AI and the brain

Dr. Charles Kind

Bristol University

May 2023

Summary of learning objectives

- What can you do that computers cannot?
- What is a perceptron?
- The corollary of biological neurons in the artificial sense.
- The structure of a computational neural network.
- Activation functions.
- How to compute a basic neural network.
- How neural networks learn.
- Gradient descent.
- Back-propagation.
- The calculus of neural network learning.

What is AI?

- Artificial intelligence is the simulation of human intelligence processes by machines.
  - Applied AI is using AI in the real world.
  - Today the predominant AI tool is the neural network.

What is AI?

- Artificial intelligence is the simulation of human intelligence processes by machines.
  - Applied AI is using AI in the real world.
  - Today the predominant AI tool is the neural network.
- How can AI be applied? What can you do that a machine cannot, are these good examples?
  - Drive a car
  - Pick a face out of a crowd
  - Design a new mechanical device
  - Write my homework
  - Feel angry or happy

What is AI?

- Artificial intelligence is the simulation of human intelligence processes by machines.
  - Applied AI is using AI in the real world.
  - Today the predominant AI tool is the neural network.
- How can AI be applied? What can you do that a machine cannot, are these good examples?
  - Drive a car ✓
  - Pick a face out of a crowd ✓
  - Design a new mechanical device ✓
  - Write my homework ✓
  - Feel angry or happy ✗

What is AI?

Driverless car in
San Francisco, 2023

What is AI?

Driverless car in San Francisco, 2023

NASA AI designed space antenna, 2006

What is AI?

Driverless car in San Francisco, 2023

NASA AI designed space antenna, 2006

Chat GPT VS turnitin™

“Empirically, we show that paraphrasing attacks, where a light paraphraser is applied on top of the generative text model, can break a whole range of detectors”, arXiv:2303.11156v1

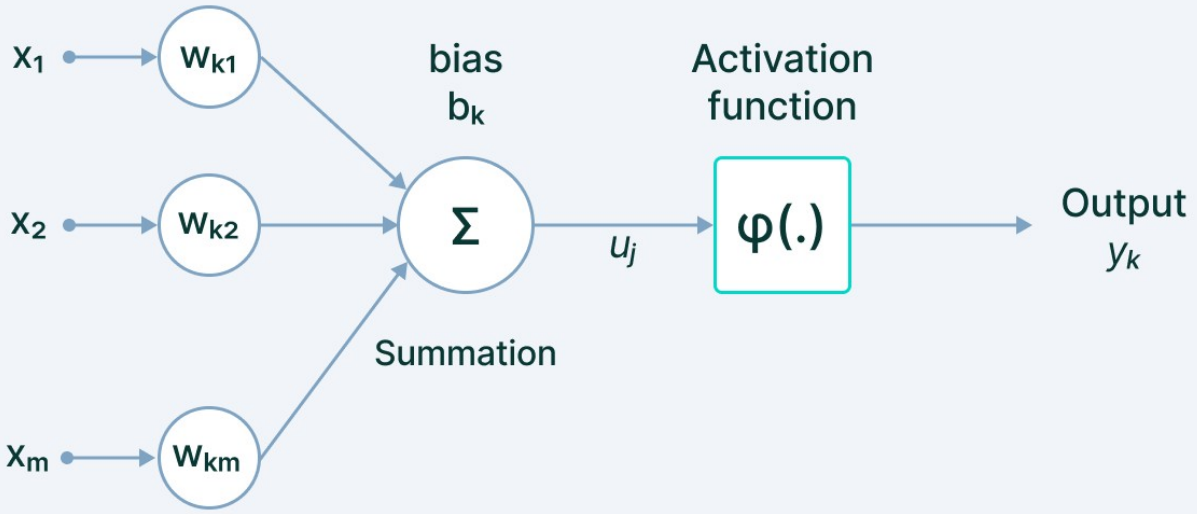
Quiz: which cars are real?

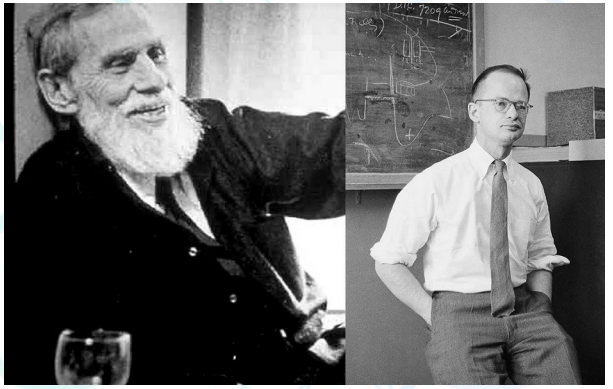
Quiz: which cars are real?

Stable Diffusion Prompt: (highly detailed:1.2),(best quality:1.2),(8k:1.2), photo of a sports car on the road, from the side, highly detailed background, (DOF), overcast sky

Modeling a Neuron

- In 1943 Warren McCulloch and Walter Pitts described the first mathematical model of the neuron.

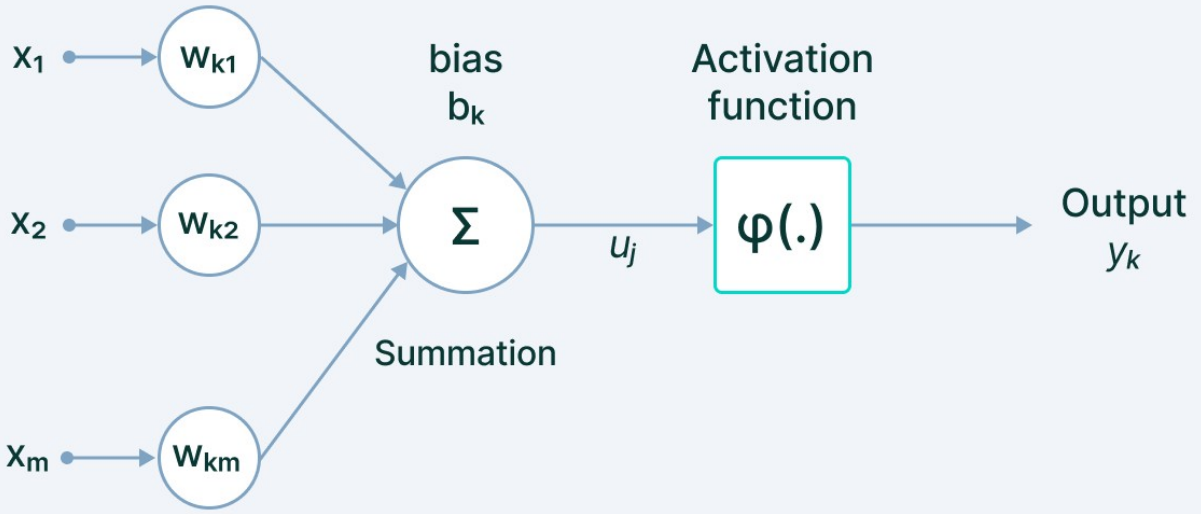


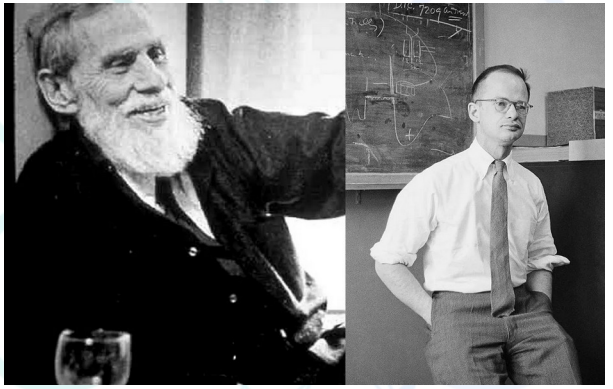


McCulloch and Pitts

Modelling a Neuron

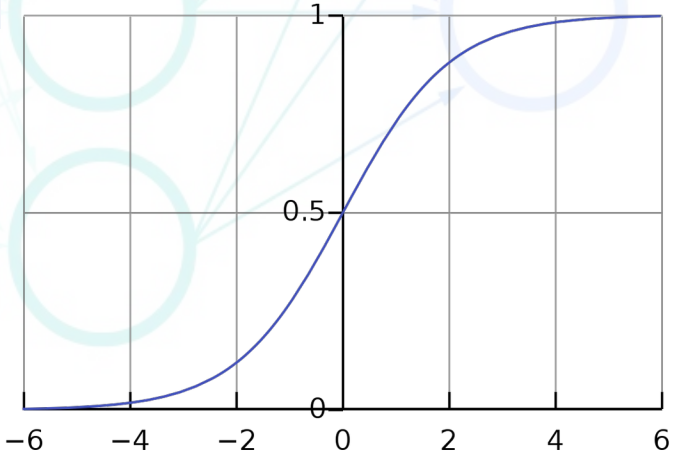
- In 1943 Warren McCulloch and Walter Pitts described the first mathematical model of the neuron.





McCulloch and Pitts

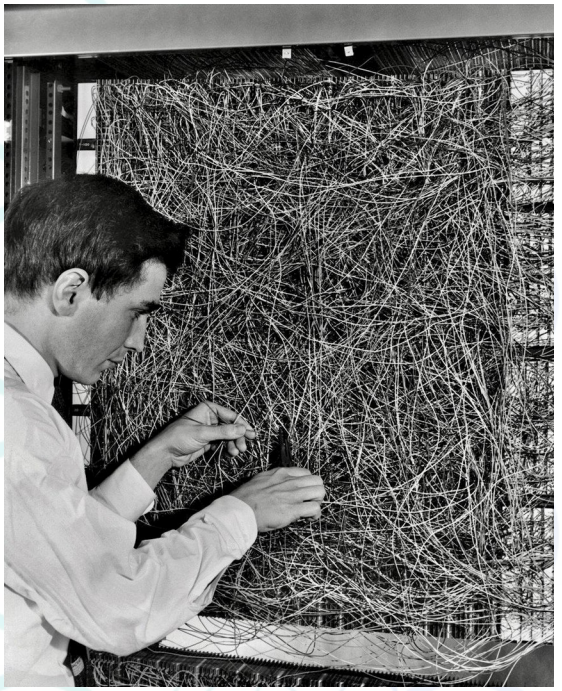
- Input weights, W_{km}, are summed
- When the summation reaches a threshold, defined by the activation function, the neuron fires.



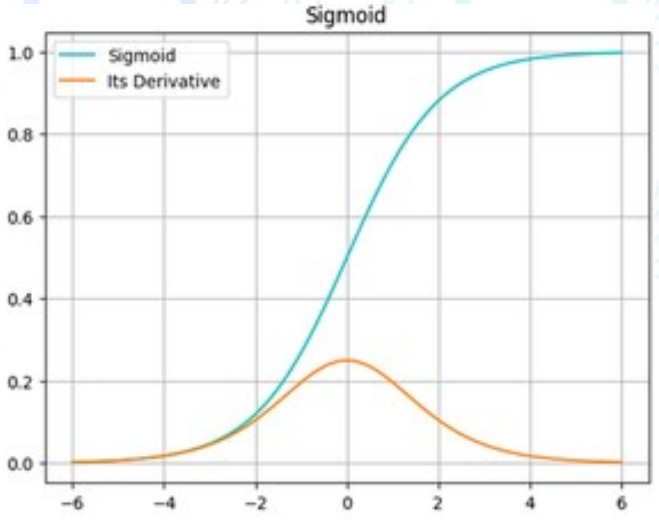
Example activation function

Perceptron

- The perceptron, or McCulloch Pitts neuron, first hardware implementation was created by Frank Rosenblatt in 1957.
- Despite great promise it was shown that Rosenblatt's perceptron could not be trained to recognise many classes of patterns … why?
- Single layer perceptrons are only capable of learning linearly separable patterns.
- Enter the long dark tea-time of neural networks!



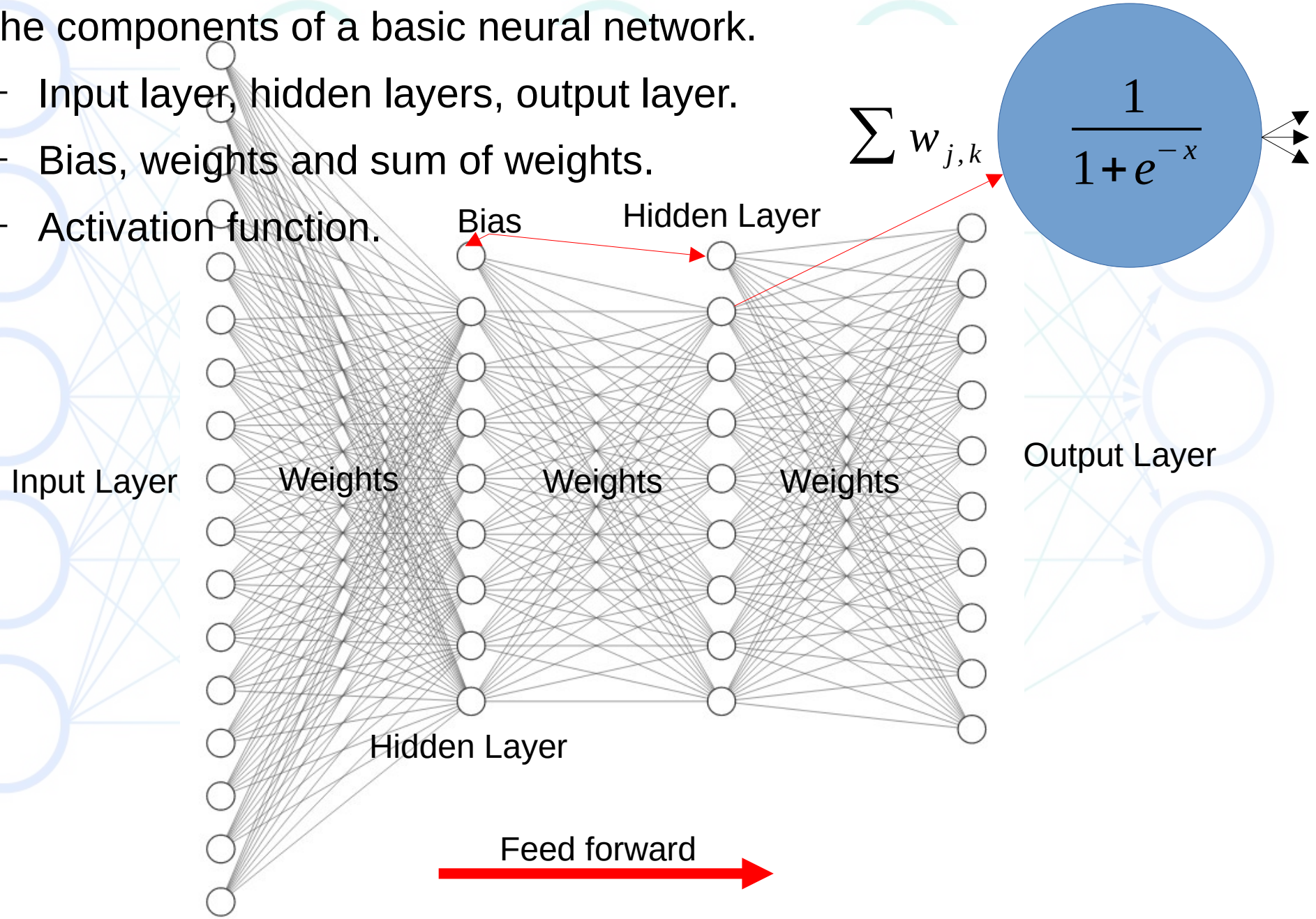
Frank Rosenblatt 1960



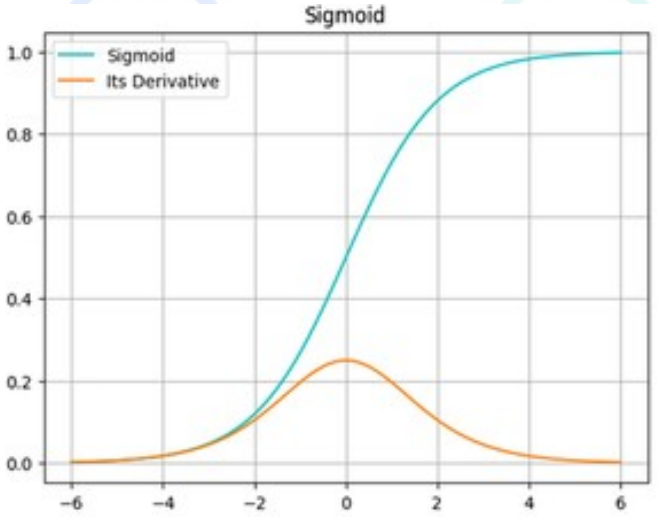
- Sigmoids, back-propagation and the 1980's
- Multiple cores and GPU's
- Computational neural networks are large linear algebra problems.

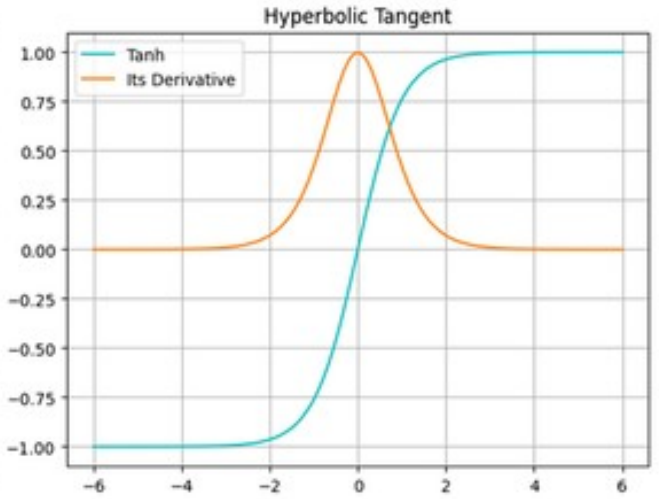
What is a neural network?

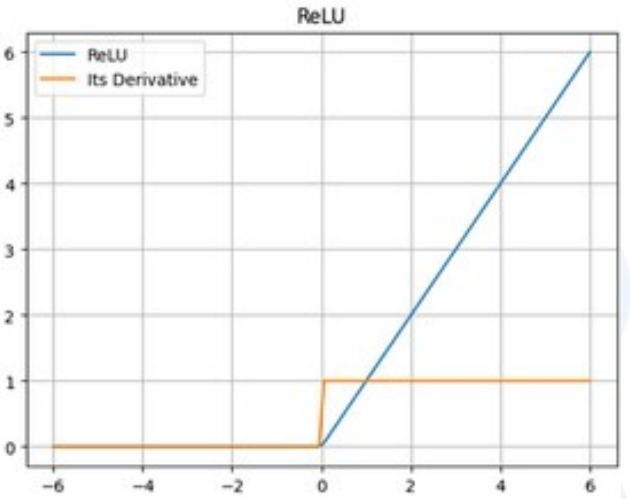
- The components of a basic neural network.
  - Input layer, hidden layers, output layer.
  - Bias, weights and sum of weights.
  - Activation function.



Activation functions







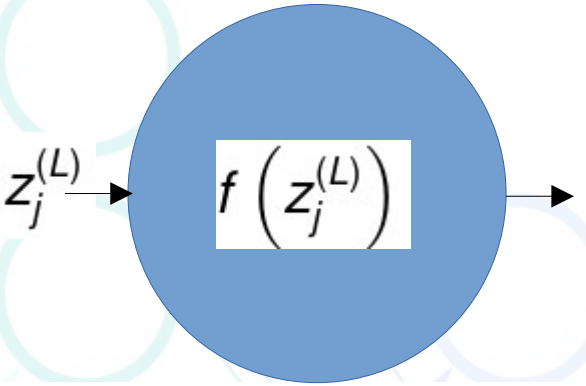
- Activation functions are typically non-linear, why?
  - Because we are modelling non-linear data (in general)
- Linear functions can only produce linear distinctions.
- But what about ReLU, it's ALMOST linear?

Linear =>

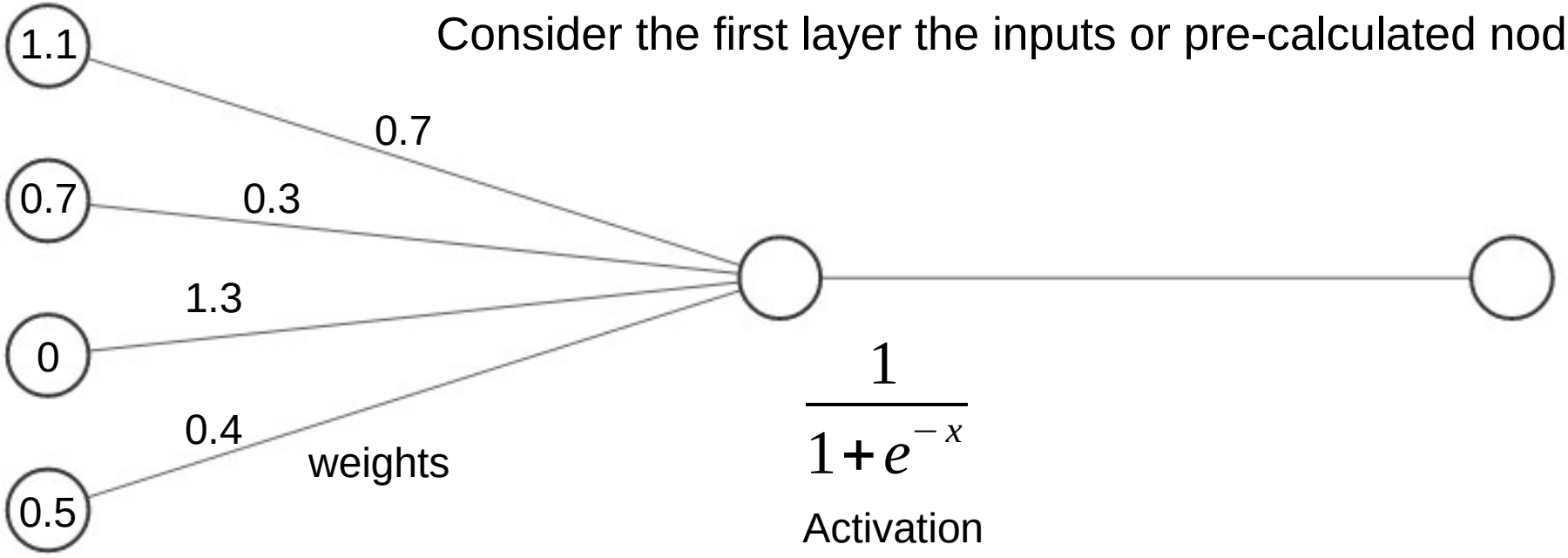
$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$$
$$f(a\mathbf{x}) = af(\mathbf{x}).$$

How a neural network works.

- A simple example.
- Given the network below let us calculate the output of the ‘neuron’
- The logistic function will be used as the activation



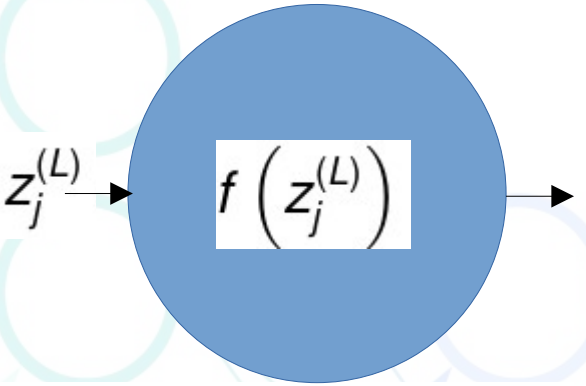
Consider the first layer the inputs or pre-calculated nodes

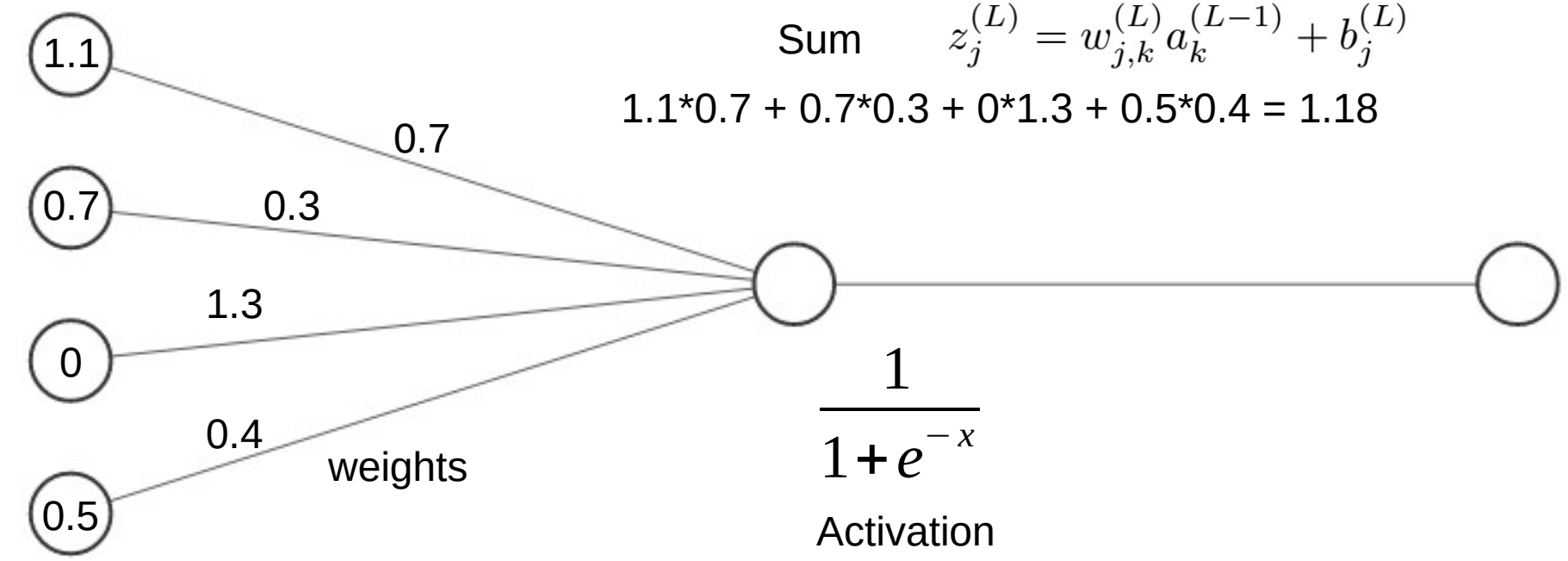


Inputs

How a neural network works.

- A simple example.
- Given the network below let us calculate the output of the ‘neuron’
- The logistic function will be used as the activation





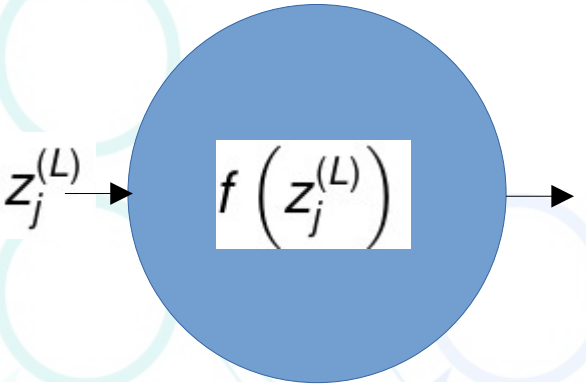
Sum $z_j^{(L)} = w_{j,k}^{(L)} a_k^{(L-1)} + b_j^{(L)}$

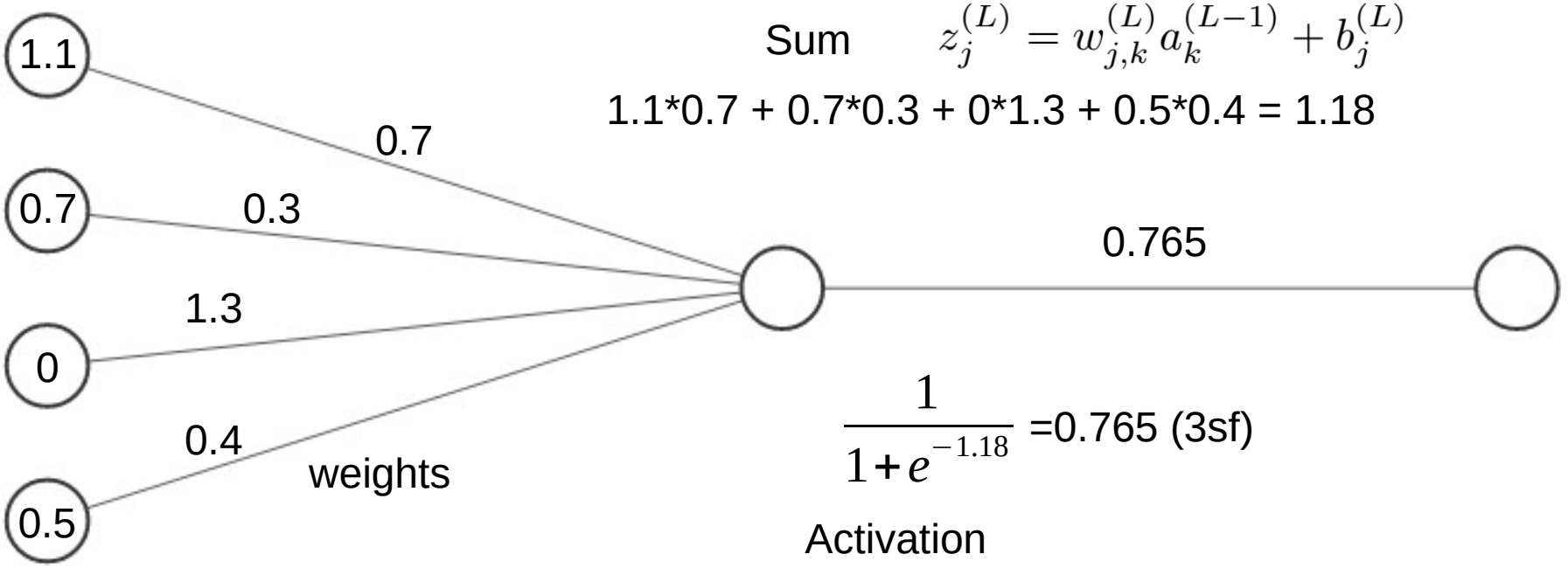
1.1*0.7 + 0.7*0.3 + 0*1.3 + 0.5*0.4 = 1.18

Inputs

How a neural network works.

- A simple example.
- Given the network below let us calculate the output of the ‘neuron’
- The logistic function will be used as the activation





Sum $z_j^{(L)} = w_{j,k}^{(L)} a_k^{(L-1)} + b_j^{(L)}$

1.1*0.7 + 0.7*0.3 + 0*1.3 + 0.5*0.4 = 1.18

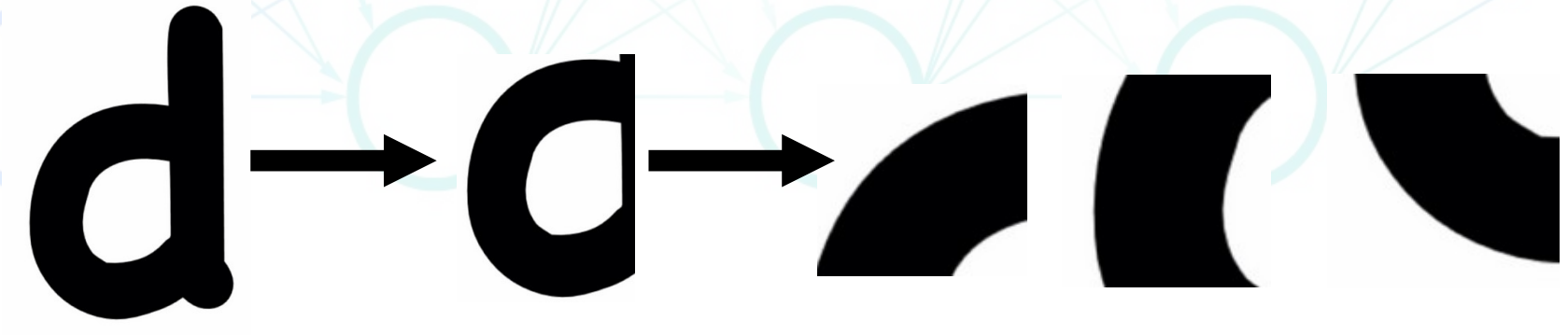
$\frac{1}{1+e^{-1.18}}$ =0.765 (3sf)

Activation

$a_j^{(L)} = f\left(z_j^{(L)}\right)$

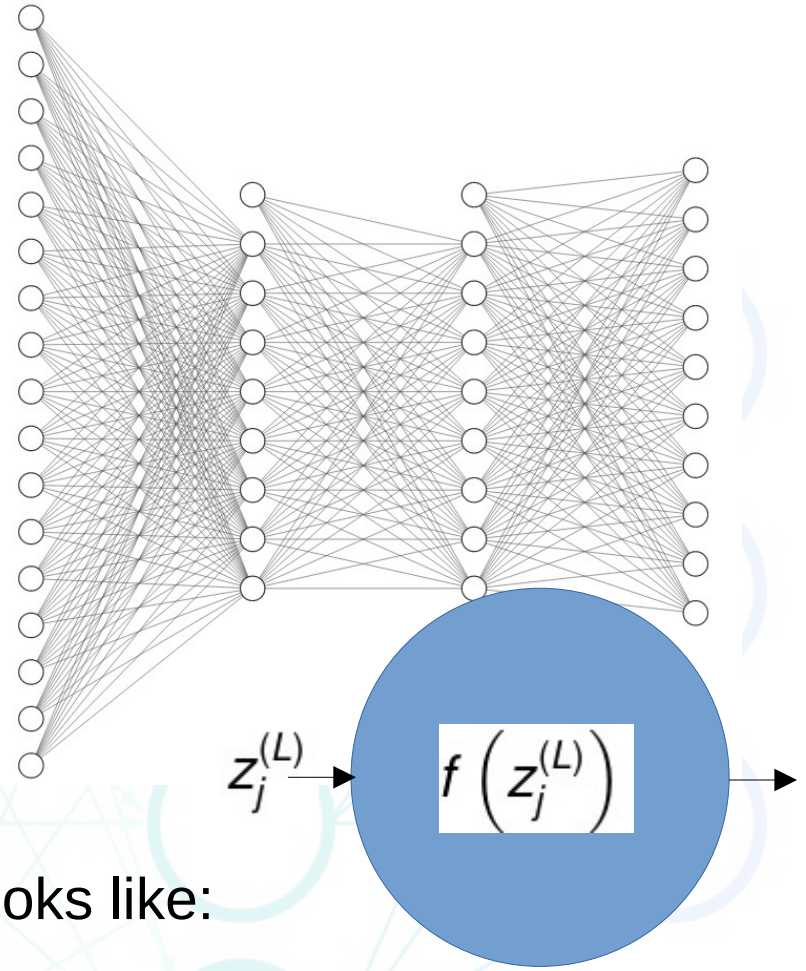
Hidden layers: a handwavy (bad?) description

- Hidden layers what are they and why bother?
- Single layer perceptrons are only capable of learning linearly separable patterns.
- The Universal Approximation Theorem: “a feedforward neural network with a single hidden layer and a non-linear activation function (under certain conditions) can approximate any continuous function.”
- Imagine you want to identify specific features of the data you are using.
  - i.e. Imagine you distil the primary shape characteristics of alphabet letters, say loops and whirls and lines
  - Then imagine that there are eight of them in this case
  - Then imagine that those primary characteristics are composed of other characteristics, say smaller parts
- Could we enforce this structure on a neural network?



How a neural network works.

- What parameters do we have?
- Inputs – real numbers
- Weights $w_{j,k}^{(L)}$ where (L) is layer j is node and k is weight
- Inputs to nodes $z_j^{(L)} = w_{j,k}^{(L)} a_j^{(L-1)} + b^{(L)}$ where z is the input, a is the activation from the previous layer and b is the bias
- Activation of a node $a_j^{(L)} = f\left(z_j^{(L)}\right)$ where f() is the activation function



Hence in matrix form the activation of a layer looks like:

$$\begin{bmatrix} a_0^{(L)} \\ a_1^{(L)} \\ \vdots \\ a_n^{(L)} \end{bmatrix} = f\left(\begin{bmatrix} w_{0,0}^{(L)} & w_{0,1}^{(L)} & \dots & w_{0,n}^{(L)} \\ w_{1,0}^{(L)} & w_{1,1}^{(L)} & \dots & w_{1,n}^{(L)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0}^{(L)} & w_{k,1}^{(L)} & \dots & w_{k,n}^{(L)} \end{bmatrix} \begin{bmatrix} a_0^{(L-1)} \\ a_1^{(L-1)} \\ \vdots \\ a_n^{(L-1)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right)$$

or

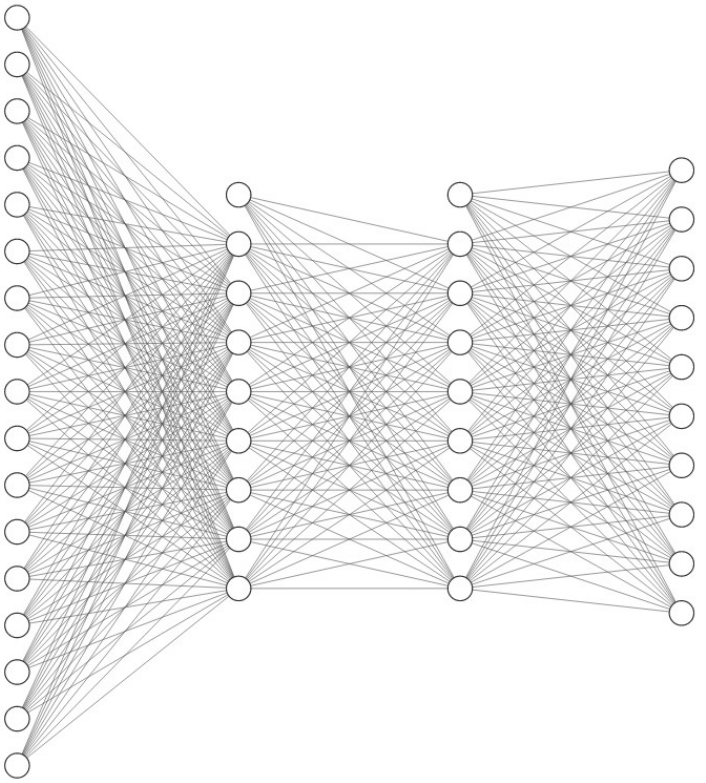
$$a^{(L)} = f\left(W a^{(L-1)} + b\right)$$

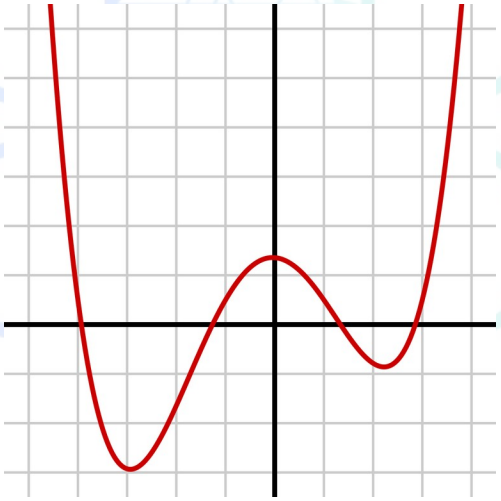
How wrong is my network?

- We can compare our output with the desired output
- In this example we assume we know what that is for example the character ‘a’
- We call the measure of expectation vs output the cost function, eg:

$$C = \sum (\text{output} - \text{expectation})^2$$

- This is the most common measure of the cost, or error, of the network.





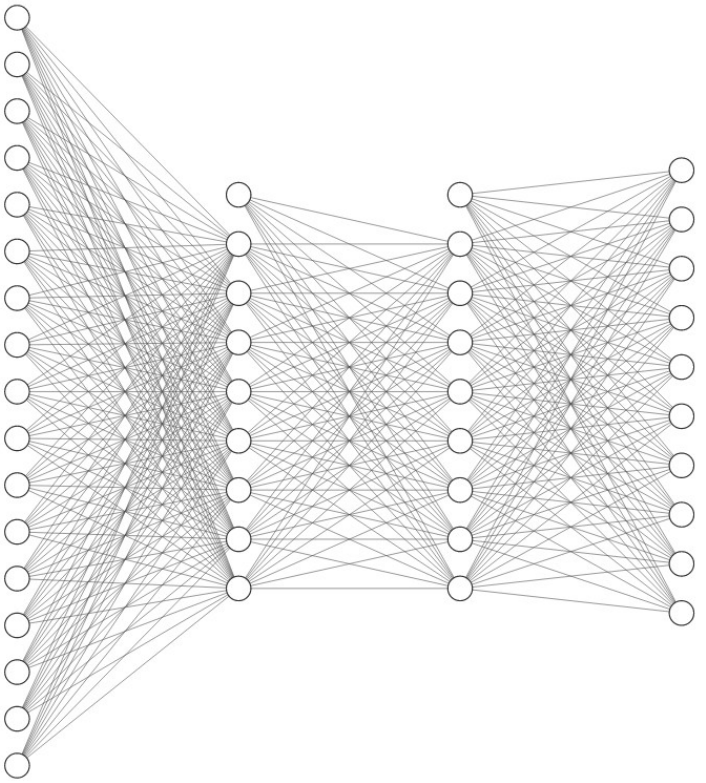
- We would like to be able to minimise our error in some way.
- How do we find the minima of functions?
- What if the function is ‘complicated’?

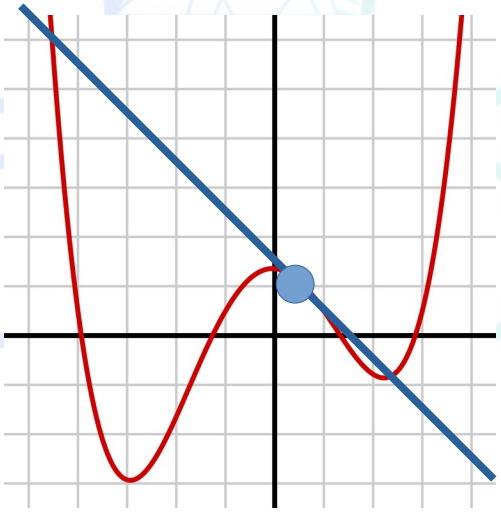
How wrong is my network?

- We can compare our output with the desired output
- In this example we assume we know what that is for example the character ‘a’
- We call the measure of expectation vs output the cost function, eg:

$$C = \sum (\text{output} - \text{expectation})^2$$

- This is the most common measure of the cost, or error, of the network.





- We would like to be able to minimise our error in some way.
- How do we find the minima of functions?
- What if the function is ‘complicated’?
- We can use gradient descent.

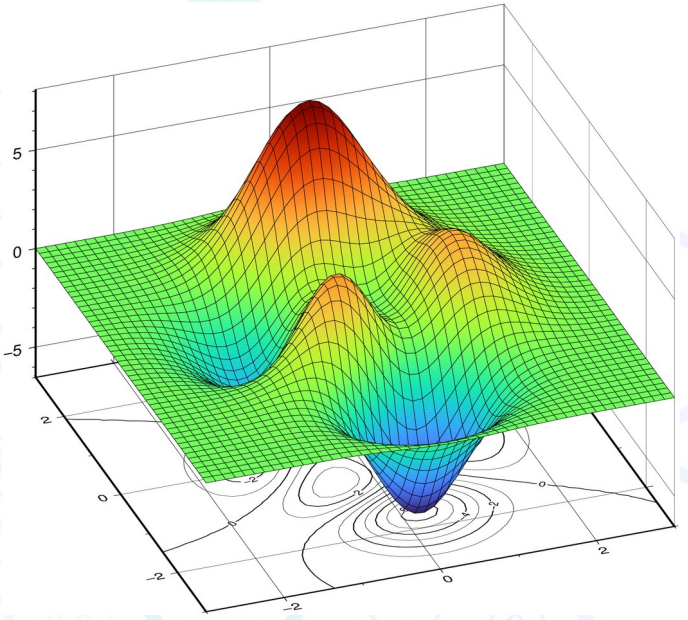
Minimising the network error.

- In more than one dimension we find

$$\nabla C\left(e_1, e_2, e_3 ... e_n\right)$$

  which gives us the direction of steepest ascent, hence it's negative gives us the steepest descent.

- We therefore require a smooth surface to traverse. Why?
- Is this like the human brain?
- If we can find the gradient with respect to the weights … what can we do?



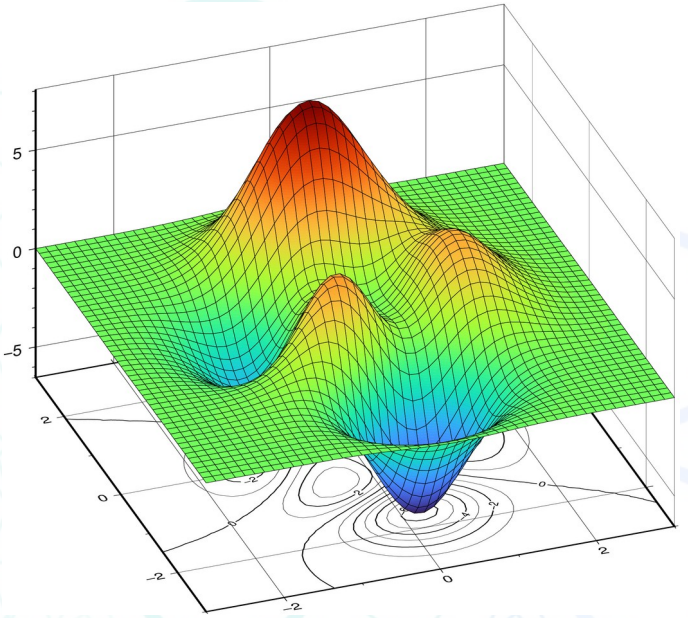
Minimising the network error.

- In more than one dimension we find

$$\nabla C\left(e_1, e_2, e_3 ... e_n\right)$$

  which gives us the direction of steepest ascent, hence it's negative gives us the steepest descent.

- We therefore require a smooth surface to traverse. Why?
- Is this like the human brain?
- If we can find the gradient with respect to the weights … what can we do?

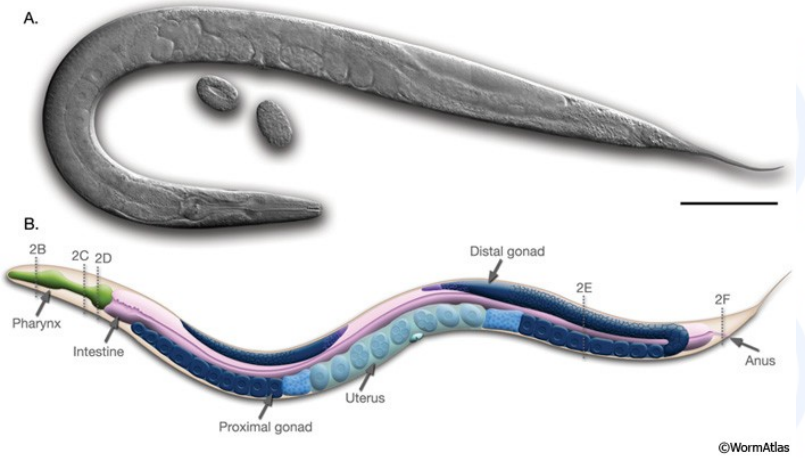


$$-\nabla C\left(W\right) = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$$

- We find the negative of the gradient.
- This tells us which weights should increase or decrease and by how much relative to the other weights.
- Now we can train our network.

Neural network vs biological brain

- Remember the hand wavy explanation? If you train a network similar to the one we have been looking at to recognise letters the hidden layers DO NOT act in this way! Why?
- Perhaps if we where as simple as a nematode (302 neurons)
- Biological neuron firing is binary
- Dendrites apply a non-linear function to their input
- Frequency of firing ‘stores’ information in a biological neural network

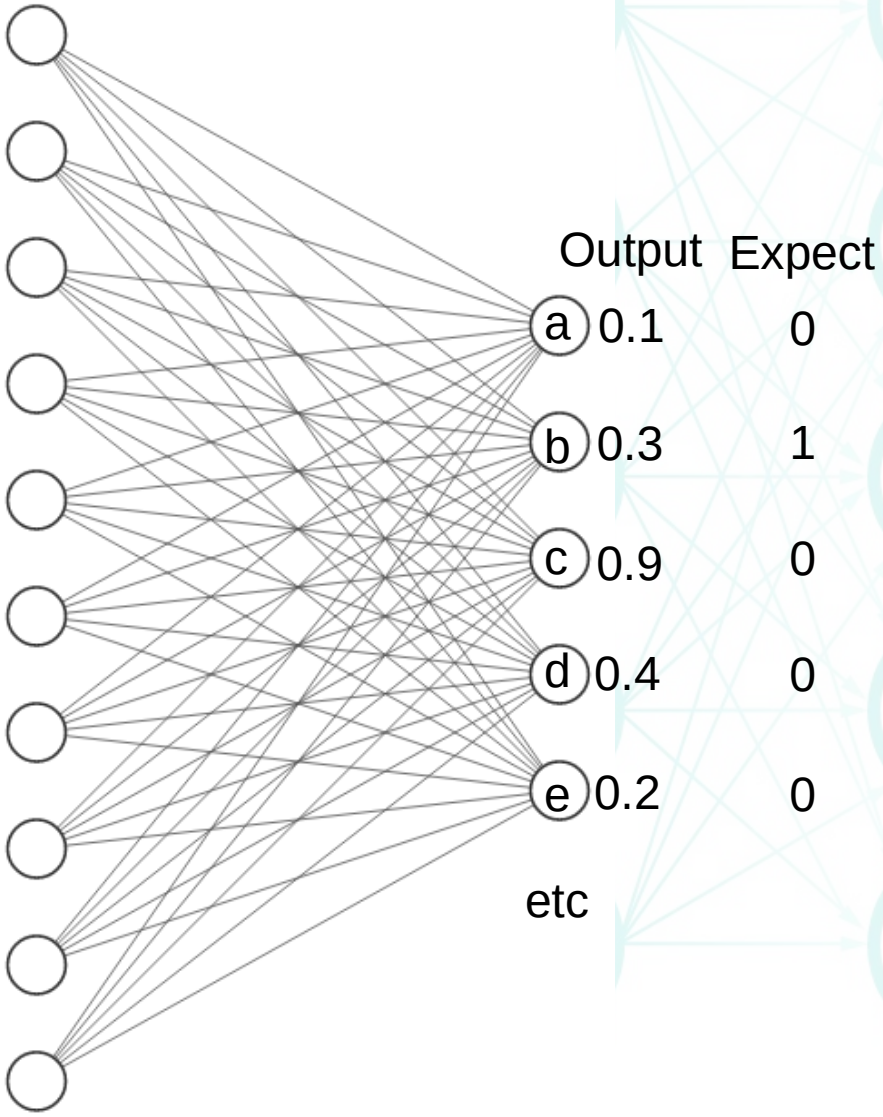


C. elegans hermaphrodite

- The same input to a computational neural network will produce the same output
- Size … matters! Your brain has around 100 billion neurons.
- Fault tolerance and energy demands … recall neuromorphic lectures.
- Can neural networks help us understand the human brain?

Back propagation

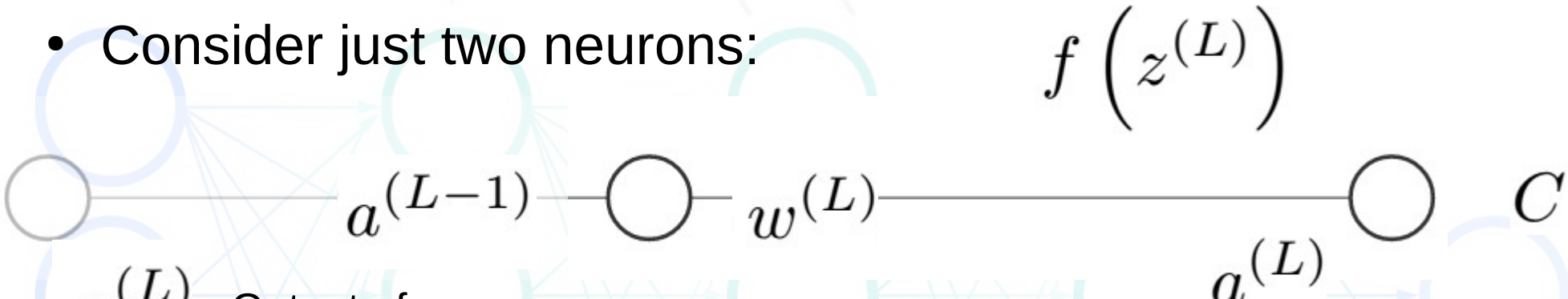
- How should our network learn?



- Let us say we wanted ‘b’ as our output.
- Looking at a single example.
- How can we make the network more likely to give us a ‘b’?
- Adjust the weights, those with higher values, connected to output ‘b’ will have the largest effect.
- Hebbian – wire together and fire together.
- In fact we (should) do this for the entire training set and average the changes then repeat ...
- In reality we often batch the training data by separating the dataset.

Back propagation in calculus

- Consider just two neurons:



$a^{(L)}$ Output of neuron

$w^{(L)}$ Weight

$z^{(L)}$ Summed input

$b^{(L)}$ Bias

$f\left(z^{(L)}\right)$ Activation function

C Cost function

y Expected output

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

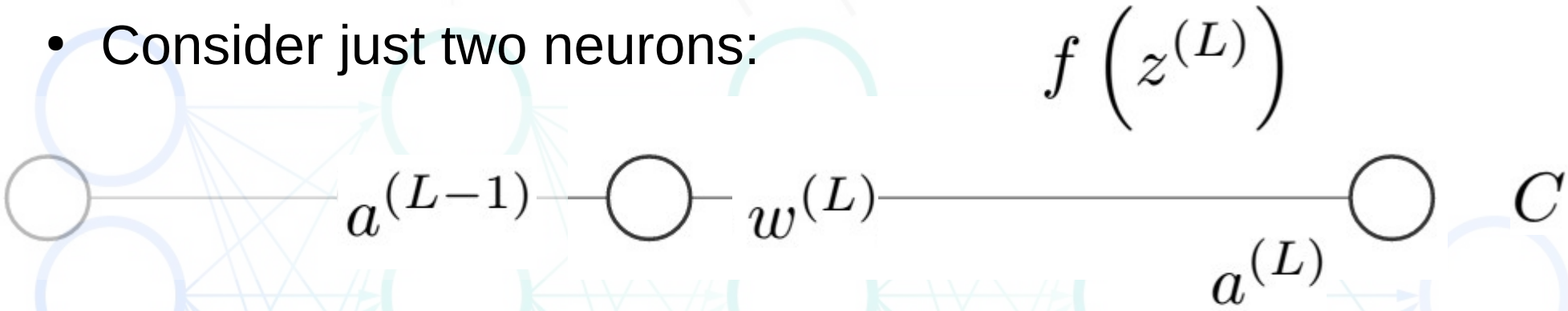
$$a^{(L)} = f\left(z^{(L)}\right)$$

$$C = \left(a^{(L)} - y\right)^2$$

The activation of the previous neuron is multiplied by the weight and summed with the bias. This is fed into the activation function and produces the output of the next neuron.

Back propagation in calculus

- Consider just two neurons:



- We want to know how the weight affects the cost function.

$$\frac{\partial C}{\partial w^{(L)}} \quad \text{where} \quad C = \left(a^{(L)} - y\right)^2$$

- We need to use the chain rule, we can see that $a^{(L)}$ is the variable component of C.

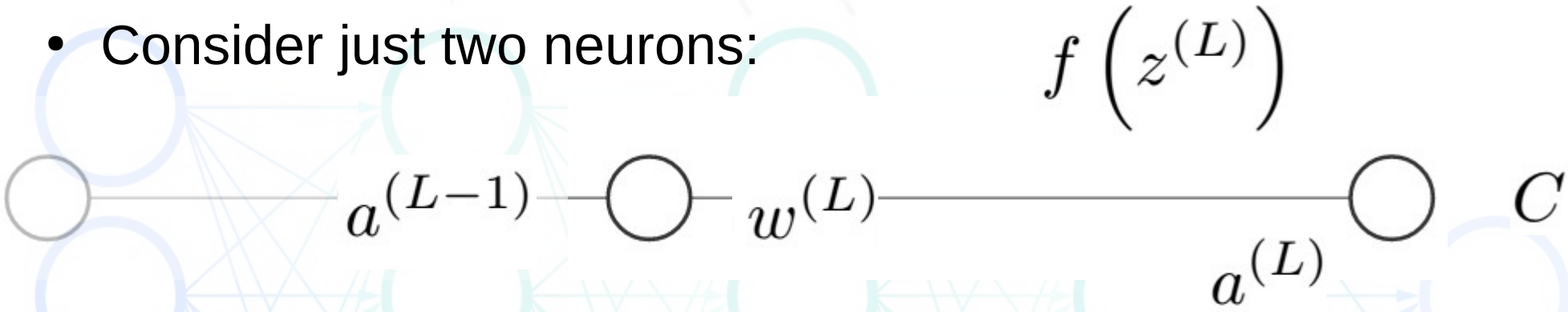
  We can see that $z^{(L)}$ is the variable component of $a^{(L)}$.

  We can see that $w^{(L)}$ is the variable component of $z^{(L)}$.

Hence ...

Back propagation in calculus

- Consider just two neurons:



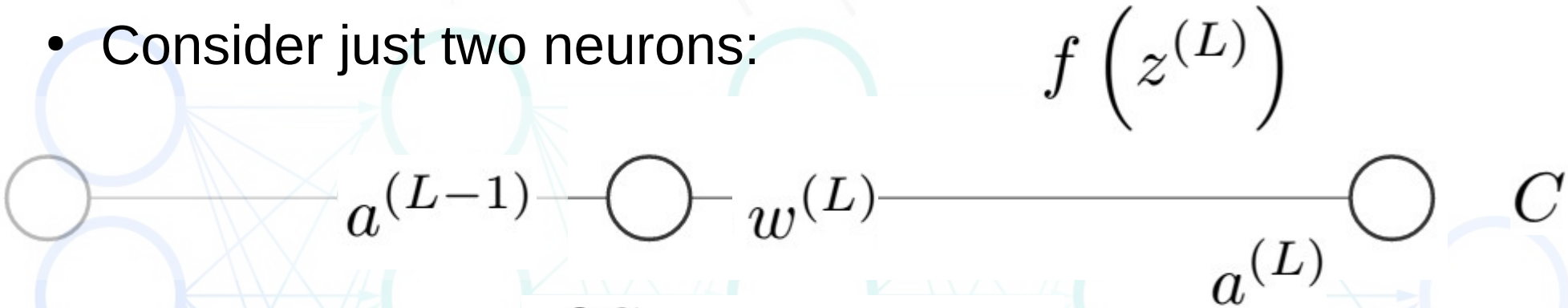
$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}}$$

The rate of change of the cost function for the output node with respect to the previous weight. When we scale this up to arbitrary size networks the main difference is there are lots of summations involved. These days this is all handled by your neural network library such as tensor flow.

Calculating ...

Back propagation in calculus

- Consider just two neurons:



$$\frac{\partial C}{\partial a^{(L)}} = 2\left(a^{(L)} - y\right)$$

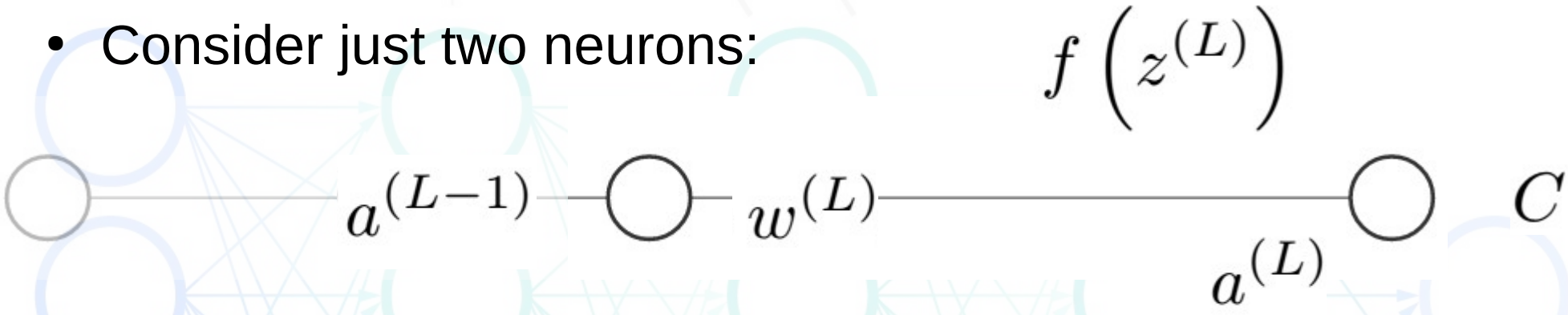
$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = f'\left(z^{(L)}\right)$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

$$\frac{\partial C}{\partial w^{(L)}} = 2\left(a^{(L)} - y\right) f'\left(z^{(L)}\right) a^{(L-1)}$$

Back propagation in calculus

- Consider just two neurons:



Average over all training examples:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}$$

Derivative of full cost function

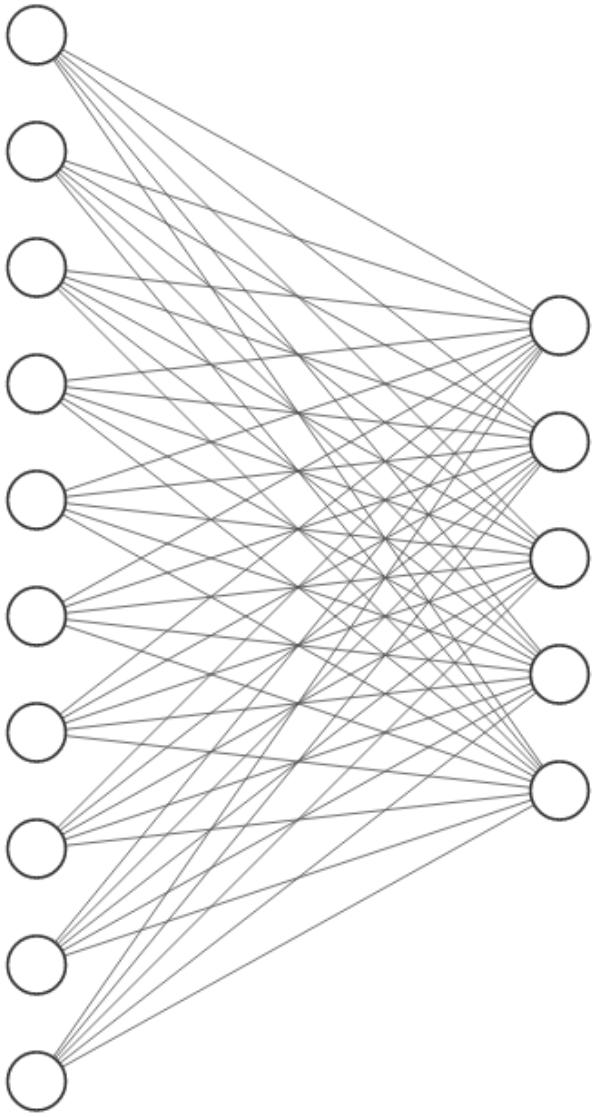
$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix}$$

And finally our gradient vector!

$$\frac{\partial C}{\partial w^{(L)}} = 2\left(a^{(L)} - y\right) f'\left(z^{(L)}\right) a^{(L-1)}$$

Back propagation in calculus

- Consider multiple neurons and layers, we then have:



$$C = \sum_{j=0}^{n_L - 1} \left(a_j^{(L)} - y_j \right)^2$$

$$z_j^{(L)} = w_{j,k}^{(L)} a_k^{(L-1)} + b_j^{(L)}$$

$$a_j^{(L)} = f\left(z_j^{(L)} \right)$$

$$\frac{\partial C}{\partial w_{j,k}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{j,k}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C}{\partial a_j^{(L)}}$$

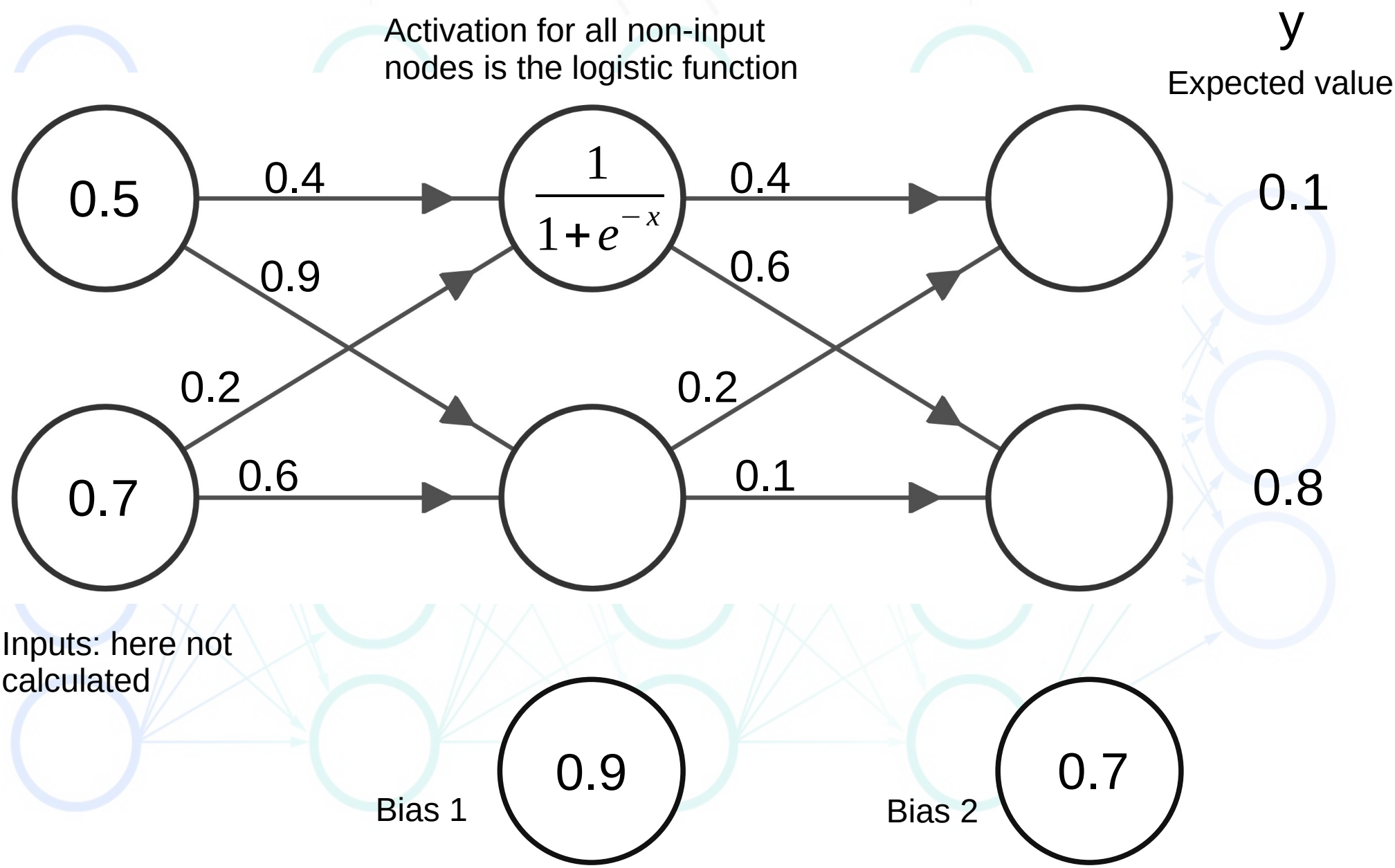
Neural networks over view

- Feed-forward is a simple mechanism involving summing the weights and biases and feeding into an activation function.
- Feed-back is also relatively simple. You feed back through the network using the total derivative to adjust the weights.
  - A training weight is used to adjust the ‘speed’ of training
  - Typically, for large datasets, the training is applied to randomised subsets of the entire data set to minimise calculations
- All of this is obscured from you with modern neural network libraries. You just specify the layers, the bias, the connectivity and the activation functions for each layer.

Neural network issues

- How many hidden layers should you have and how large should they be?
  - No standard answer, more layers more non-linearity
  - As a rule of thumb your hidden layers should be around 1.5X the size of your input layer
  - However, less connections is less computations ...
- How do you deal with large differences in the magnitude of your dataset?
  - If your network is trained on values between [0 … 1] then if you later feed in much larger values what happens?
  - Data may need to be normalised in some sense that does not harshly effect the underlying information content.
- What activation function should I use?
  - ReLU unless you have a very good reason not to.
- What learning rate should I use and what do I do about network ‘stiffness’
  - If you rate is too small the network will take a long time to train. If your rate is too large you may jump around the fitness landscape too much.
  - If you overtrain your network will have problems generalising, you can add in noise or new training data or even retrain from the beginning.
- There are many aspects of neural networks that are poorly understood and/or managed by rules of thumb and trial and error.

An example back-propagation



An example back-propagation

- First we feed forward.
- Sum inputs to middle layer: $z_j^{(L)} = w_{j,k}^{(L)} a_k^{(L-1)} + b_j^{(L)}$

  z_1 = 0.5*0.4 + 0.7*0.2 + 0.9 = 1.24
  z_2 = 0.5*0.9 + 0.7*0.6 + 0.9 = 1.77

- Activation function

  $f(x) = \frac{1}{1+e^{-x}}$

  f(z_1) = 0.776 (3 sf)
  f(z_2) = 0.854 (3 sf)

- That is the middle layer complete!

An example back-propagation

- First we feed forward.
- Sum inputs to middle layer: $z_j^{(L)} = w_{j,k}^{(L)} a_k^{(L-1)} + b_j^{(L)}$

  z_1 = 0.5*0.4 + 0.7*0.2 + 0.9 = 1.24
  z_2 = 0.5*0.9 + 0.7*0.6 + 0.9 = 1.77

- Activation function

  $f(x) = \frac{1}{1+e^{-x}}$

  f(z_1) = 0.776 (3 sf)
  f(z_2) = 0.854 (3 sf)

- That is the middle layer complete!
- After calculating the final layer the same way we have outputs:

  Output of node 0: 0.765
  Output of node 1: 0.777

An example back-propagation

- Now we feed back and find: $$\frac{\partial C}{\partial w_{j,k}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{j,k}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C}{\partial a_j^{(L)}}$$
- We know: $$\frac{\partial z_j^{(L)}}{\partial w_{j,k}^{(L)}} = a_k^{(L-1)}$$

$$\frac{\partial C}{\partial a_j^{(L)}} = 2\left(a_j^{(L)} - y\right)$$

$$\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = f'\left(z_j^{(L)}\right)$$

- All of these values we have to hand apart from $f'\left(z_j^{(L)}\right)$

An example back-propagation

- $\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = f'\left(z_j^{(L)}\right)$
- For the logistic function f’(x) = f(x)*(1-f(x)) which makes the calculation a little more simple.
- Hence for the weight w_{0,0} we seek:

$a_k^{(L-1)}$ where k=0 is the output of node 0 of layer 1, the hidden layer: = 0.776

$2\left(a_j^{(L)} - y\right)$ for j=0 is: 2*(0.765-0.1) = 1.33

$f'\left(z_j^{(L)}\right)$ for j=0 is 0.765*(1-0.765) = 0.18

$\frac{\partial C}{\partial w_{j,k}^{(L)}}$ = 0.776*1.33*0.18 = 0.185

An example back-propagation

- Hence for all the weights in between the hidden and the output layers we have:
- $\frac{\partial C}{\partial w_{j,k}^{(L)}}$ is for {j,k} = {0,0}: 0.185

  {j,k} = {0,1}: -0.006

  {j,k} = {1,0}: 0.204

  {j,k} = {1,1}: -0.007

- I recommend that you check these values are correct and work through the example yourselves.

AI: further reading

- ‘Artificial Intelligence: A Modern Approach’, Russell and Norvig
- ‘Deep Learning’, Goodfellow et al.
- ‘Deep Learning with Python’, Chollet
- ‘Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems’, Geron

Available free online

- ‘Neural Networks and Deep Learning’, Nielsen
- ‘An Introduction to Statistical Learning’, Gareth James et al.