Neural Networks

Machine Learning and Data Mining

Dr Muhammad Hammad Saleem

Outline

- Introduction
- What is Neural Networks
- A Biological Neuron
- Mathematical Model
- Example
- Perceptron
- Artificial neural networks
- Learning algorithms (Case Study)
- Applications

Machine Learning

Machine learning is a general method of using a computer to extract meaning from data

Usually this involves identifying patterns in training set of data and then using those patterns to predict features of a future data

Here we will discuss the increasing use of **Artificial Neural Networks**

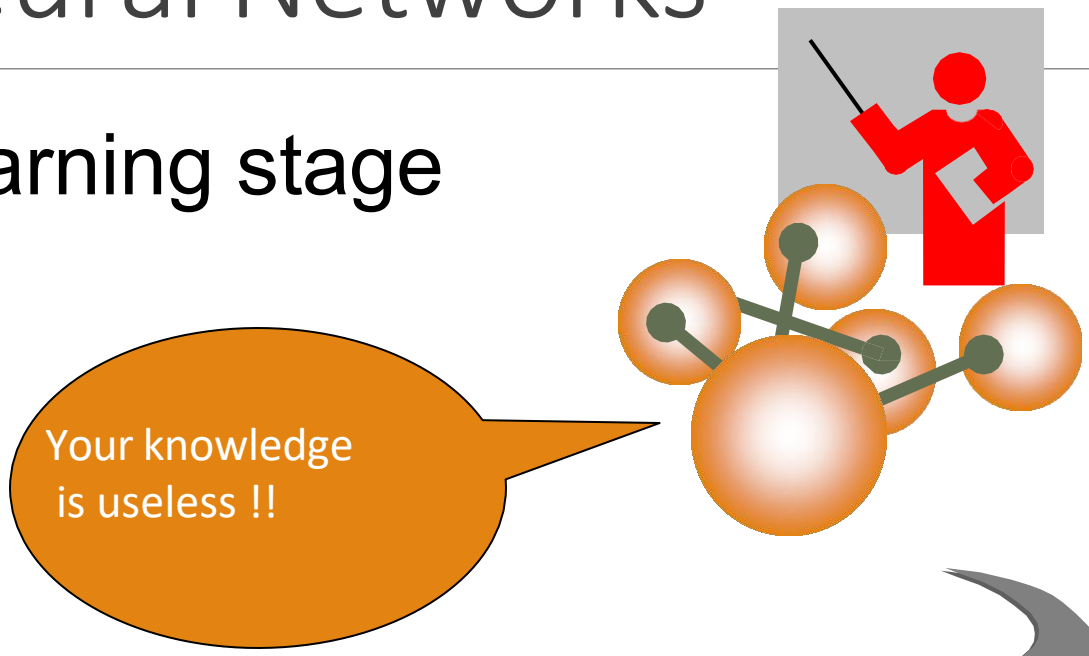
These were popularized by the work of John J Hopfield beginning in 1982

Earlier work was by McCulloch and Pitts (1943) and Hodgekin and Huxley (1952)

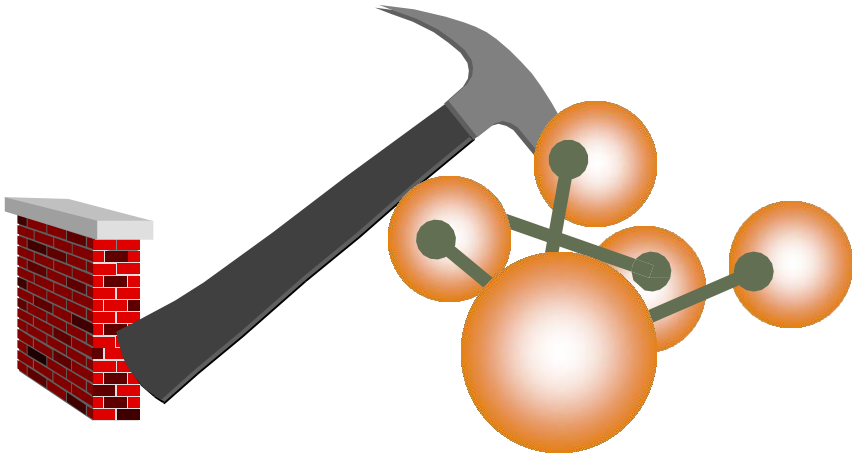
Nobel Physics Prize 2024: Geoffrey Hinton and John Hopfield

Neural Networks

1. Learning stage



2. Test stage
   (working stage)



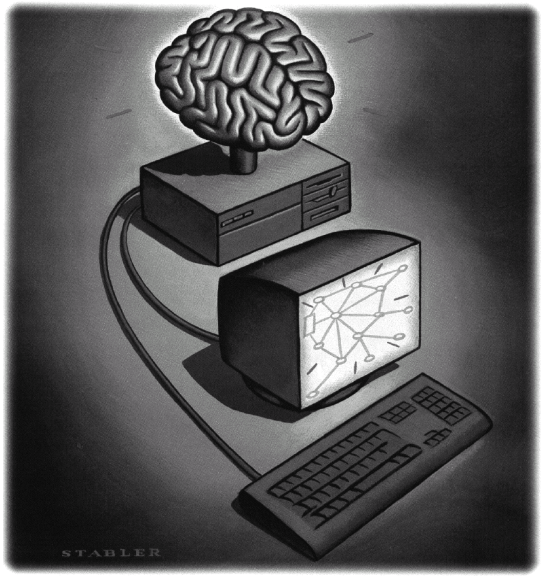
Why Use Neural Networks

- Neural networks are also ideally suited to help people solve complex problems in real-life situations.
- Can improve decision processes in areas such as:

- Credit card and Medicare fraud detection.
- Optimisation of logistics for transportation networks.
- Character and voice recognition, also known as natural language processing.
- Medical and disease diagnosis.
- Targeted marketing.
- Financial predictions for stock prices, currency, options, futures, bankruptcy and bond ratings.
- Robotic control systems.
- Electrical load and energy demand forecasting.
- Process and quality control.
- Chemical compound identification.
- Ecosystem evaluation.
- Computer vision to interpret raw photos and videos (for example, in medical imaging and robotics and facial recognition).

What is Neural Network?

•Biologically motivated approach to machine learning

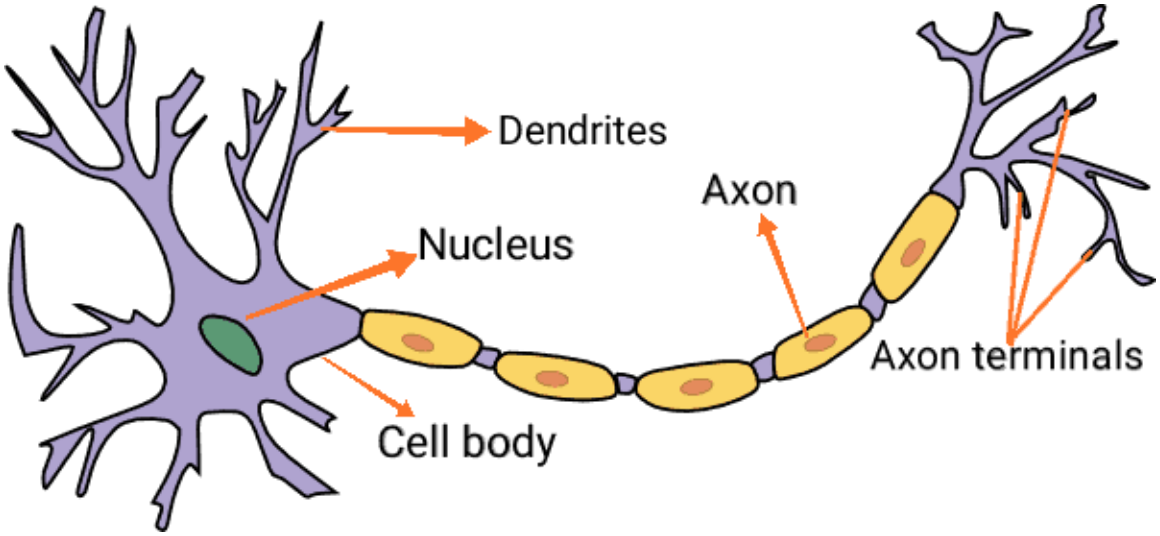


Similarity with biological network

Fundamental processing elements of a neural network is a neuron

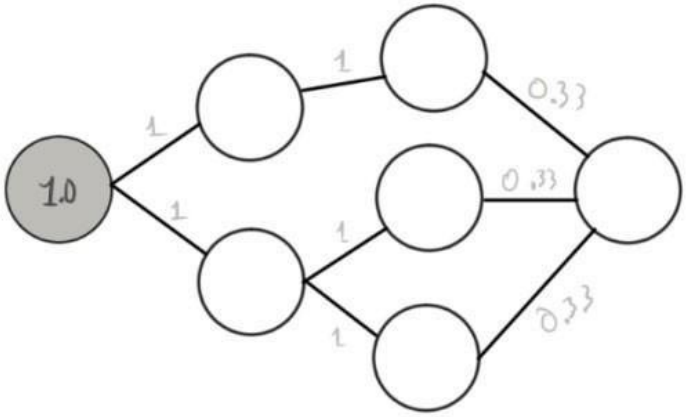
1. Receives inputs from other source
2. Combines them in someway
3. Performs a generally nonlinear operation on the result
4. Outputs the final result

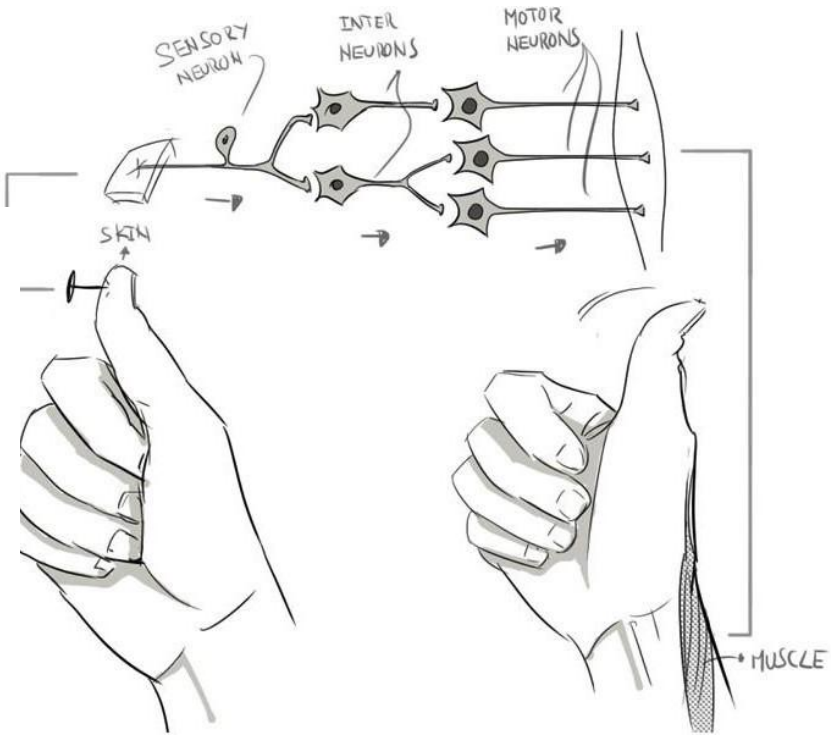
A Biological Neuron



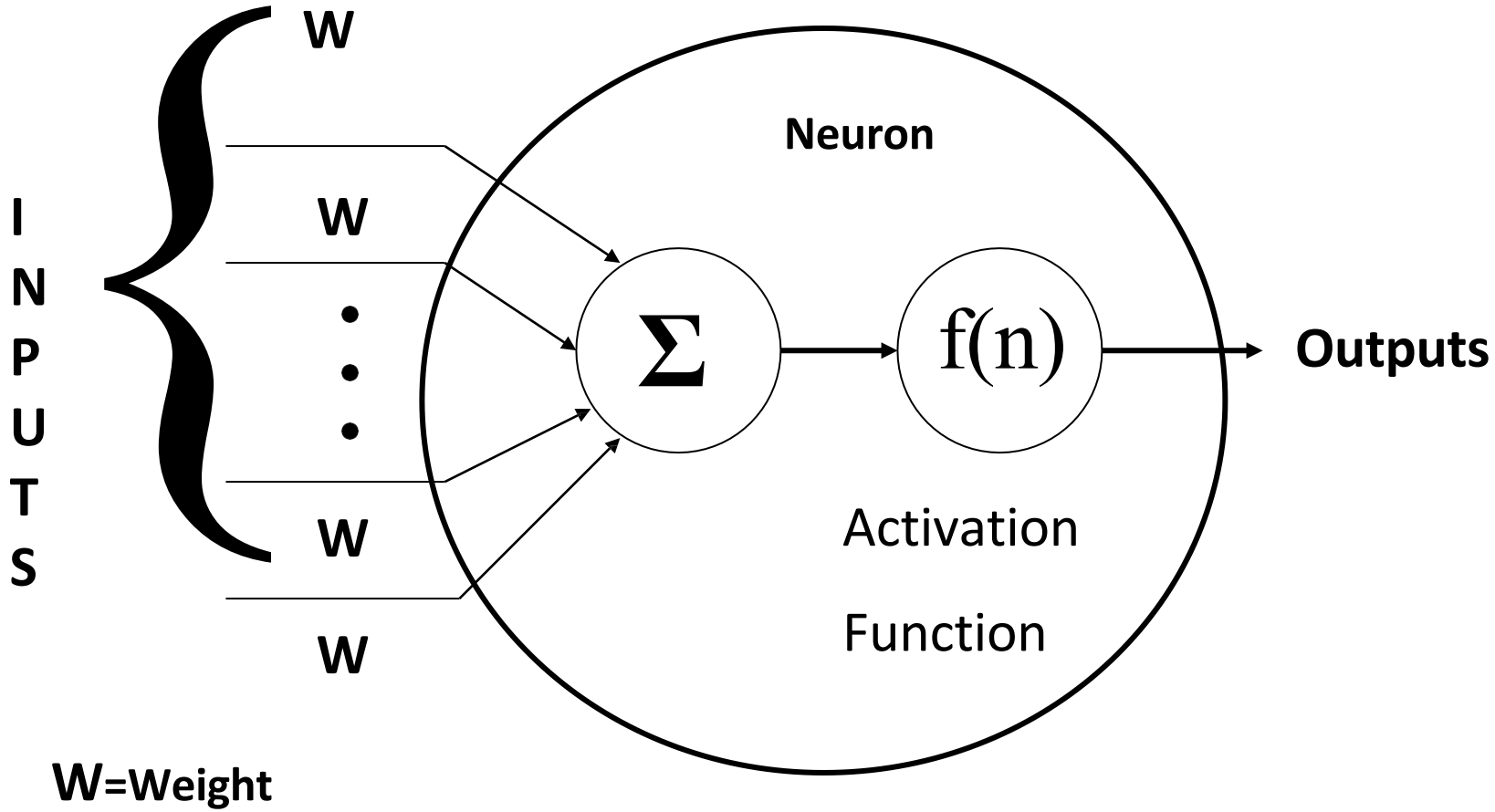
Minimal Biological Network

Generally described in a formal way with a ***graph.***

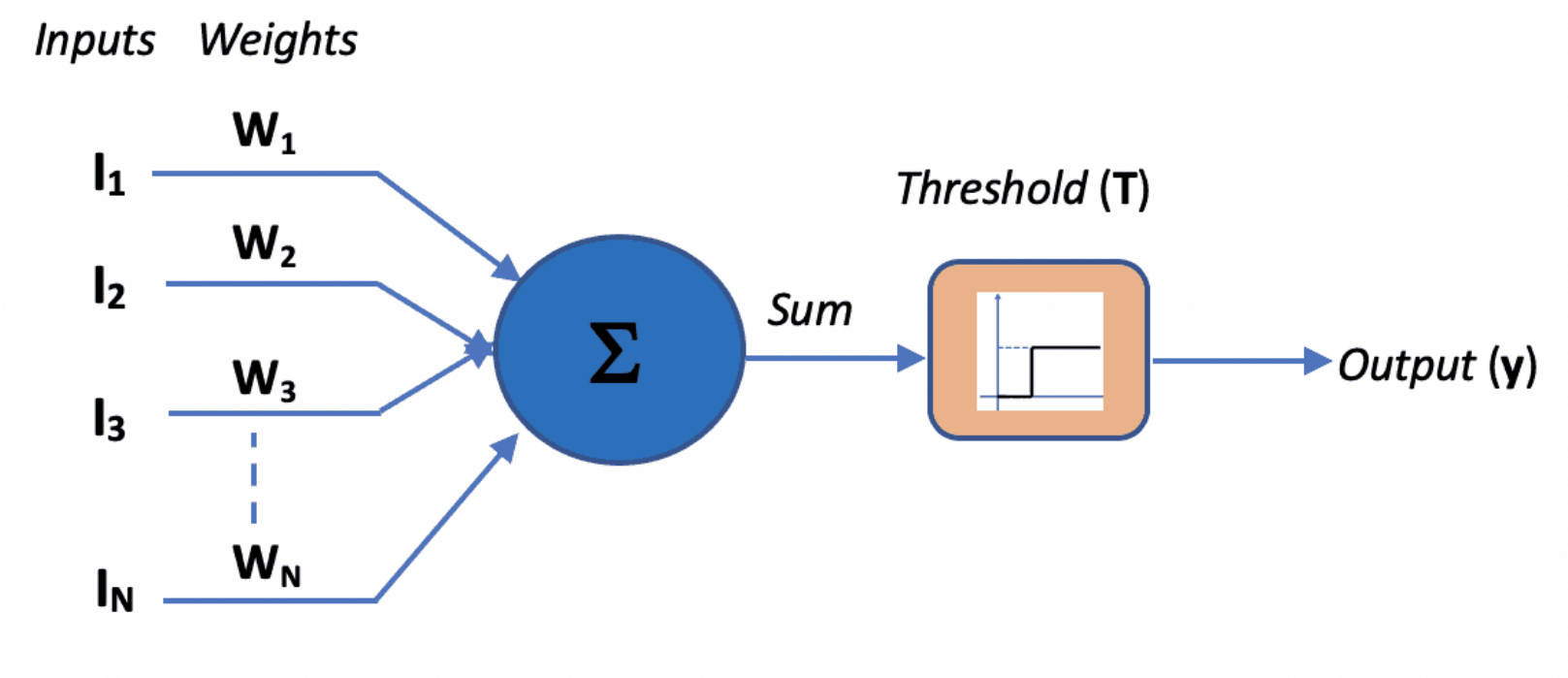




Mathematical Model



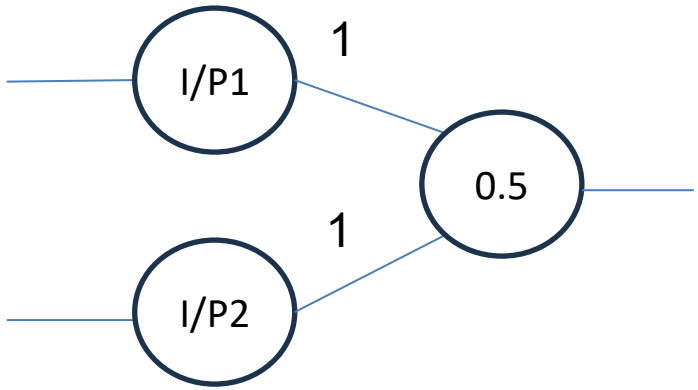
Simplest ANN



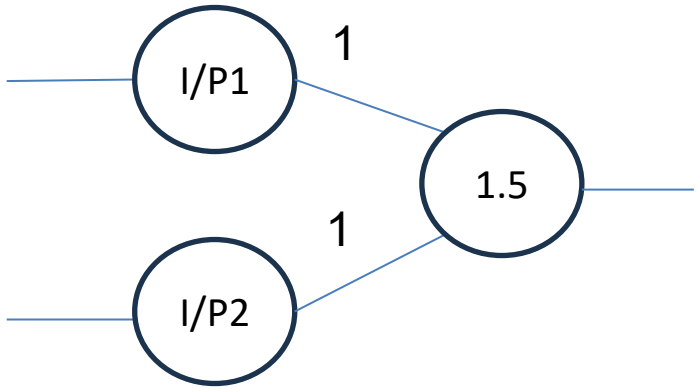
Simplest ANN (continued)

OR Gate		
I/P1	I/P2	O/P
0	0	0
0	1	1
1	0	1
1	1	1

AND Gate		
I/P1	I/P2	O/P
0	0	0
0	1	1
1	0	1
1	1	1



(0, 0): (0*1) + (0*1) = 0
(1, 0): (1*1) + (0*1) = 1



Example

Say you want to decide whether you are going to attend a music festival this upcoming weekend. There are three variables that go into your decision:

1. Is the weather good?
2. Does your friend want to go with you?
3. Is it near public transportation?

We'll assume that answers to these questions are the only factors that go into your decision.

A simple decision (continued)

We can use binary variables x_i to write the answers to the questions, with **zero** being the answer `no' and **one** being the answer `yes':

1. Is the weather good? $X_1 = 0$ or $X_1 = 1$
2. Does your friend want to go with you? $X_2 = 0$ or $X_2 = 1$
3. Is it near public transportation? $X_3 = 0$ or $X_3 = 1$

Can you think of a way to describe the decision statement that results from these inputs?

And what if not all features are equally important?

A simple decision (continued)

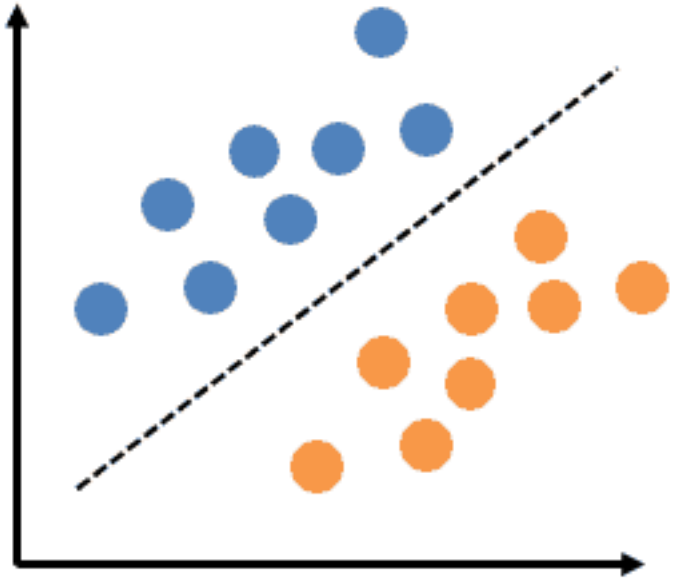
We could determine weights w_i indicating how important each feature is to whether you would like to attend.

Then

$$x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \geq \textit{threshold}$$

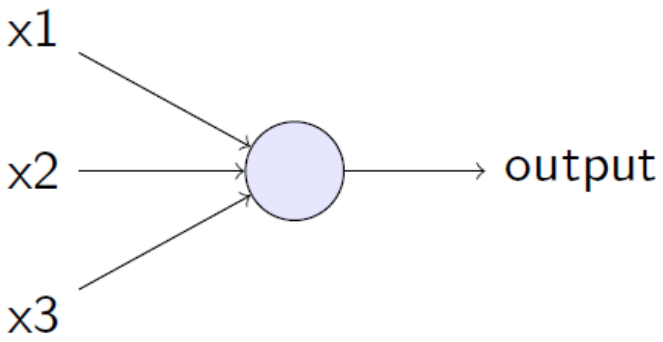
for some pre-determined *threshold*. If this statement is true, we would attend the festival, and otherwise we would not.

Perceptron



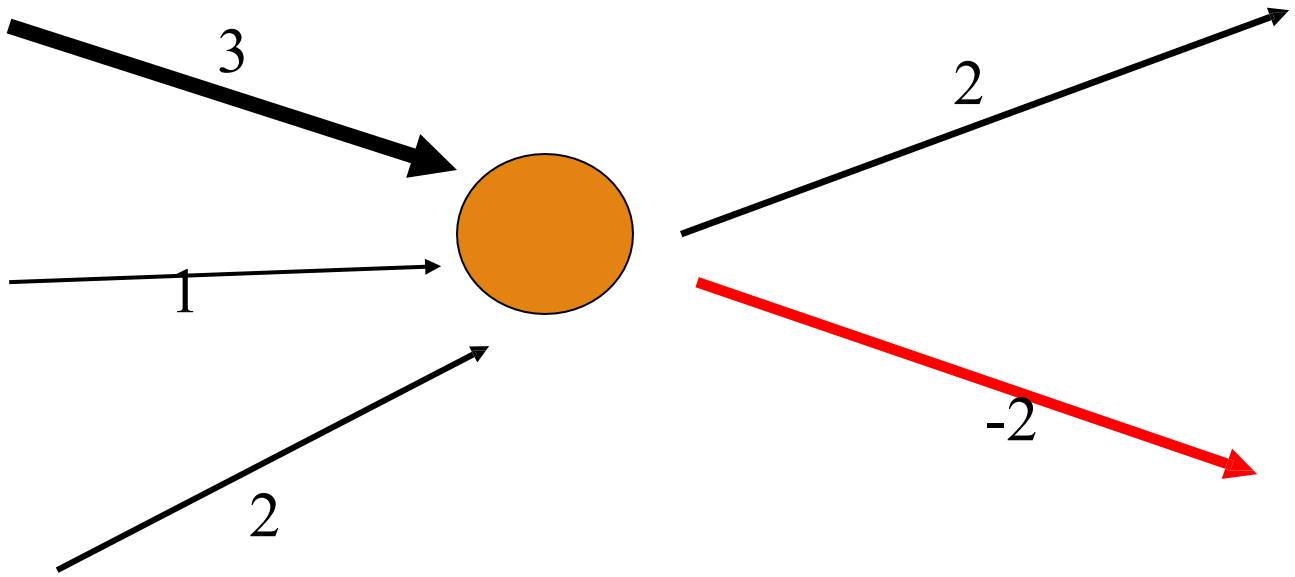
Perceptron (continued)

We can graphically represent this decision algorithm as an object that takes 3 binary inputs and produces a single binary output:

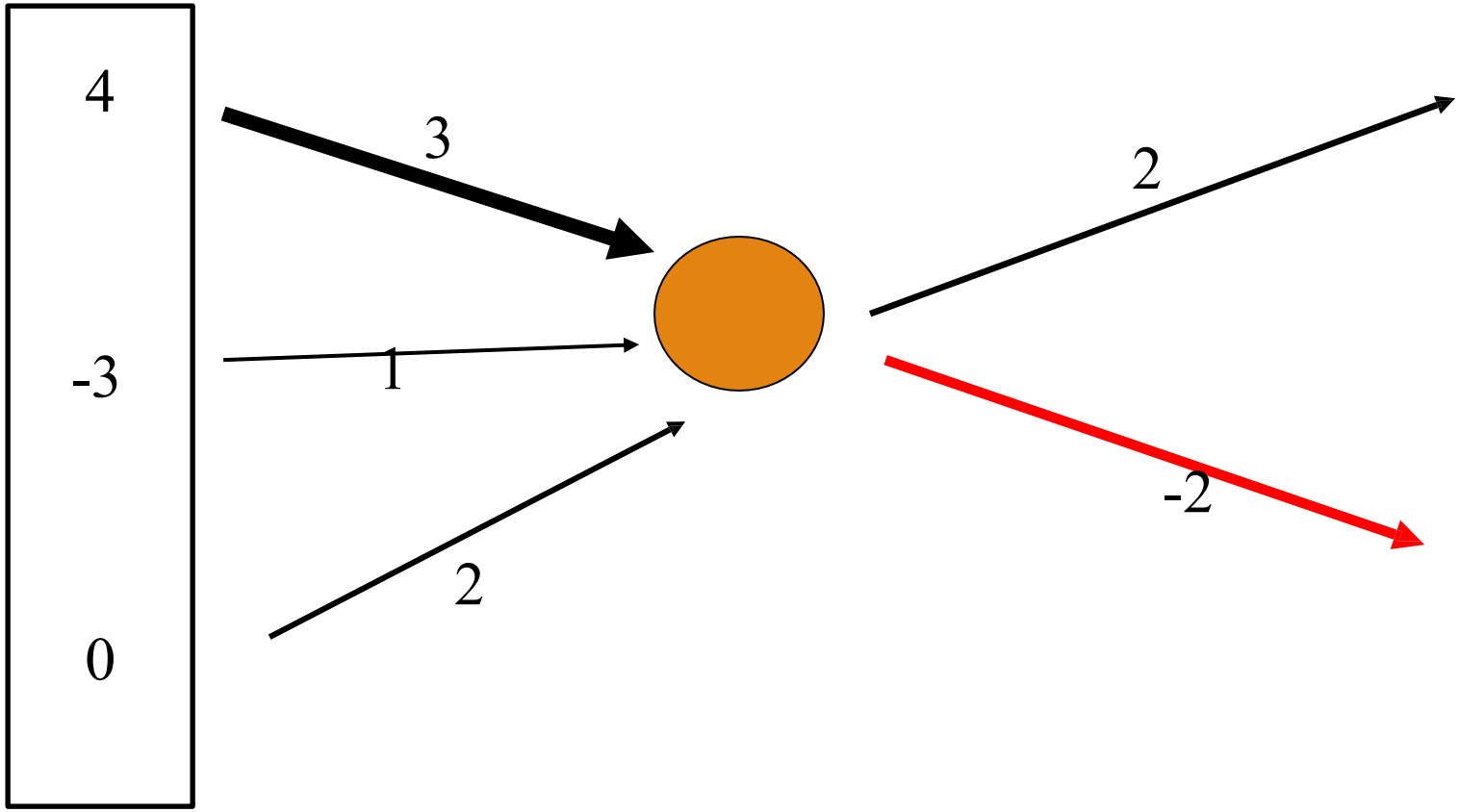


This object is called a perceptron when using the type of weighting scheme we just developed.

A single node (artificial neuron) works like this

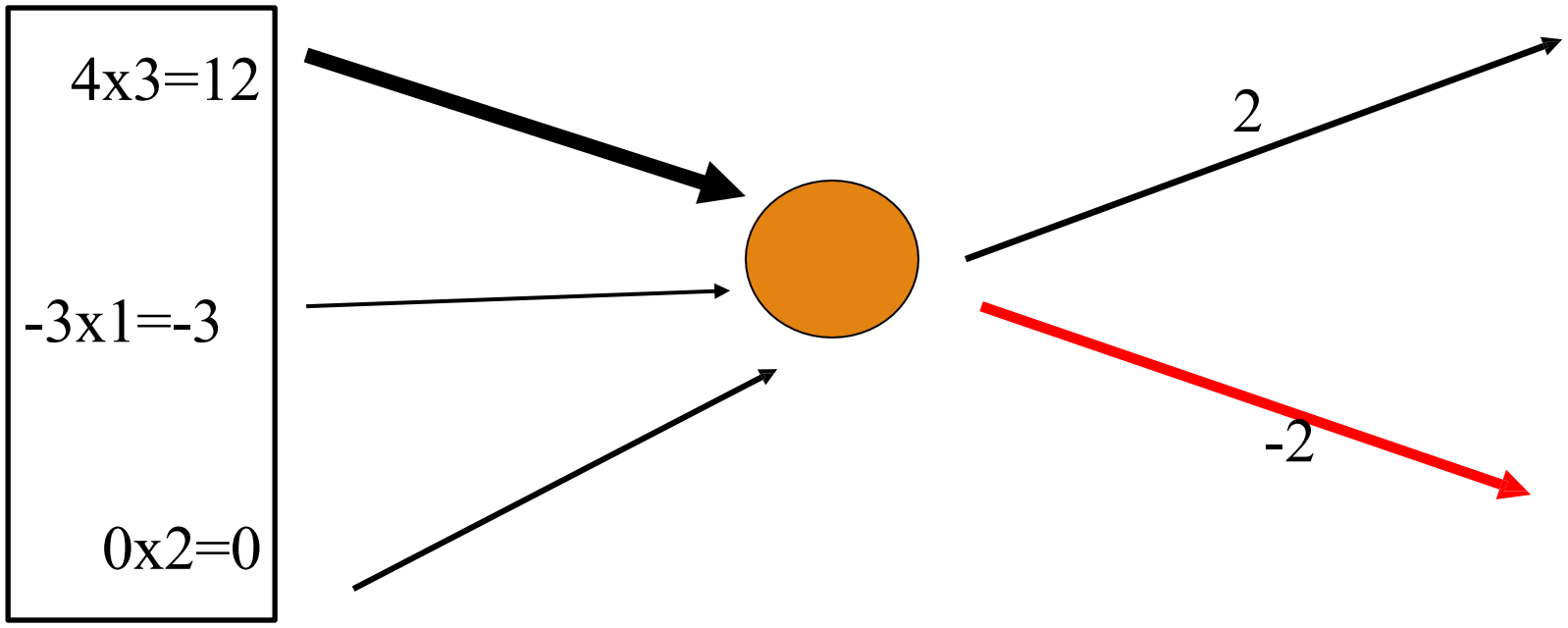


A single node (artificial neuron) works like this



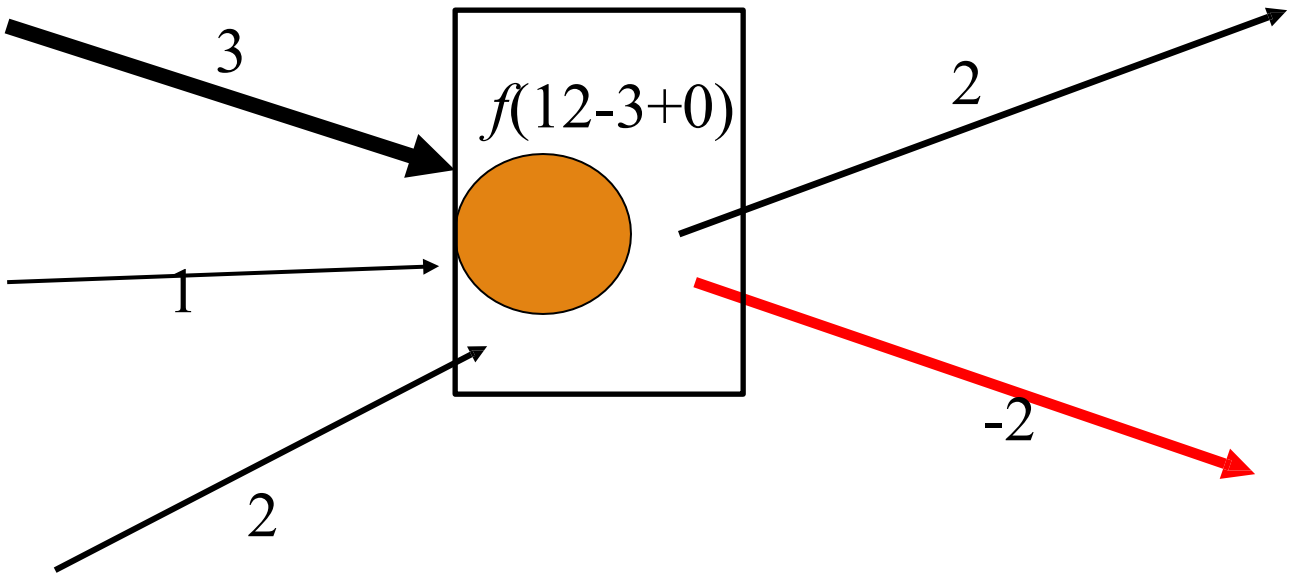
Field values come along (inputs from us, or from other nodes)

A single node (artificial neuron) works like this



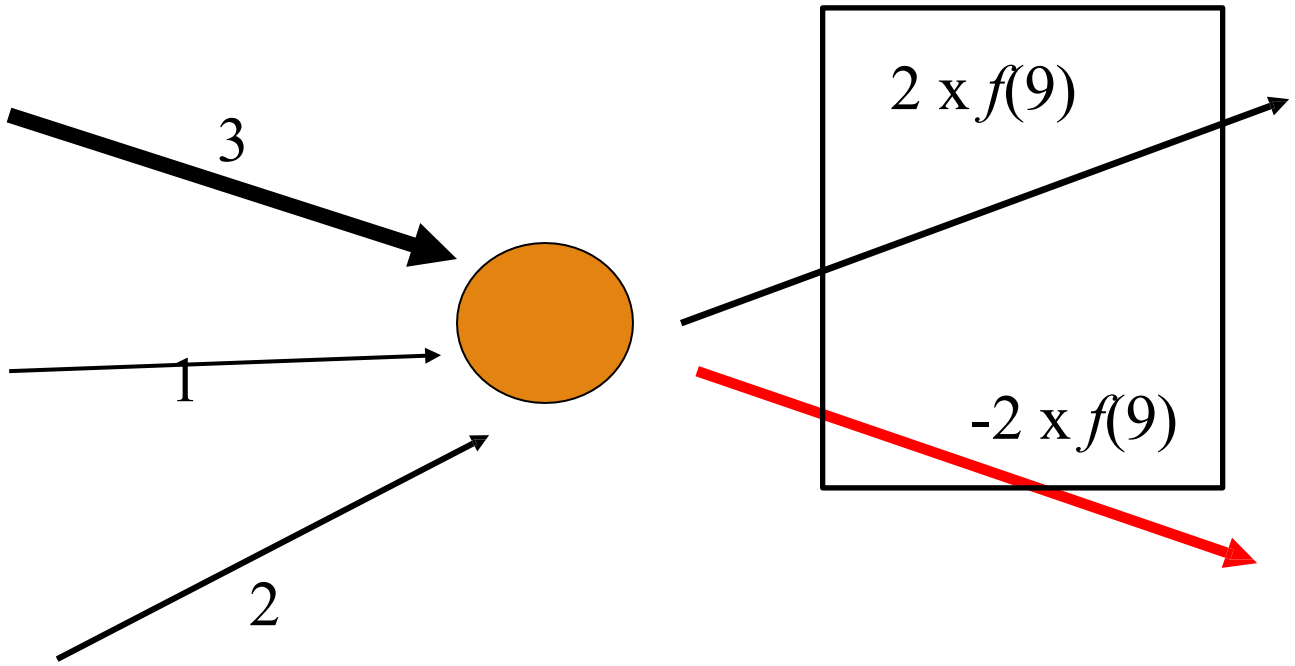
They get multiplied by the strengths on the input lines …

A single node (artificial neuron) works like this



The node adds up its inputs, and applies a simple function to it

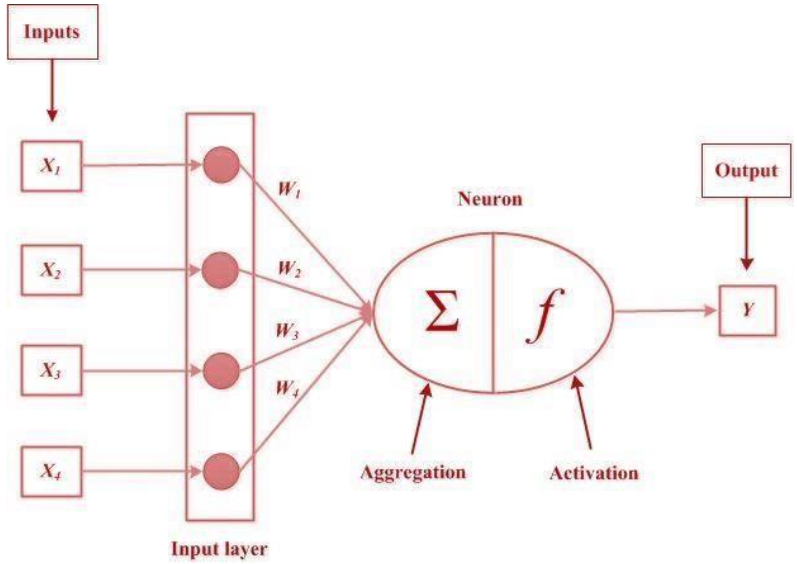
A single node (artificial neuron) works like this



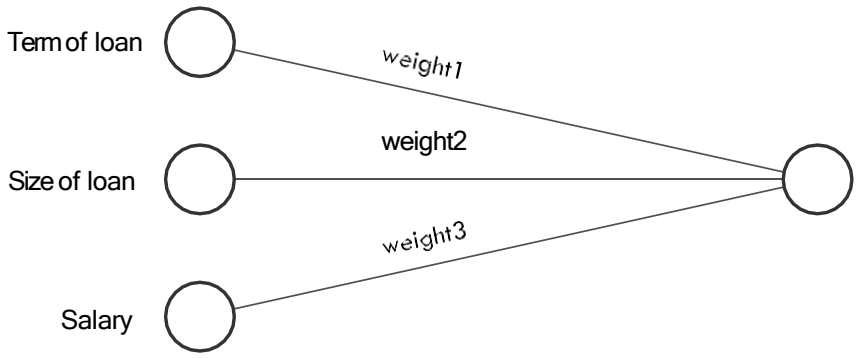
It sends the result out along its output lines, where it will in turn get multiplied by the line *weights* before being delivered …

Activation Function

- The activation function plays the key role of ensuring the output is mapped to the value range we want. In a binary classification problem, we would like the output to be between 0 and 1.
- This means we can interpret the output as the probability that the observation belongsto the positive class
- We'll cover other activation functions in next slides.



Predicting Loan Defaults



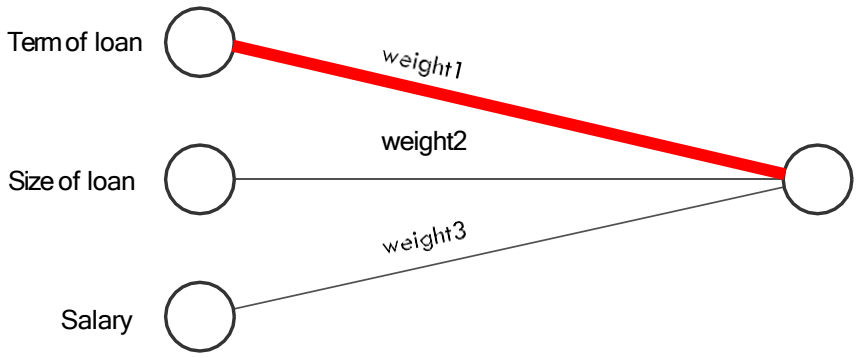
Will the customer default?
Value is between 0 and 1 and represents the probability the customer will default.
For example, an output of 1 would mean 100% chance of default

Input Layer $\in \mathbb{R}^3$

Output Layer $\in \mathbb{R}^1$

Multiply each input by its weight → Sum them all together → Apply sigmoid function

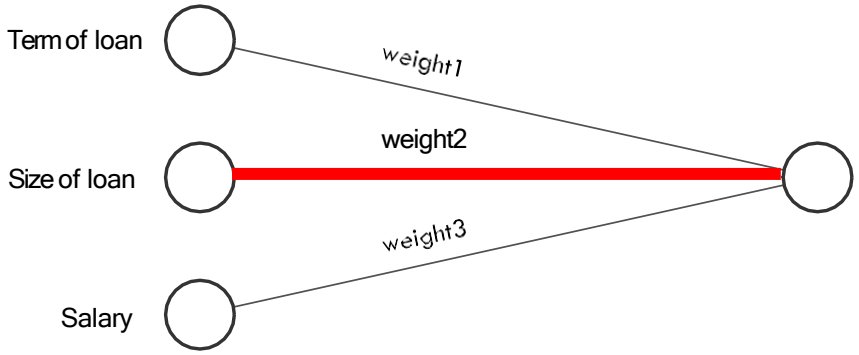
Predicting Loan Defaults



For example, imagine that the term of the loan is not important for predicting whether the customer will default. We can then set the weight close to 0 so it doesn't impact the prediction much

Multiply each input by its weight → Sum the mall together → Threshold

Predicting Loan Defaults

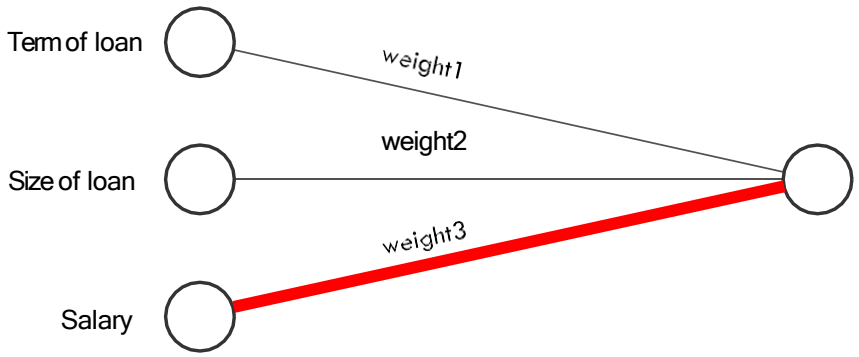


Imagine that the size of the loan is more important, and that as this increases, the chance of default increases too. We can therefore have a higher value for the weight associated with this neuron, so that higher values of loan leads to an output closer to 1.

Input Layer $\in \mathbb{R}^3$ Output Layer $\in \mathbb{R}^1$

Multiply each input by its weight → Sum the mall together → Threshold

Predicting Loan Defaults



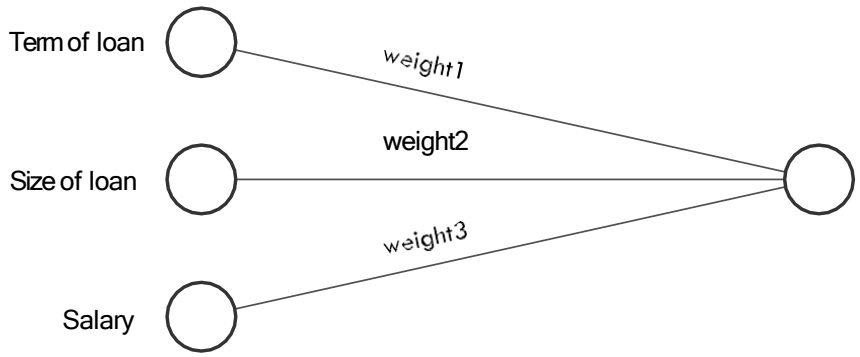
And maybe the lower the salary the greater the chance of default. In this case we want the weight to be a <u>negative</u> number

Input Layer $\in \mathbb{R}^3$ Output Layer $\in \mathbb{R}^1$

Multiply each input by its weight → Sum the mall together → Threshold

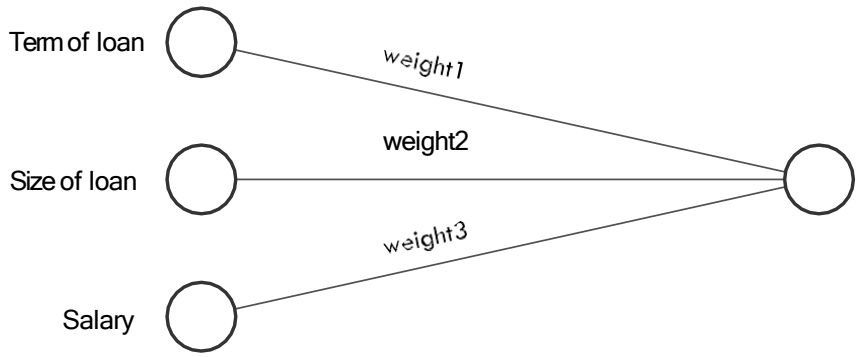
Predicting Loan Defaults

- So far, with a single perceptron using a sigmoid activation function we can represent simple relationships between the input and the output – for example, as the size of the loan increases, so does the probability of default.
- But what about if the situation is more complicated? For example, what if defaults are more common on loans with under 2 years or over 10 years?



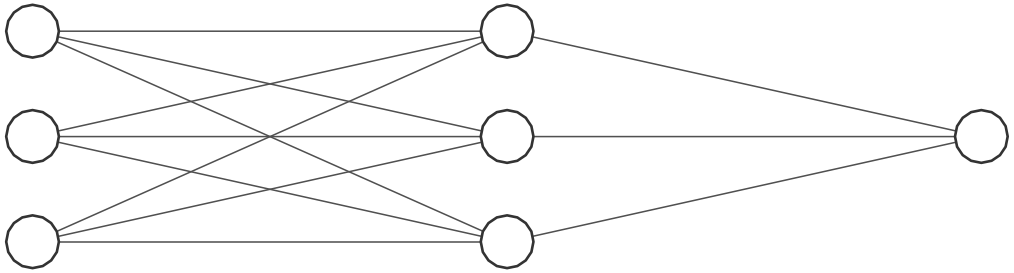
Predicting Loan Defaults

- Or what if there is a more complex relationship between the input variables?
- E.g. people with high salaries are less likely to default on smaller loans but more likely to default on large loans?
- Our simple perceptron cannot capture these kinds of relationship!

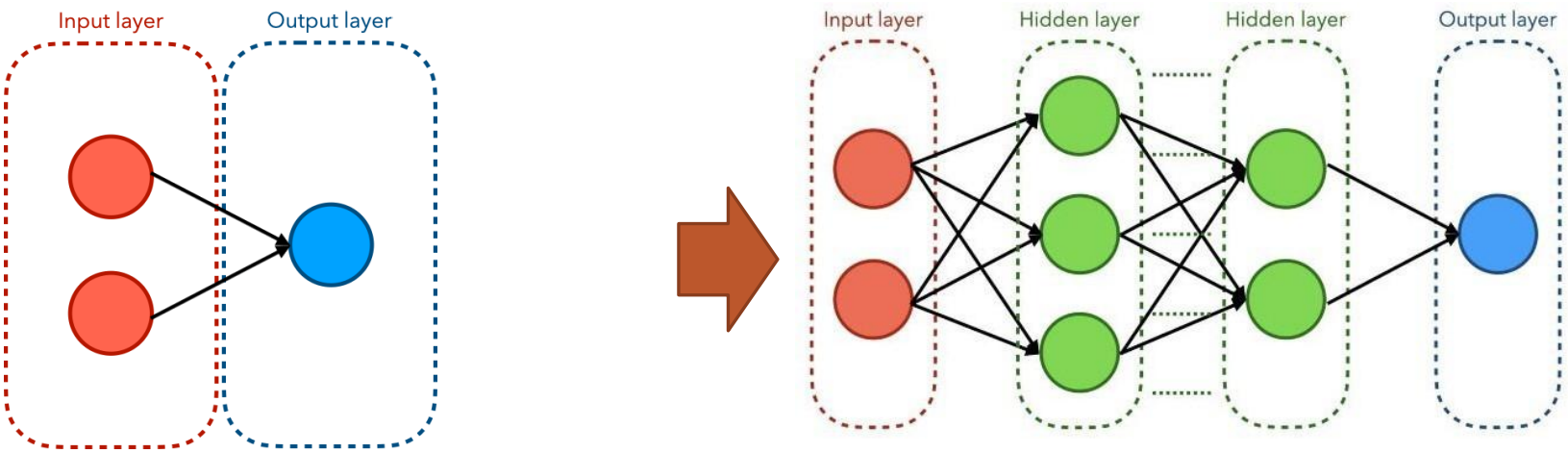


Hidden Layers

- The solution is to add **hidden layers.**
- These allow the model to learn more complex relationships between the input and the output.
- In practise, we will always use a neural network with at least one hidden layer
- In deep learning, neural networks can have hundreds of hidden layers!



Neural Network Layers



Motivation for Artificial Neural Networks

Algorithms experience the world through data — by training a neural network on a relevant dataset, we seek to decrease its ignorance. The way we measure progress is by monitoring the error produced by the network.

Before delving into the world of neural networks, it is important to get an understanding of the motivation behind these networks and why they work.

We will see that neural networks are arranged in feedforward layers, called *perceptrons*

Theory of ANN

An artificial neural network is a supervised learning algorithm which means that we provide it the input data containing the independent variables and the output data that contains the dependent variable.

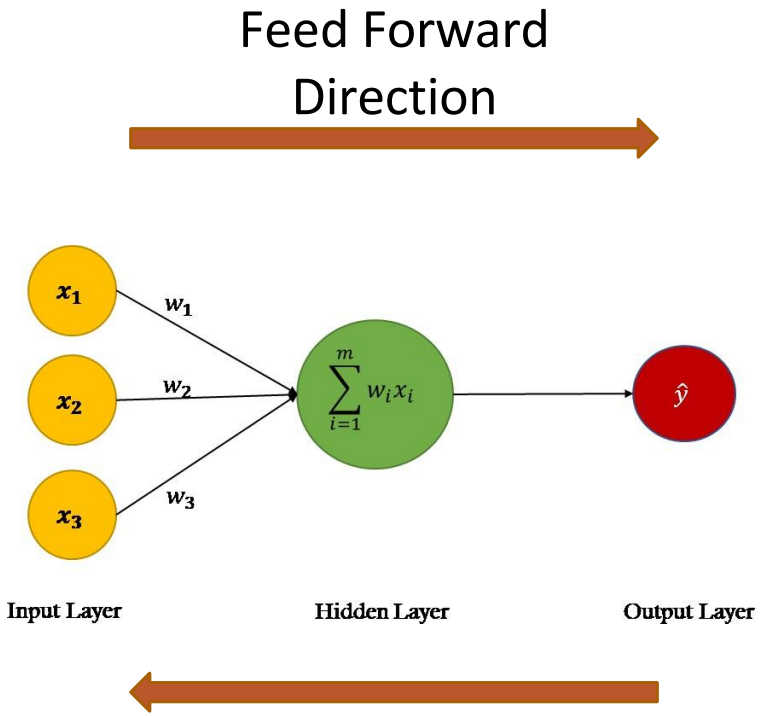
For instance, in our example our independent variables are X1, X2 and X3. The dependent variable is Y.

Thus **X** -> Y, where **X** is a *vector of data or features*, and Y is a *label*

- Number of input nodes: determined by the dimension of dataset
- Number of output nodes: determined by the number of dependent variables
  - For classification: determined by the number of classes and coding scheme
  - For regression: same as the number of dependent variables

Theory of ANN

A neural network executes in two phases: **Feed Forward** phase and **Back Propagation** phase.

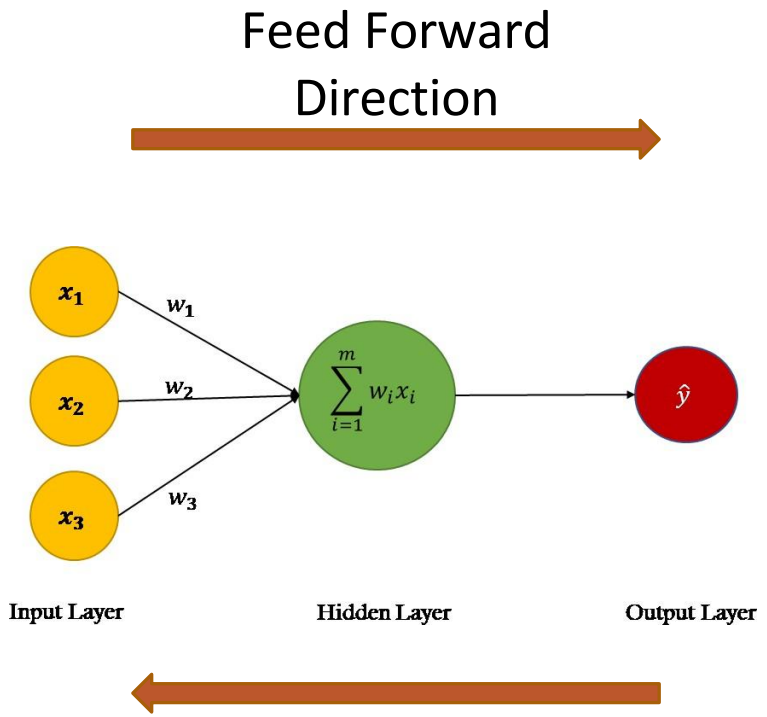


Feedforward Phase of ANN

Step 1: Calculate the dot product between inputs and weights

Mathematically, the summation of the dot product is:

X*W=x1*w1 + x2*w2 + x3*w3 + b



Bias Term b

Suppose if we have a person who has input values (0,0,0), the sum of the products of input nodes and weights will be zero.

In that case, the output will always be zero no matter how much we train the algorithms.

Therefore, in order to be able to make predictions, even if we do not have any non-zero information about the person, we need a bias term.

The bias term is necessary to make a robust neural network.

$X*W=x1*w1 + x2*w2 + x3*w3 + b$

Activation Term

Step 2: Pass the summation of dot products (X.W) through an activation function

The dot product XW can produce any set of values.

However, in our output, we have the values in the form of 1 and 0.

We want our output to be in the same format.

To do so we need an **Activation Function**, which restricts the input values between 0 and 1.

Activation Functions

- Perceptron shortcoming: small change in the input values can cause a large change the output because each node (or neuron) only has two possible states: 0 or 1.

- Want: continuum of values, say any number between 0 and 1.
  - Activation function

Activation functions

Controls when unit is "active" or "inactive"

The main purpose of most activation functions is to introduce non-linearity in the network so it would be capable of learning more complex patterns.

Handy for it to be differentiable (have a slope): When updating the curve, to know in which direction and how much to change or update the curve depending upon the slope.

Activation Functions

Step Function

Binary step function depends on a threshold value that decides whether a neuron should be activated or not

$f(x) = 0 \text{ for } x<0$

$f(x)=1 \text{ for } x>=0$

Linear

The linear activation function, also known as "no activation," or "identity function" (multiplied x1.0), is where the activation is proportional to the input.

$F(x)=x$

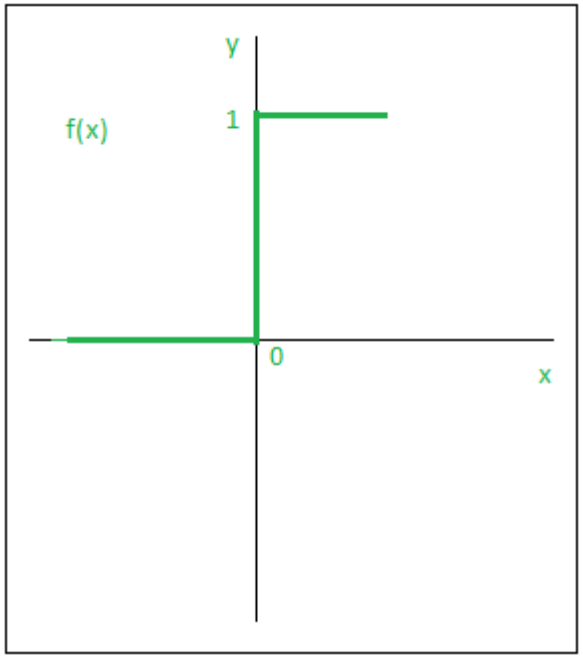
Non-Linear

Sigmoid, Tanh, Relu

Activation functions: Step function

$$\text{output} = \begin{cases} 1 & \text{if input} > 0 \\ 0 & \text{if input} \leq 0 \end{cases}$$

Either 0 or 1

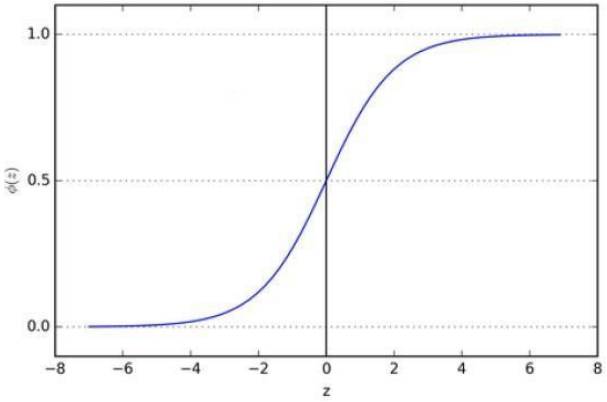


Activation functions: sigmoid function

- As discussed at the start, we can use a sigmoid activation function on the output layer when we have a binary classification problem because this always **outputs a value between 0 and 1**
- We can also use it as the activation function for the hidden layers too.

- Sigmoid function

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

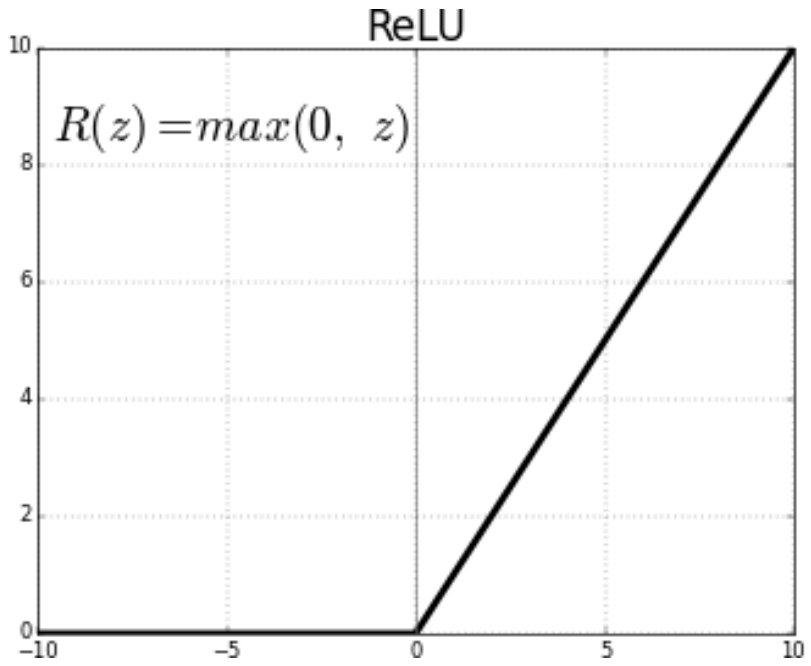


Important points:

- Exists between (0,1) → can predict probability.
- It is differentiable.
- However, can get stuck at training time.

Activation functions: ReLU function

- An activation function which is widely used in the hidden layers in called the Rectified Linear Unit, or ReLU for short.
- If the weighted sum of the inputs is less than 0, it outputs 0
- If the weighted sum of the inputs is more than 1, it outputs the weighted sum of the inputs

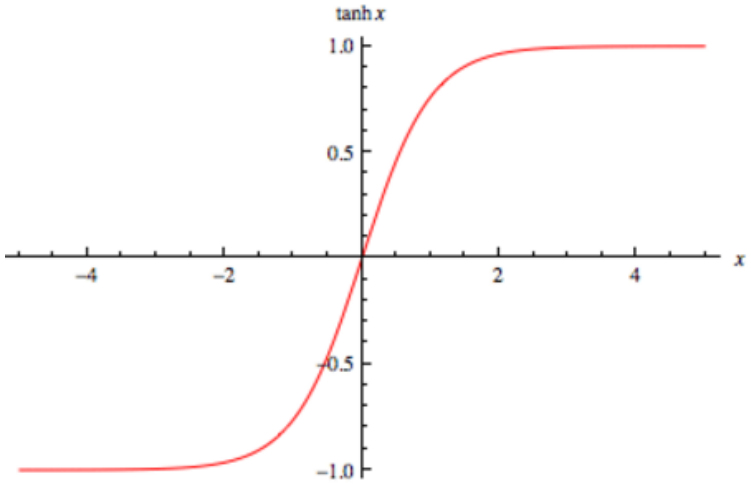


if it's positive, keep it; if it's negative, throw it away.

Activation functions: Tanh function

A smoother version of the Sigmoid, but with outputs ranging from -1 to 1 instead of 0 to 1.

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



Activation functions: Softmax

- The softmax function is used in the output layer when we are conducting multiclass classification
- The softmax function ensures that **the sum of the output neurons is equal to 1.**
- This means we can interpret the value for any one output neuron as the probability that the observation belongs to that class.

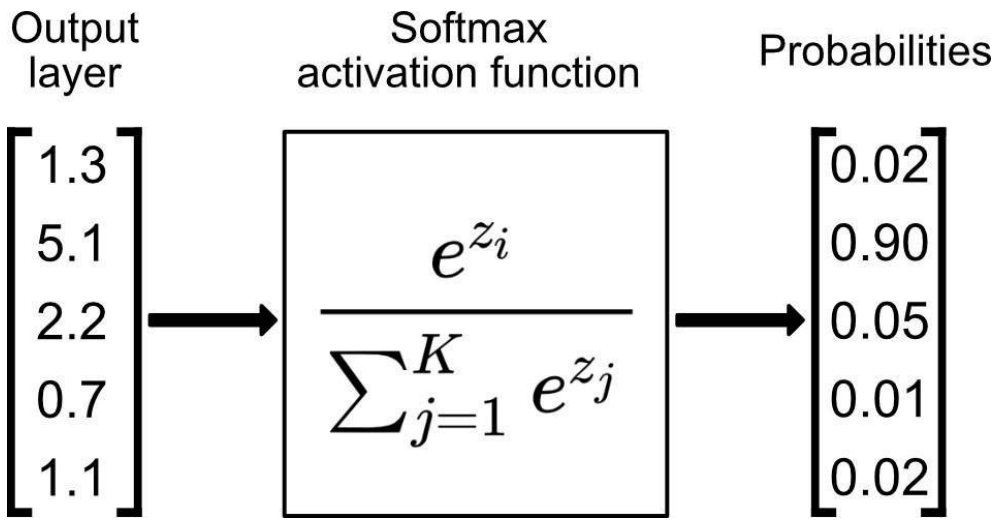


Image credit: https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60

Activation functions: Key Points!

- For hidden layers, we need a non-linear activation function – often we choose ReLU
- For the output layer, the key point is that we can choose the activation function which fits our problem
- For the output layer, we choose a function which ensures the inputs to the output neuron(s) are mapped into **outputs which make sense for our problem**, whether that is binary classification, multi-class classification or regression.

Step 3: Training the Neural Network

- We've discussed how the weights can be used to generate a prediction for a given set up values for the input variables.
- But how do we set these weights in the first place? How do we train the network?

Back Propagation is the General Training Method

In the beginning, before you do any training, the neural network makes random predictions which are of course incorrect.

We start by letting the network make random output predictions.

We then compare the predicted output of the neural network with the actual output.

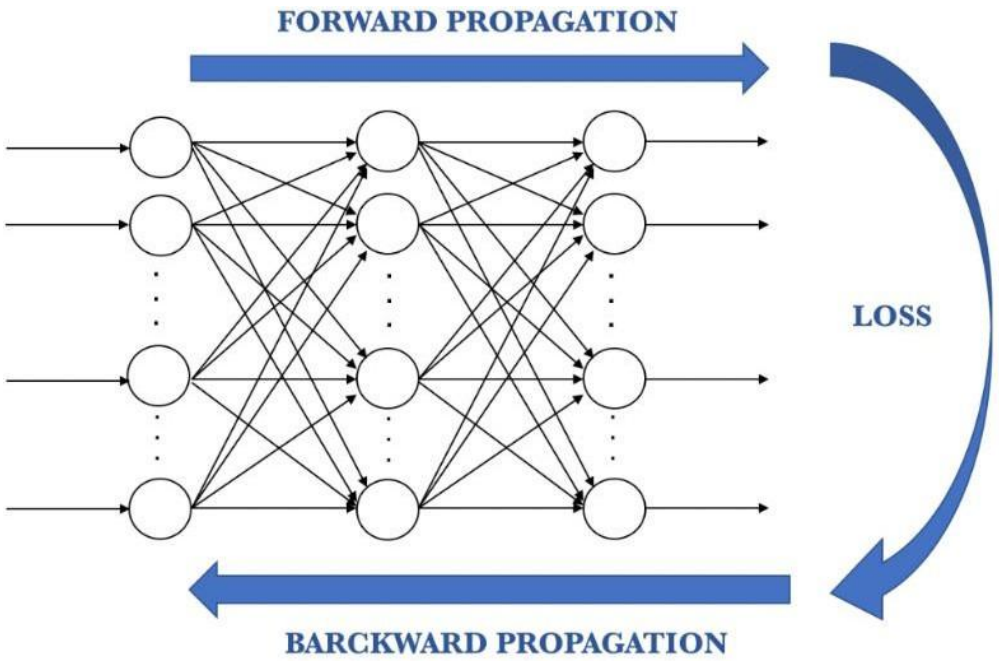
We then calculate the error or loss function

Next, we update the weights and the bias in such a manner that our predicted output comes closer to the actual output.

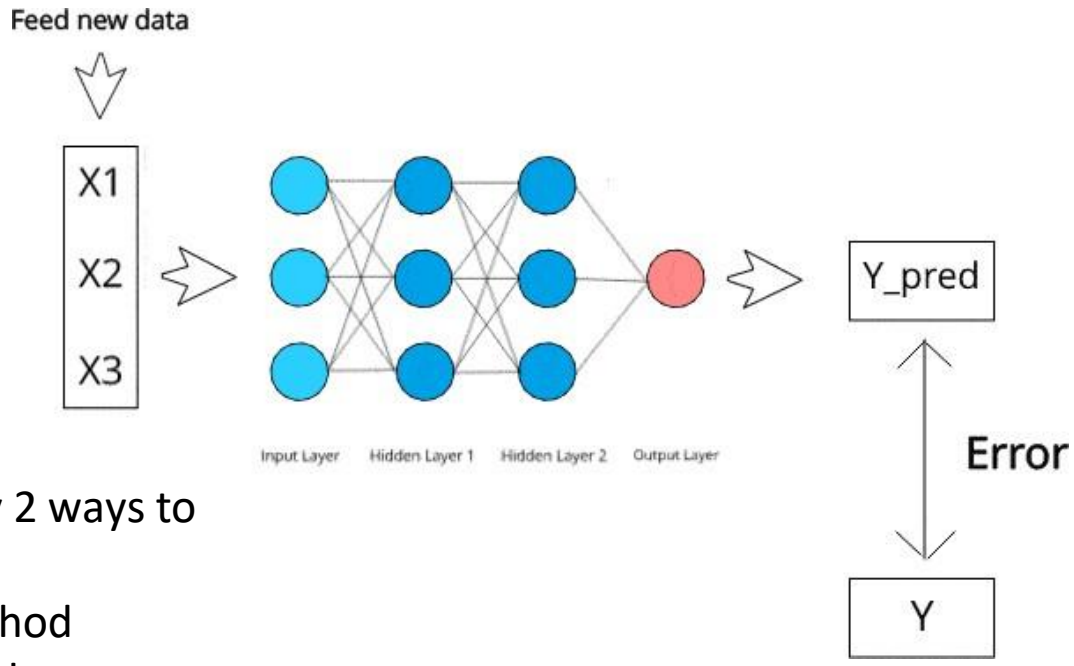
In this phase, we can train our algorithm via *backpropgation*.

Let's take a look at the steps involved in the backpropagation phase.

Training an ANN by Backpropagation



Backpropagation to Adjust Weights



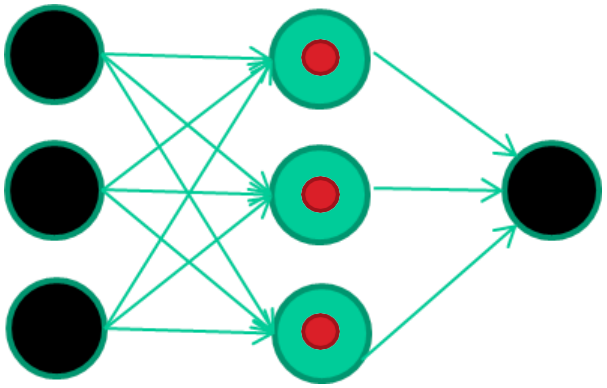
There are basically 2 ways to adjust weights:

1. Brute-force method
2. Batch-Gradient descent

Training the neural network

Dataset

X1	X2	X3	Class
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

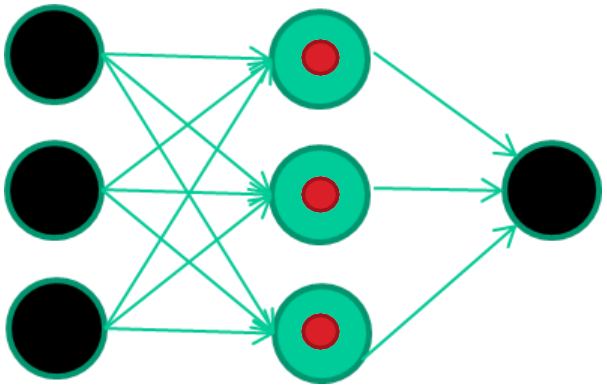


Training the neural network

Dataset

X1	X2	X3	Class
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

Initialise with random weights



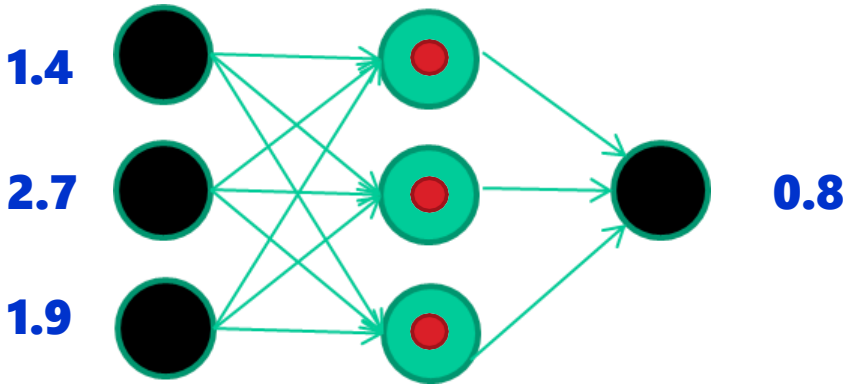
Training the neural network

Dataset

1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

Initialise with random weights

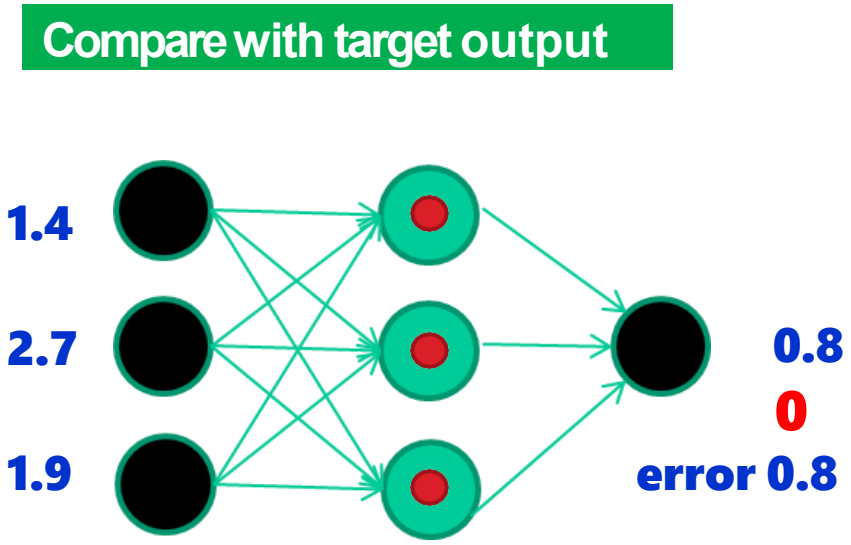
Feed it through to get output



Training the neural network

Dataset

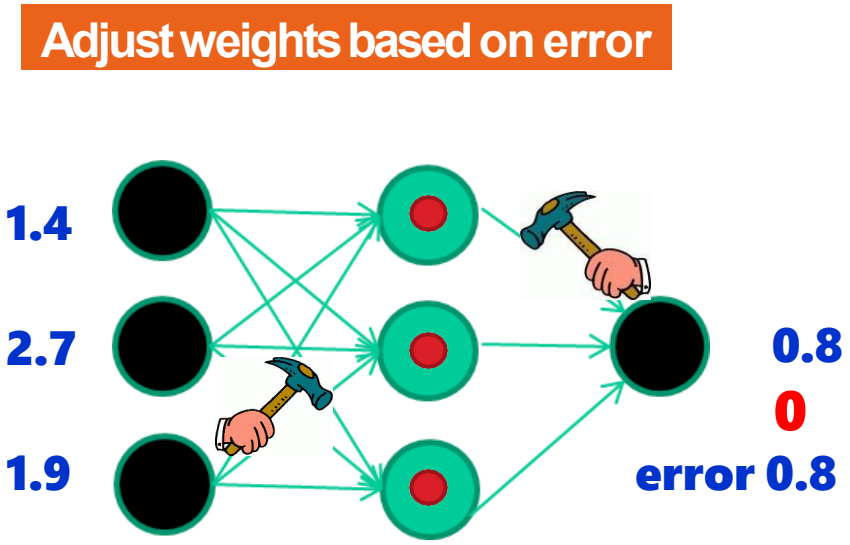
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0



Training the neural network

Dataset

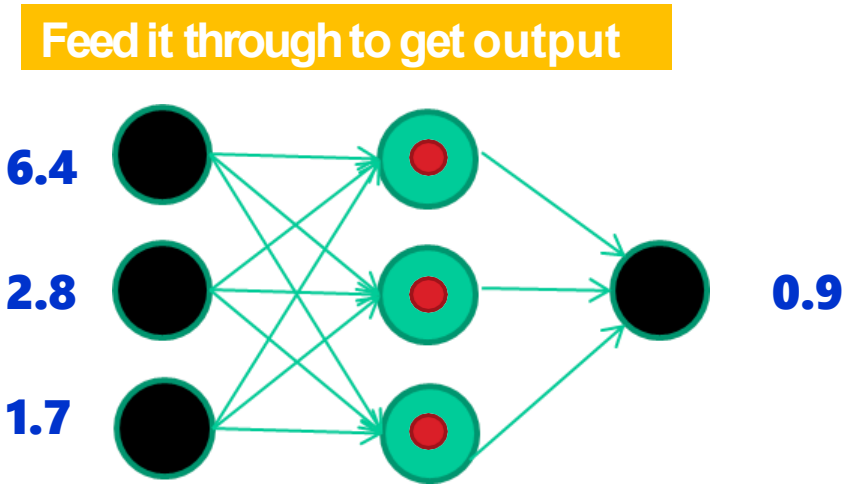
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0



Training the neural network

Dataset

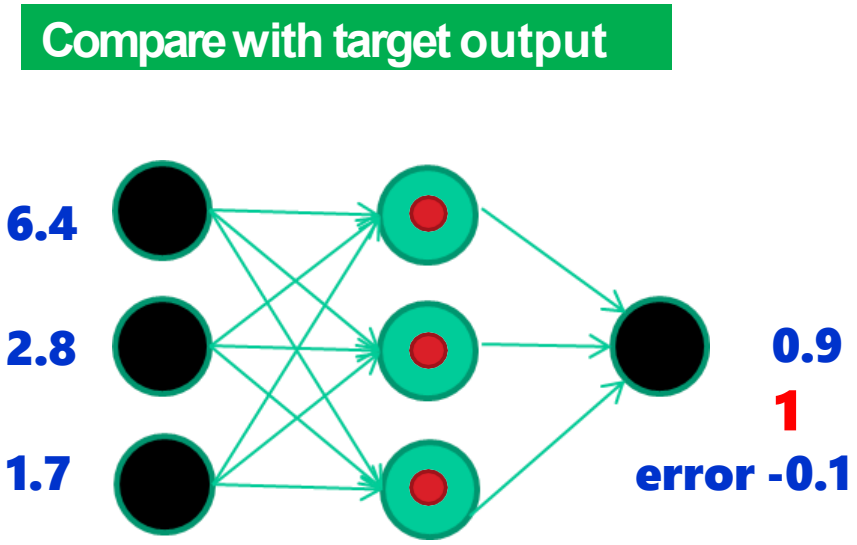
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0



Training the neural network

Dataset

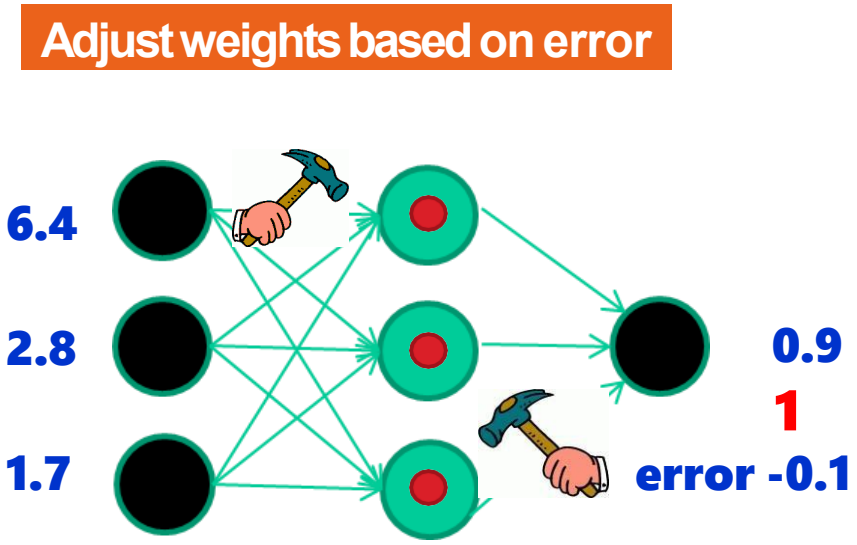
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0



Training the neural network

Dataset

1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

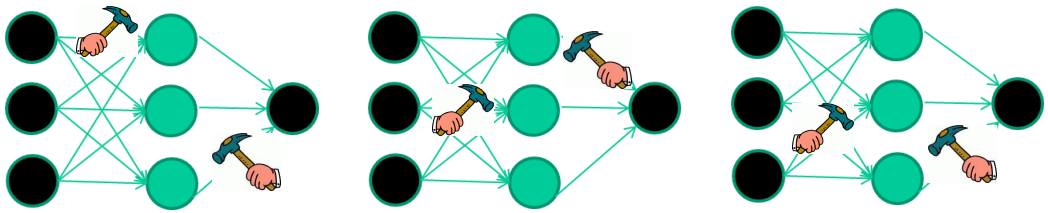


Training the neural network

And so on ....

Dataset

1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

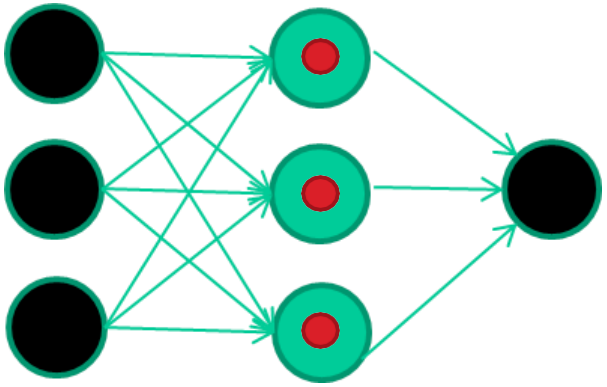


Repeat this process thousands, maybe milions of times - each time taking a random training instance, and making slight weight adjustments

– Algorithms for weight adjustment are designed to make changes that will reduce the error

The loss function

- We mentioned that the weights in the network are updated based on the error, but how do we measure this?
- We use a function called the loss function to quantify how well the neural network models the training data
- The backpropagation algorithm updates the weights to try to minimise the loss
- We have to choose the loss function to fit the problem. For multiclass classification we use categorical cross-entropy, for binary classification we use binary cross-entropy



The loss function: Binary cross-entropy

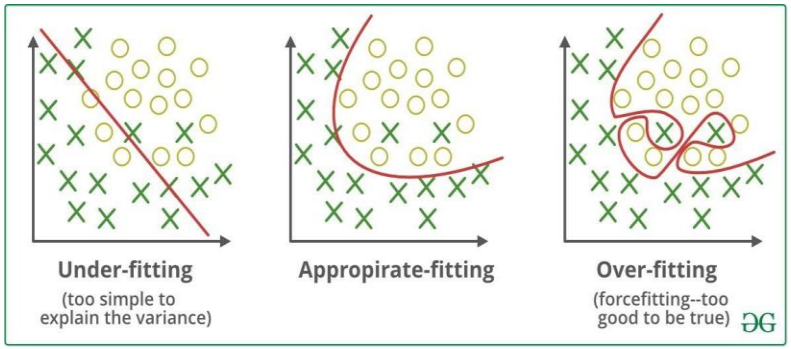
- Cross-entropy is used to quantify the difference between two probability functions. In the context of machine learning, binary cross-entropy is a measure of the error in a binary classification problem.

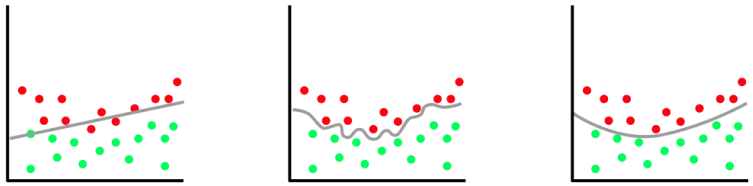
$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

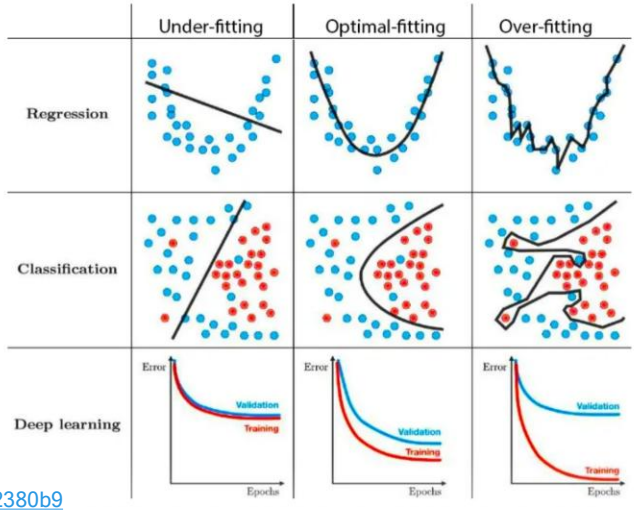
- If the output was 0.51 and the correct value of y was 0 the loss for that prediction would be $-(0 \times \log_2(0.51) + 1 \times \log_2(0.49)) = 1.03$
- If the output was 0.99 and the correct value of y was 0 the loss for that prediction would be $-(0 \times \log_2(0.99) + 1 \times \log_2(0.01)) = 6.64$
- **The main thing to remember is that the further our predicted probability is from the true class label, the higher the binary cross-entropy is!**

Training the neural network

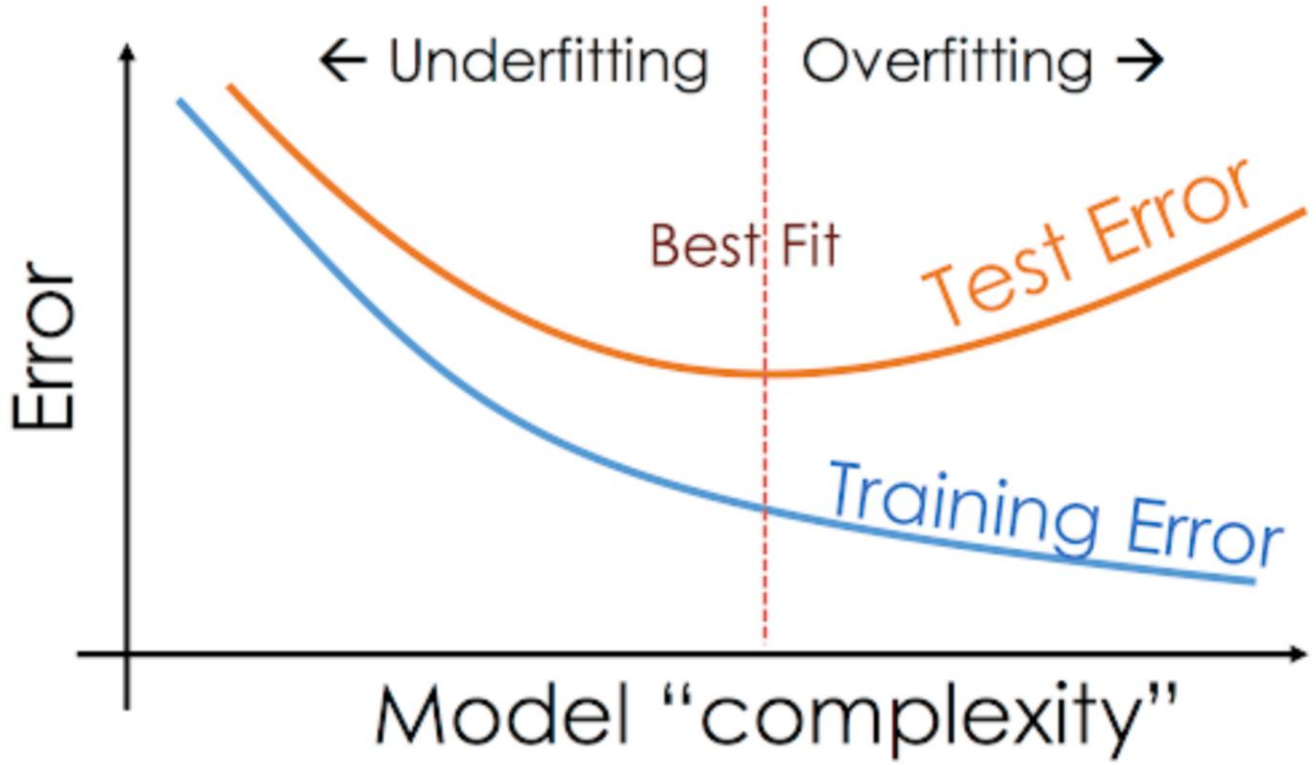
- In one epoch, the full training dataset is passed through the network once.
- Getting the number of epochs right is important. Too many can lead to the network overfitting the training data and too few can lead to the network underfitting the training data.
- We can use early stopping to bring the training to the halt early in the error on the validation set is no longer improving





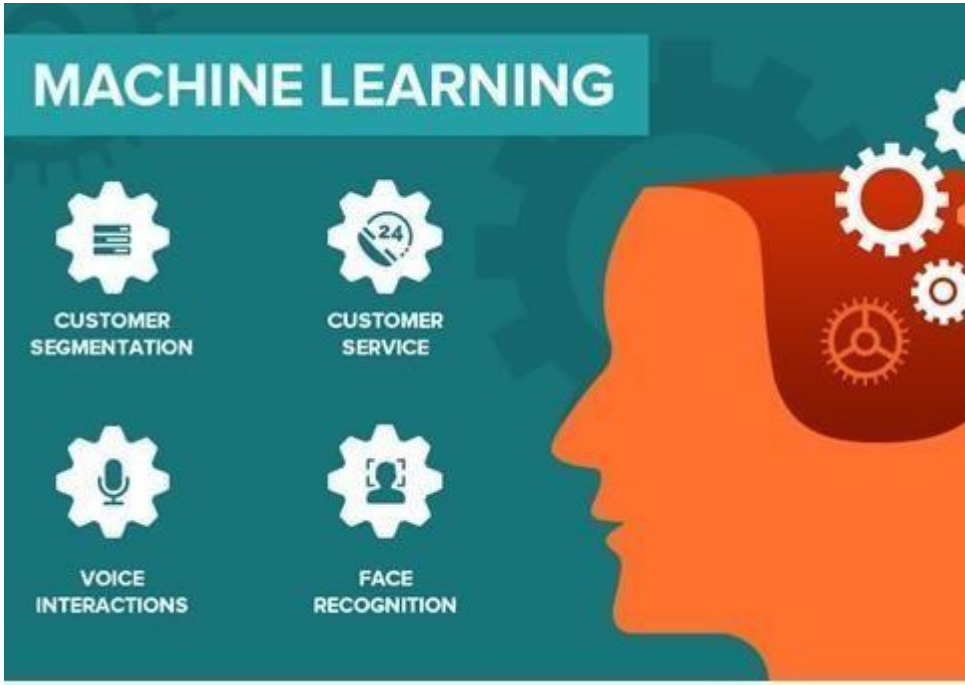


Picture credit:
https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/
https://towardsdatascience.com/techniques-for-handling-underfitting-and-overfitting-in-machine-learning-348daa2380b9
https://towardsdatascience.com/8-simple-techniques-to-prevent-overfitting-4d443da2ef7d


← Underfitting
Overfitting →
Best Fit
Test Error
Training Error
Error
Model "complexity"

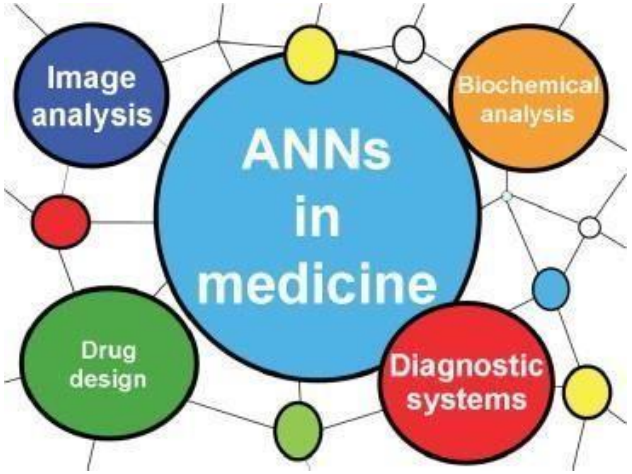
Applications of Artificial Neural Networks

Since artificial neural networks allow modeling of nonlinear processes, they have turned into a very popular and useful tool for solving many problems such as classification, clustering, regression, pattern recognition, dimension reduction, structured prediction, machine translation, anomaly detection, decision making, visualization, computer vision, and others.



More possible applications

- Pattern recognition
  - Network attacks
  - Breast cancer
  - ...
  - Handwriting recognition
- Pattern completion
- Auto-association
  - ANN trained to reproduce input as output
    - Noise reduction
    - Compression
    - Finding anomalies



Uses of Artificial Neural Networks

Deep Learning models can be used for a variety of complex tasks:

- Artificial Neural Networks(ANN) for Regression and classification
- Convolutional Neural Networks(CNN) for Computer Vision
- Recurrent Neural Networks(RNN) for Time Series analysis
- Self-organizing maps for Feature extraction
- Deep Boltzmann machines for Recommendation systems
- Auto Encoders for Recommendation systems

A typical example is Google Translate

Refer to the TensorFlow playground on the web that allows one to experiment with ANNs in an interactive setting