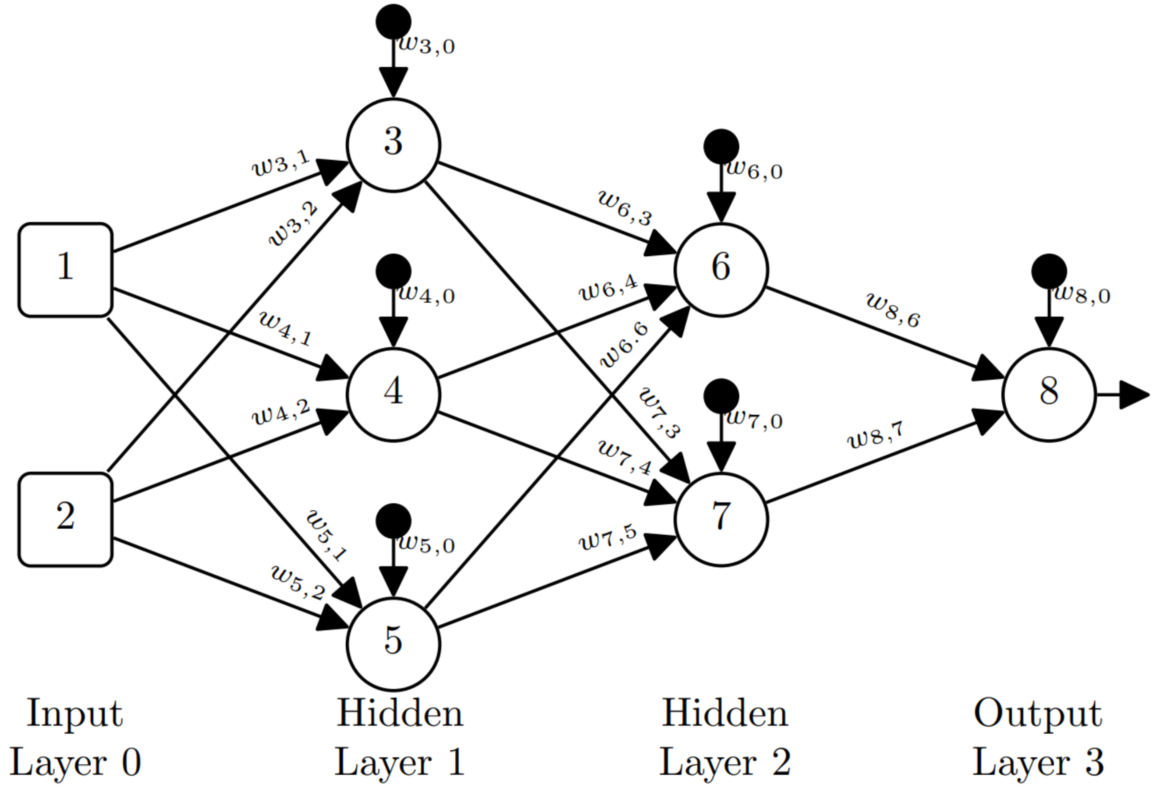


OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Artificial neural networks

The two squares on the left of the figure represent the two memory locations through which inputs are presented to this network. These locations can be thought of as sensing neurons that permit the network to sense the external inputs.

Although we consider these memory locations as (sensing) neurons within the network, the inputs presented to the network are not transformed by these sensing neurons.



OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

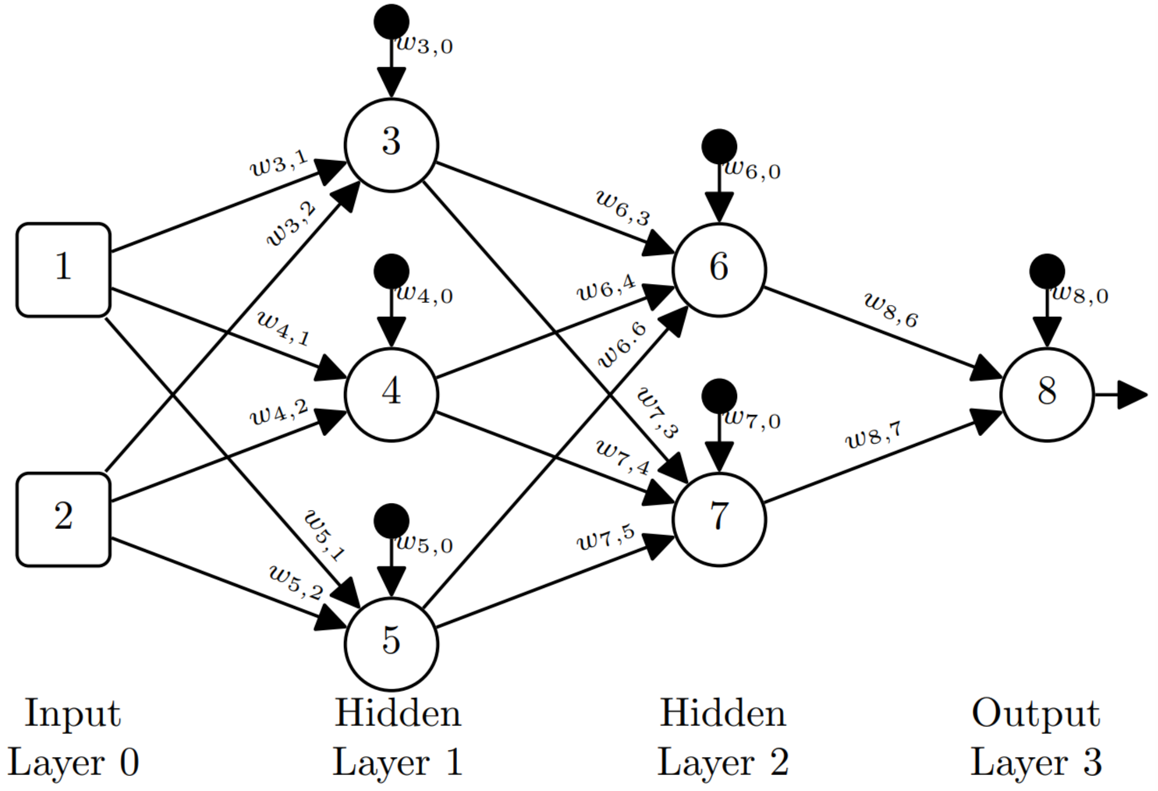
Artificial neural networks

Circles in the network represent processing neurons that transform input using the previously described two-step process of a weighted sum followed by an activation function

Arrows connecting the neurons in the network indicate the flow of information through the network

Input to processing neurons can be:

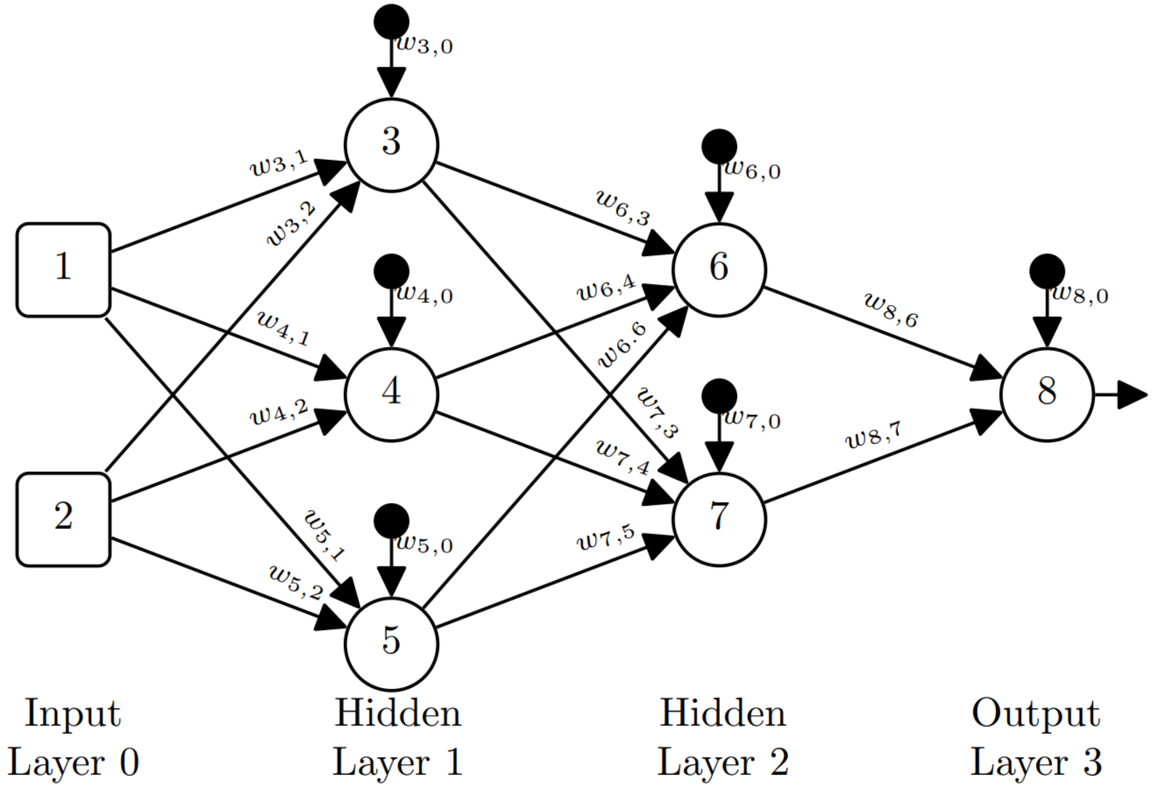
- external input from sensing neurons
- the output activation of another processing neuron in the network
- a dummy input that is always set to 1 (the input from a black circle)



Artificial neural networks

In feedforward networks there are no loops or cycles in the network connections that would allow the output of a neuron to flow back into the neuron as an input (even indirectly).

In a feedforward network the activations in the network always flow forward through the sequence of layers.
This ANN is **fully connected** because each of the neurons in the network is connected so that it receives inputs from all the neurons in the preceding layer and passes its output activation to all the neurons in the next layer
The depth of a neural network is the number of hidden layers plus the output layer



When is a neural network considered ‘deep’?

The number of layers required for a network to be considered deep is an open question
Cybenko (1988) proved that a network with three layers of (processing) neurons (i.e., two hidden layers and an output layer) can approximate any function to arbitrary accuracy.
So, here we define the minimum number of hidden layers necessary for a network to be considered deep as two; under this definition the network shown on the previous slide would be described as a deep network.
However, most deep networks have many more than two hidden layers.
Some deep networks have tens or even hundreds of layers

Notes on activation functions

It is not a coincidence that the most useful activation functions are non-linear.

Introducing non-linearity into the input to output mapping defined by a neuron enables an artificial neural network to learn complex non-linear mappings

This ability to learn complex non-linear mappings that makes artificial neural networks such powerful models, in terms of their ability to be accurate on complex tasks.

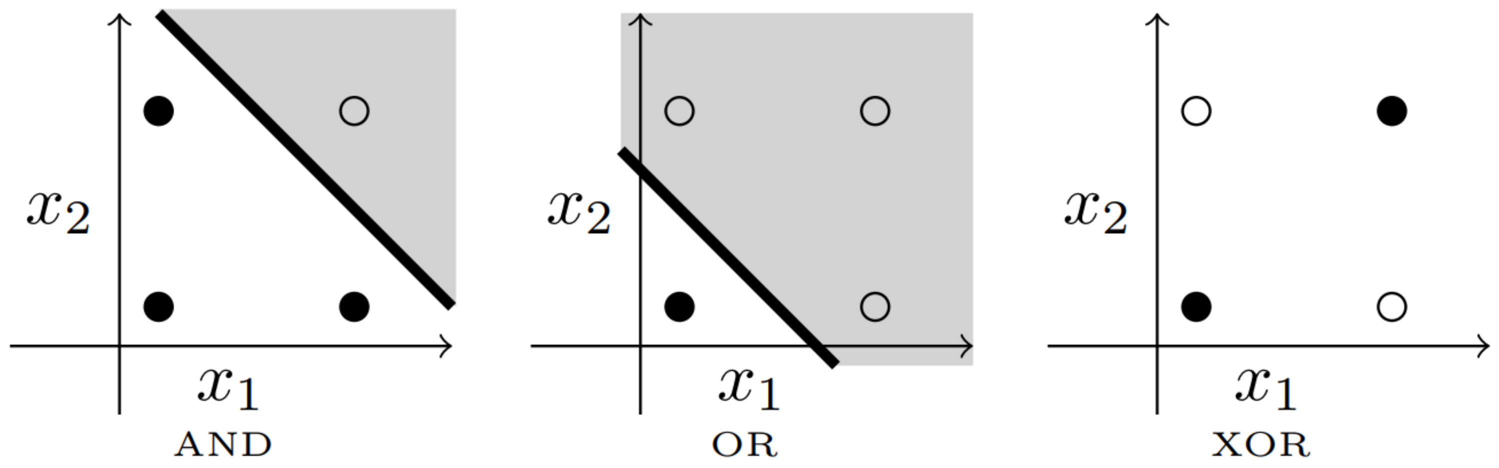
A multi-layer feedforward neural network that uses only linear neurons (i.e., neurons that do not include a non-linear activation function) is equivalent to a single-layer network with linear neurons; in other words, it can represent only a linear mapping on the inputs. This equivalence is true no matter how many hidden layers we introduce into the network.

Introducing even simple non-linearities in the form of logistic or rectified linear units is sufficient to enable neural networks to represent arbitrarily complex functions, if the network contains enough layers; in other words, if the networks are deep enough

Linear Separability - examples from Boolean logic

All three functions have the same structure; they all take two inputs that can be either TRUE (1) or FALSE (0), and they return either TRUE (1) or FALSE (0).

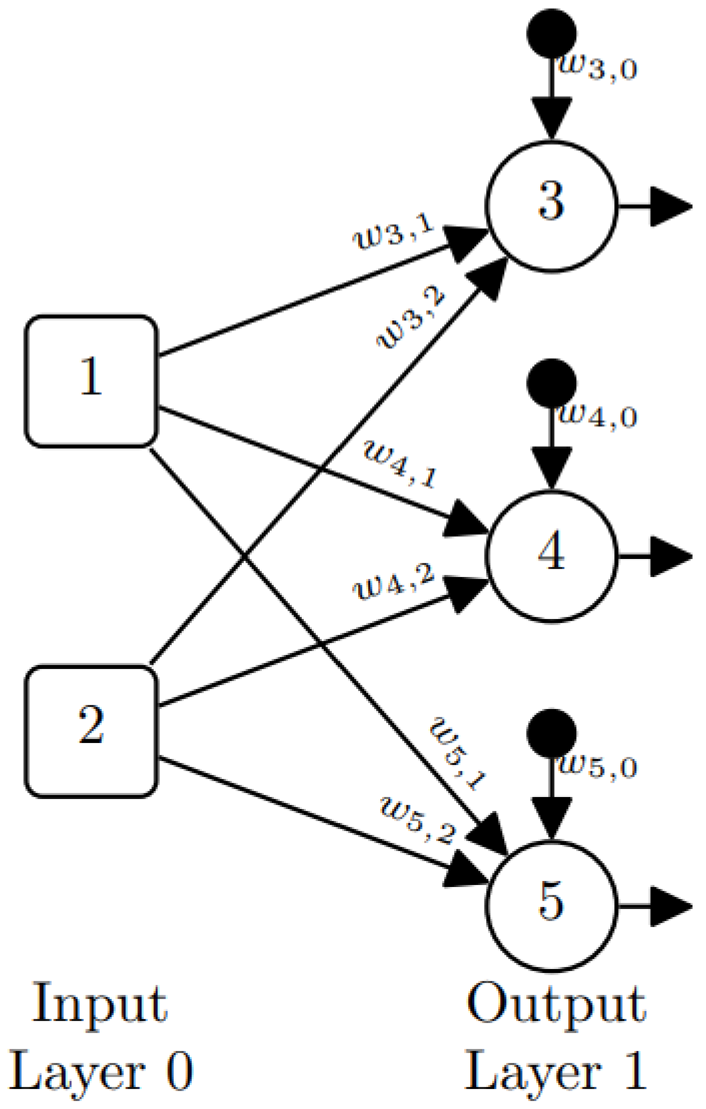
Solid black dots represent FALSE outputs and clear dots represent TRUE outputs
AND returns TRUE if both inputs are TRUE (linearly separable)
OR returns TRUE if either input is TRUE (linearly separable)
XOR returns TRUE if only one of the inputs is TRUE (not linearly separable)

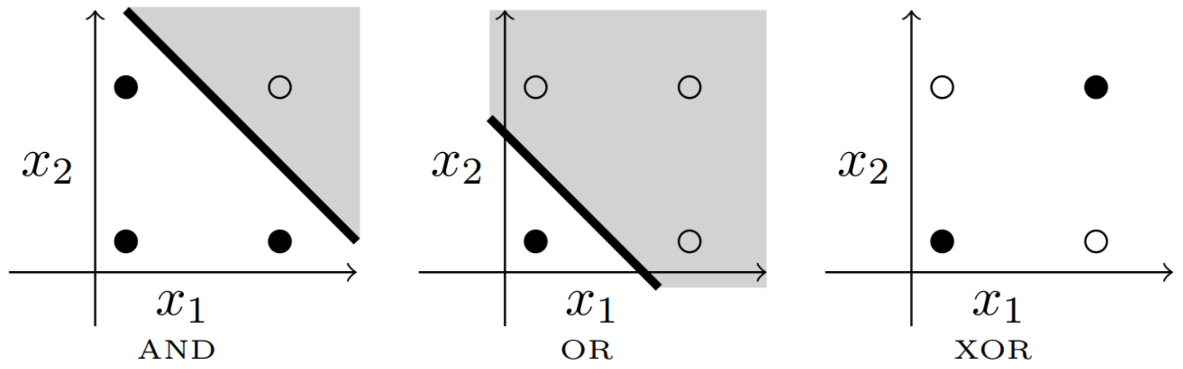


OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Why is network depth important?

Shown on the right – a single layer neural network
Notorious finding in the history of ANNs - although the XOR function is very simple, a perceptron cannot represent it because it is not linearly separable
Minsky and Papert, 1969 published a book highly critical of perceptrons – set research back a decade!
The representational limitation of single-layer networks can be overcome by adding a single hidden layer to the network





Network depth vs. learning rate

Adding depth to a network comes with a cost.
As we see when we discuss the **vanishing gradient problem**, adding depth to a network can slow down the rate at which a network learns.
Therefore, when we wish to increase the representational power of a network, there is often a trade-off between making a network deeper and making the layers wider.

Finding a good equilibrium for a given prediction task involves experimenting with different architectures to see which performs best (i.e., we can treat the **neural network architecture as a hyperparameter** that must be optimised)

Overall, there is a general trend that deeper networks have better performance than shallower networks, but as networks become deeper, they can become more difficult to train

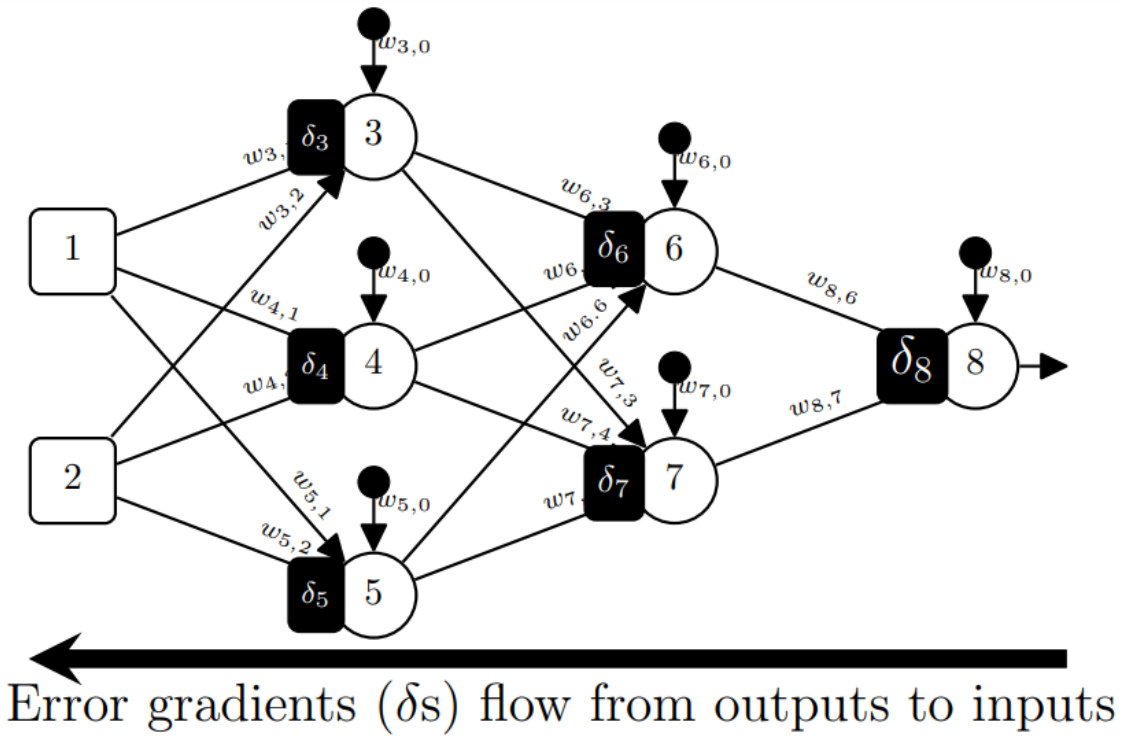
Training neural networks

Blame assignment problem – when learning neural network weights, how to calculate how much of the error in the entire network is due to an individual neuron

The **backpropagation algorithm** solves the blame assignment problem

At each learning step, once we have used backpropagation to calculate errors for individual neurons, we can then use **gradient descent** to update the weights for each neuron in the network

(we will not cover backpropagation or gradient descent)



Example learning process for a feedforward ANN applied to regression

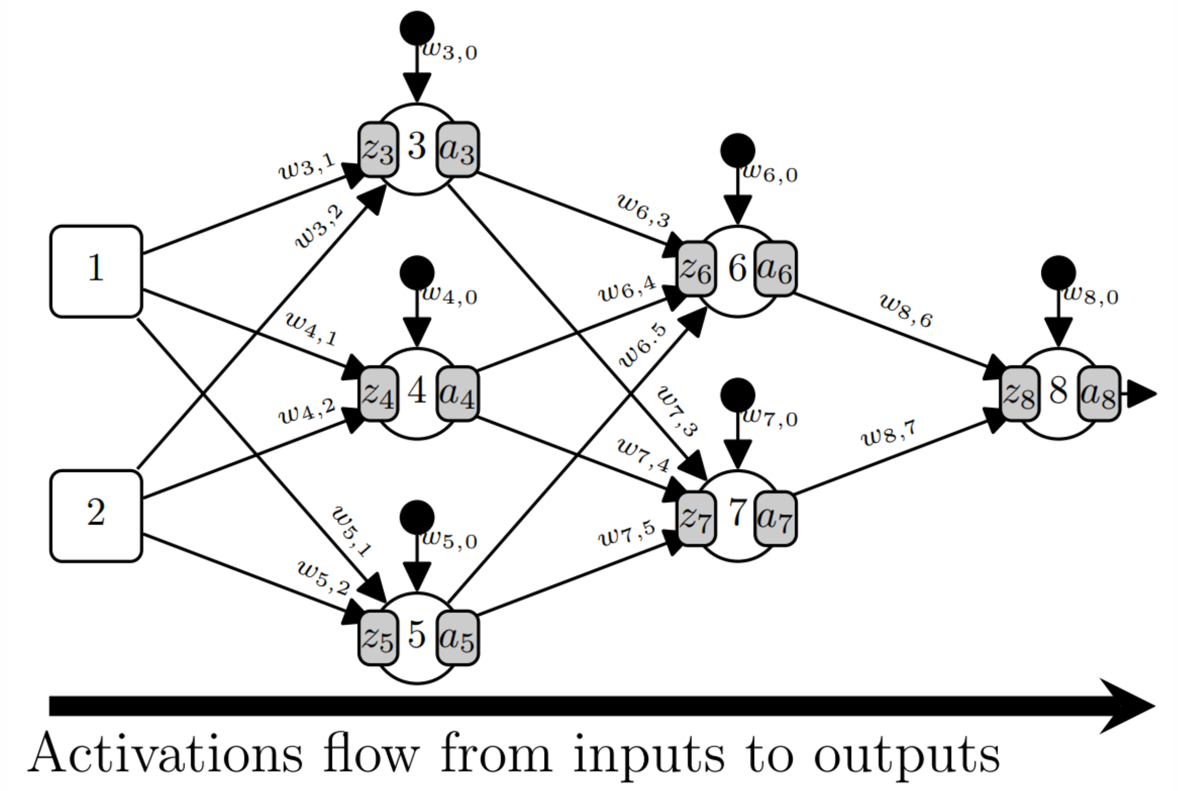
For example, consider a regression problem where we predict a continuous target value based on the values of two independent attributes
We could use the feedforward network structure outlined earlier to tackle this

Example process:

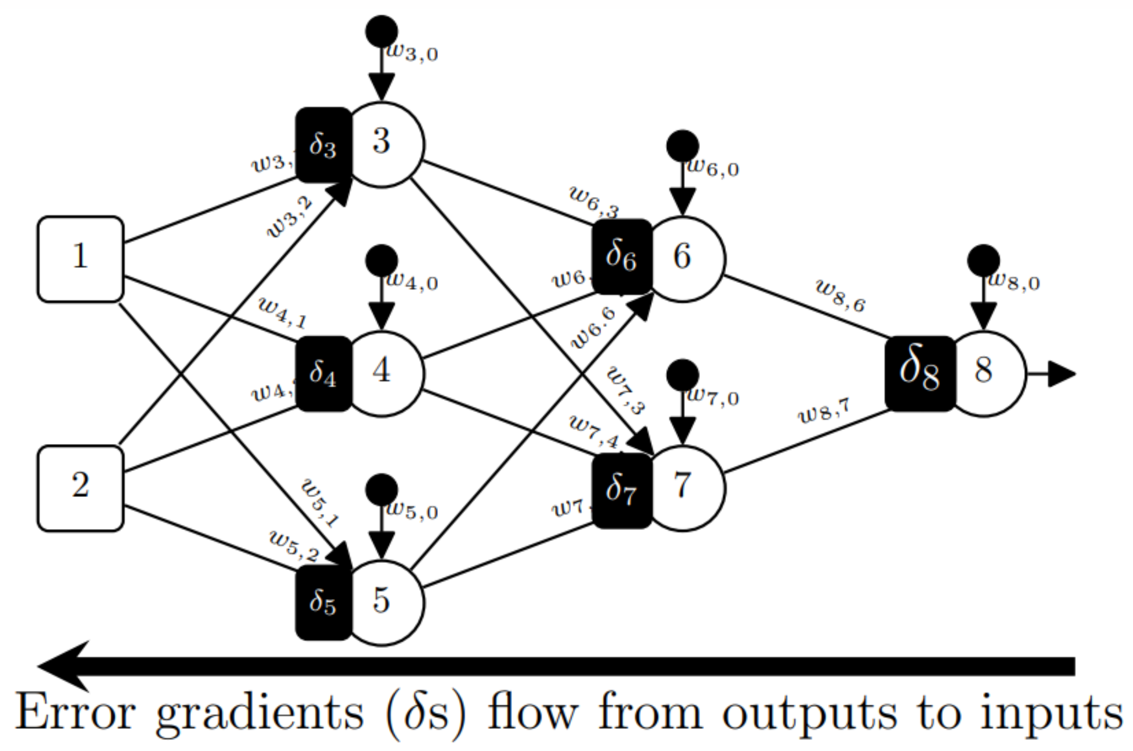
- We initialise the weights *randomly* at first
- We make predictions and then use these to compute the error of the network using a **loss function** (e.g., sum of squared errors on training data)
- We use **back propagation** to compute errors on each neuron
- We use **gradient descent** to update neuron weights
- We repeat this for **multiple training steps** (epochs) until convergence

Flows of activations and errors

Calculate predicted values and use the loss function (e.g., sum of squared errors) to calculate the error of



Activations flow from inputs to outputs

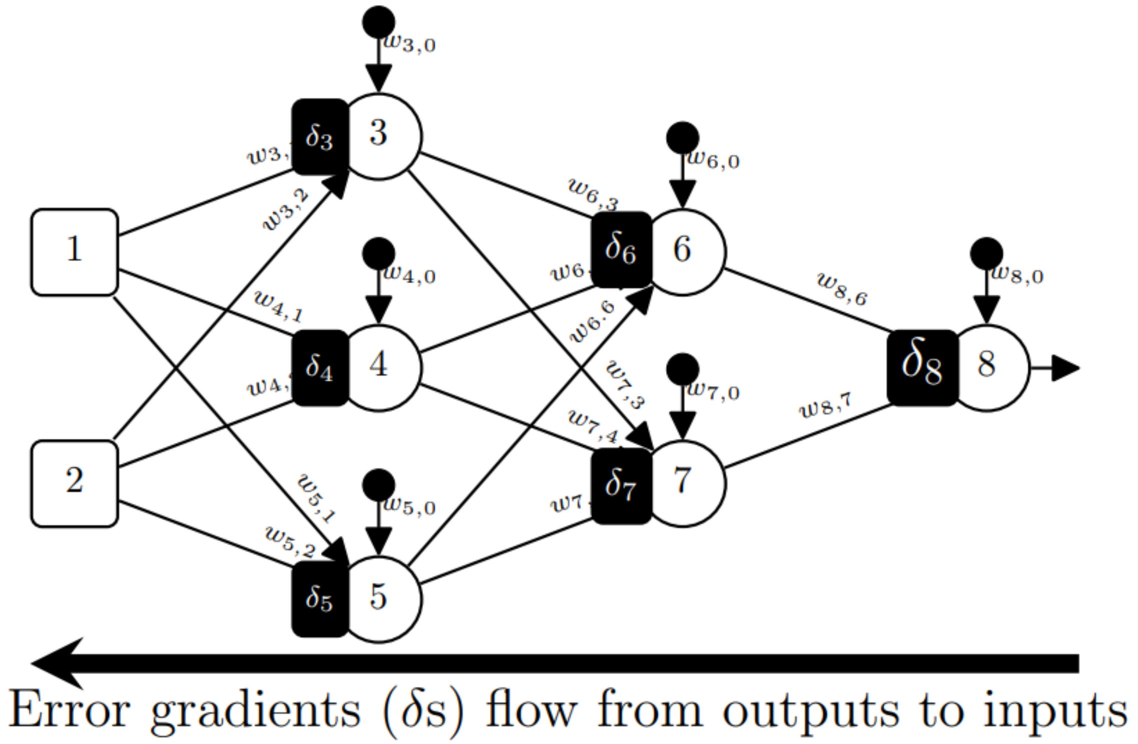


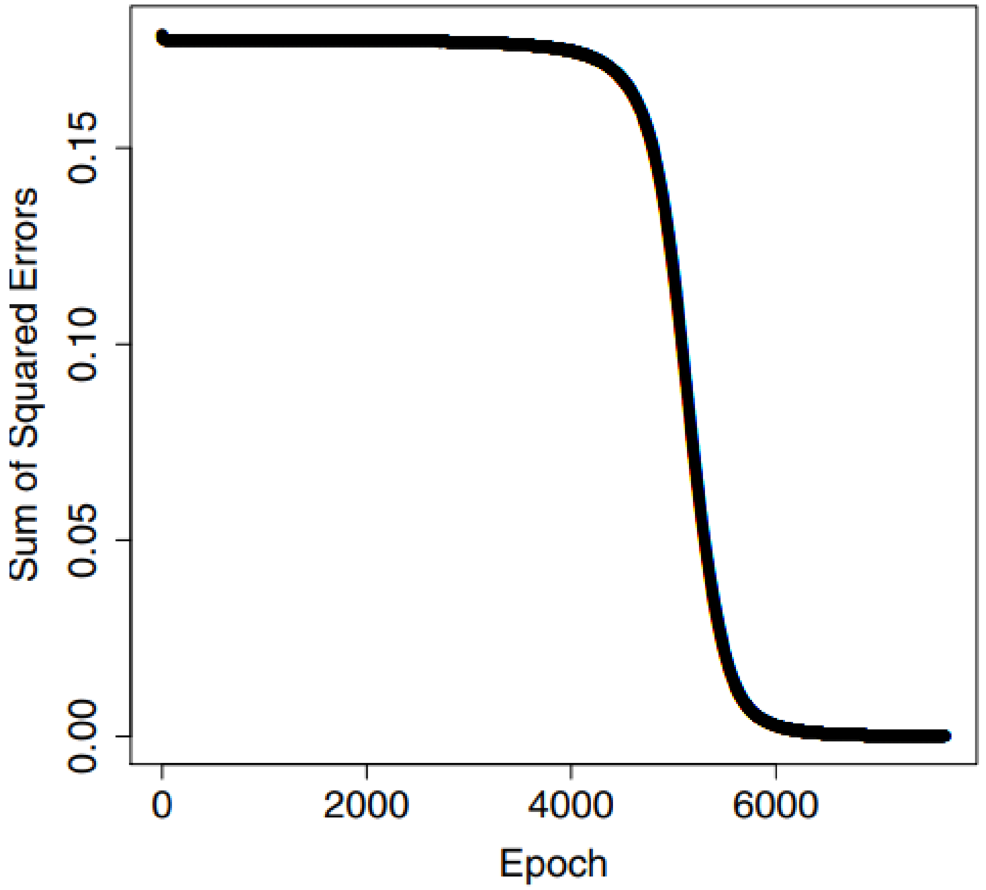
Error gradients (δs) flow from outputs to inputs

$$L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \frac{1}{2}\sum_{i=1}^{n}\left(t_i - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i)\right)^2$$

Example learning process for a feedforward ANN applied to regression

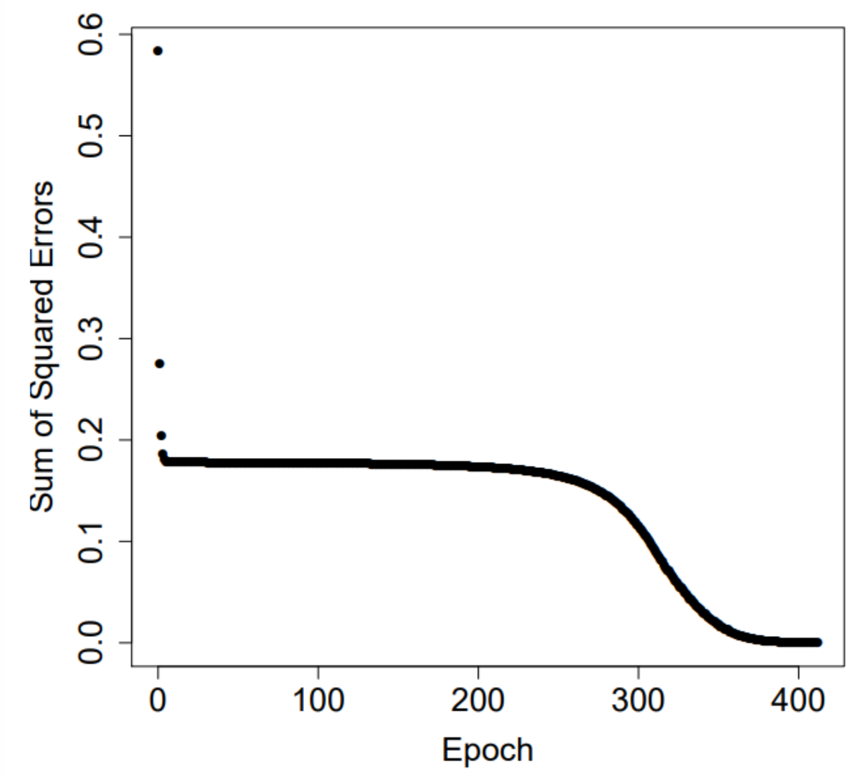
At every epoch there is a **small improvement in the network weights** until convergence is reached

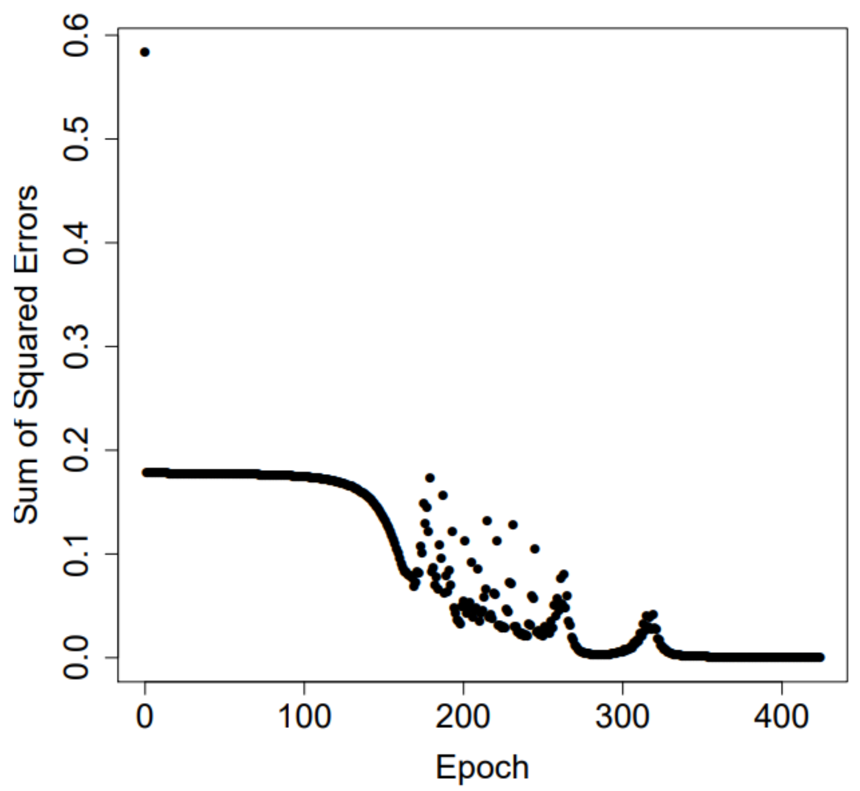




Ollscoil na Gaillimhe
University of Galway

The effect of the learning rate





- One important hyperparameter that must be set in ANNs (and many other ML algorithms) is the **learning rate** – this controls how large updates to the model weights are at each epoch
- **Left**: example of an appropriate learning rate.
- **Right**: example of a learning rate set too high

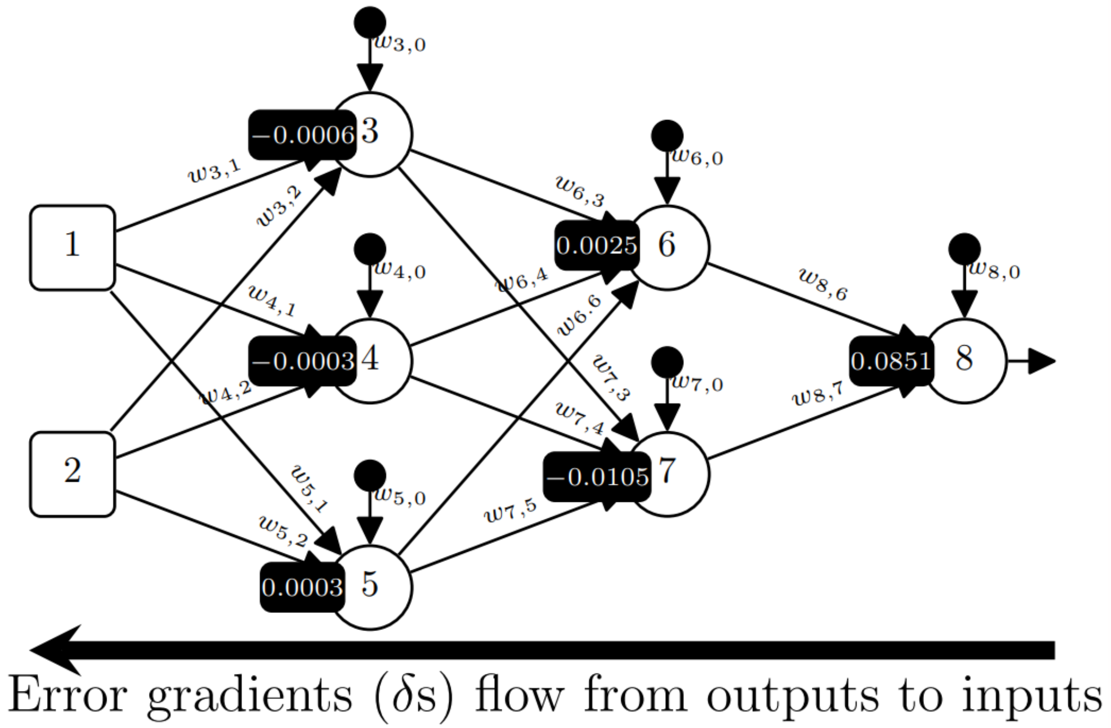
The vanishing gradient problem

The errors for each individual neuron are calculated by backpropagating the error from the output layer through each hidden layer of the network

The magnitudes of these errors tend to decrease as we move from the output layer back to earlier layers in the network

This phenomenon is known as the **vanishing gradient problem**. This is a serious challenge for deep networks as these errors are the learning signals used to optimise individual neuron weights

Rectified linear activation functions can lead to quicker training times than logistic functions



OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Ollscoil na Gaillimhe
University of Galway

Neural Networks: Handling categorical features

University
*of*Galway.ie

Handling categorical descriptive features

To apply many ML algorithms to datasets with categorical descriptive features, it is first necessary to encode the categorical features as numerical features

This is necessary for many common ML algorithms including ones we have covered such as linear regression, neural networks, …

Some possible options:

- Binary encoding: if the feature has only two possible values (e.g., TRUE/FALSE, yes/no), encode the positive examples as 1s and the negative examples as 0s
- One-hot encoding: if the feature has >2 possible values

One-hot encoding

A **one-hot encoding** is a vector-based representation of a categorial feature value

A one-hot vector has one element per level of the categorical feature.

For example, if a feature can take three levels (e.g., low, medium, high), then the vector would have three elements

It is known as a one-hot representation because at most one element in the vector will have the value 1 and all the other elements will be 0, with the value of the feature indicated by whichever element is 1.

For our three-level categorical feature we might decide that:

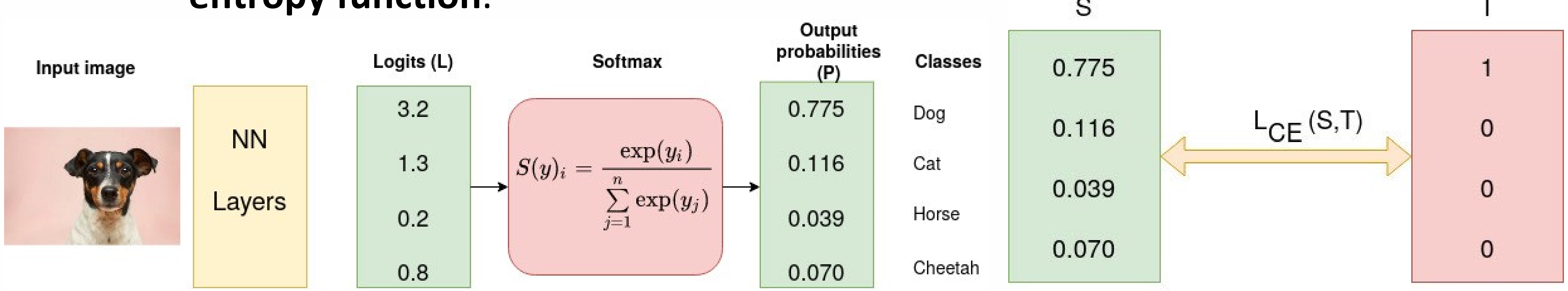
- [1, 0, 0] indicates low
- [0, 1, 0] indicates medium
- [0, 0, 1] indicates high

Handling categorical target features

All the examples that we have looked at so far have been regression problems

To create a neural network that can predict a multi-level categorical feature, we make three adjustments:

1. we represent the target feature using one-hot encoding
2. we change the output layer of the network to be a **softmax layer**
3. we change the error (or loss) function we use for training to be the **cross-entropy function**.



Classification example - Powerplant Dataset

Note that the target feature Electrical Output has been **one hot encoded**

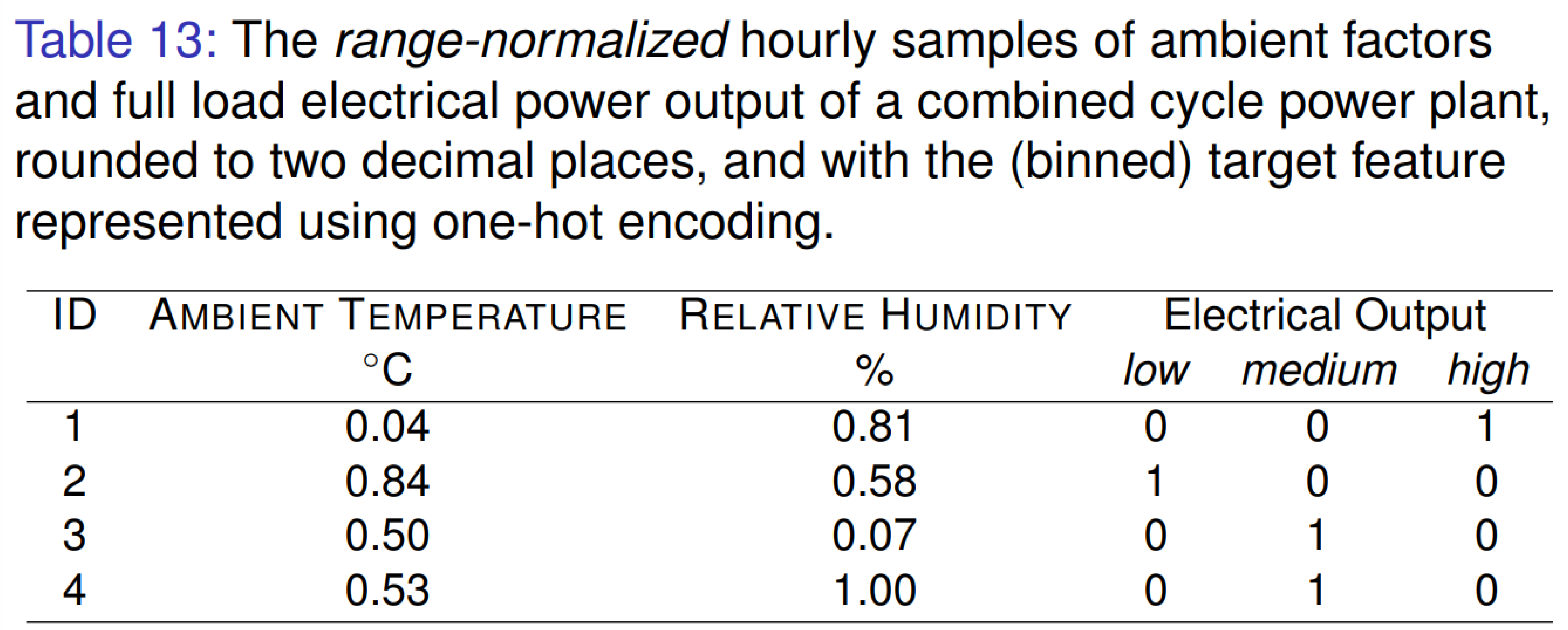


Table 13: The *range-normalized* hourly samples of ambient factors and full load electrical power output of a combined cycle power plant, rounded to two decimal places, and with the (binned) target feature represented using one-hot encoding.

ID	AMBIENT TEMPERATURE	RELATIVE HUMIDITY	Electrical Output		
	°C	%	*low*	*medium*	*high*
1	0.04	0.81	0	0	1
2	0.84	0.58	1	0	0
3	0.50	0.07	0	1	0
4	0.53	1.00	0	1	0

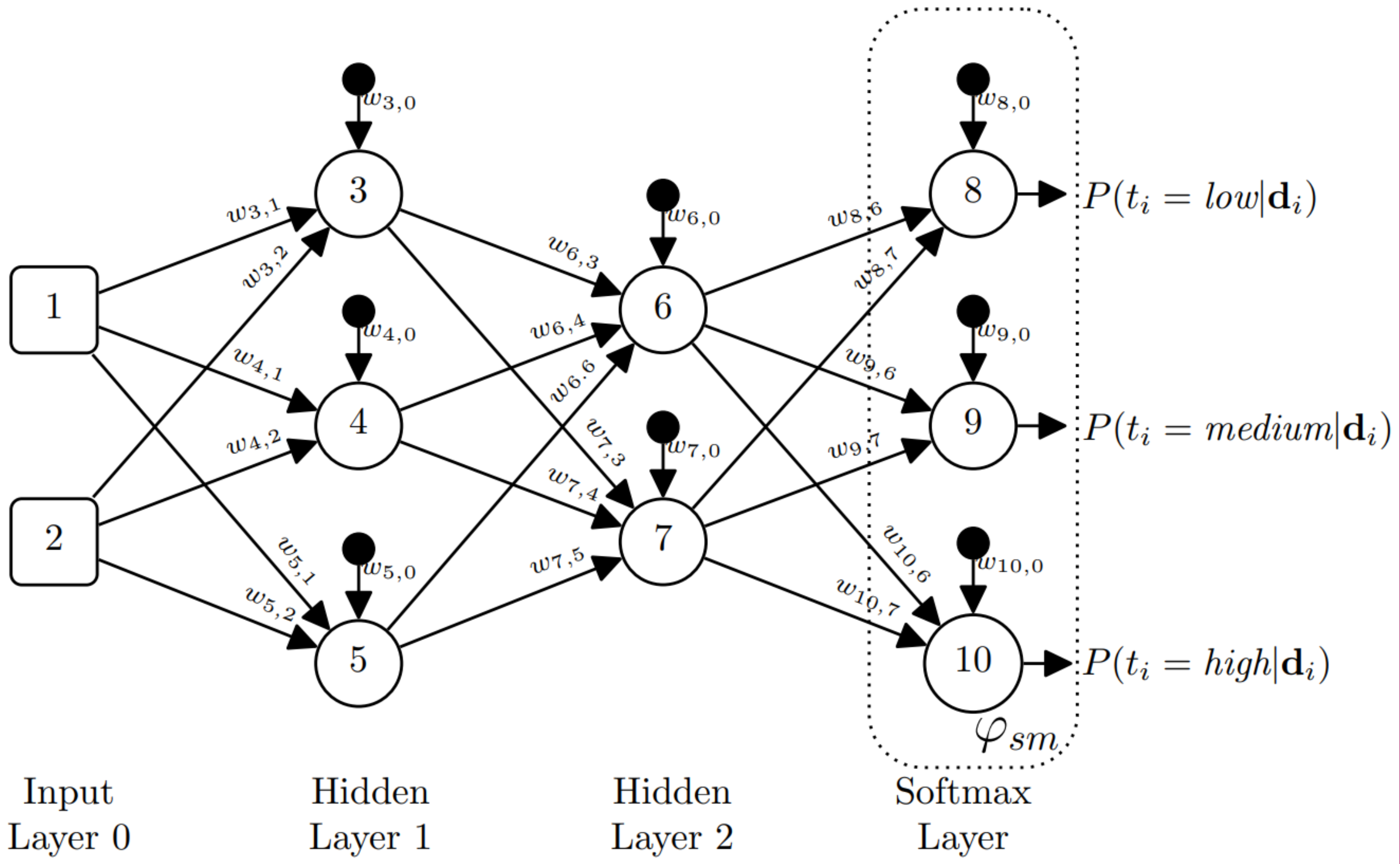
The softmax activation function

$$\varphi_{sm}\left(z_i\right) = \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_m}}$$

In a softmax output layer there is a single neuron for each level of the target feature.
E.g. if the prediction task is to predict the level of a categorical feature that can take three levels (e.g., low, medium, high), then the output layer of the network would have three neurons.
The activation function used by the neurons in a softmax layer is the softmax function; for an output layer with m neurons, the softmax activation function is defined as above
The softmax activation function normalizes the z scores for a layer of neurons so that the sum of the activations of the neurons is 1
The softmax function will always return a positive value for every neuron because e z is always positive, even if z is negative

Example neural network for the Powerplant Dataset

Because softmax returns a normalized set of positive values across the output layer, we can interpret the activation of each neuron in the layer as a probability.
There is one neuron in an output softmax layer per target feature level, and so the softmax function returns one probability per level.
The final answer is the level whose neuron predicts the highest probability



Libraries for neural networks

Scikit-learn can be used to create basic MLPs for classification and regression tasks

E.g., `sklearn.neural_network.MLPClassifier`

- https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

E.g., `sklearn.neural_network.MLPRegressor`

- https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

Scikit-learn does not have GPU acceleration, and lacks many of the features of full dedicated deep learning libraries

More advanced deep learning libraries that include GPU acceleration include:

- Tensorflow/Keras
- Deeplearning4j