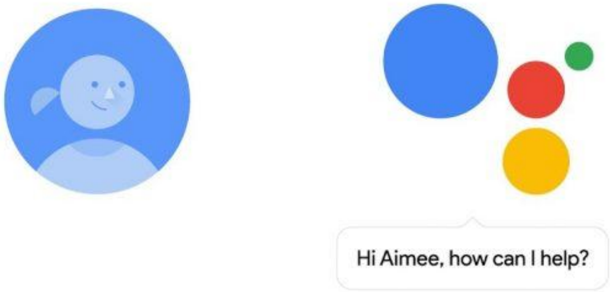
DATA X

Introduction to Deep Learning & Neural Networks

Berkeley SCET

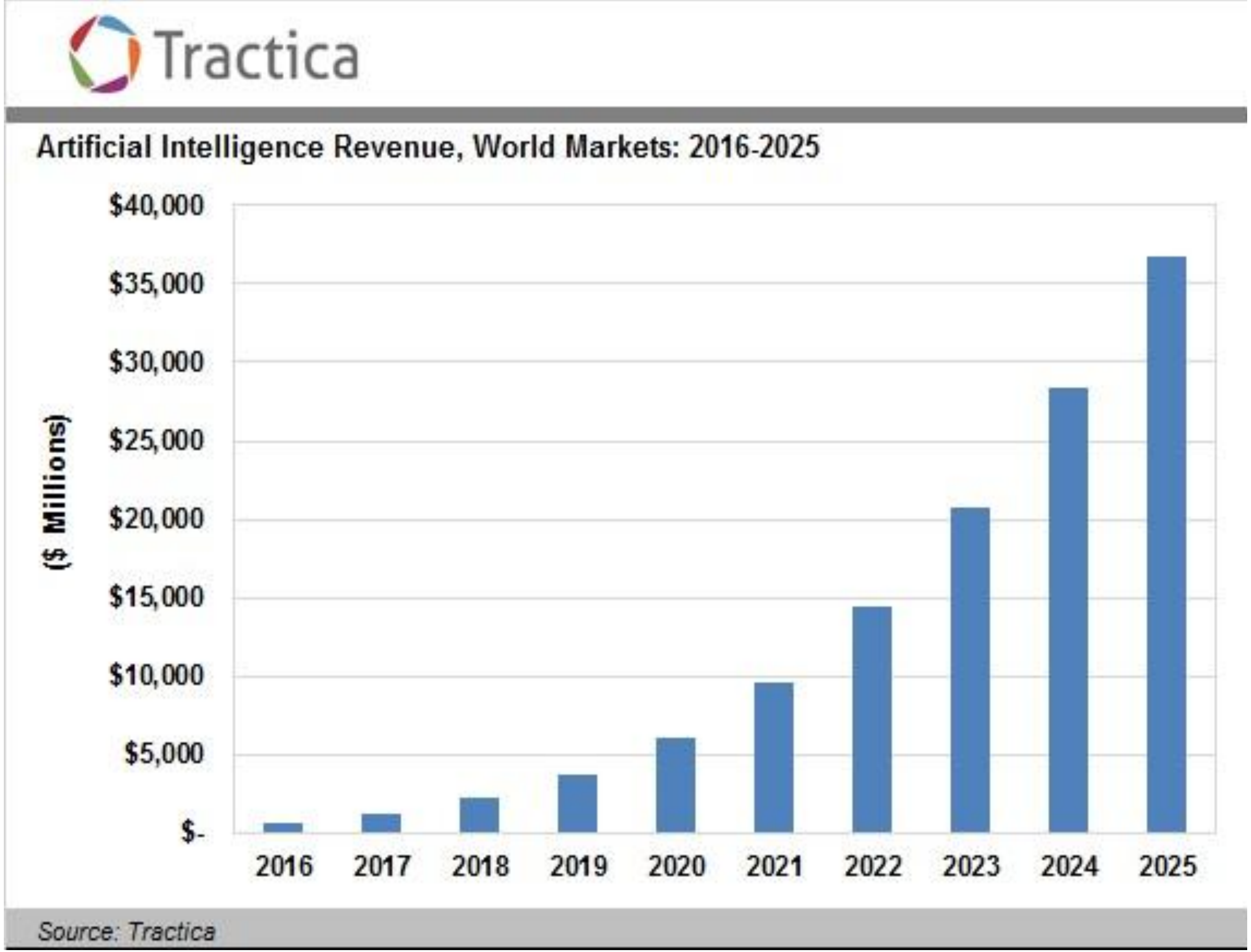
Created By: Arash Nourian



Cortana
Microsoft's virtual Assistant.

Socratic

An AI-powered app to help students with math and other homework. It is now acquired by Google.


Tractica
Artificial Intelligence Revenue, World Markets: 2016-2025
($ Millions)
$40,000
$35,000
$30,000
$25,000
$20,000
$15,000
$10,000
$5,000
$-
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Source: Tractica

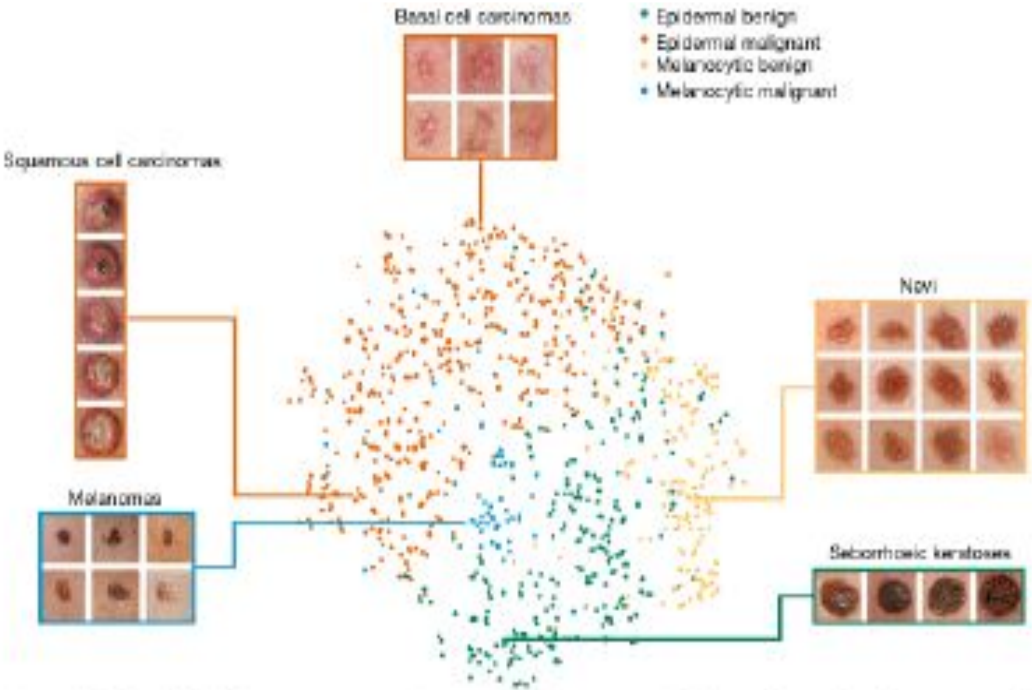
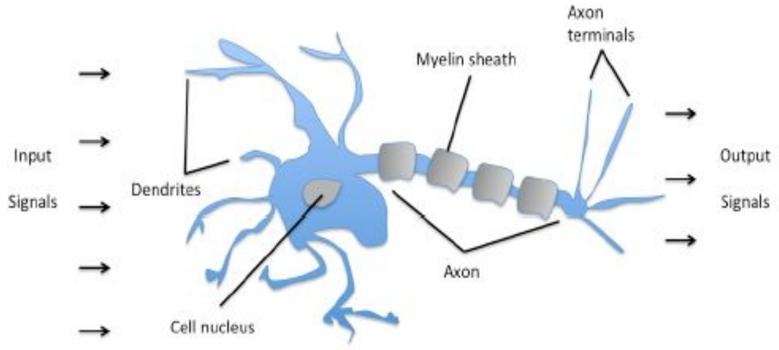


Figure 4 | t-SNE visualization of the last hidden layer representations in the CNN for four disease classes. Here we show the CNN's internal

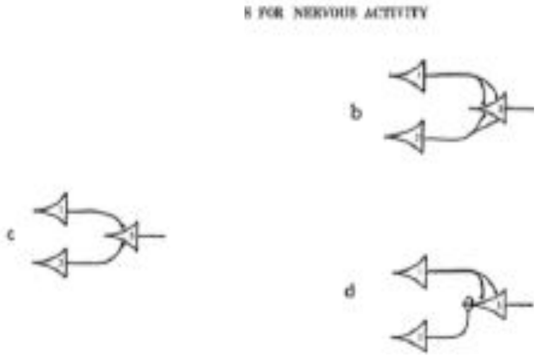
(932 images). Coloured point clouds represent the different disease categories, showing how the algorithm clusters the diseases. Insets show

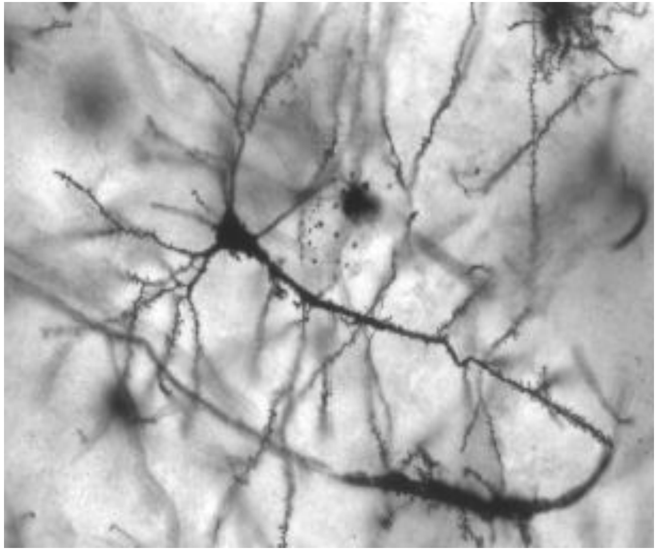
Neural Networks

McCulloch & Pitt’s Neuron Model (1943)



Schematic of a biological neuron.





Gates
AND, OR, NOT gates can be solved by the mathematical formulation of a biological neuron

List of Animals by Number of Neurons

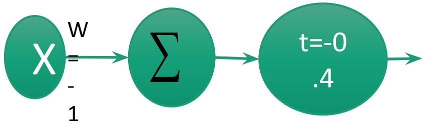
Whole nervous system [edit]

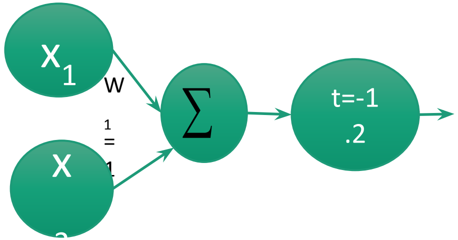
This list is incomplete; you can help by expanding it.

All numbers for neurons (except Caenorhabditis and Ciona), and all numbers for synapses (except Ciona) are estimations.

Name	Neurons in the brain/whole nervous system	Synapses	Details	Image	Source
Sponge	0				[3]
Trichoplax	0		Despite no nervous system, it exhibits coordinated feeding and response behaviors.[4]		[5]
Asplanchna brightwellii (rotifer)	about 200		Brain only		[6]
Human	8.6×10^{10}	$\sim 1.5 \times 10^{14}$	Neurons for average adult		[49][50][51]
African elephant	2.57×10^{11}				[52][53]

McCulloch & Pitt’s Neuron Model (1943)





NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

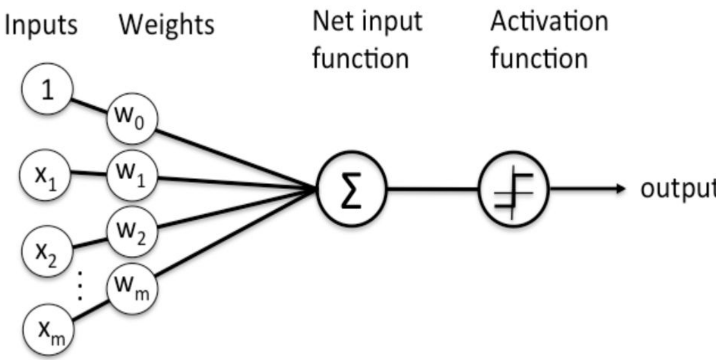
x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

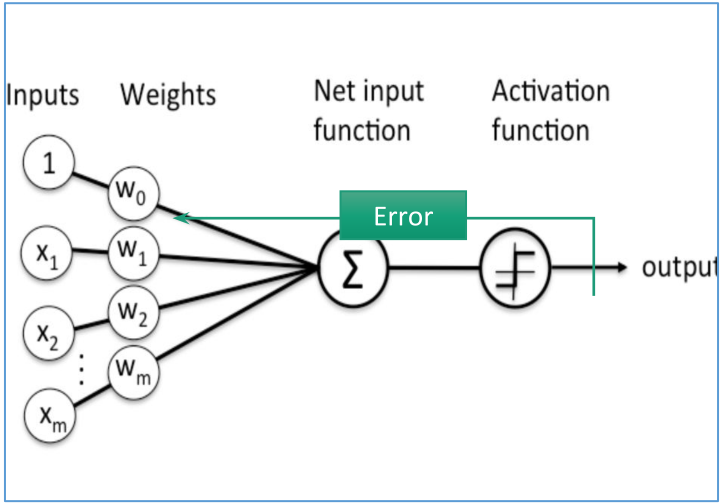
Quiz:
Could you do this for XOR?

Frank Rosenblatt's Perceptron (1957)



Schematic of Rosenblatt's Perceptron
A learning algorithm for the neuron model

Widrow and Hoff’s ADALINE (1969)



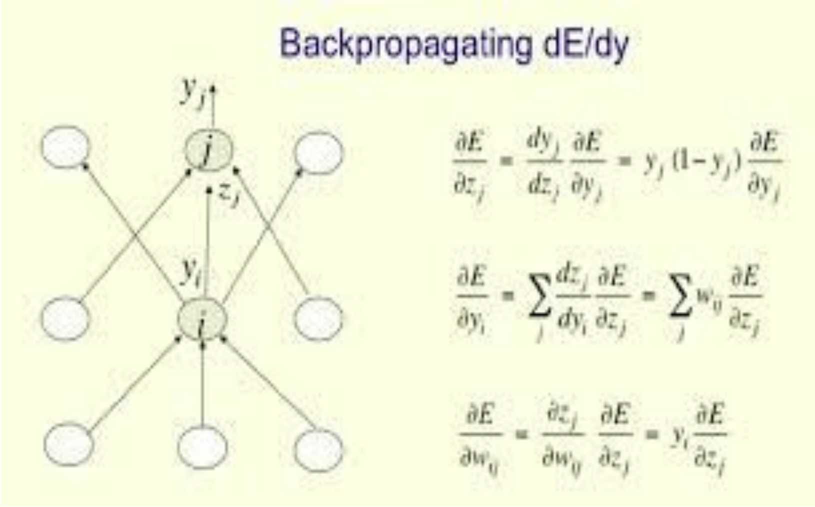
Schematic of Rosenblatt’s Perceptron

A nicely differentiable neuron model

Multilayer Perceptrons

- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533.

- According to BackProp, they showed a formulation on how to train and set the basis for later research in deep learning



What is a Neural Network?

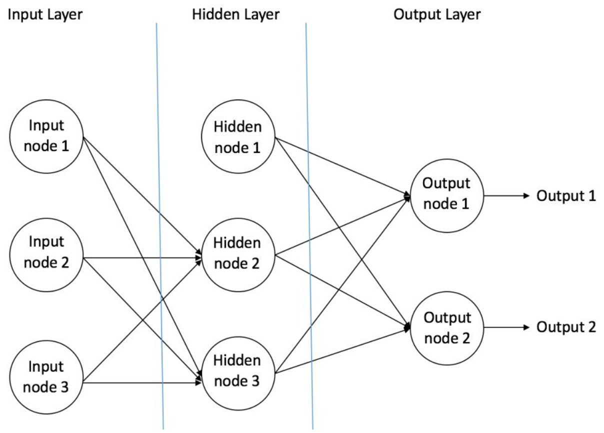
Like other machine learning methods that we saw earlier in class, it is a technique to:

- Map features to labels or some dependent continuous value
- Compute the function that relates features to labels or some dependent continuous value.

Neural Network

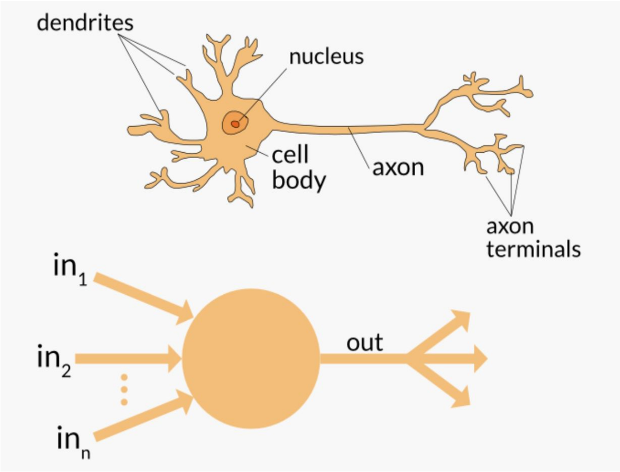
Network:

- A network of neurons/nodes connected by a set of weights.

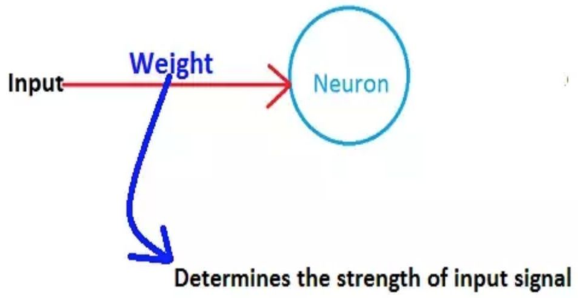


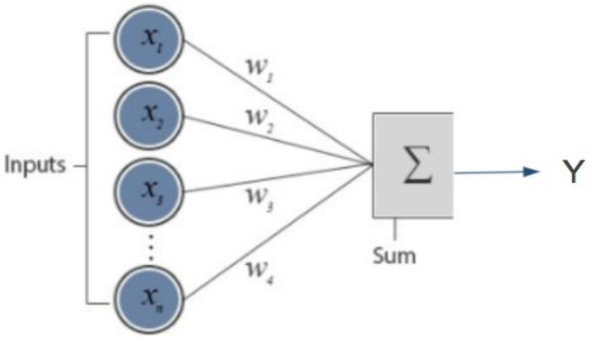
Neural:

- Loosely inspired by the way biological neural networks in the human brain process



Example: Linear Regression



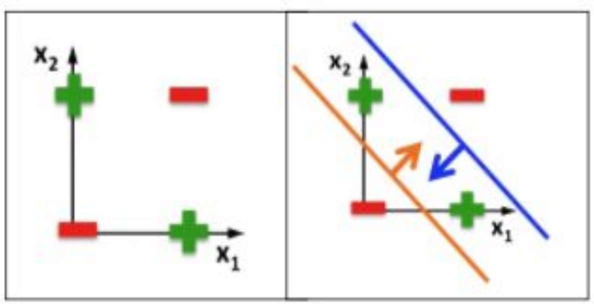


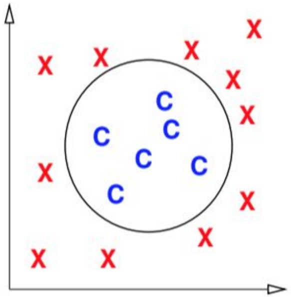
$Y = x1*w1 + x2*w2 + x3*w3 + \cdots + xn*wn$ **--linear regression**

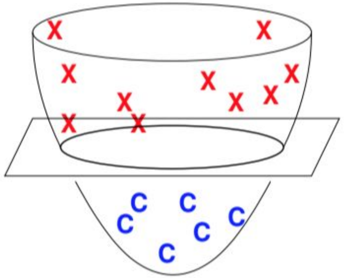
Activation Functions

- We use activation functions in neurons to induce nonlinearity in the neural nets so that it can learn complex functions

- All mapping functions are **NOT** linear



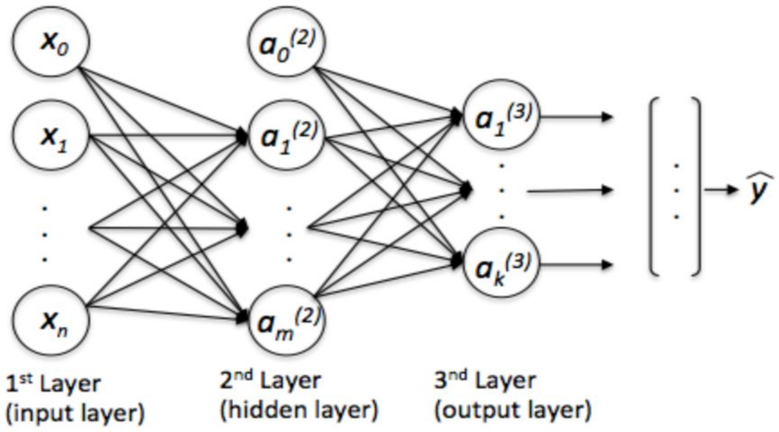




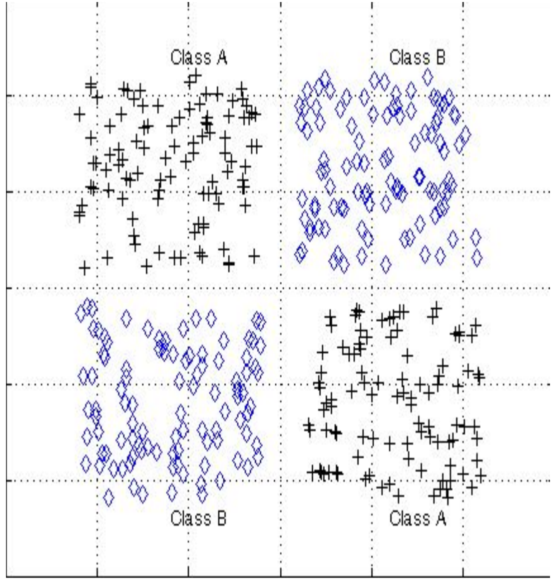
Perceptron Update Rule

- If we misclassify a data point x_i, with label y_i simple update the weights by $w_{new} = w_{old} + \lambda (d_i - y_i)$ for some λ between 0 and 1, where the d_i is the desired 0 or 1 label.
- Simply update all weights a step higher or lower in the direction of the desired classification.

The Multilayer Perceptron

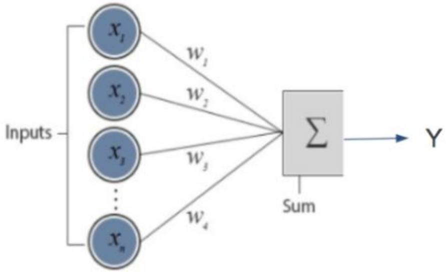


Schematic of a multi-layer perceptron.



Schematic of Rosenblatt's Perceptron
Can represent more complex functions like XOR

Example



Y = x1∗w1 + x2∗w2 + x3∗w3 +·····+ xn∗wn **--linear regression**

For sample 1:

x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**

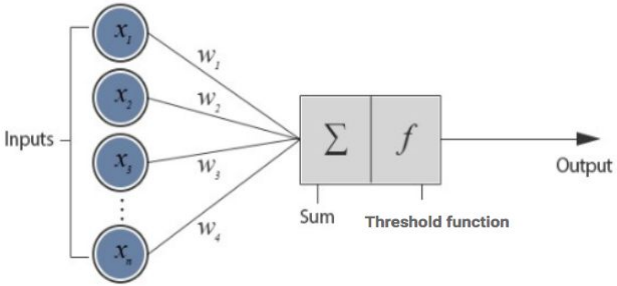
Y = sum(x * w) =**1.3**

For sample 2:

x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**

Y = sum(x * w) = 5.5

Lets apply a **threshold function** on the output:



$$f(t) = \begin{cases} t & \textit{if } t < 3 \\ 0 & \textit{otherwise} \end{cases}$$

For sample 1:

x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**

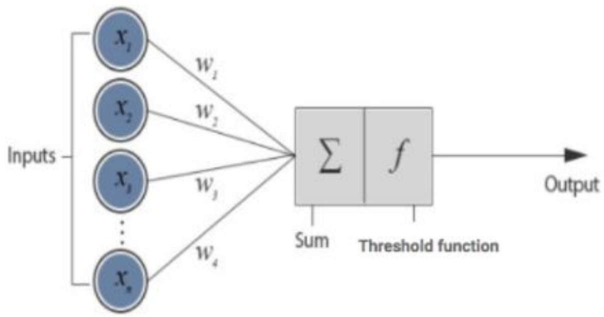
Y = f(sum(x * w)) = f(1.3)= 1.3

For sample 2:

x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**

Y = f(sum(x * w))= f(5.5)=0

Now, if we apply a **logistic/sigmoid function** on the output, it will squeeze all the output between 0 and 1:



Y = Sigmoid(x1 * w1 + x2 * w2 + .. + xn * wn) --**logistic regression**

Logistic/sigmoid function

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

For sample 1:

x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**

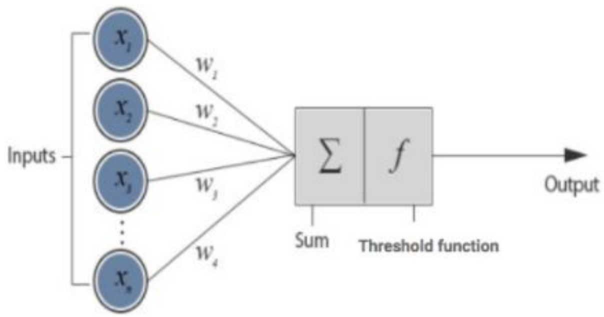
Y = **σ**(sum(x * w)) = **σ(1.3) = 0.78**

For sample 2:

x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**

Y = **σ**(sum(x * w)) = **σ(5.5) = 0.99**

Now, if we apply a **logistic/sigmoid function** on the output, it will squeeze all the output between 0 and 1:



Y = Sigmoid(x1∗w1 + x2∗w2 + .. + xn∗wn) --**logistic regression**

Logistic/sigmoid function

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

For sample 1:

x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**

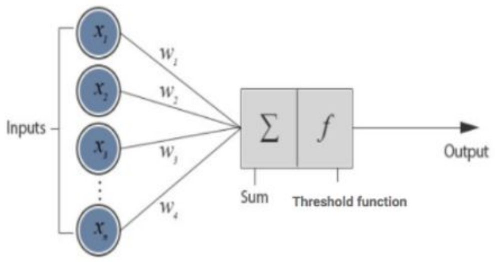
Y = **σ**(sum(x * w)) = **σ(1.3) = 0.78**

For sample 2:

x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**

Y = **σ**(sum(x * w)) = **σ(5.5) = 0.99**

Now, if we apply a **logistic/sigmoid function** on the output, it will set the final output as 0 or 1:



Logistic/sigmoid function

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

f(t) = { 1 if t > 0.6
0 *otherwise* }

For sample 1:

x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**

Y = **f** (**σ**(sum(x * w)))= f(**σ**(1.3)) = f (0.78) =1

For sample 2:

x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**

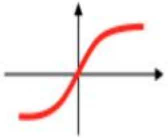
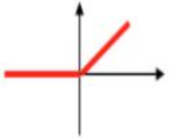
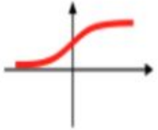
Y = **f**(**σ**(sum(x * w))) = f (**σ**(5.5)) = f (0.99)=1

Neural Network Playground

https://playground.tensorflow.org/

Activation Functions

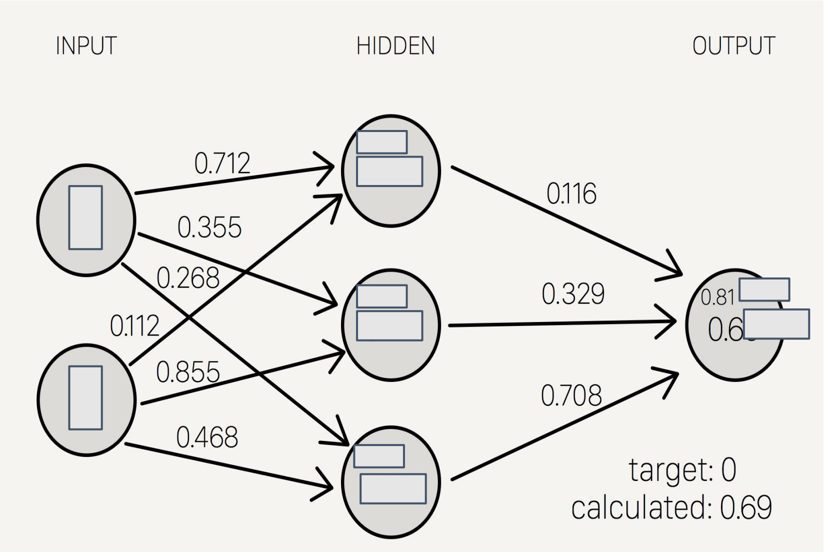
1. Sigmoid (0,1)
2. Tanh (-1,1)
3. Relu (0, x)
4. Softmax (0,1)

Logistic (sigmoid)	$\phi(z) = \frac{1}{1+e^{-z}}$	
Hyperbolic tangent	$\phi(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	
Softmax output	$z_j(t) = \frac{e^{t_j}}{\sum_{i=1}^{k} e^{t_i}}.$	

Network and Forward Propagation

Activation Function—Logistic Sigmoid Function

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

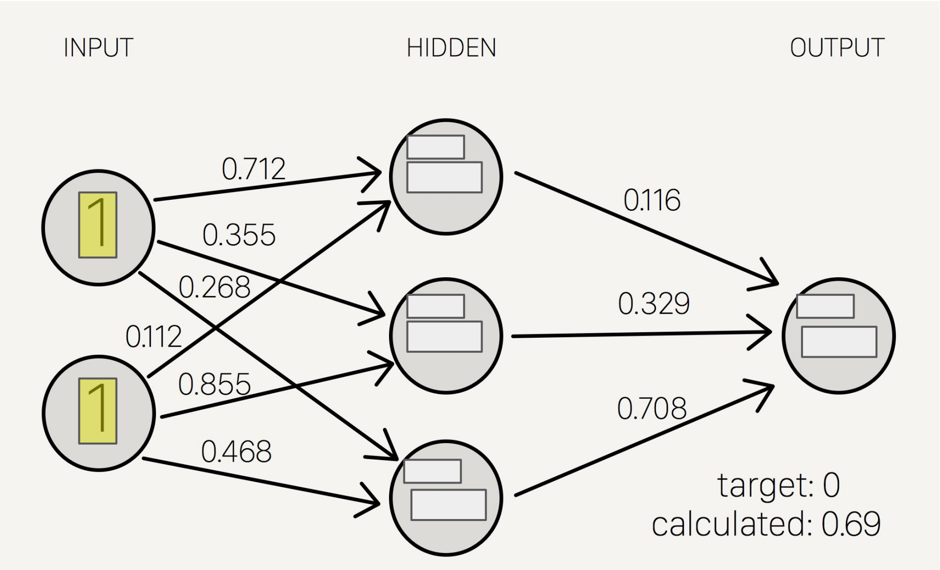


This is the architecture of a neural network. In order to make a prediction, we do what is called forward propagation.

Example Network and Forward Propagation

Activation Function—Logistic Sigmoid Function

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



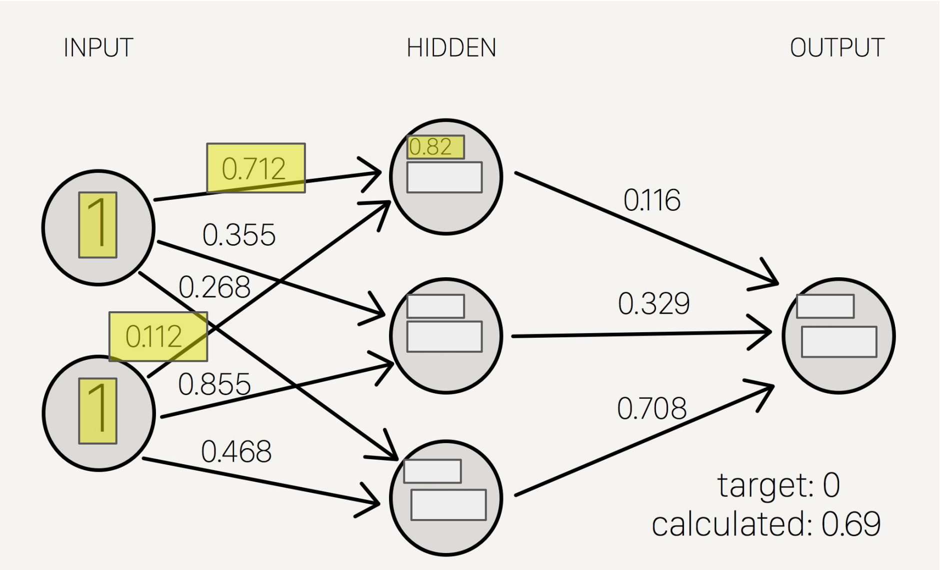
http://i.imgur.com/UNIffE1.png

First, we feed in our inputs into the input layer. In this case, we have two features, each with a value of 1.

Example Network and Forward Propagation

Activation Function—Logistic Sigmoid Function

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



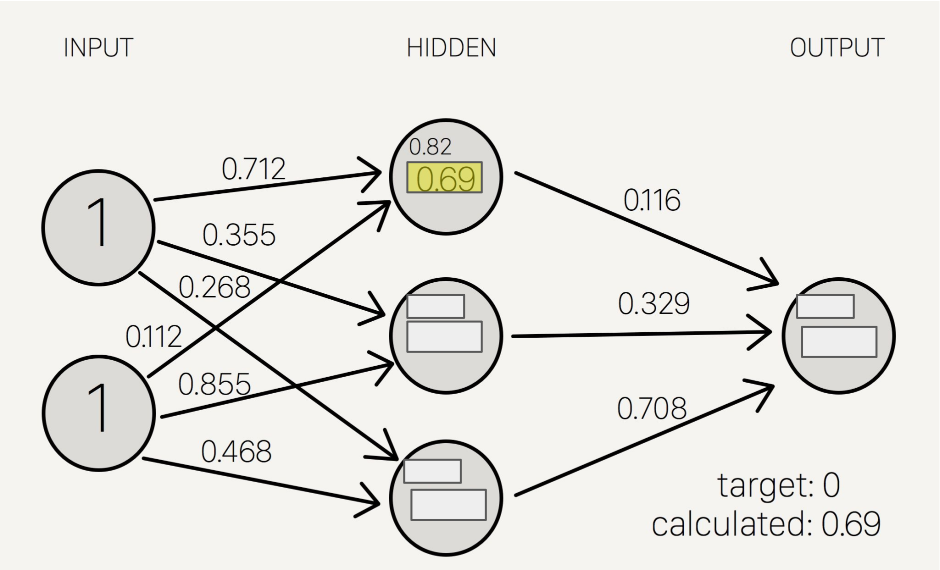
http://i.imgur.com/UNIffE1.png

The relevant axon weights will then take weighted sums and send them to the next layer. In this case, the first hidden neuron gets a value of 1*(0.712) + 1*(0.112)

Example Network and Forward Propagation

Activation Function—Logistic Sigmoid Function

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



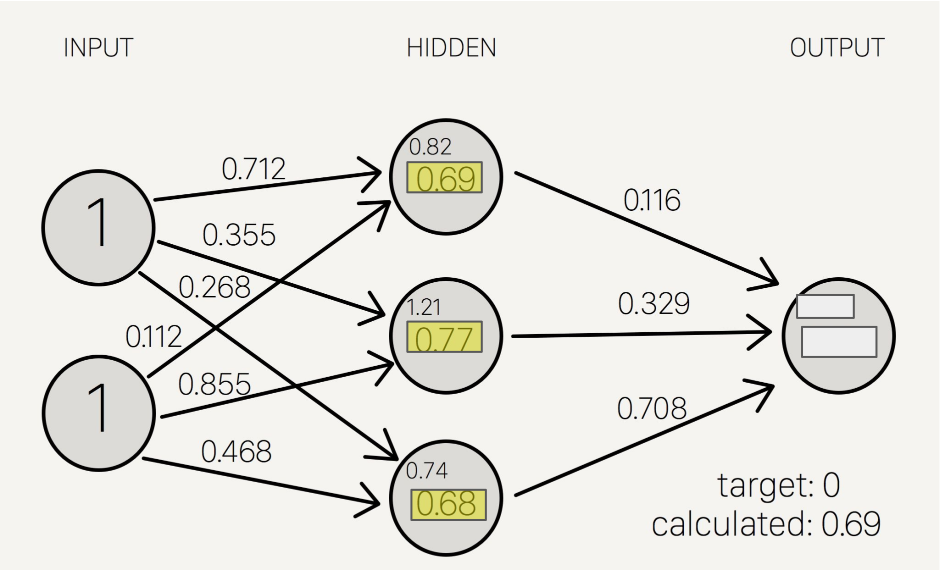
http://i.imgur.com/UNIffE1.png

But 0.82 is only the pre-activation value. We need to then apply our activation function to 0.82, giving us f(0.82) = 0.69. This is the value for the neuron.

Example Network and Forward Propagation

Activation Function—Logistic Sigmoid Function

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



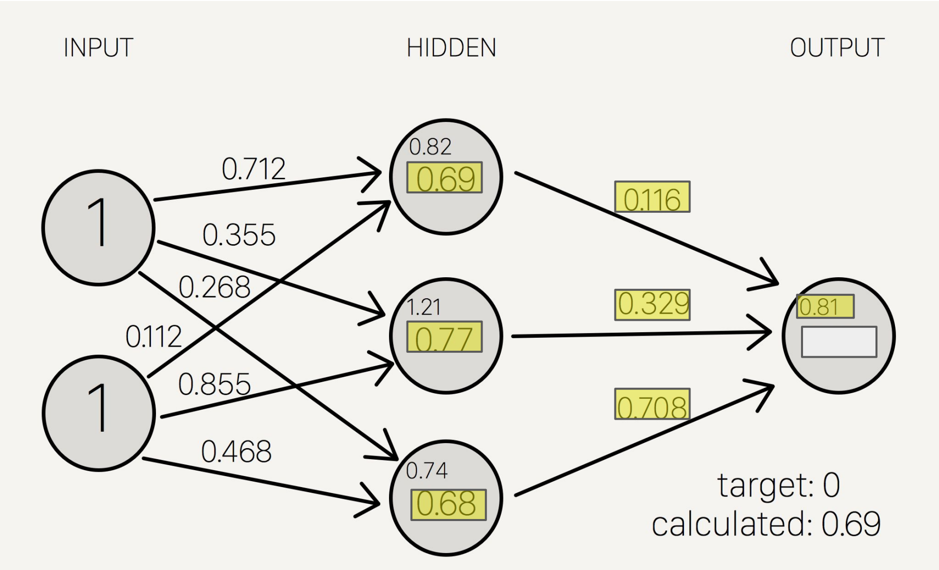
http://i.imgur.com/UNIffE1.png

The same process is applied to each neuron.

Example Network and Forward Propagation

Activation Function—Logistic Sigmoid Function

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



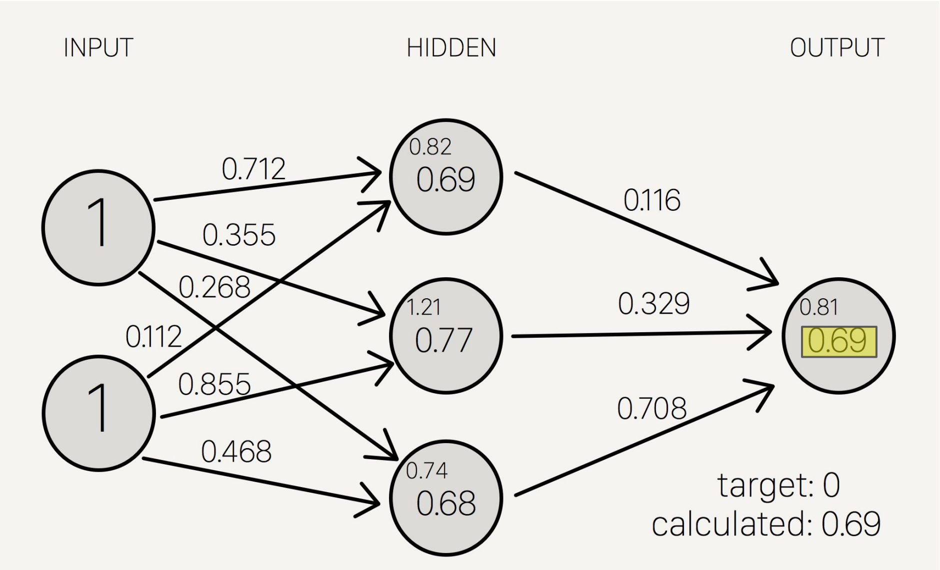
http://i.imgur.com/UNIffE1.png

0.69*(0.116) + 0.77*(0.329) + 0.68*(0.708) = 0.81

Example Network and Forward Propagation

Activation Function—Logistic Sigmoid Function

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



http://i.imgur.com/UNIffE1.png

f(0.81) = 0.69. This is the output, or prediction, or our model.

Mathematical Representation

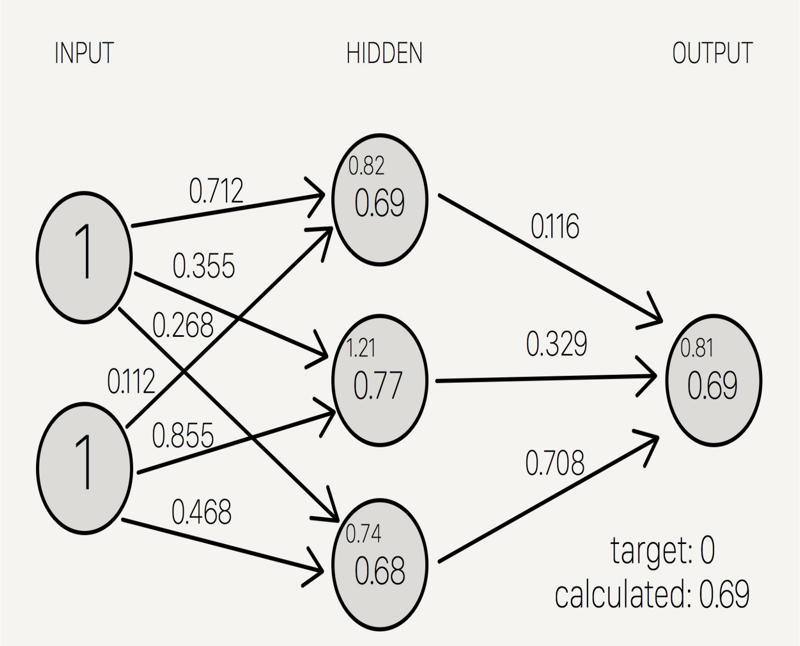
We can represent this model using matrices, for example:

X gets input values:

$$X = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

W gets axon weights:

$$W = \begin{bmatrix} 0.712 & 0.355 & 0.268 \\ 0.112 & 0.855 & 0.468 \end{bmatrix}$$



http://i.imgur.com/UNIffE1.png

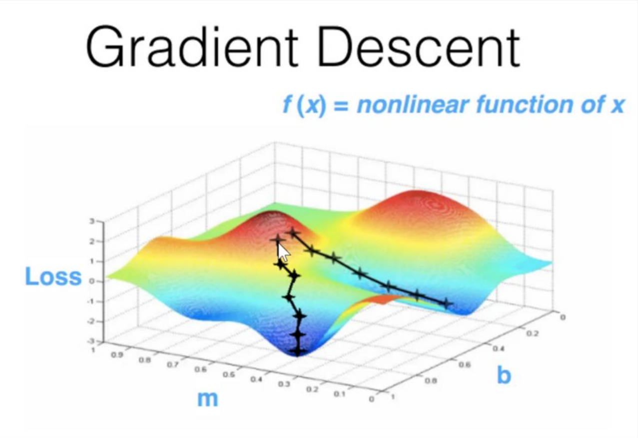
One step of forward propagation looks like:

$$\begin{bmatrix} 0.712 & 0.355 & 0.268 \end{bmatrix}$$

How Does it work?

How does the network know the strength of connections between neurons? It learns them!

- We start with random weights
- Input a set of features
- Calculate the output
- Calculate the loss wrt to actual output value in the data
- Find the gradient of the cost function
- Backpropagation: The gradients are pushed back into the network and used for adjusting the weights
- The whole process is repeated again till we train a model of acceptable performance

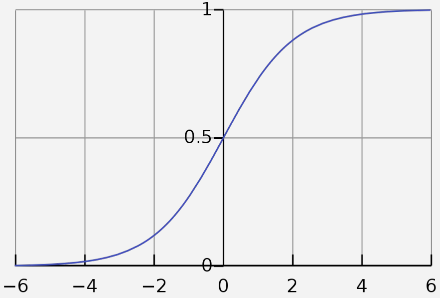


$$D\{f(g(x))\} = f'(g(x))\, g'(x)$$

Different Activation Functions

*Recall the **sigmoid function** is:*

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

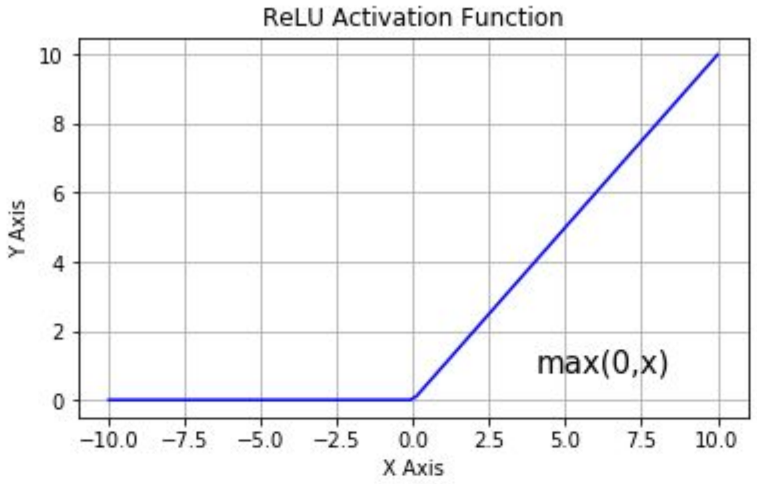


During back propagation we calculate gradients of activation functions, for **s =**

$$\begin{aligned} f'(x) &= -(1 - e^{-x})^{-2} e^{-x} \\ &= \frac{1}{1 - e^{-x}} \frac{-e^{-x}}{1 - e^{-x}} \\ &= f(x)(1 - f(x)) \end{aligned}$$

So when f(x) is close to 1 or 0, this means the gradient will be very close to 0, so learning may happen slowly! This is called **vanishing gradients.**

A Solution: The Relu Activation Function

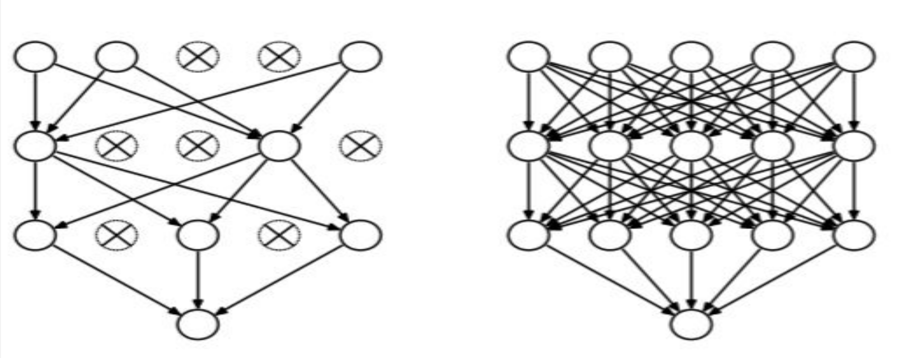


ReLU Activation Function Graph

Notice that the derivative for x > 0 is constant, unlike the sigmoid activation function

Regularization in Neural Nets

Dropout is an approach to regularization in neural networks which helps **reducing interdependent learning** amongst the neurons.



Review

1. Neural nets want to find the function that maps features to outputs
2. Neuron takes in weighted input(s)
3. Functions are used for transforming neuron output

Berkeley SCET

Exercises

1. Draw a 2 hidden layer neural net with input of size 2 units with the following information:
   Hidden Layer-1 with 3 nodes
   Hidden Layer-2 with 4 nodes
   The output should be a number. How many weights are there?

2. Given an input = [3, 2]

Weight matrix W =

w1	w2
0.2	0
0.91	2.25
0.7	0.6

Calculate the output, if the activation function is sigmoid.

Solution

$$\begin{bmatrix}0.2 & 0\\ 0.91 & 2.25\\ 0.7 & 0.6\end{bmatrix} \mathrm{X} \begin{bmatrix}3\\ 2\end{bmatrix} \longrightarrow \begin{bmatrix}0.2 \text{ x } 3 & 0 \text{ x } 2\\ 0.91 \text{ x } 3 & 2.24 \text{ x } 2\\ 0.7 \text{ x } 3 & 0.6 \text{ x } 2\end{bmatrix} \longrightarrow \begin{bmatrix}0.6\\ 7.23\\ 3.3\end{bmatrix} \xrightarrow[\text{Sigmoid}]{} \begin{bmatrix}0.64\\ 0.99\\ 0.94\end{bmatrix}$$

Solution

Then calculate the Mean Square Loss: sq(z-y)

Given:

$$y = \begin{bmatrix} 0.6 \\ 7.23 \\ 3.3 \end{bmatrix} \qquad \text{Actual}(z) = \begin{bmatrix} 0.64 \\ 0.99 \\ 0.94 \end{bmatrix}$$

Review

Pros of Neural Nets

1. It finds the best function approximation from a given set of inputs, we do not need to define features.
2. Representational Learning
   a. Used to get word vectors
   b. We do not need to handcraft image features

Cons of Neural Nets

1. It needs a lot of data, heavily parametrized by weights

Berkeley SCET