UNIT 1

Deep Learning Basics

History

- **1943:** ***Walter Pitts*** *and* ***Warren McCulloch*** *created a* ***computer model based on the neural networks*** *of the human brain.*
- **1960:** ***Henry J. Kelley*** *is given credit for developing the basics of a continuous* ***Back Propagation*** *Model. It was inefficient, and would not become useful until 1985.*
- **1985:** *David E Rumelhart;* ***Geoffrey E Hinton****; Ronald J Williams* ***demonstrated back propagation*** *in a neural network could provide "interesting" distribution representations*
- **1989:** ***Yann LeCun*** *provided the* ***first practical demonstration of backpropagation*** *at Bell Labs. He combined convolutional neural networks with back propagation onto read "handwritten" digits using* ***convolutional neural networks****.* ***Yoshua Bengio*** *publishes his paper on* ***"****Connectionist model applied* ***to speech recognition"***
- **1999:** *computers started becoming faster at processing data and GPU (graphics processing units) were developed increasing computational speeds by 1000 times*
- **2006:** ***Geoffrey E Hinton*** *introduced the idea of* ***unsupervised pretraining*** *and* ***deep belief nets*** *through deeper networks; hence rebranding of* ***neural networks*** *->* ***Deep Learning*** *for Handwritten digits*

1943

1985

1999

1960

?

1989

2006

The Great AI Awakening

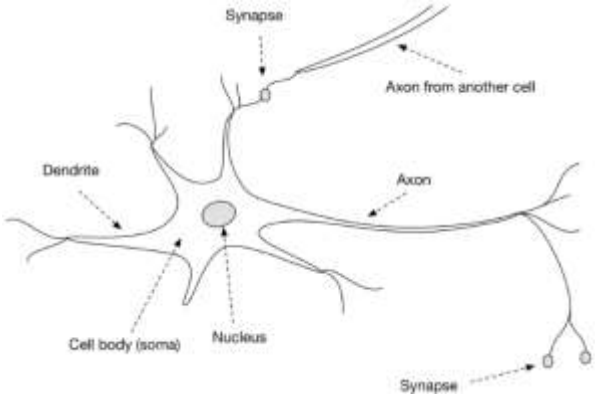
- **Appearance of large, high-quality labeled datasets** - Data along with GPUs probably explains most of the improvements we've seen. Deep learning is a furnace that needs a lot of fuel to keep burning, and we finally have enough fuel. (MNIST, etc.)
- **Massively parallel computing with GPUs** -
  - It turns out that neural nets are actually just a bunch of floating point calculations that you can do in parallel.
  - It also turns out that GPUs are **great** at doing these types of calculations.
  - The transition from CPU-based training to GPU-based has resulted in massive speed ups for these models, and as a result, allowed us to go bigger and deeper, and with more data.
- **Backprop-friendly activation functions** - The transition away from saturating activation functions like tanh and the logistic function to things like ReLU have alleviated the vanishing gradient problem

The Great AI Awakening

- **Improved architectures** - Resnets, inception modules, and Highway networks keep the gradients flowing smoothly, and let us increase the depth and flexibility of the network
- **Software platforms** - Frameworks like tensorflow, keras, pytorch, Theano,etc. that provide automatic differentiation allow for seamless GPU computing and make prototyping faster and less error-prone. They let you focus on your model structure without having to worry about low-level details like gradients and GPU management.
- **New regularization techniques** - Techniques like dropout, batch normalization, and data-augmentation allow us to train larger and larger networks without (or with less) overfitting

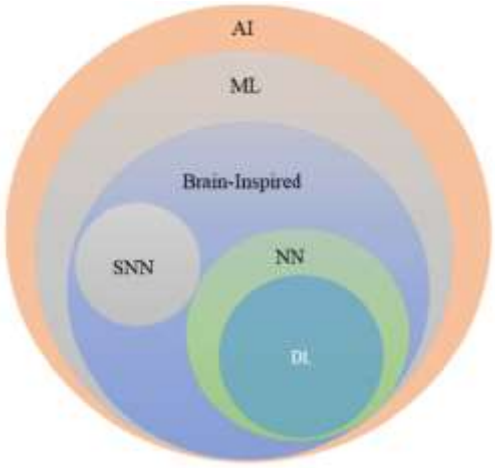
Definition

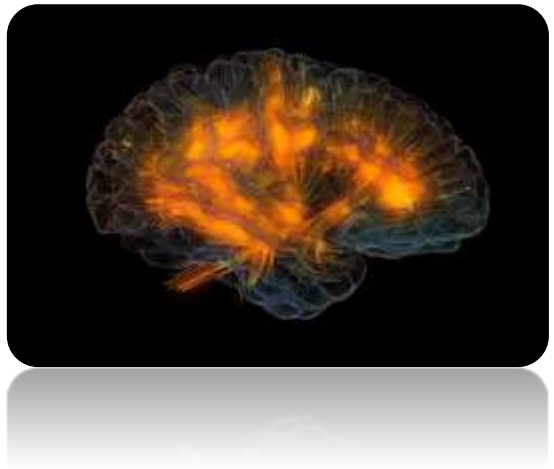
Is a branch of ***Machine Learning*** based on a set of algorithms that attempt to model ***high level abstractions*** in data by using ***deep graphs*** with multiple ***hierarchal processing layers***, composed of multiple ***linear*** and ***non-linear*** transformations.



Important features

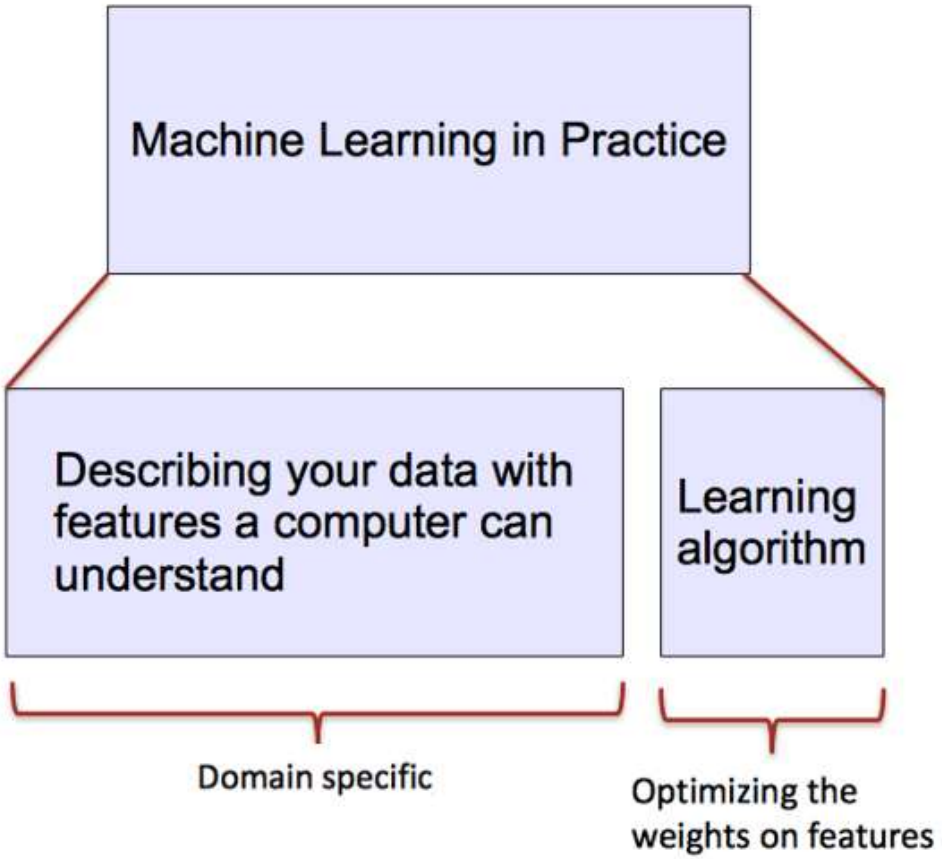
- More neurons than traditional neural networks
- Rather more complex architectures of connecting layers/neurons in the underlying neural networks
- Requires explosive amount of computing power to be available to train and automatic feature extraction





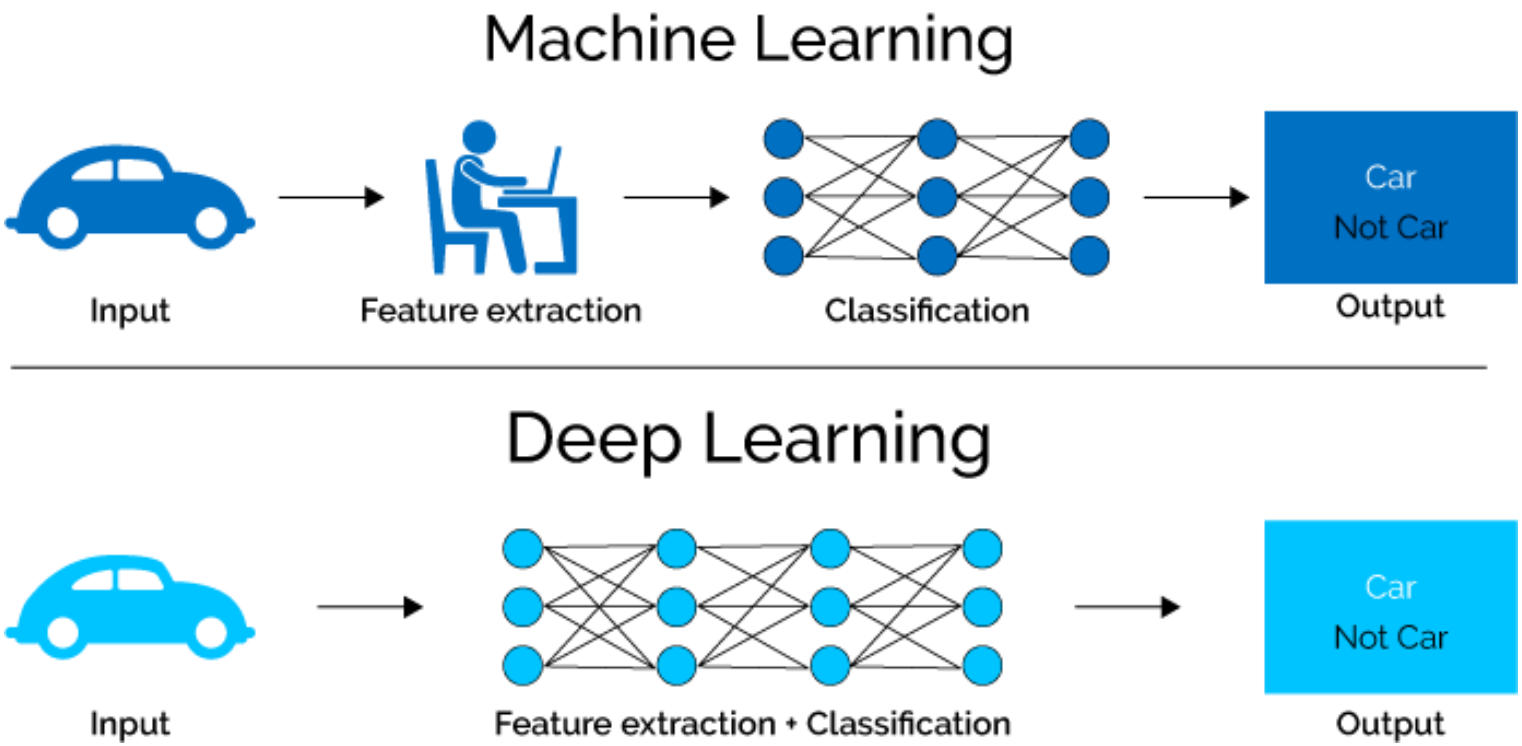
ML vs. Deep Learning

- Conventional machine learning methods rely on human-designed feature representations
  - ML is just optimizing weights to best make a final prediction



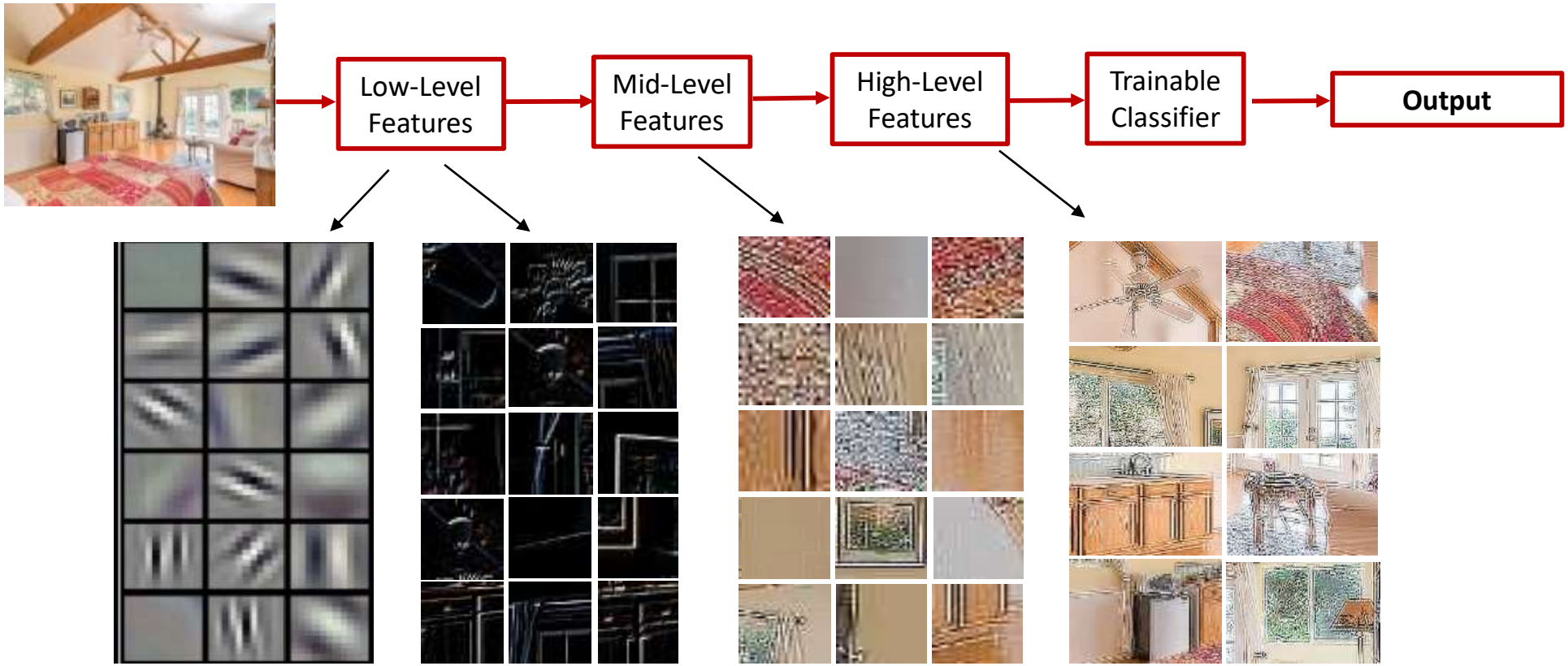
ML vs. Deep Learning

- *Deep learning* (DL) is a machine learning subfield that uses multiple layers for learning data representations
  - DL is exceptionally effective at learning patterns



ML vs. Deep Learning

- DL applies a multi-layer process for learning rich hierarchical features (i.e., data representations)
  - Input image pixels → Edges → Textures → Parts → Objects

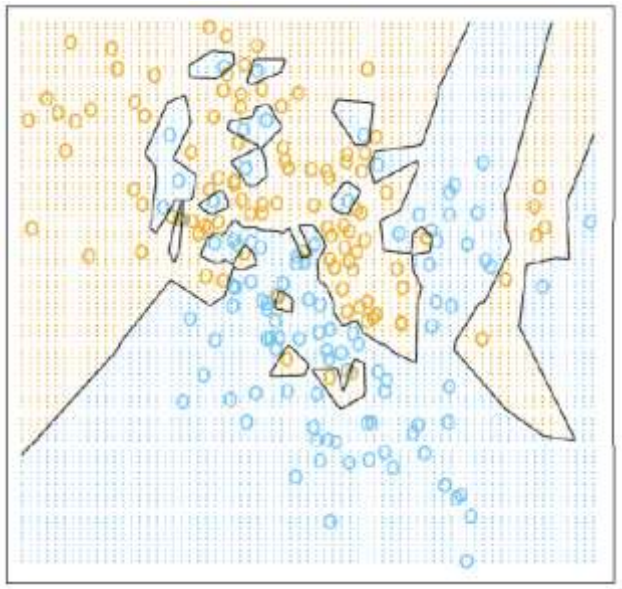


Why is DL Useful?

- DL provides a flexible, learnable framework for representing visual, text, linguistic information
  - Can learn in supervised and unsupervised manner
- DL represents an effective end-to-end learning system
- Requires large amounts of training data
- Since about 2010, DL has outperformed other ML techniques
  - First in vision and speech, then NLP, and other applications

Representational Power

- NNs with at least one hidden layer are universal approximators
  - Given any continuous function $h(x)$ and some $\epsilon > 0$, there exists a NN with one hidden layer (and with a reasonable choice of non-linearity) described with the function $f(x)$, such that $\forall x, |h(x) - f(x)| < \epsilon$
  - I.e., NN can approximate any arbitrary complex continuous function
- NNs use nonlinear mapping of the inputs x to the outputs $f(x)$ to compute complex decision boundaries
- But then, why use deeper NNs?
  - The fact that deep NNs work better is an practical observation
  - Mathematically, deep NNs have the same representational power as a one-layer NN



UNDERSTANDING THE BIOLOGICAL NEURON

- The human nervous system has two main parts –
  - the central nervous system (CNS) consisting of the brain and spinal cord
  - the peripheral nervous system consisting of nerves and ganglia outside the brain and spinal cord.
- The CNS integrates all information, in the form of signals, from the different parts of the body.
- The peripheral nervous system, on the other hand, connects the CNS with the limbs and organs.
- Neurons are basic structural units of the CNS.
- A neuron is able to receive, process, and transmit information in the form of chemical and electrical signals

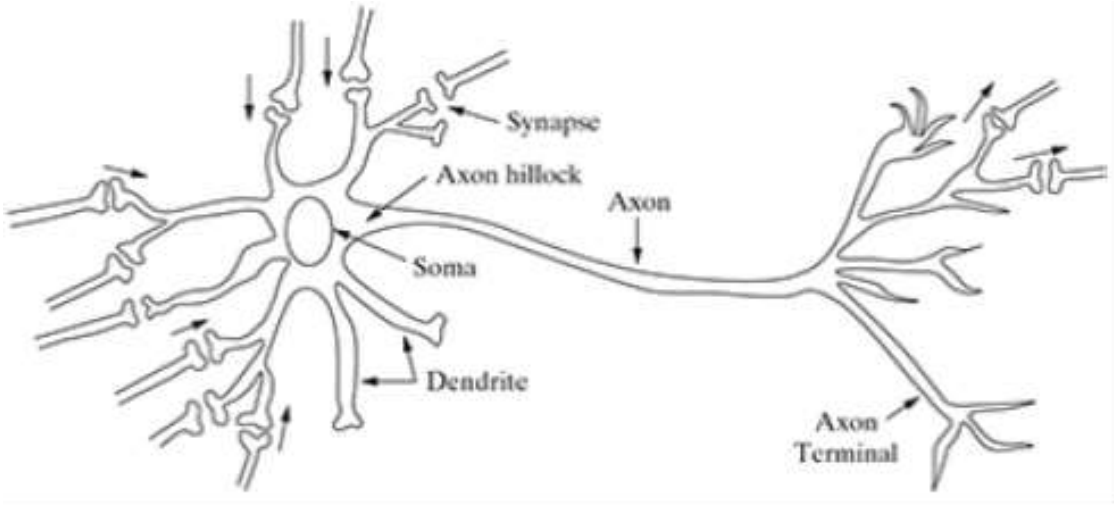
Neuron

- The structure of a neuron three main parts to carry out its primary functionality of receiving and transmitting information:

  1. Dendrites – to receive signals from neighbouring neurons.

  2. Soma – main body of the neuron which accumulates the signals coming from the different dendrites. It 'fires' when a sufficient amount of signal is accumulated.

  3. Axon – last part of the neuron which receives signal from soma, once the neuron 'fires', and passes it on to the neighbouring neurons through the axon terminals (to the adjacent dendrite of the neighbouring neurons).



- There is a very small gap between the axon terminal of one neuron and the adjacent dendrite of the neighbouring neuron. This small gap is known as **synapse**.
- The signals transmitted through synapse may be excitatory or inhibitory.

Human Brain

- The adult human brain, which forms the main part of the central nervous system, is approximately 1.3 kg in weight and 1200 cm^3 in volume.
- It is estimated to contain about 100 billion (i.e. 10^{11}) neurons and 10 times more glial or glue cells.
- Glial cells act as support cells for the neurons.
- It is believed that neurons represent about 10% of all cells in the brain. On an average, each neuron is connected to 10^5 of other neurons, which means that altogether there are 10^{16} connections.
- The axon, a human neuron, is 10–12 μm in diameter.
- Each synapse spans a gap of about a millionth of an inch wide.

THE ARTIFICIAL NEURON

- Each neuron has three major components:

1. A set of '*i*' **synapses** having weight w_i . A signal x_i forms the input to the *i*-th synapse having weight w_i . The value of weight w_i may be positive or negative.

A positive weight has an excitatory effect, while a negative weight has an inhibitory effect on the output of the summation junction, y_{sum} .

2. A **summation junction** for the input signals is weighted by the respective synaptic weight. Because it is a linear combiner or adder of the weighted input signals, the output of the summation junction, y_{sum} , can be expressed as :

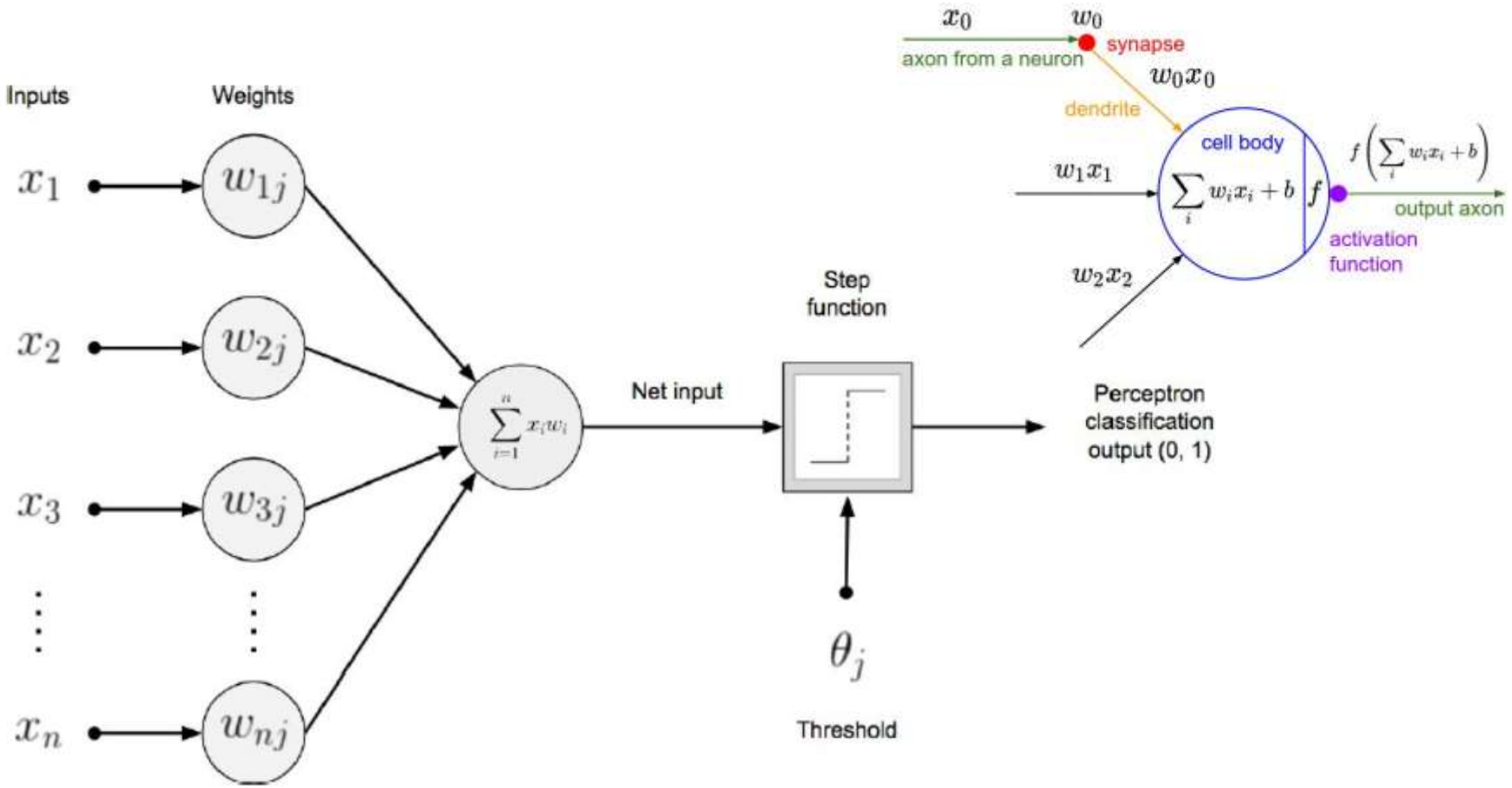
$$y_{\text{sum}} = b + \sum_{i=1}^{n} w_i x_i$$

3. A threshold activation function (or simply activation function, also called squashing function) results in an output signal only when an input signal exceeding a specific threshold value comes as an input. It is similar in behaviour to the biological neuron which transmits the signal only when the total input signal meets the firing threshold.

Perceptron

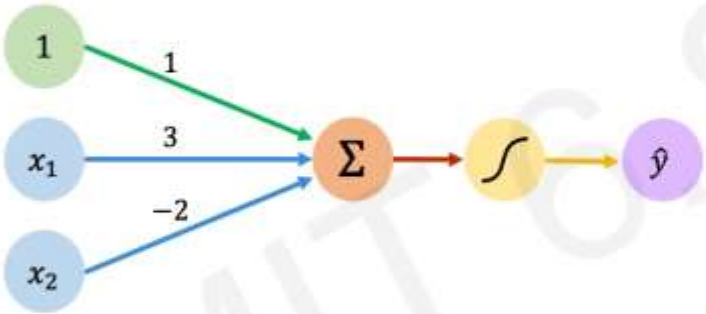
- A perceptron is the fundamental building block of a neural network.
- It's a simplified model of a biological neuron, capable of making binary decisions.
- In its simplest form, a perceptron takes in multiple inputs, each with an associated weight, calculates the weighted sum of inputs, applies an activation function to the sum, and produces an output (usually 0 or 1).
- Perceptrons were introduced by Frank Rosenblatt in the late 1950s.

The Perceptron



- A perceptron is a simple binary classification algorithm, proposed by Cornell scientist Frank Rosenblatt.
- But unlike many other classification algorithms, the perceptron was modeled after the essential unit of the human brain—the neuron and has an uncanny ability to learn and solve complex problems.
- A perceptron is a very simple learning machine. It can take in a few inputs, each of which has a weight to signify how important it is, and generate an output decision of “0” or “1” —“yes” and “no”.
- However, when combined with many other perceptrons, it forms an artificial neural network.
- A neural network can, theoretically, answer any question, given enough training data and computing power.

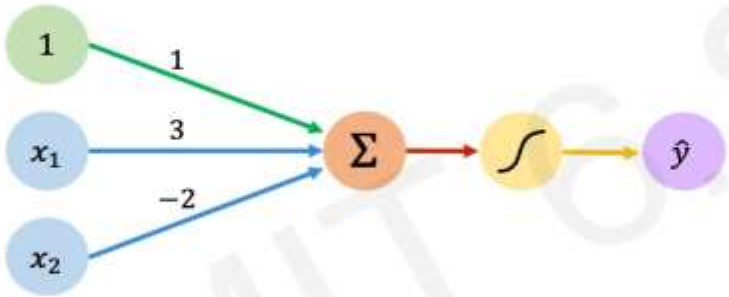
The Perceptron: Example



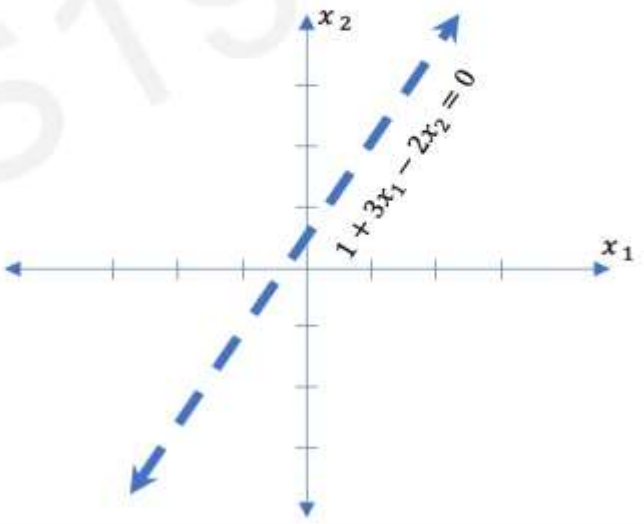
We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$
$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}})$$

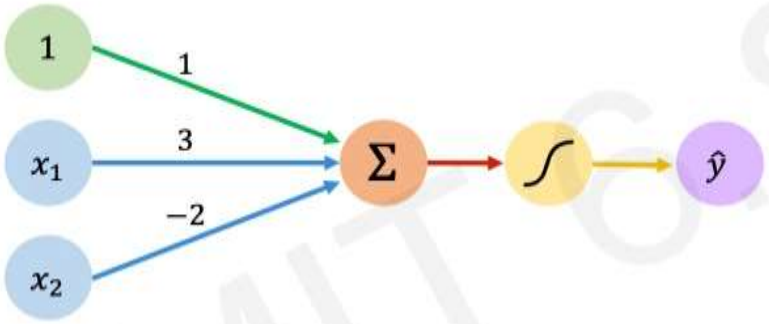
The Perceptron: Example



$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

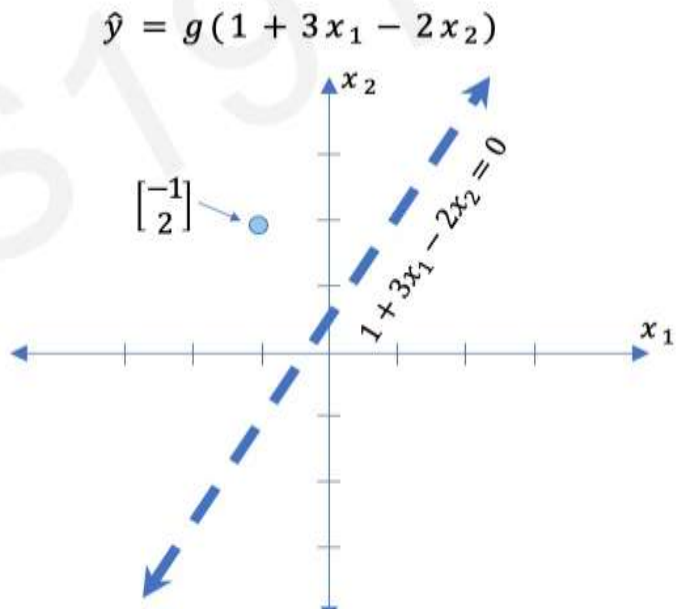


The Perceptron: Example

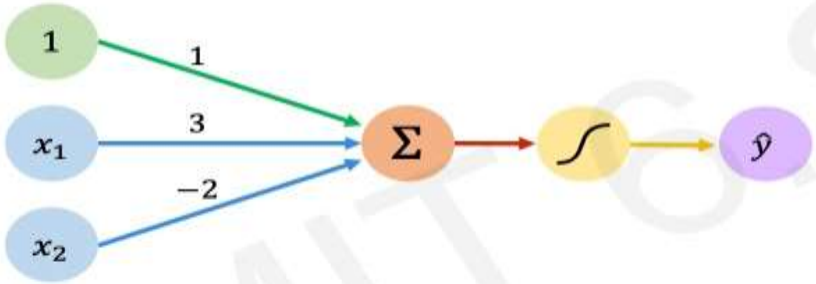


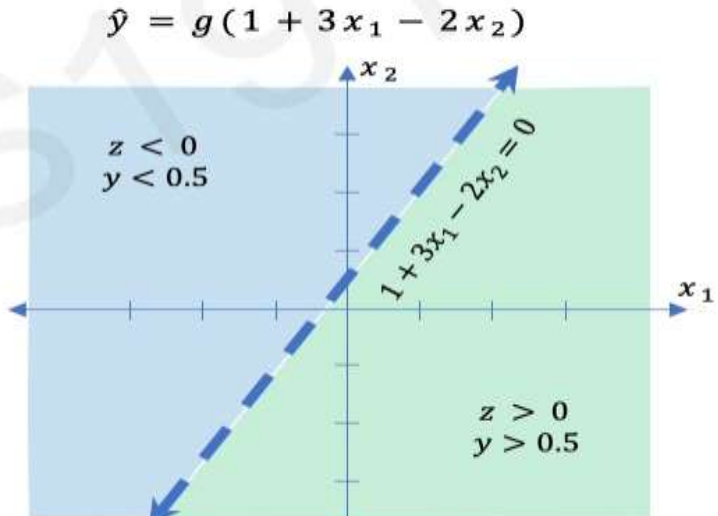
Assume we have input: $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

$$\hat{y} = g(1 + (3 * -1) - (2 * 2))$$
$$= g(-6) \approx 0.002$$



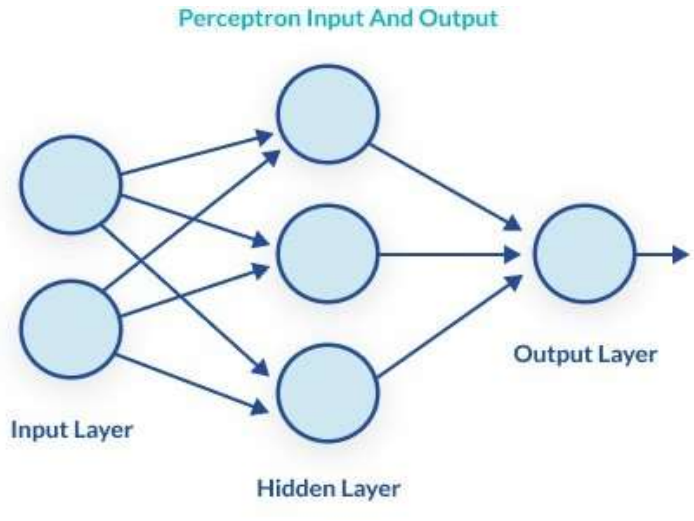
The Perceptron: Example





Multilayer Perceptron

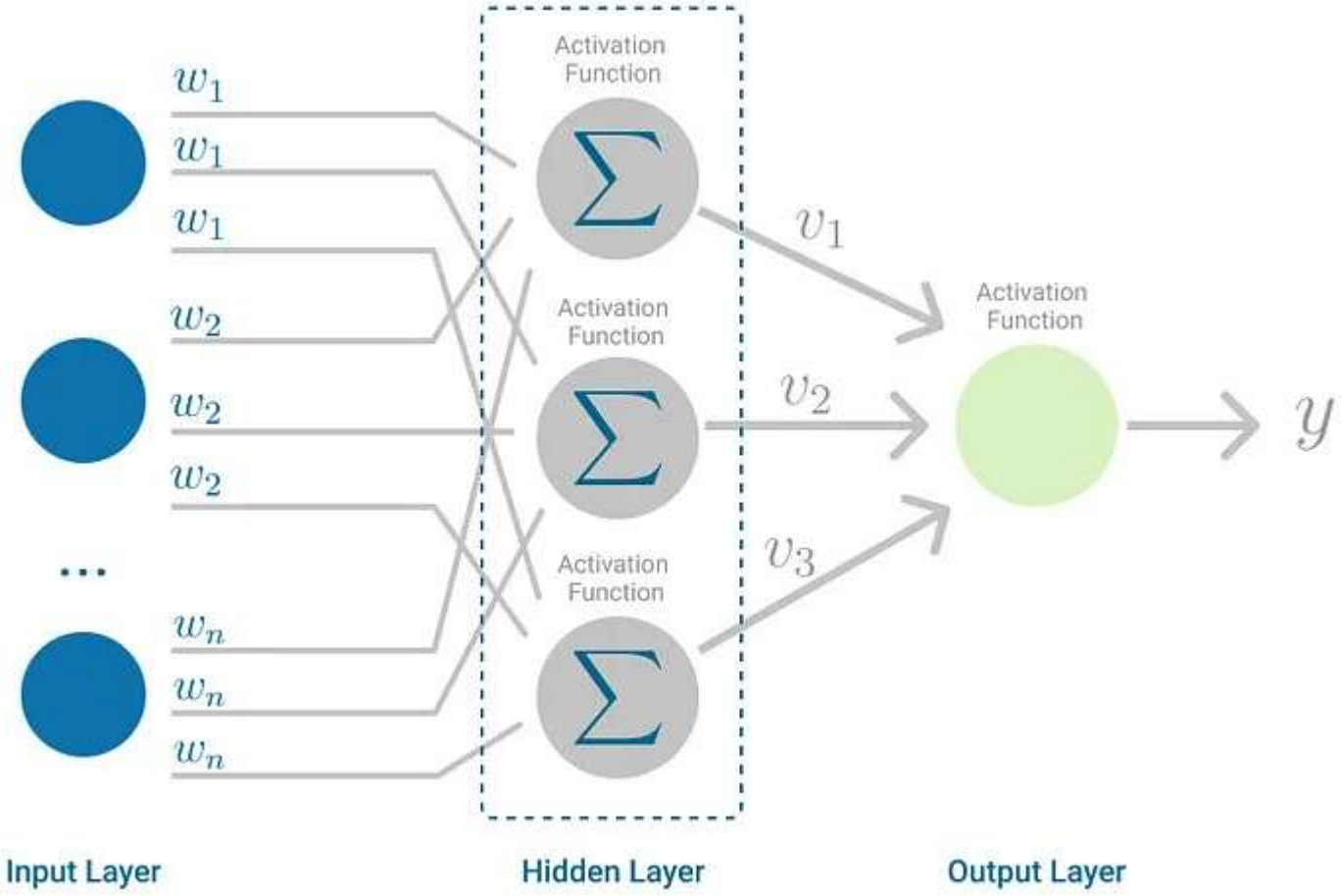
- A multilayer perceptron (MLP) is a perceptron that teams up with additional perceptrons, stacked in several layers, to solve complex problems.
- Each perceptron in the first layer on the left (the input layer), sends outputs to all the perceptrons in the second layer (the hidden layer), and all perceptrons in the second layer send outputs to the final layer on the right (the output layer).
- The mapping between the input and the output is non-linear
- Each layer can have a large number of perceptrons, and there can be multiple layers, so the multilayer perceptron can quickly become a very complex system.



Multilayer Perceptron

- The multilayer perceptron has another, more common name—a neural network.
- A three-layer MLP is called a Non-Deep or Shallow Neural Network.
- An MLP with four or more layers is called a Deep Neural Network.
- One difference between an MLP and a neural network is that in the classic perceptron, the decision function is a step function and the output is binary.
- In neural networks that evolved from MLPs, other activation functions can be used which result in outputs of real values, usually between 0 and 1 or between -1 and 1.
- This allows for probability-based predictions or classification of items into multiple labels.

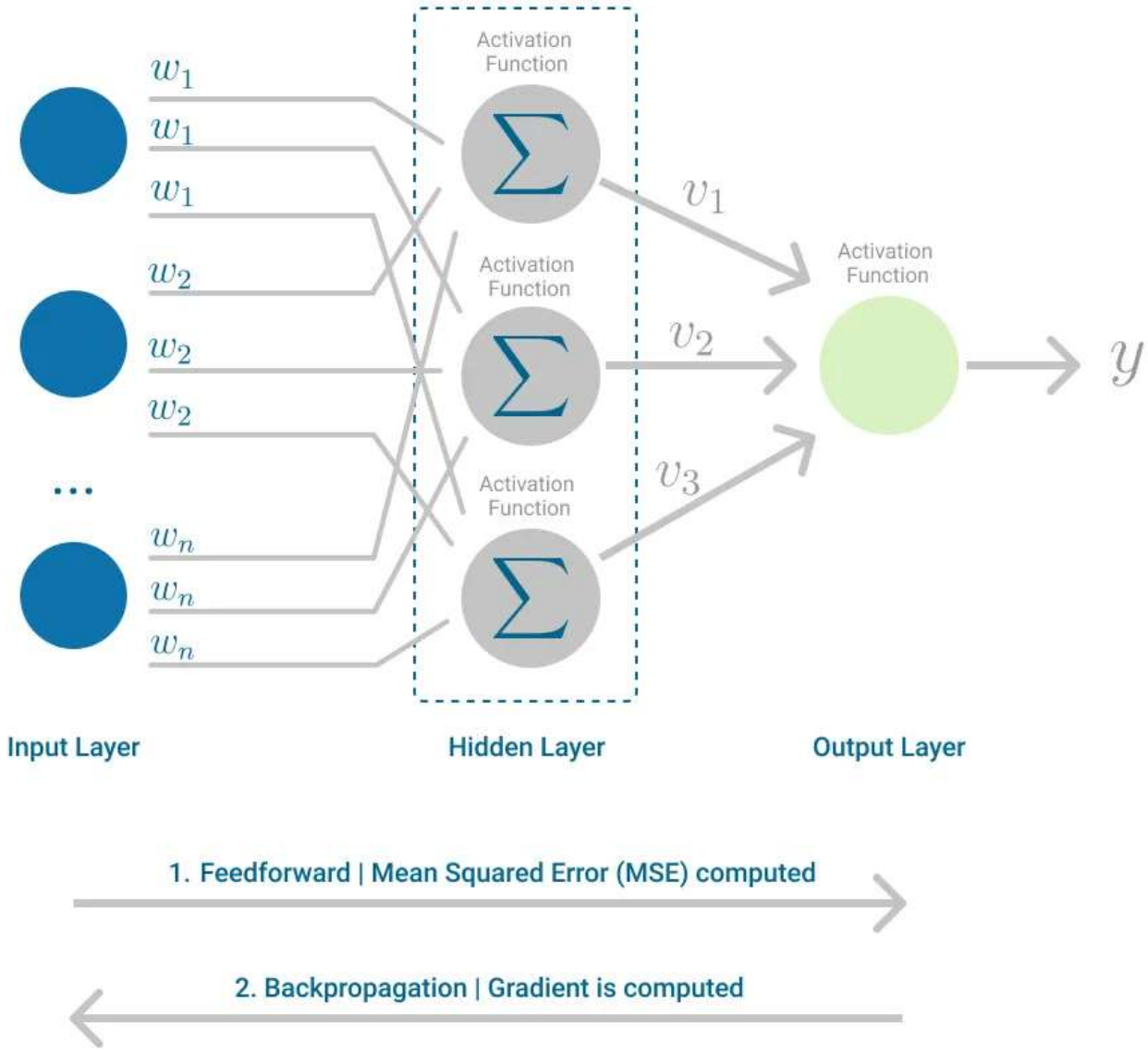
Multilayer Perceptron



Multilayer Perceptron

- Multilayer Perceptron falls under the category of feedforward algorithms, as inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like in the Perceptron.
- The difference is that each linear combination is propagated to the next layer.
- Each layer is *feeding* the next one with the result of their computation, their internal representation of the data.
- This goes all the way through the hidden layers to the output layer.
- If the algorithm only computed one iteration, there would be no actual learning.
- For carrying out the learning, back propogartion algorithm is used.

Multilayer Perceptron with feedback



Difference between perceptron and neuron

- The main difference between a perceptron and a neuron is that
  - a perceptron refers to a basic binary classification unit with a linear activation function,
  - whereas a neuron is a more general computational unit used in modern neural networks that can have various activation functions and play different roles in complex tasks.

Need of activation function

- They are used in the hidden and output layers.
- It adds nonlinearity to the model.
- Activation function is a function that is added into an artificial neural network in order **to help the network learn complex patterns in the data**.
- The activation function will decide what is to be fired to the next neuron

Can ANN work without an activation function?

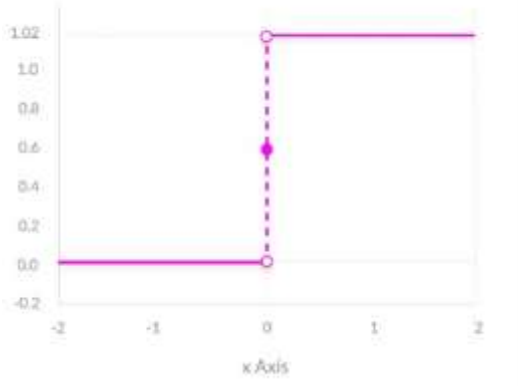
- Then it becomes linear.
- Basically, the cell body has two parts, one is linear and another one is non-linear. Summation part will do linear activity, and activation function will perform non-linear activity.
- If activation function is not used then, every neuron will only be performing a linear transformation on the inputs using the weights and biases.
- Although linear transformations make the neural network simpler, but this network would be less powerful and will not be able to learn the complex patterns from the data. Hence the need for activation function.

Popular Activation Functions

1. Popular types of activation functions are:
    1. Step function
    2. Sign function
    3. Linear function
    4. ReLU (Rectified Linear Unit): no –ve value
    5. Tanh
    6. Sigmoid
    7. softmax

1. Step function

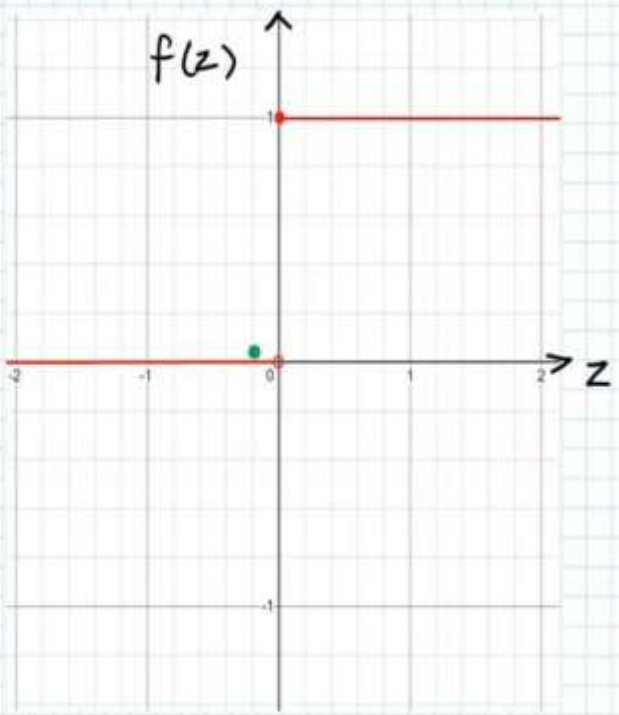
- A binary step function is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.
- The problem with a step function is that it does not allow multi-value outputs—for example, it cannot support classifying the inputs into one of several categories.



Step Function

1. Step function

$$f(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

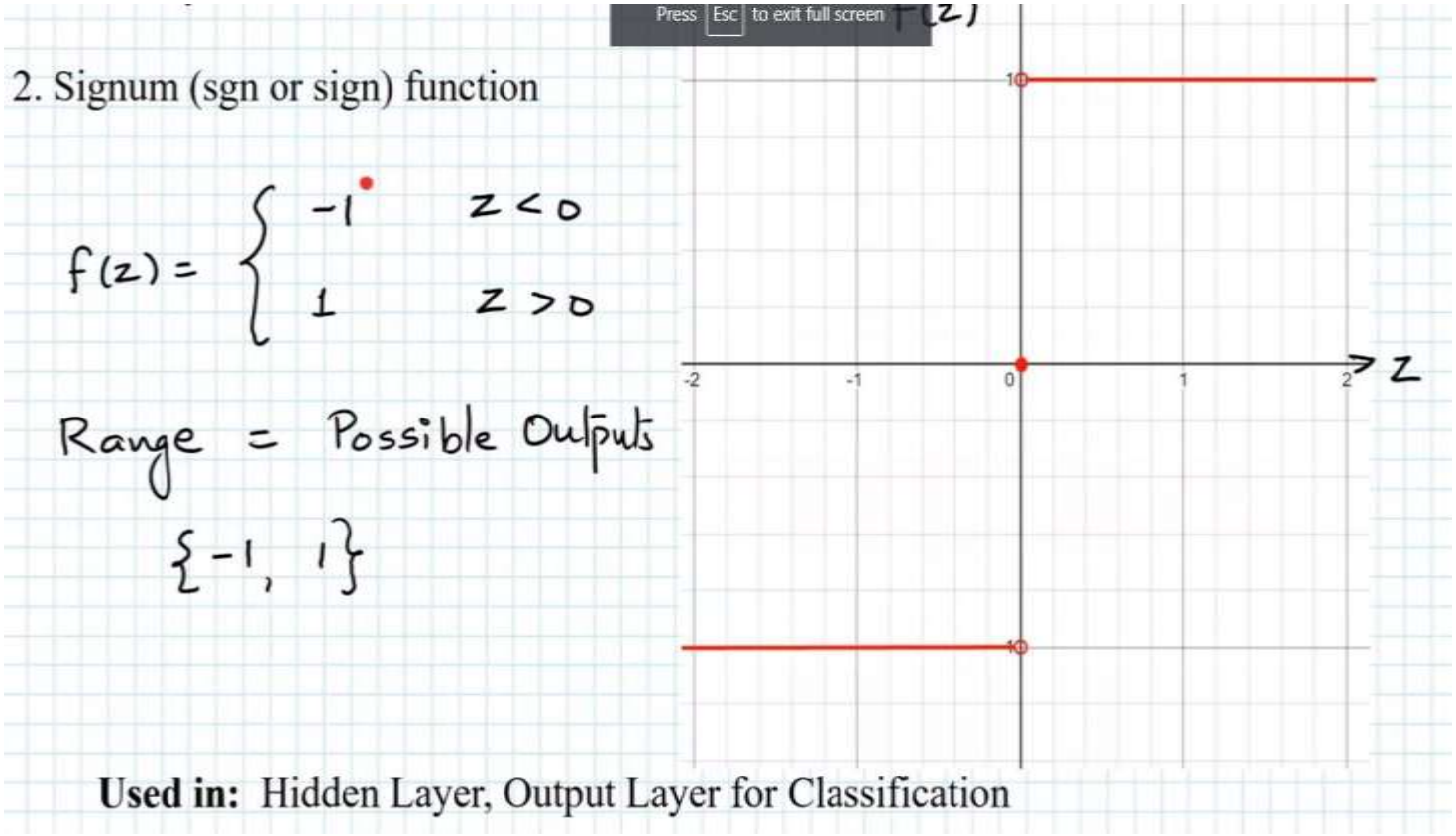


Range = Possible Output

$\{0, 1\}$

Used in: Hidden Layer, Output Layer for Classification

2. Sign function



3. Linear Function

- It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input.
- In one sense, a linear function is better than a step function because it allows multiple outputs, not just yes and no.
- All layers of the neural network collapse into one —
  - With linear activation functions, no matter how many layers in the neural network, the last layer will be a linear function of the first layer (because a linear combination of linear functions is still a linear function). So a linear activation function turns the neural network into just one layer.

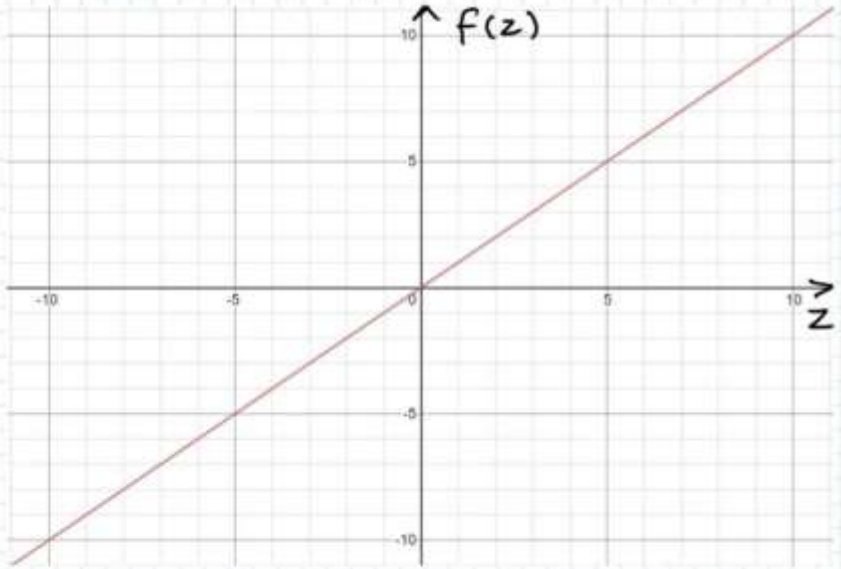
Linear function

3. Linear Function

$f(z) = z$

Range = Possible Outputs

$(-\infty, \infty)$

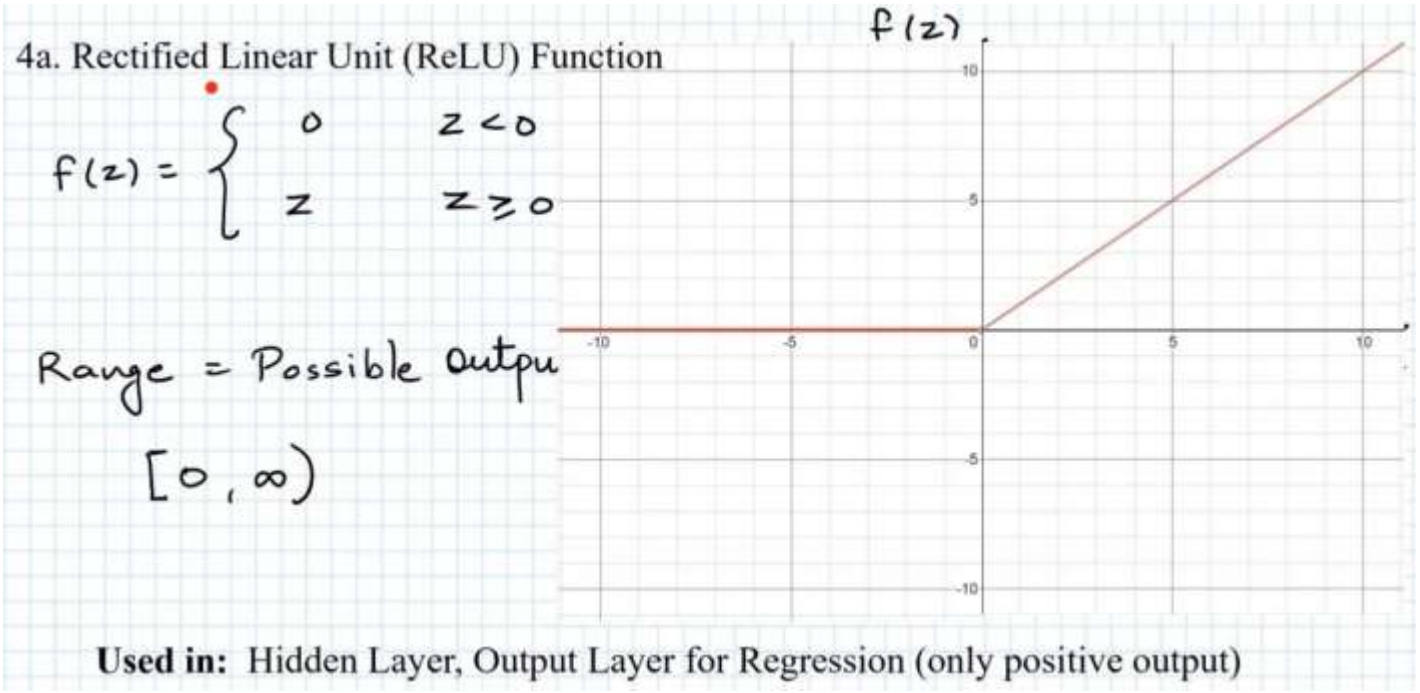


Used in: Hidden Layer, Output Layer for Regression

4. ReLU (Rectified Linear Units)

- Advantages
  - Computationally efficient—allows the network to converge very quickly
  - Non-linear—although it looks like a linear function, ReLU has a derivative function and allows for backpropagation
- Disadvantages
  - The Dying ReLU problem—when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

ReLU function



5. Sigmoid Function

- Advantages
  - Smooth gradient, preventing “jumps” in output values.
  - Output values bound between 0 and 1, normalizing the output of each neuron.
  - Clear predictions—For X above 2 or below -2, tends to bring the Y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.
- Disadvantages
  - Vanishing gradient—for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.
  - Outputs not zero centered.
  - Computationally expensive

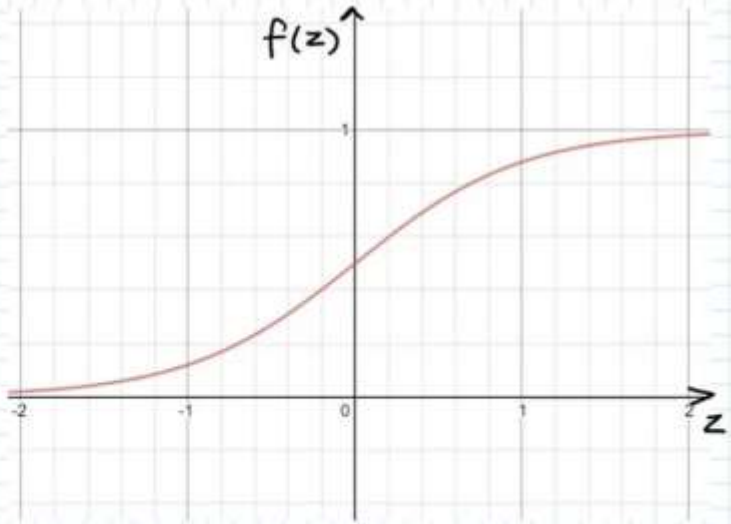
Sigmoid Function

6a. Sigmoid Function

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

Range = Possible Outputs

(0, 1)



Used in: Hidden Layer, Output Layer for Classification

6. Tanh- Hyperbolic Tangent

- Advantages
  - Zero centered—making it easier to model inputs that have strongly negative, neutral, and strongly positive values.
  - Otherwise like the Sigmoid function.
  - Disadvantages
  - Like the Sigmoid function

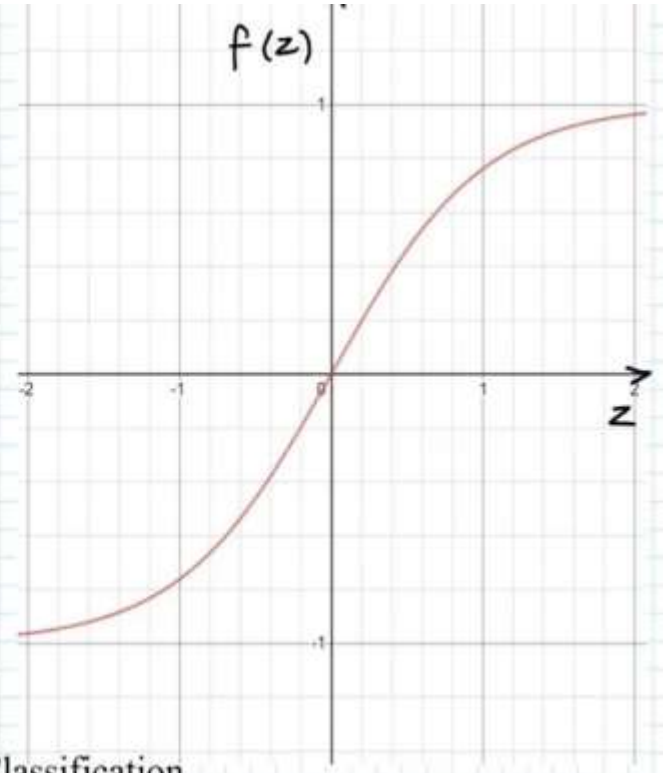
Tanh: Hyperbolic Tangent

5. Hyperbolic tangent; tanh(z)

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

Range = Possible Outputs

(-1, 1)



Used in: Hidden Layer, Output Layer for Classification

7. SoftMax function

- It is the variant of sigmoid function with multi class classification
- Advantages
  - Able to handle multiple classes only one class in other activation functions—normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class.
  - Useful for output neurons—typically Softmax is used only for the output layer, for neural networks that need to classify inputs into multiple categories.

SoftMax function

6b. Softmax Function

$$z_1, z_2, \ldots, z_n$$

$$f(z_1) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_n}}$$

$$f(z_i) = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \cdots + e^{z_n}} = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}}$$

Used in: Output Layer for Multiclass Classification

Number of Hidden Layers

- For many problems, you can just begin with a single hidden layer and you will get reasonable results.
- It has actually been shown that an MLP with just one hidden layer can model even the most complex functions provided it has enough neurons.
- For a long time, these facts convinced researchers that there was no need to investigate any deeper neural networks.
- But they overlooked the fact that deep networks have a much higher parameter efficiency than shallow ones.
- They can model complex functions using exponentially fewer neurons than shallow nets, making them much faster to train
- Very complex tasks, such as large image classification or speech recognition, typically require networks with dozens of layers (or even hundreds) and they need a huge amount of training data.
- However, you will rarely have to train such networks from scratch: it is much more common to reuse parts of a pretrained state-of-the-art network that performs a similar task. Training will be a lot faster and require much less data

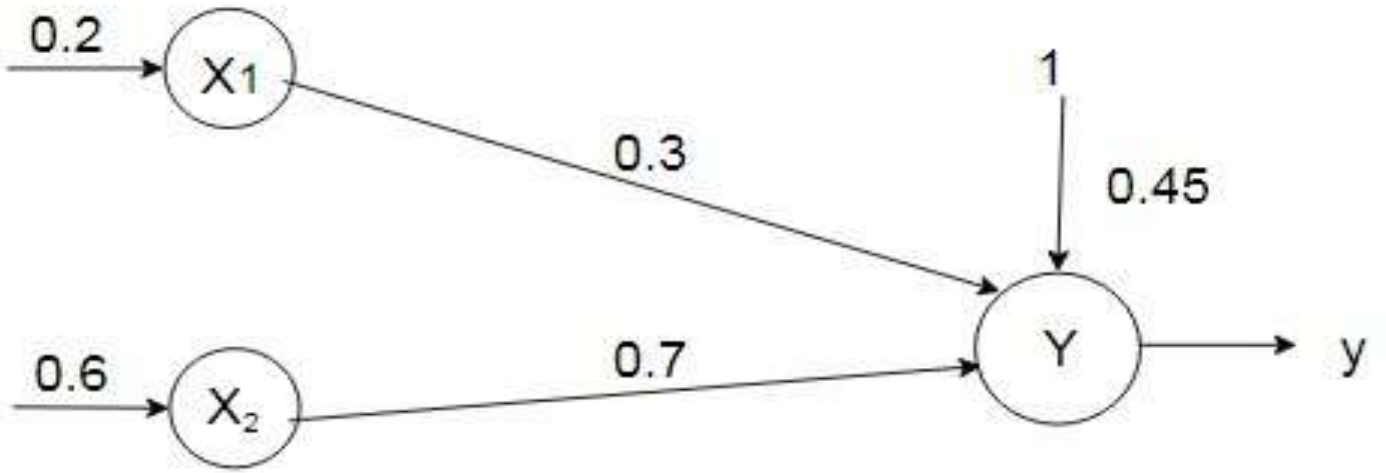
Number of Neurons per Hidden Layers

- Usually the number of neurons in the input and output layers is determined by the type of input and output your task required.
- For the hidden layer the common practice is to size them to form a funnel, with fewer and fewer neurons at each layer.
- For example a typical neural network may have two hidden layers, the first with 300 neurons and the second with 100.
- However, this practice is not as common now, and you may simply use the same size for all hidden layers; for example all hidden layers with 150 neurons.
- Neurons can be gradually increased until the network starts overfitting.

Activation Functions

- In most of the cases you can use the ReLU activation function in the hidden layers. It is a bit faster to compute than other activation functions.
- For the output layer, the softmax activation function is generally a good choice for classification tasks.

Q2. Calculate the net input for the network shown in figure with bias included in the network?



Solution: $[x_1, x_2, b] = [0.2, 0.6, 0.45]$

$[w_1, w_2] = [0.3, 0.7]$

The net input can be calculated as,

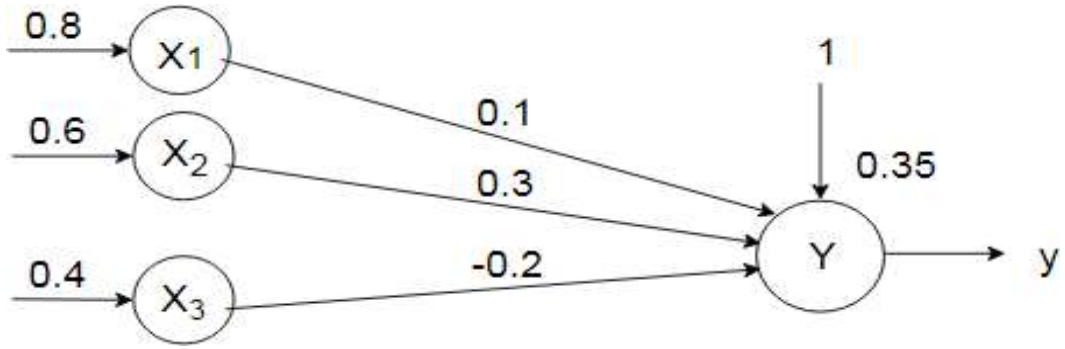
$y_{in} = b + x_1 w_1 + x_2 w_2$

$y_{in} = 0.45 + 0.2 \times 0.3 + 0.6 \times 0.7$

$y_{in} = 0.45 + 0.06 + 0.42 = 0.93$

Problem: (to compute Y out): apply activation function

- Calculate the output ‘y’ of a 3 input neuron with bias, with data as given in the diagram. Use sigmoidal activation function



$$[x_1, x_2, x_3] = [0.8, 0.6, 0.4]$$

The weights are,

$$[w_1, w_2, w_3] = [0.1, 0.3, -0.2]$$

The net input can be calculated as,

$$y_{in} = b + \sum_{i=1}^{n} (x_i w_i)$$

$$y_{in} = 0.35 + 0.8 \times 0.1 + 0.6 \times 0.3 + 0.4 \times (-0.2)$$

$$y_{in} = 0.35 + 0.08 + 0.18 - 0.08 = 0.53$$

(i) For Binary Sigmoidal Function,

$$y = f(y_{in}) = \frac{1}{1 + e^{-y_{in}}} = \frac{1}{1 + e^{-0.53}} = 0.62$$

Thankyou