

CA318

Advanced Algorithms and AI Search

Theory of Neural Networks



Theory of Neural Networks

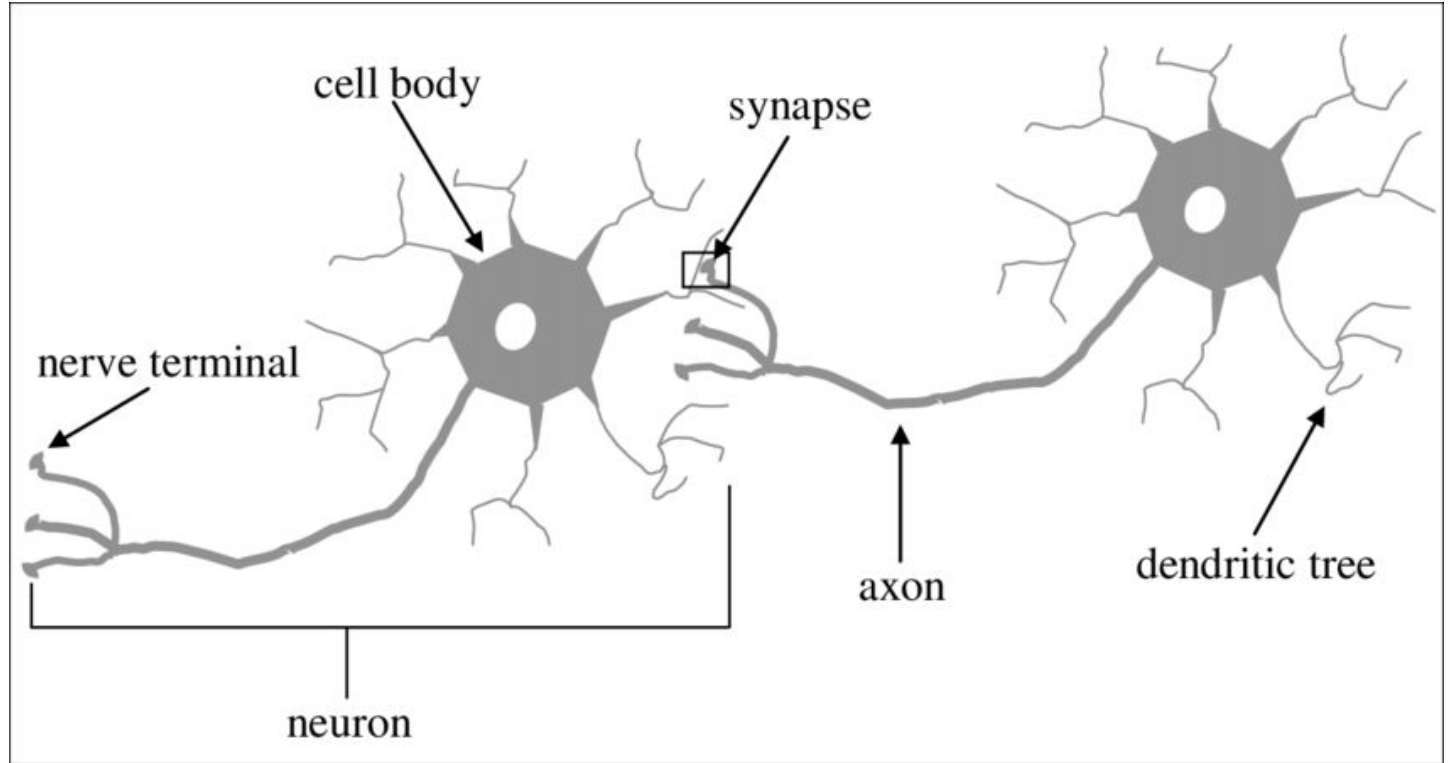
Biologically Inspired Learning

Biologically Inspired Learning

- Although we have studied many elegant methods of
 - searching
 - optimization
 - classification
- We still have not adequately explained how **learning** arises
 - The human brain learns well
 - So why not try to model this in software
- This is the goal of Neural Networks and Deep Learning

Neuron Structure

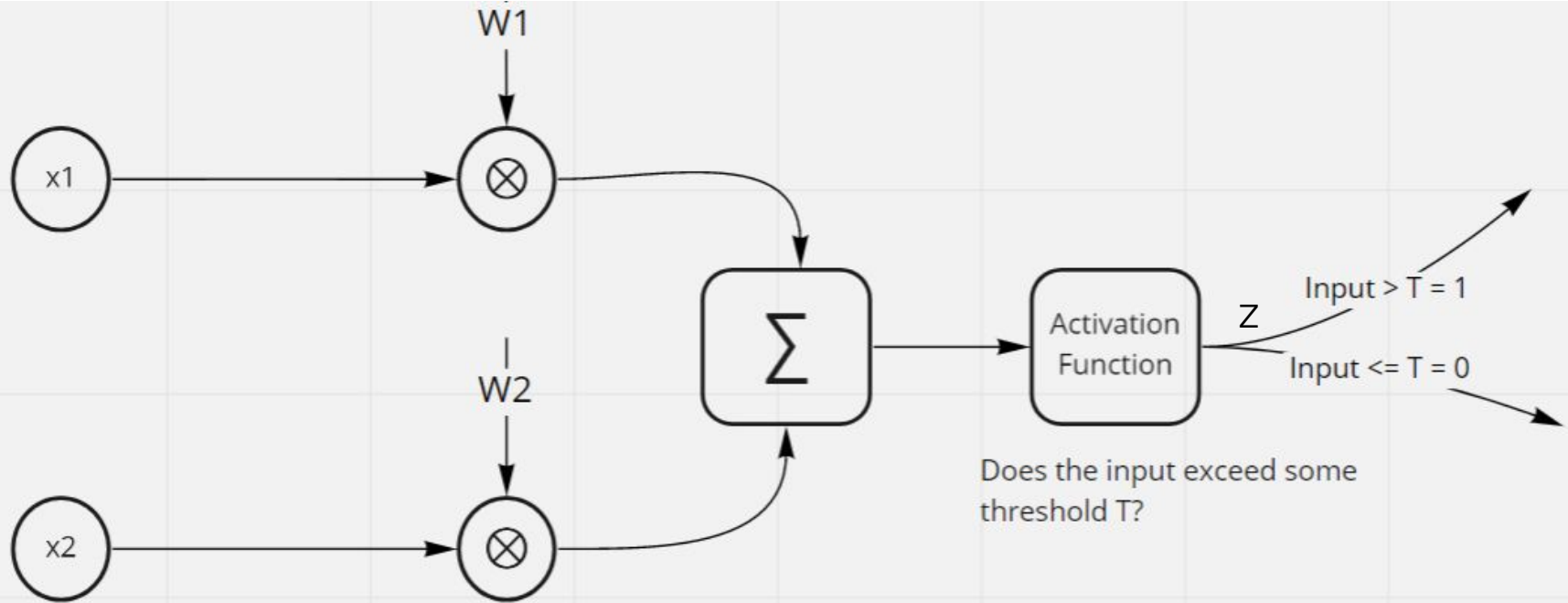
There are
86 billion
neurons in
the human
brain



Modelling a Neuron

- Neurons firing is binary
 - Given some stimulus event
 - They either fire \otimes
 - or do not fire
- When neurons connect to each other, they do so with some level of strength
 - Synaptic Strength
- Lets consider how two neurons might be connected in our model

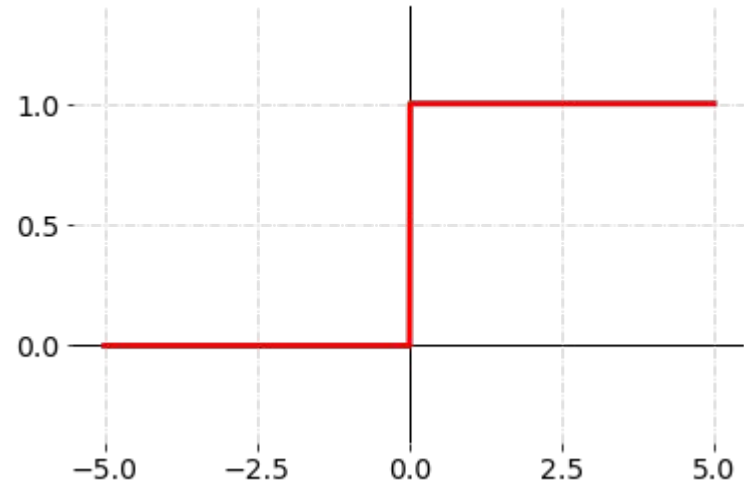
Two Neurons Connected



Activation Function

The activation function defined in this initial example is a binary activation (step) function.

This is the most simplistic approach possible, but we shall see later, it is not very desirable



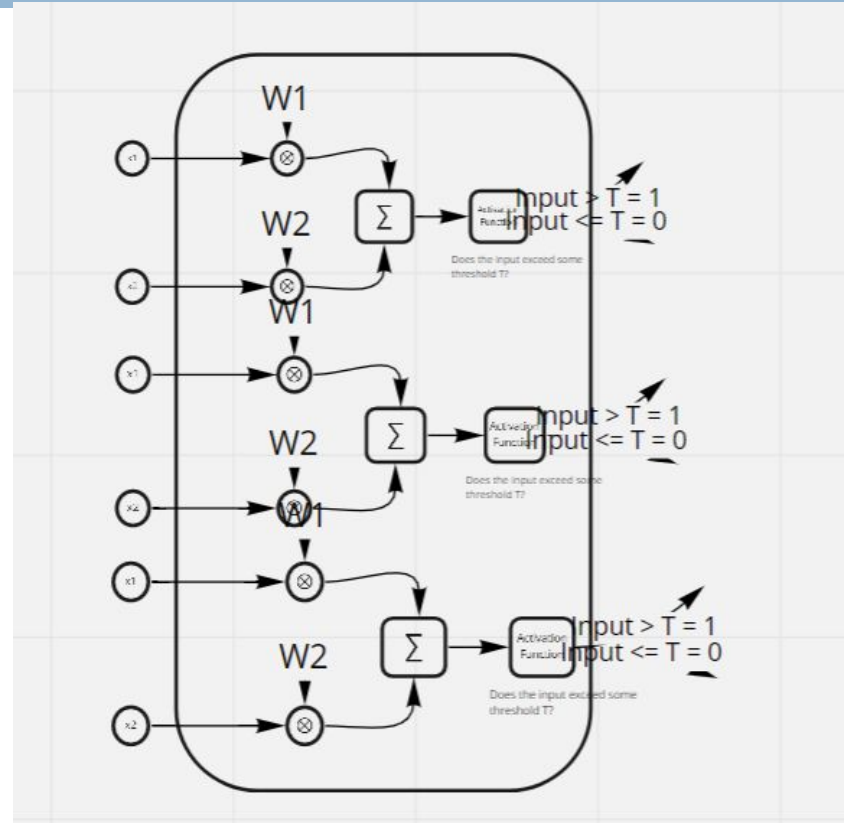
Model Characteristics

- All or none
- Cumulative Influence
- Synaptic Weight
- This is model that is inspired by the human brain
- But it is not clear that this model is the essential parameters of how a human can do what they do
 - We may be missing many other parameters including timing characteristics

Another View

A “view” of your head,
which could be thought
of as a very large
function calculator:

$$\bar{z} = f(\bar{x}, \bar{w}, \bar{T})$$



Objectives

$$\bar{z} = f(\bar{x}, \bar{w}, \bar{T})$$

What we need to do is to adjust the vectors **w** and **T**, until the vector **z** takes the form of what we want to see.

There may be millions of elements in the vector **w**.

NN as a function approximator

$$\bar{z} = f(\bar{x}, \bar{w}, \bar{T})$$

Clearly, we can think of the neural network as being a **function approximator**.

Desired Output

- If we consider our NN as a **function approximator**
- we can compute the desired output as some kind of ideal function **g**
- **g** yields the desired output vector **d** from the input **x**:

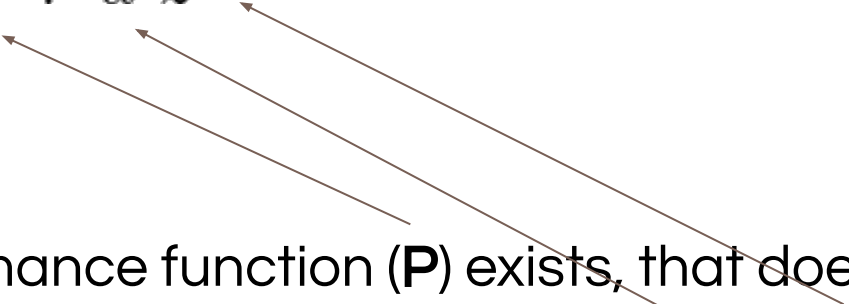
$$\bar{d} = g(\bar{x})$$

How well are we doing?

From the simple observations on the previous slides:

- We can figure out how well our function approximator is doing, by comparing the actual value vector (\mathbf{z}) with the desired value vector (\mathbf{d}) in a special function
 - This is called a **performance function**

Performance function P

$$P : \bar{d} \bar{z}$$


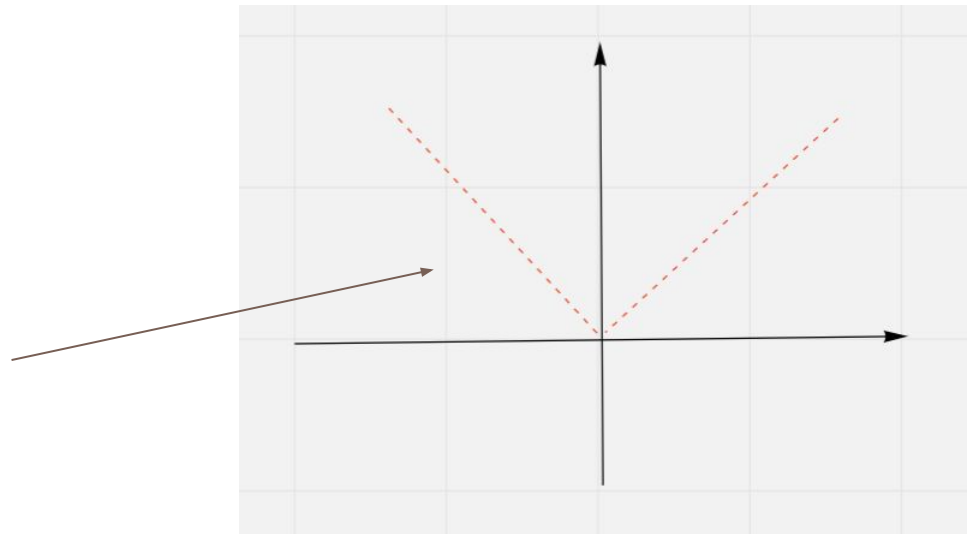
So clearly some performance function (P) exists, that does some kind of operation or measurement on the vectors \bar{d} and \bar{z} and reports a quantitative measurement of performance

Performance function P

One simple implementation of P is to simply measure the magnitude of the difference thus:

$$P : ||\bar{d} - \bar{z}||$$

In turn, P yields a performance function thus

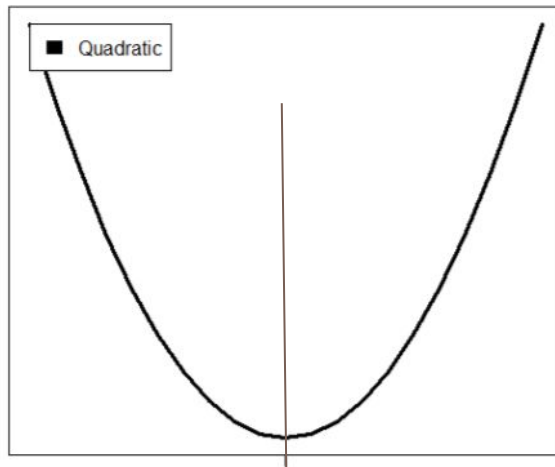


Performance function P

However, this kind of function is awkward to deal with mathematically, so we square the magnitude of the difference:

$$P : ||\bar{d} - \bar{z}||^2$$

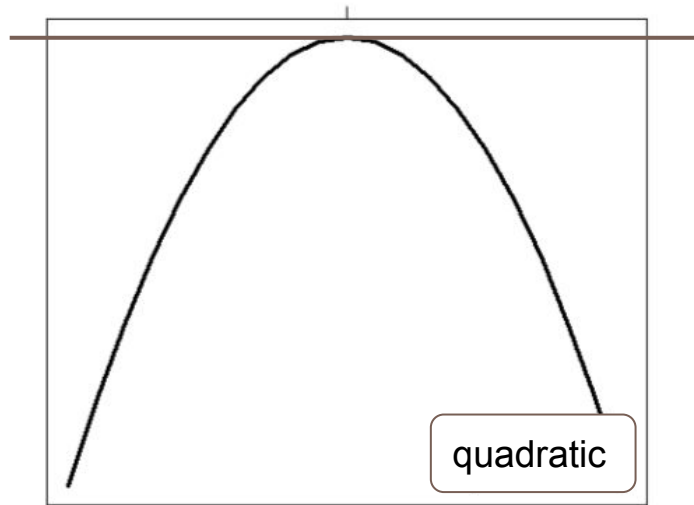
In turn, P yields a performance function thus



Refinement of P

We can now slightly refine P to express the idea that P returns a negative quantity which gets closer to zero the better input and output matches:

$$P : - ||\bar{d} - \bar{z}||^2$$



Optimization (again)

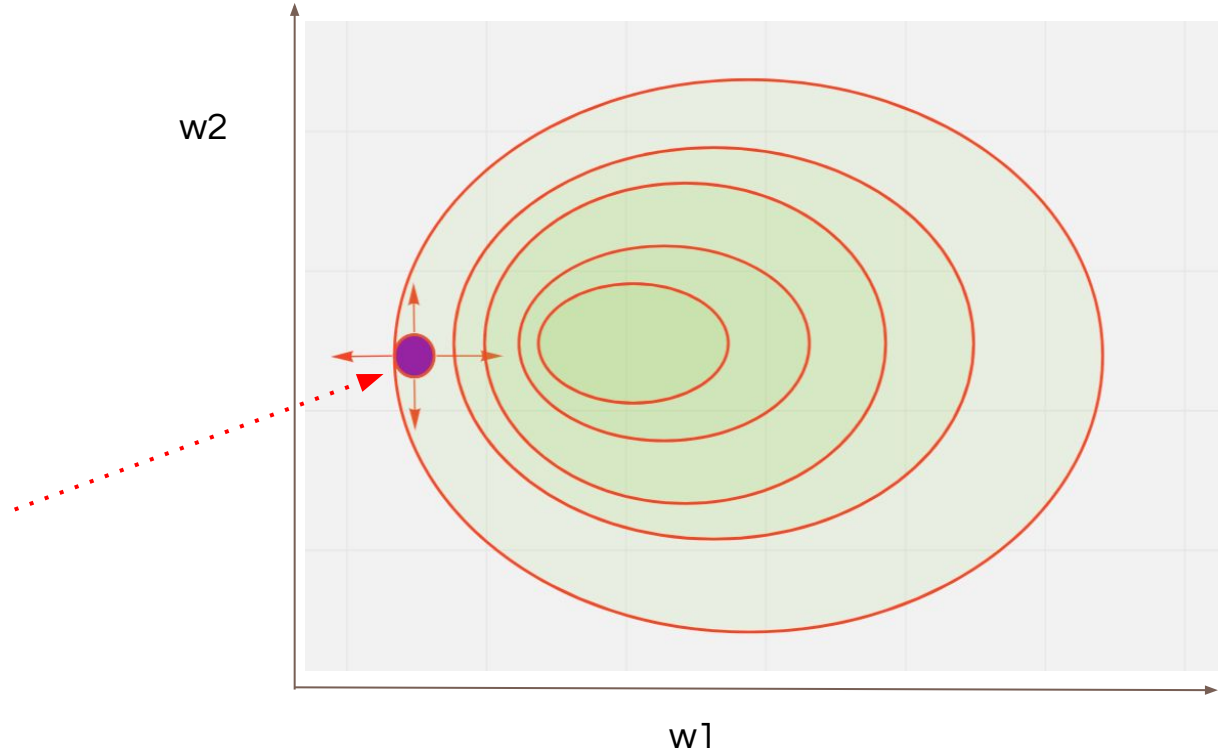
We can now see that the objective is to find the minimum value for the performance function **P**

So we seek to satisfy this **constraint** to find the best vector **w**:

$$P : \min(- ||\bar{d} - \bar{z}||^2)$$

Contour Map

So we can see a way to improve the weights...by implementing a Hill Climbing **heuristic**, from the starting point shown here



Simplification

A problem with the

$$\bar{z} = f(\bar{x}, \bar{w}, \bar{T})$$

is that threshold measures are not easy to deal with. So we will replace the threshold value \mathbf{T} with an equivalent weight \mathbf{w}_T shown here

$$\bar{z} = f(\bar{x}, \bar{w})$$

Intractability

Unfortunately, another problem arises when we have several million weights, we cannot adjust each of them simultaneously as this quickly becomes intractable (a search in n -dimensional space)

Instead, we use partial derivatives to tell us which of the weights has the most influence, and then we modify that weight only.

Partial Derivates - Hill Climbing

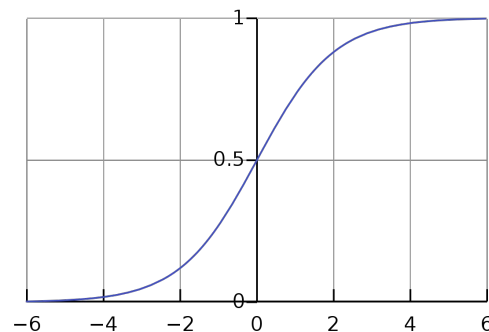
$$\Delta \bar{w} = r \left(\frac{\partial P}{\partial w_1} i + \frac{\partial P}{\partial w_2} j \right)$$

Here, the change in **w** is dependent on some rate constant **r** into the partial derivates shown here

Sigmoid Activation Functions

- Finally, we need to replace the binary activation function with a smoother (sigmoid) activation function.
- This is because it is mathematically inconvenient to perform Hill Climbing when our functions contains discontinuities.
- Sigmoid activation functions are generally of the form:

$$S(\alpha) = \frac{1}{1 + e^{-\alpha}}$$





Deep Learning

An Introduction

Modelling the Brain

- Deep learning has its origins in early work that tried to model networks of neurons in the brain (1943) with computational circuits.
- For this reason, the networks trained by deep learning methods are often called **neural networks**
 - even though the resemblance to real neural cells and structures is superficial

Linear Regression

- Although methods such as linear and logistic regression can handle a large number of input variables,
 - the computation path from each input to the output is very short: multiplication by a single weight, then adding into the aggregate output.
- Moreover, the different input variables contribute independently to the output, without interacting with each other

Linear Regression



They can represent only linear functions and boundaries in the input space, whereas most real-world concepts are far more complex.

Long Paths for Complex Problems

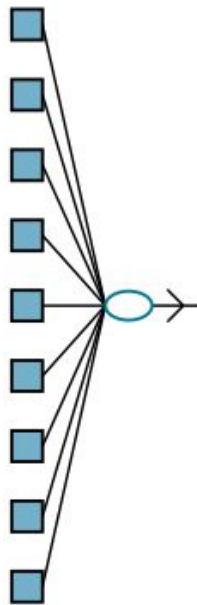
- The basic idea of deep learning is to train circuits such that the computation paths are long, allowing all the input variables to interact in complex ways
 - See Fig (c).
- These circuit models turn out to be sufficiently expressive to capture the complexity of real-world data for many important kinds of learning problems.

Network Depth

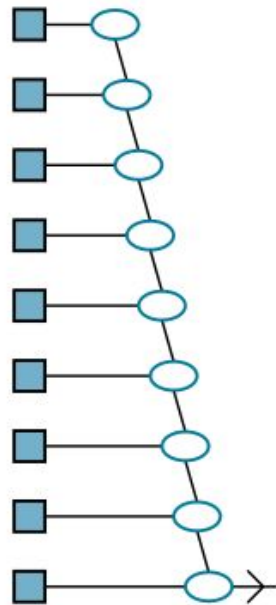
(a) A shallow model, such as linear regression, has short computation paths between inputs and output.

(b) A decision list network has some long paths for some possible input values, but most paths are short.

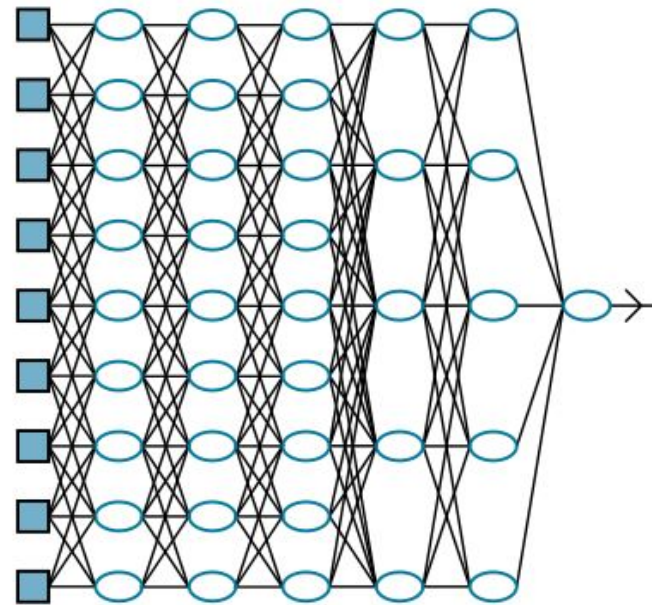
(c) A deep learning network has longer computation paths, allowing each variable to interact with all the other



(a)



(b)



(c)

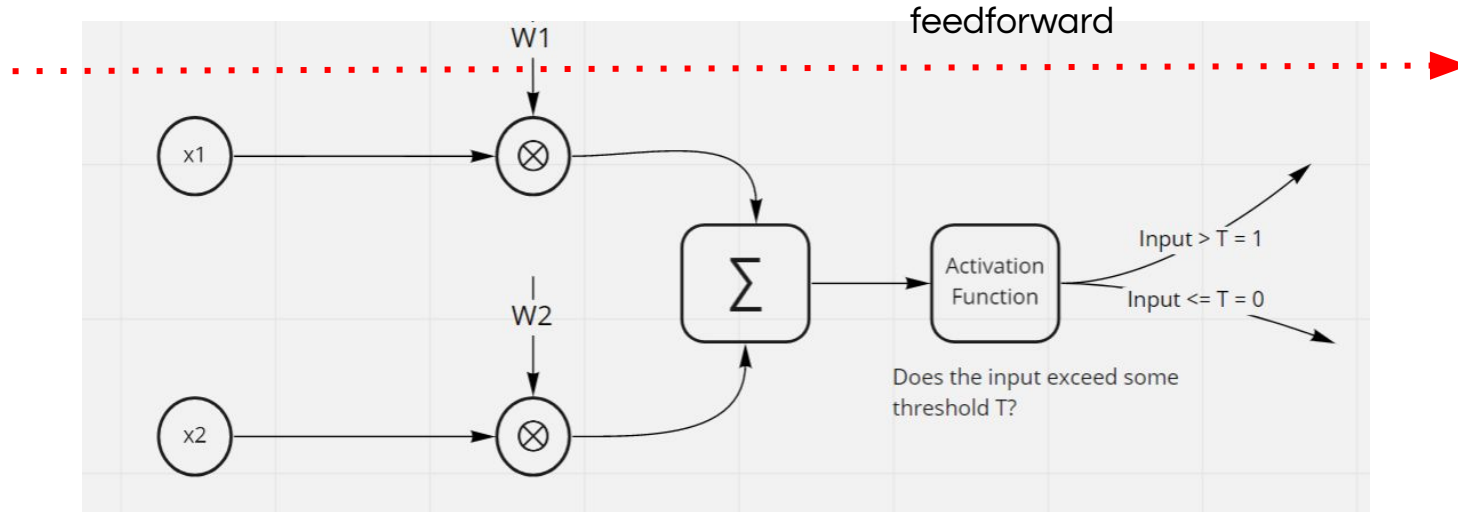


Simple Feedforward Networks

Connections only in one direction

Feedforward Network

In the previous section we connected our trivial neural network in such a way that the data flowed from left to right, otherwise known as a Feedforward Network



Feedforward Network

- Each node computes a function of its inputs
 - then passes the result to its successors in the network.
- Information flows through the network from the input nodes to the output nodes
 - there are no loops

Recurrent networks

- A recurrent network, on the other hand, feeds its intermediate or final outputs back into its own inputs.
- This means that the values within the network form a dynamical system
 - that has internal state or memory.
- We will consider these in the next lecture

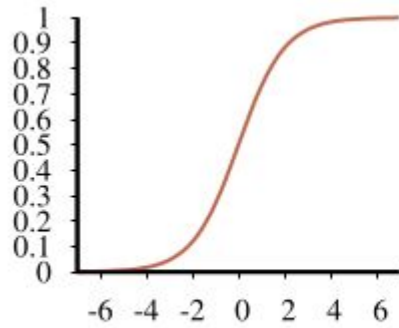
More Activation Functions

In the previous section we discussed sigmoid based activation functions:

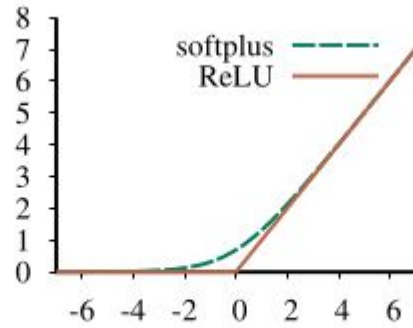
$$S(\alpha) = \frac{1}{1 + e^{-\alpha}}$$

However, sigmoid activation functions are not the only kind. Typically we see two others: **tanh** and **softplus/ReLU**

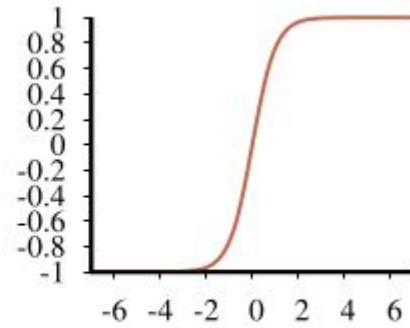
More Activation Functions



(a)



(b)



(c)

Activation functions commonly used in deep learning systems: (a) the logistic or sigmoid function; (b) the ReLU function and the softplus function; (c) the tanh function.

Activation Functions

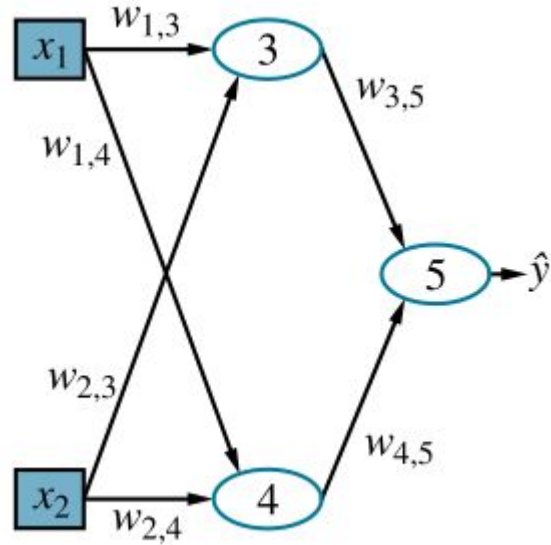
Sigmoid $S(\alpha) = \frac{1}{1 + e^{-\alpha}}$

ReLU $\text{ReLU}(x) = \max(0, x) .$

Softplus $\text{softplus}(x) = \log(1 + e^x) .$

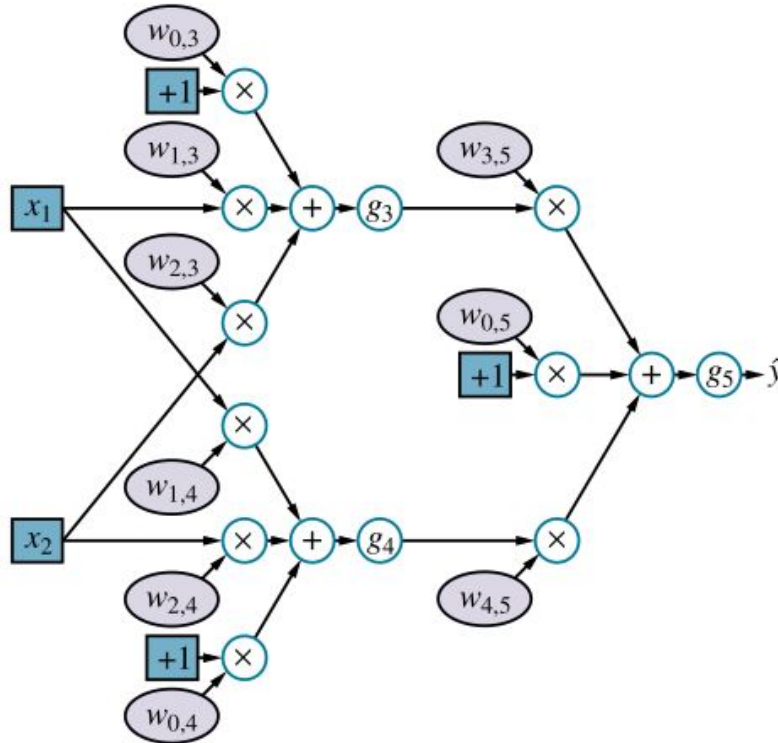
tanh $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} .$

Hidden Layers



A neural network with two inputs, one hidden layer of two units, and one output unit.

Computational Expansion



The network is unpacked into its full computation graph.

Hidden layers are those not input or output layers.

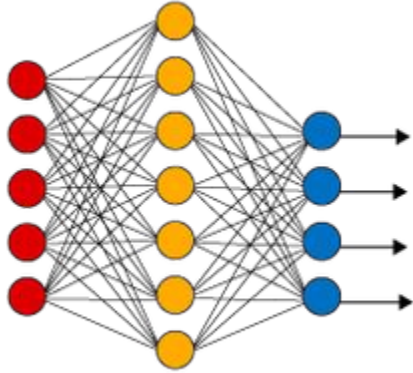
In this model there is **one** hidden layer.

Hidden Layers

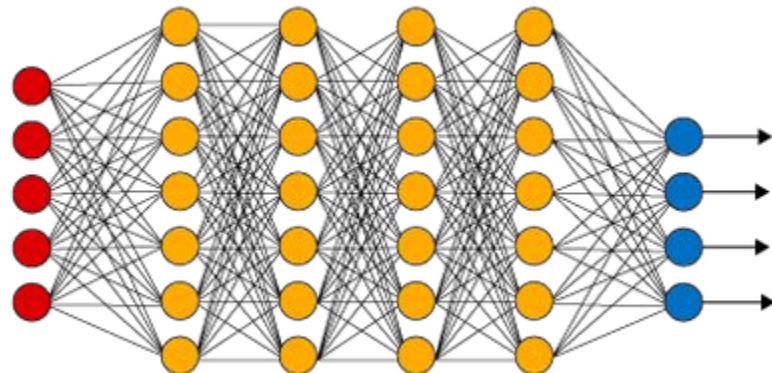
- Hidden layers allow for the function of a neural network to be broken down into specific transformations of the data.
- Each hidden layer function is specialized to produce a defined output.
- For example, a hidden layer functions that are used to identify human eyes and ears may be used in conjunction by subsequent layers to identify faces in images.
- While the functions to identify eyes alone are not enough to independently recognize objects, they can function jointly within a neural network.

Hidden Layers

Simple Neural Network



Deep Learning Neural Network



● Input Layer ● Hidden Layer ● Output Layer

Ref: <https://deeptai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning>

Function Approximator

Coupling multiple units together into a network creates a complex function that is a composition of the algebraic expressions represented by the individual units.

For example, the network shown here represents a function $h_w(\mathbf{x})$, parameterized by weights \mathbf{w} , that maps a two-element input vector \mathbf{x} to a scalar output value \hat{y} .

$$\begin{aligned}\hat{y} &= g_5(in_5) \\ &= g_5(w_{0,5} + w_{3,5}g_3(in_3) + w_{4,5}g_4(in_4)) \\ &= g_5(w_{0,5} + w_{3,5}g_3(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g_4(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2))\end{aligned}$$

Computational Graph

- The computational graph makes each element of the overall computation explicit.
- It also distinguishes between the inputs (in blue) and the weights (in light mauve):
 - the weights can be adjusted to make the output \hat{y} agree more closely with the true value y in the training data.
- Each weight is like a volume control knob that determines how much the next node in the graph hears from that particular predecessor in the graph

Graph Complexity

- The computation graph here is relatively small and shallow,
- but the same idea applies to all forms of deep learning:
 - we construct computation graphs and adjust their weights to fit the data.
- The graph above is also fully connected,
 - meaning that every node in each layer is connected to every node in the next layer.
 - Except for nodes leading into the output layer

Graph Connectedness

- The default connection (or wiring) of the graph is **connected**
 - every node in each layer is connected to every node in the next layer.
- This is in some sense the default,
- Choosing the connectivity of the network is also important in achieving effective learning

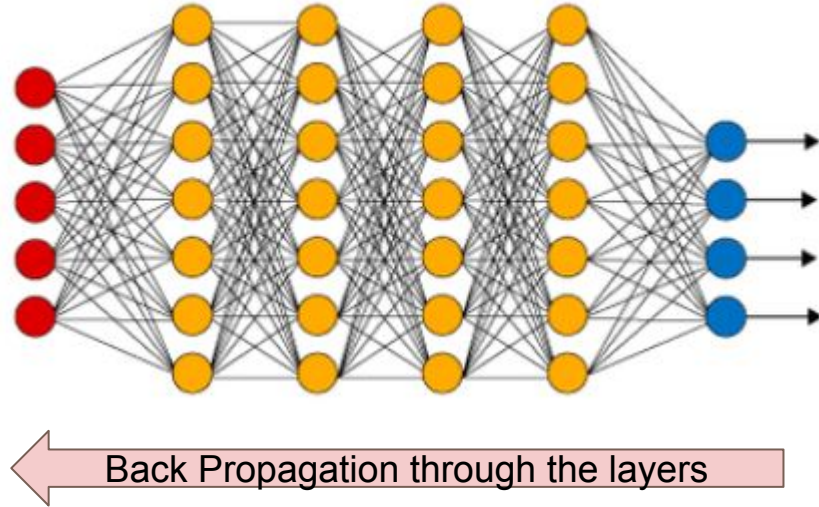
Gradients and learning

- For the weights leading into units in the output layer the ones that produce the output of the network,
 - the gradient calculation is essentially identical to the process the partial differential model seen earlier
- However, units in the hidden layers, which are not directly connected to the outputs, calculating the decent gradient is more complicated.

Backpropagation

Clearly, the error function is computed first at the output layer.

In order to improve the output layer the error function for the hidden layers are computed in reverse order from the output back to the first hidden layer - this is called **back propagation**



Backpropagation

- It is important to remember that Back Propagation is a weight adjustment process.
- It does not mean connecting the output of nodes in later layers with nodes in earlier layers.
- This type of network is called a Recurrent Neural Network and will be the subject of our next lecture
- BP works by iterating backward from the last layer to
 - avoid redundant calculations of intermediate terms in the chain rule
- BP is an example of **dynamic programming**

Backpropagation

At the last layer L , we define a Cost Function as follows, where $a^{(L)}$ is the activation function of the last layer

$$C_0 = (a^{(L)} - y)^2$$

Note though that the input to $a^{(L)}$ can be represented as $z^{(L)}$

Thus $a^{(L)}$ is really this, where σ is ReLU for example

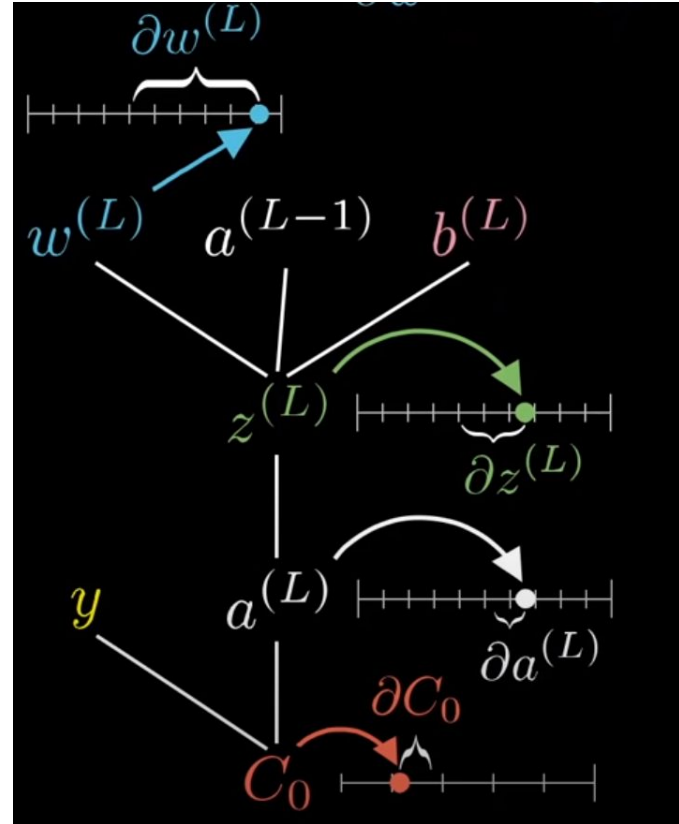
$$a^{(L)} = \sigma(z^{(L)})$$

Backpropagation

If the weights at L are written as $w^{(L)}$, we can think of $z^{(L)}$ as

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

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$



Backpropagation

If the weights at L are written as $w^{(L)}$, we can think of $z^{(L)}$ as

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

And the total cost function is written now as a partial derivative using the chain rule

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

Thank You



Any Questions?