## Table of Contents

# Deep Learning course Transcript

## What is deep learning?

Let's begin the course by reviewing the concept of Deep Learning. What is Deep Learning? Deep Learning is a field within machine learning that deals with building and using neural network models. Neural networks with more than three layers are typically categorized as Deep Learning networks. Neural networks mimic the functioning of a human brain. They're organized similar to the brain cells and imitate how humans process data and make decisions. Deep Learning is a field that has seen exponential growth in the last few years. While the algorithms for neural networks have existed for some time the advances in large scale data processing, as well as inference technologies like GPU's have spurred their popularity for real world applications. Deep Learning has been extremely popular in natural language processing as the neural network architectures are ideal for dealing with unstructured data. For the same reason, they are also popular for speech recognition and synthesis applications. Image recognition is another domain where Deep Learning has made inroads. Self-driving cars is a bleeding edge technology that is being powered by Deep Learning. Applications that require complex learning of behaviors are usually suited for Deep Learning. The applications of Deep Learning are getting wide popularity in very domains like customer experience, healthcare and robotics. We will now start with the concepts of Deep Learning.

## Concepts of Deep learning

### Linear regression

One of the basic statistical concepts that is used in machine learning is linear regression. It forms a key foundation for deep learning. Linear regression is a linear model that explains the relationship between two or more variables. We have a dependent variable y and an independent variable x. The model provides an equation to compute the value of y based on the value of x. To compute this, we need two constants called a, which is the slope, and an intercept, which is b. The formula for computing y is ax plus b. This provides a linear relationship between y

and x. In reality, the relationship may not be perfectly linear, so there will be errors in predictions. Linear regression is used in regression problems to predict continuous variables. It can be applied for multiple independent variables like x1, x2, up to xn. In which case, there will be an equal n slope of values a1, a2, up to an. Let's look at an example for building a linear regression model. When we say we build a linear regression model, we are determining the value of the slope and intercept that models the relationship between the dependent and independent variables. To determine the slope and intercept, we will start with known values of x and y. You may have learned this in your math classes in school or college. For one independent variable, we start with two equations and then make substitutions to determine the value of a and b. Then the values of a and b can be applied to situations where y is not known. When there are multiple independent variables, this situation becomes complex. A related technique that is most used in deep learning is logistic regression. Logistic regression is a binary model that defines the relationship between two variables. The output y in this case is either zero or one. The formula is similar to linear regression, except that we use an additional activation function called f to convert the continuous variable coming out of ax plus b into a boolean value of zero or one. There are multiple options and variations of the function f. This equation, again, can be extended to multiple independent variables x1 to xn. We will look at an analogy for deep learning using the linear regression equation next.

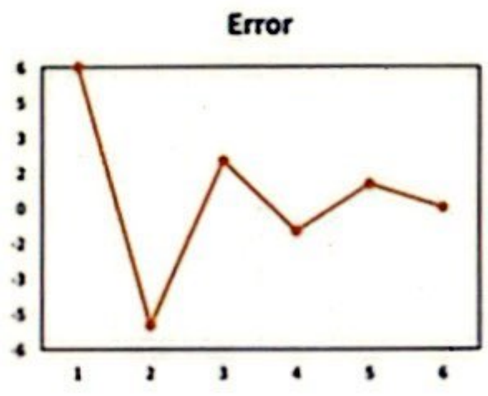
## An analogy for deep learning

- Let's use an analogy to understand how deep learning works. We will use this analogy later in the course. Deep learning is a complex and iterative process that requires a series of trials to narrow down the parameters for the model. It starts actually with random initialization of the model parameters and works towards the right values of these parameters by trial and error. In the case of a linear regression model, the model parameters are A and B. The values of A and B form the model that determines the relationship between X and Y. For analogy, let's try to find the values of A and B in the formula, 10 = 3A + B. We will follow a similar trial and error process to narrow down these

values. Let's start the trials. We will initialize the values of A and B to a random value. In this case, we initialize both to 1. We then compute the value of Y using the formula, 3A + B. The result is 4. The expected result is 10, though. We end up with an error of 6. Now, we will use this error value to adjust the values of A and B and see if we are getting closer to the expected result. Let's bump A to 4 and B to 3. Now, we have the value of Y to be 15, and an error of -5. We went from a positive error to a negative error, so we will adjust to lower values of A and B. We reduce A to 2 and B to 2. Now, the error has lowered down to 2, and we are getting closer to the results. Let's just change A to 3 and see what happens. The error is -1 now. We are getting closer, but went to the negative side. We now switch the values of A to 2 and B to 3. But the error goes to the other side, to positive 1. We then adjust only B to 4, and we now get an error of 0. So we determined that A is 2 and B is 4 by trial and error. As we do these iterations, we are progressively reducing the error towards 0 as shown in the graph. While the mathematical approach of using sample values and reducing the equation works well with a small number of independent variables, it becomes incredibly complex when the number of variables is really high. Also, the equation may not always provide a zero error,

**10 = 3a + b**

| Trial | a | b | 3a + b | Error |
|---|---|---|---|---|
| 1 | 1 | 1 | 4 | 6 |
| 2 | 4 | 3 | 15 | -5 |
| 3 | 2 | 2 | 8 | 2 |
| 4 | 3 | 2 | 11 | -1 |
| 5 | 2 | 3 | 9 | 1 |
| 6 | 2 | 4 | 10 | 0 |




## The perceptron

- The perceptron is the unit for learning in an artificial neural network. A perceptron represents the algorithm for supervised learning in an artificial neural network. It resembles a human brain cell. Multiple inputs are fed into the perceptron, which in turn does computations and outputs a boolean variable. It represents a single cell or node in a

neural network. It is built based on logistic regression. To derive the formula for the perceptron, we use the logistic regression formula discussed in the earlier video. Here, we replace the slope, a, with a weight called w, and intercept b with the bias called b. Weights and biases become the parameters for a neural network. We then apply an activation function f that outputs a boolean result based on the values. This formula for the perceptron is fundamental to deep learning. The same perceptron is shown here in a figure. We have multiple independent input variables, x1 to xn, that are fed to the perceptron. Each of them is multiplied by a corresponding weight. The number of weights equal the number of inputs. We also feed in a 1 that is multiplied by the bias. All the results are then summed up. An activation function is applied, that delivers the value of y, which is either a 1 or a 0. We build neural networks by connecting a number of perceptrons. Let's discuss more of that in the next video.

## Perceptron Formula

$$y = f(a_1x_1 + a_2x_2 \ldots + a_nx_n + b)$$

$$y = f(w_1x_1 + w_2x_2 \ldots + w_nx_n + b)$$

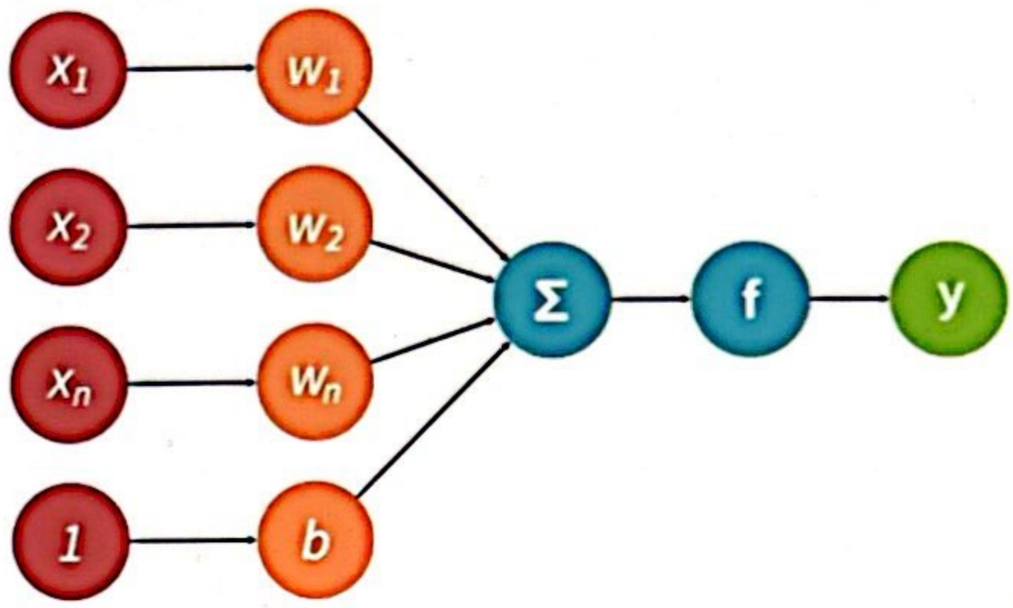
| | |
|---|---|
| w | Weight |
| b | Bias |
| f | Activation Function |

***Activation Function Example***

*1 if value > 0*

*0 otherwise*

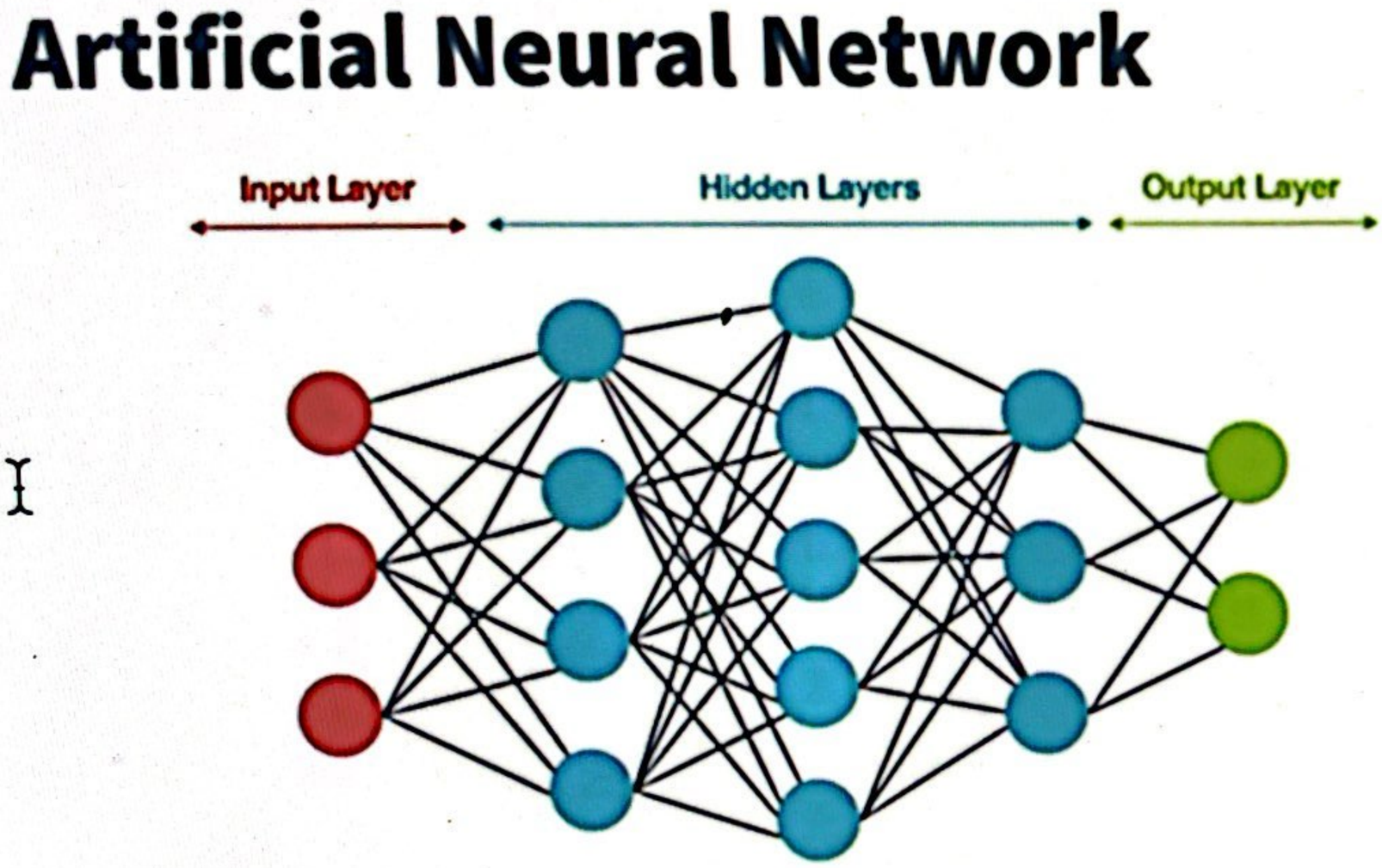
# Perceptron




## Artificial neural networks

- How do we use perceptrons to build an artificial neural network, or in short, ANNs. An ANN is a network of perceptrons. We discussed earlier that a perceptron imitates the human brain cell. Similar to how a human brain is created with a network of cells, an ANN is created with a network of perceptrons. The perceptron is also called a node in the neural network. We use the node as the term to represent perceptrons going forward in the course. Nodes are organized into multiple layers in a neural network. A deep neural network usually has three or more layers. Each node has its own weights, biases, and activation function as discussed in the previous video. Each node is connected to all the nodes in the next layer forming a dense network. The nodes within a layer are not connected with each other. There are some exceptions to these in advanced use cases though. This diagram shows an example neural network. Each neural network has one input layer, one or more hidden layers, and one output layer. In the input layer, there is one node for each independent variable. In this example, there are three. The hidden layer has three layers in this example. Layer one has four nodes. Layer two has five nodes. And layer three has three nodes. The number of layers and the number of nodes in each layer are determined by experience and trials, and it will vary from case to case. The number of nodes in the output layer will vary based on the type of

predictions. The output layer in this example has two nodes. This arrangement of nodes represents the architecture of a given neural network. How does an ANN work for predictions? The inputs, or independent variables, are sent from the input layer of the network. Data may be pre-processed before using them. Inputs are passed on to the next layer. Each node is a perceptron containing weights, bias, and an activation function. The formula is applied on the inputs and the outputs derived. This repeats for each node in the layer. the results from all the nodes in the layer are then passed on to the next layer, and this process is repeated. As the process reaches the output layer, the final predictions will be derived. We will discuss more on this process as we progress through the course.




## Artificial neural networks

- How do we use perceptrons to build an artificial neural network, or in short, ANNs. An ANN is a network of perceptrons. We discussed

earlier that a perceptron imitates the human brain cell. Similar to how a human brain is created with a network of cells, an ANN is created with a network of perceptrons. The perceptron is also called a node in the neural network. We use the node as the term to represent perceptrons going forward in the course. Nodes are organized into multiple layers in a neural network. A deep neural network usually has three or more layers. Each node has its own weights, biases, and activation function as discussed in the previous video. Each node is connected to all the nodes in the next layer forming a dense network. The nodes within a layer are not connected with each other. There are some exceptions to these in advanced use cases though. This diagram shows an example neural network. Each neural network has one input layer, one or more hidden layers, and one output layer. In the input layer, there is one node for each independent variable. In this example, there are three. The hidden layer has three layers in this example. Layer one has four nodes. Layer two has five nodes. And layer three has three nodes. The number of layers and the number of nodes in each layer are determined by experience and trials, and it will vary from case to case. The number of nodes in the output layer will vary based on the type of predictions. The output layer in this example has two nodes. This arrangement of nodes represents the architecture of a given neural network. How does an ANN work for predictions? The inputs, or independent variables, are sent from the input layer of the network. Data may be pre-processed before using them. Inputs are passed on to the next layer. Each node is a perceptron containing weights, bias, and an activation function. The formula is applied on the inputs and the outputs derived. This repeats for each node in the layer. the results from all the nodes in the layer are then passed on to the next layer, and this process is repeated. As the process reaches the output layer, the final predictions will be derived. We will discuss more on this process as we progress through the course.

## Training an ANN

In the previous video we discussed how a neural contains layers, notes, weights, and biases. How do we create that network? An ANN is created through a model training process. What does training actually mean? A neural network model is represented by a set of parameters and hyperparameters. This includes the values of weights and biases for all the nodes, and the number of nodes and the layers in the network. There are more hyperparameters which we will discuss later in the course. Training an ANN model means determining the right values for these parameters and hyperparameters such that it maximizes the accuracy of predictions for the given use case. They may also compromise accuracy for performance. Do note that the inputs, weights and biases may each be n-dimensional arrays depending on the use case. For example, an image maybe represented by an array of pixel values. Let's go back to the analogy we discussed earlier in the course. Deep learning training works similarly with weights and biases, replacing A and B here respectively. We start with random values for weights and biases, and then work our way towards minimizing the error. As we progress, we will get closer to the values of weights and biases that can predict the outcomes accurately. The training process would then look as follows. We use training data like regular machine learning, where we know both the dependent and independent variables. We start with the network architecture by intuition. We also initialize weights and biases to random values. Then we repeat the iterations of applying weights and biases to the inputs and computing the error. Based on the error found, we will adjust the weights and biases to reduce the error. We keep repeating the process of adjusting weights and biases until the error gets to an acceptable value. We will also fine tune the network hyperparameters to improve training speed and reduce iterations. Finally, we will save the model as

represented by its parameters and hyperparameters, and then use it for predictions. Now that we have an understanding of what an ANN is, let's explore the training process in detail in the next chapter.

## The input layer

- In this chapter, let's deep dive into the architecture of an ANN and explore various layers, parameters, and functions used in the architecture. We will start with the input layer. Let's first start with the concept of vectors. A vector is an ordered list of numeric values. The input to deep learning is usually a vector of numeric values. A vector can be set to be a tuple of one or more values. Vectors are usually defined using a NumPy array. It represents the feature variables or independent variables that are used for predictions as well as training. Input data sets that are available from the real world for machine learning contain samples and features. A sample is one instance of a real world example. It is equivalent to records in a database. Features are individual attributes in the sample.

# Input Preprocessing

**Features need to be converted to numeric representations.**

| Input Type | Preprocessing Needed |
|---|---|
| Numeric | Centering and scaling |
| Categorical | Integer encoding, one-hot encoding |
| Text | TF-IDF, embeddings |
| Image | Pixels – RGB representation |
| Speech | Time series of numbers |

In the sample, we have a data set of employees. Each employee record represents a sample. Individual attributes like age, salary, and service are considered features. For text data, each document is a sample and it's numeric representation becomes its features. For images, each image is a sample and its pixel representation becomes its features. Similar to regular machine learning, input data needs to be pre-processed and transformed to appropriate numeric representations before they are fed into a neural network. Here are some popular pre-processing techniques used. For input data, we usually center and scale them to normalize the values. For categorical variables, we encode them using either integer encoding or one hot encoding techniques. Text data needs to be converted to equal numeric representations. TF-IDF, or text frequency-inverse document frequency, is a classic example to represent documents. Embeddings are becoming more popular in the deep learning world. Images are represented by pixels. We will create a vector of pixel values for this. Speech may be represented as a time series of numbers. This list is no way exhaustive. There are a number of advanced pre-processing techniques that are applied to data to prepare them for deep learning. This is an example of how employee data can be pre-processed. Here, we represent each attribute like age, salary, and service as individual features x1, x2, and x3. We will center and scale the values to normalize them into standard ranges. Optionally, we will also transpose them to represent each sample as a column. Once the input data is ready, it can be passed to the deep learning network for learning.

# Samples to Input Vectors

**Raw Data**

| Age | Salary | Service |
|---|---|---|
| 28 | 40,000 | 5 |
| 47 | 60,000 | 20 |
| 35 | 55,000 | 10 |

→

**Centered and Scaled**

| x1 | x2 | x3 |
|---|---|---|
| -1.10 | -1.37 | -1.07 |
| 1.32 | 0.98 | 1.34 |
| -0.21 | 0.39 | -0.27 |

→

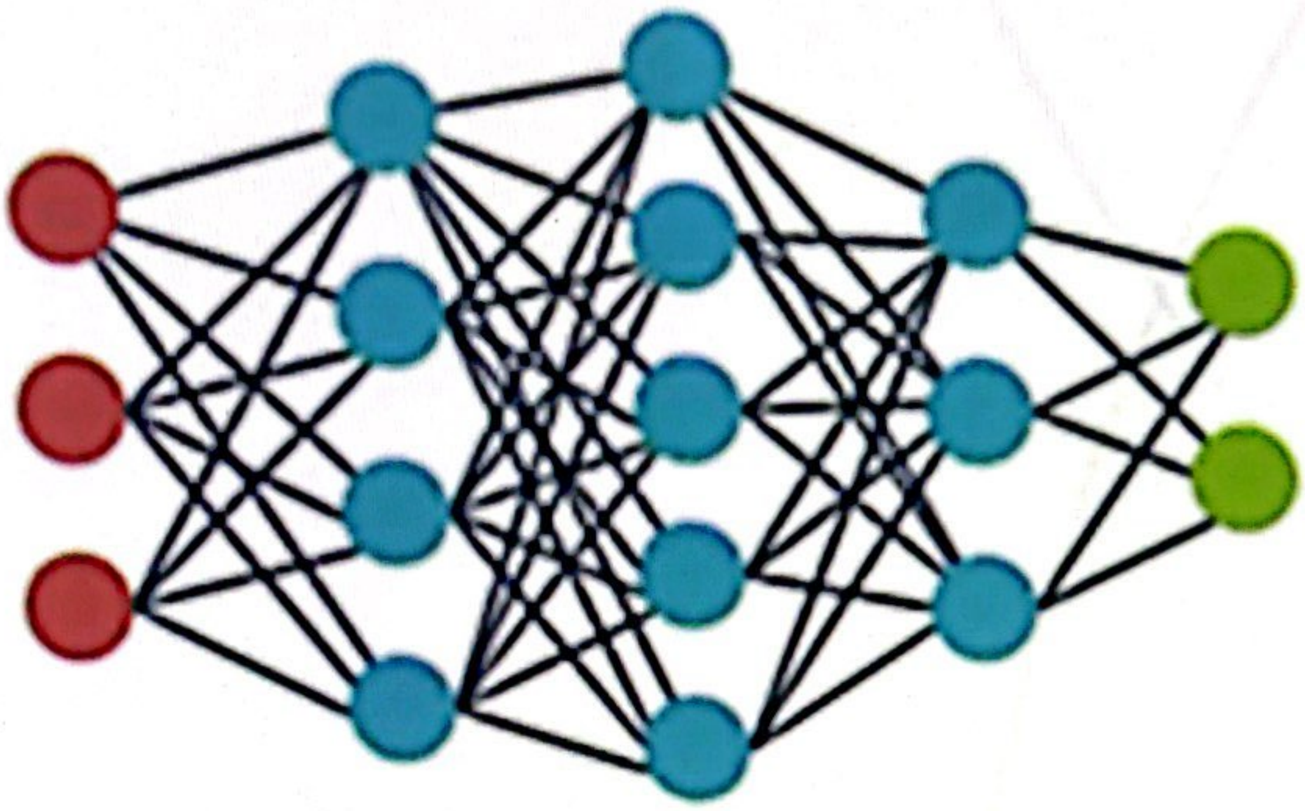
**Transposed**

| | | | |
|---|---|---|---|
| x1 | -1.1 | 1.32 | -0.21 |
| x2 | -1.37 | 0.98 | 0.39 |
| x3 | -1.07 | 1.34 | -0.27 |

## Hidden layers

- Hidden layers in a neural network form the brain where knowledge is acquired and used. An ANN can have one or more hidden layers. The more the number of layers, the deeper the network is. Each hidden layer can have one or more nodes. Typically node count is configured in the range of 2 power n Example counts may be 8, 16, 32, 64, 128, et cetera. A neural networks architecture is defined by the number of layers and nodes in that layer.

How are the inputs and outputs connected? The output of each node in the previous layer will become the input for every node in the current layer. Similarly, the output of each node in the current layer is passed to every node in the next layer. In the example shown, there are four nodes in the first hidden layer, producing four outputs from their activation functions.



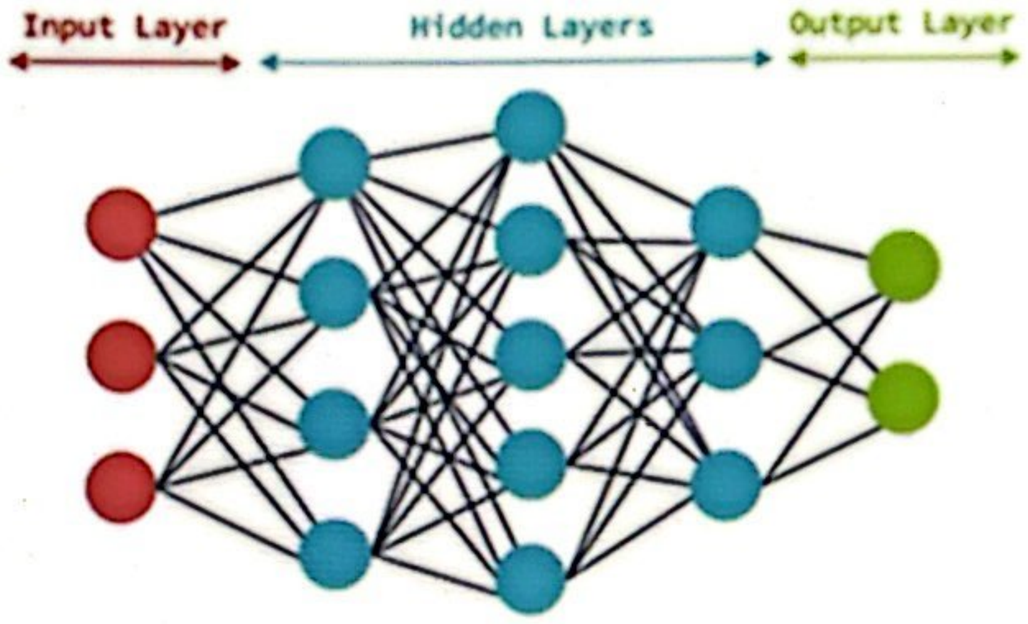
There are five nodes in the second layer. The four outputs from the first layer are passed to each of the five nodes in the second layer. How do we determine the right number of layers and nodes in a network? Each node in the neural network learns something about the relationship between the feature variables and the target variables. This knowledge is persisted in its weights and biases. When there are more nodes and layers, it usually results in better accuracy. Do note that this is always not true. More layers would also mean more compute resources and time for both training and inference. The right architecture for a given problem is determined by experimentation. As a general practice, start with small numbers and keep adding until acceptable accuracy levels are obtained.

## Weights and biases

- Weights and biases form the foundation for deep learning algorithms. How are they structured for a given neural network architecture. Weights and biases are the trainable parameters in a Neural Network. During the training process, the values for these weights and biases are determined such that they provide accurate predictions. Weights and biases are nothing but a collection of numeric values. Each input for each node will have an associated weight with it. A given node will

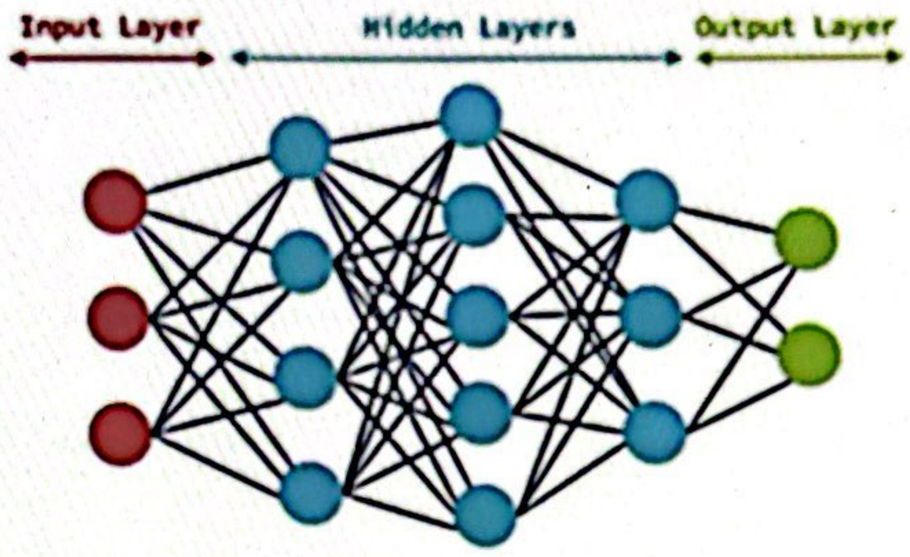
have one bias value associated with it. At a layer level these weights and biases are handled as RAs. Let's compute the number of weights and bias values, for the example network. For the first hidden layer, there are three inputs and it has four layers. Each node will then have three weight values, so for all the four layers, there will be 12 weights. There is one bias value per node, so there will be a total of four bias values for that layer. In a similar fraction, we can compute the remaining hidden layers and the output layer. This network has a total of 53 weights, and 14 bias values. The network does has 67 total parameters.

## Weights and Biases for a Layer




| Layer | Inputs | Nodes | Weights | Biases |
|---|---|---|---|---|
| HL 1 | 3 | 4 | 12 | 4 |
| HL 2 | 4 | 5 | 20 | 5 |
| HL 3 | 5 | 3 | 15 | 3 |
| Output | 3 | 2 | 6 | 2 |
| **Total** | | | **53** | **14** |

When training a Neural Network, computation for the hidden layer is done together, and not for each node, for optimization reasons. The weights and biases for each layer are maintain as matrices. The input is also taken as a matrix, and the output is produced as a matrix. This example shows the computation for hidden layer two which has four inputs and five nodes.




$$\begin{bmatrix} w11 & w21 & w31 & w41 \\ w12 & w22 & w32 & w42 \\ w13 & w23 & w34 & w43 \\ w14 & w24 & w34 & w44 \\ w15 & w25 & w35 & w45 \end{bmatrix} * \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \\ b3 \\ b4 \\ b5 \end{bmatrix} = \begin{bmatrix} y1 \\ y2 \\ y3 \\ y4 \\ y5 \end{bmatrix}$$

$$W^T X + B = Y$$

The four inputs are received as a metrics of one by four. The weights for all the nodes in the layer, are maintain as a four by five matrix. Bias is a one by five matrix covering the five nodes in the layer. Matrix multiplication is performed between the input and weight matrices to get an output of one by five, this is added to the bias metrics to produce an output of one by five. The output matrix is then passed as input to the next layer. I recommend additional reading on matrix multiplication, if you are interested in more details around these computations. Deep learning applications take care of the computation work behind the scenes, based on the configured inputs and settings.

## Activation functions

- An Activation function plays an important role in creating the output of a node in the neural network. An Activation function, takes in the matrix output of the node and determines if and how the node will propagate its information to the next layer. Activation functions act as filters to reduce noise and also normalize the output, which can get fairly large due to matrix multiplications. It converts the output to a nonlinear value. They serve as a critical step in helping a neural network learn specific patterns in data. Here is the list of some of the most popular activation functions. Each of these functions have specific advantages, shortcomings and applications. They can take in an output matrix and deliver another output matrix of the same dimension.

# Popular Activation Functions

| Activation Function | Output |
|---|---|
| Sigmoid | 0 to 1 |
| Tanh | -1 to +1 |
| Rectified Linear Unit (ReLU) | 0 if x < 0; x otherwise |
| Softmax | Vector of probabilities, with sum=1 |

**Choice depends on problem and experimentation.**

A sigmoid function delivers an output in the range of zero to one, based on the input values. When it has a value of zero, it means that it does not pass its learnings to the next layer. A Tanh function normalizes the output in the range of minus one to plus one. A rectified linear unit or ReLU, produces a zero if the output is negative. Else, it reproduces the same input verbatim. A Softmax activation function is used in case of classification problems. It produces a vector of probabilities for each of the possible classes in the outcome. The sum of probabilities will be equal to one. The class with the highest probability will be considered for prediction. I recommend further reading on the math concepts between activation functions and their advantages and shortcomings if you are interested. Otherwise, they are implemented in various deep learning libraries already, and need to be just added as a hyper parameter to the model.

## The output layer

- The output layer is the final layer in the neural network where desired predictions are obtained. There is one output layer in a neural network that produces the desired final prediction. It has its own set of weights and biases that are applied before the final output is derived. The activation function for the output layer may be different than the hidden layers based on the problem. For example, Softmax activation

is used to derive the final classes in a classification problem. The output is a vector of values that may need further post-processing to convert them to business related values. For example, in a classification problem, the output is a set of probabilities that needs to be mapped to the corresponding business classes. How do we determine the number of notes in the output layer? It depends on the problem. In a binary classification problem, there is only 1 note that provides a probability of the positive outcome. In the case of n-class classification, there are n notes, each producing the probability for a given class. For regression, there is only 1 note that produces the output. In this way, the number of notes may vary based on the type of problem being solved. This completes our discussion on the structure of the neural network. In the next chapter, we will explore more about training the network.

## Training a neural network

### Setup and initialization

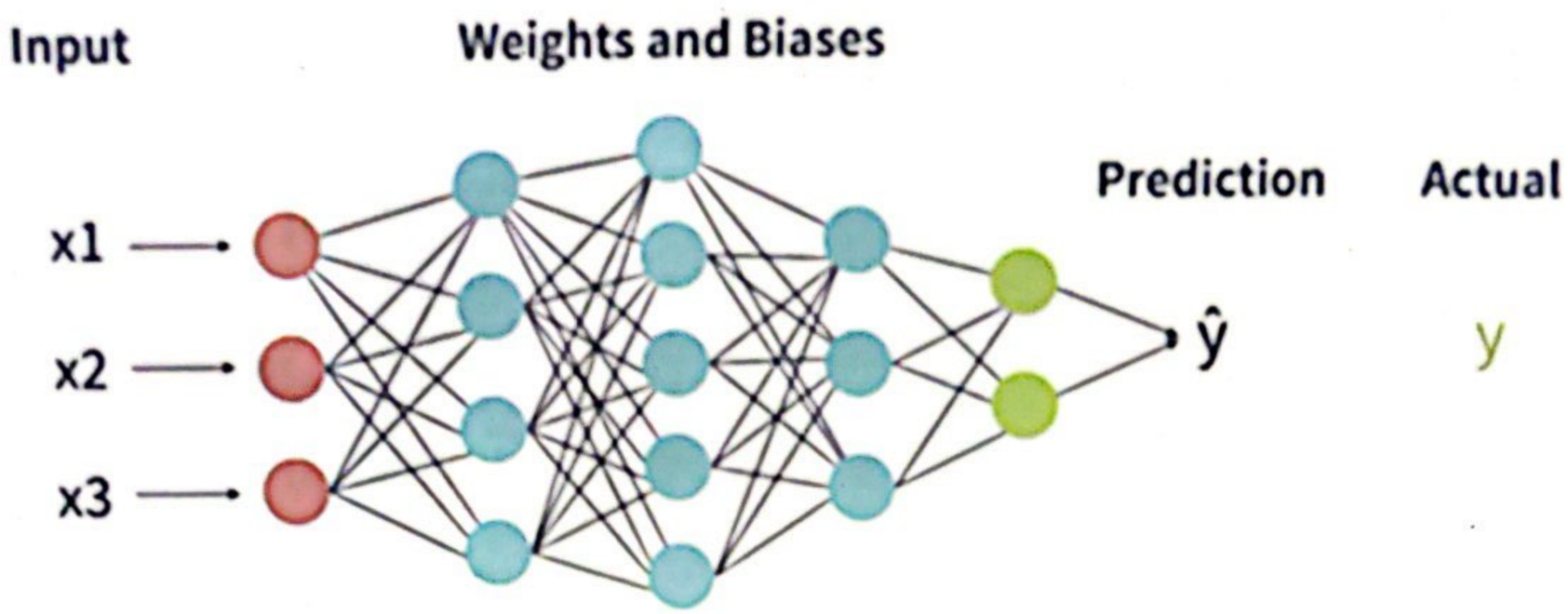
- How does a deep learning model get trained? We will explore this process in detail in this chapter. We will start with setup and initialization for the training process in this video. Before we start training the neural network, the input data needs to be prepared. This includes applying a number of processing techniques to convert samples into numeric vectors. They are then transposed optionally to create the input vectors. The target variables may also undergo similar transformations. To help with training, the input data is usually split into training, test, and validation sets. A training data set is used to run through the neural network and fit the parameters like weights and biases. Once a model is created, the validation data set is used to check for its accuracy and error rates. The result from this validation is then used to refine the model and recheck. When a find a model is obtained, it is used to predict on the test set to measure the final model performance. The usual split of input data between the training, validation, and test sets is 80 to 10 to 10. In order to create the initial model, a set of values need to be selected for various parameters and hyper-parameters. This includes the number of layers and the number

of nodes in each layer. We also need to select the activation functions for each layer. Then, there are hyper parameters like epoch, batch sizes, and error functions that need to be selected. How do we make the initial selection? It may be based on our own intuition and experience. It can also be based on references in best practices and suitability of techniques to the specific problem. Whatever values are selected, they are then refined as the model is trained. If the final results of the model are not acceptable, then we will go back, adjust the parameters, and then retrain the model. Finally, we also need to initialize the weights and biases for each of the nodes in the neural network. We will start with some value and then the neural network will learn the right values for these based on the error rates are obtained during the training process. Multiple techniques for initialization are available. In zero initialization, we initialize all values to zeros. The preferred technique though is random initialization. In random initialization, we initialize the weights and biases to random values. obtained from a standard normal distribution whose mean is zero and standard deviation is one. Once we are done with setup and installation, we are ready to do some training.

## Forward propagation

- Once we are ready with the input training data, we are ready to do a round of forward propagation. To recollect, we have the input data organized as samples and features. The data has been pre-split into training, validation, and test sets. For the training set, for each sample, we have a target or the value to predict, called y. y is the actual value of the target in the training set. y hat will be the value that will be predicted through forward propagation. The forward propagation step is exactly the same as doing an actual prediction with the neural network. For each sample, the inputs are sent through the designated neural network. For each node, we compute the outputs based on the perceptron formula and pass them to the next layer. The final outcome, y hat, is then obtained at the end.

# Forward Propagation: 1 Sample




As we keep sending the samples to the neural network, we will collect values of y hat for each sample. This process is repeated for all samples in the training data set. We will then compare the values of y hat and y and compute the error rates. We will discuss more on computing error rates in the next video. ,

## Measuring accuracy and error

- Accuracy and Error are alternating terms that can be used to represent the gap between the predicted values and the actual values of the target variables. As we go through forward propagation, we end up with a set of y-hat values that need to be then compared with the actual values of y to compute the error. For computing error, we use two functions, namely, the loss function and the cost function. A loss function measures the prediction error for a single sample, a cost function measures the error across a set of samples. The cost function provides an averaging effect over all the errors found on the training dataset. The terms loss function and cost function are used almost interchangeably and are used to measure the average error over a set of samples. There are a number of popular costs functions available, and they are implemented in all popular Deep Learning Libraries.

# Popular Cost Functions

| Cost Functions | Applications |
|---|---|
| Mean Square Error ( MSE ) | Regression |
| Root Mean Square Error ( RMSE ) | Regression |
| Binary Cross Entropy | Binary classification |
| Categorical Cross Entropy | Multi-class classification |

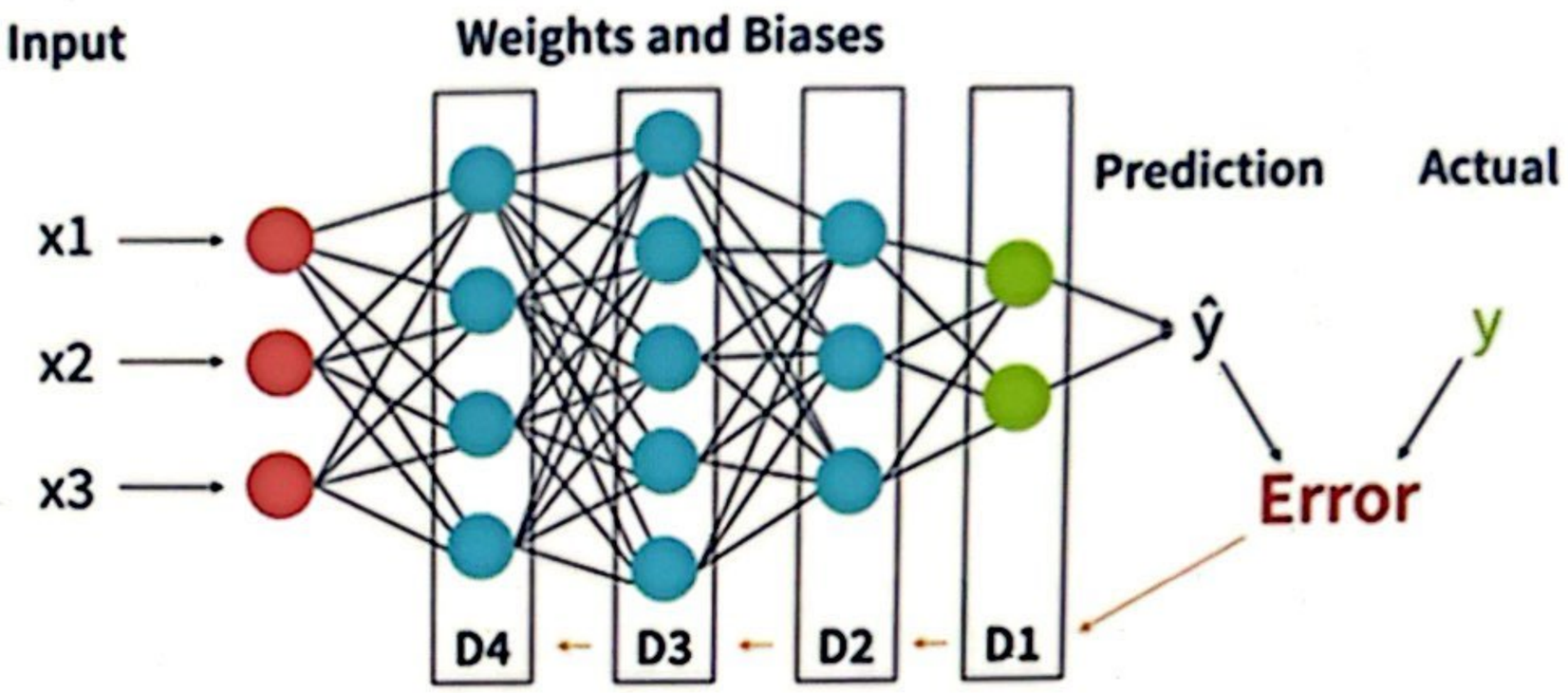
The Mean Square Error or MSE measures errors in case of regression problems. It computes the difference between the predictor and the actual values, squares them, and sums them across all samples, and finally divides by the number of samples. The Root Mean Square Error or RMSE is more popular, as it provides error values in the same scale as the target variables. This is again used for regression problems. For binary classification, we use Binary Cross Entropy to compute the error. A similar function called Categorical Cross Entropy exist for multi-class classification problems. If you wish to understand the math behind these cost functions, I would recommend referring to additional documentation. So with the cost function, how do we measure accuracy? We send a set of samples through the ANN for forward propagation and predict the outcomes. We estimate the prediction error between the predicted outcome and expected outcome using a cost function. We will then use backward propagation to adjust the weights and biases in the model based on the error obtained. We will discuss backward propagation in the next video.

## Back propagation

Once we have estimated the prediction error from forward propagation, we need to go back to back propagation to adjust the

weights and biases. What is the purpose of back propagation? We found an overall error based on the entire network during forward propagation. Each node in the neural network contributes to this overall error. And nodes contribution is determined by the values for the weights and biases, it has. Different nodes contribute differently, based on how well their weights and biases model the relationship between the feature and target variables. As we tuned the network, the weights and biases for each node needs to be adjusted in order to lower the error contribution, by that node. How does back propagation work? It works in the reverse direction as the forward propagation. We start from the output layer. We will compute a delta value for this layer based on the overall error. This delta value is an adjustment that is then applied to all the weights and biases in the layer. This results in new values of weights and biases. Then we derive a new delta value for the previous layer based on the new values in the current layer, it is then applied to the weights and biases in the previous layer. The process of computing deltas, applying it to the weights and biases, and then back propagating continues until we reached the input layer. This image shows the same back' propagation process for the network example. The deltas D1 to D4 are computed at each layer and applied to their weights and biases. They also propagate to the previous layer and influence their deltas.

# Back Propagation




If you are interested in the math behind these computations, I recommend additional readings. Again, the deep learning libraries take care of these computations. At the end of the back propagation process, we will have an updated set of weights and biases that would reduce the overall prediction error. How do we continue to reduce error and improve accuracy? Let's discuss that in the next video.

## Gradient descent

- Gradient descent is the process of repeating forward and backward propagations in order to reduce error and move closer to the desired model. To recollect one run of the forward propagation results in predicting the outcomes based on weights and biases. We compute the error using a cost function. We then use back propagation to propagate, edit, and adjust the weights and biases. This is one pass of learning. We have to now repeat this pass again and again, as the weights and biases get refined, and the error gets reduced. This is called gradient descent. In gradient descent, we repeat the learning process

of forward propagation, estimating error, backward propagation, and adjusting weights and biases. As we do this, the overall error estimated by the cost function will oscillate around and start moving closer to zero. We keep measuring the error and computing deltas that would minimize the error contribution of individual notes. There are situations where the error will stop producing and there are additional hyperparameters to control that. There are also hyperparameters to speed up or slow down the learning process. We will discuss them in the followup course, deep learning model optimization and tuning.

## Batches and epochs

- Batches and epochs help control the number of passes through the neural network, during the learning process. What is a batch? A batch is a set of training samples, that are sent through the neural network in one single pass. The training data set is divided into one or more batches. The neural network receives one batch at a time, and does forward propagation. Cost functions are executed and the weights and biases updated after each batch. When the next batch comes in, it will have a new set of weights and biases to work with. There are two types of gradient descent. When the batch size is equal to the training set size, it's called batch gradient descent. When the batch size is less than the training data set size, it's called mini-batch gradient descent. Batch sizes are typically based on two power and numbers like 32 to 64, 128, et cetera. A training data set is sent through the neural network multiple times during the learning process. The total number of times the entire training data set is sent through the neural network is called epoch. An epoch obviously would have one or more batches. As more epochs happen, the same batch is sent repeatedly through the neural network, but will get to work with a different set of weights and biases each time. When all epochs are completed, the training process is complete. Higher epoch sizes may lead to better accuracy while also delaying the learning process.

## Epoch and Batch Example

- Training set size = 1000, batch size = 128, epoch = 50
- Batches per epoch = ceil ( 1000 / 128 ) = 8
- Total iterations (passes) through ANN = 8 * 50 = 400
- Batch size and epoch are hyperparameters that can be tuned to improve model accuracy

## Validation and testing

As we build models, we need to also validate and test them against independent data sets to measure out of sample error. During the input preparation process, we usually isolate validation and test data sets for this purpose. What is validation? While performing, learning, and model improvement, we are comparing the predictions provided by the neural network for the training samples against its actual values and measuring errors. However, this is an in sample error and the model has no guarantee that it will perform the same against independent datasets. So after each epoch is completed and the weights and biases updated, we will also use the network to predict for the validation data set. We will measure accuracy and loss for the validation data set, and also investigate the same to make sure that it does not deviate significantly from the in sample errors observed. The model can be fine tuned, and the learning process repeated based on the results seen against the validation dataset. The final step in model building is evaluation. After all the fine tuning is completed, and the final model obtained, the test data set is used to evaluate the model. This is done only once at the end. The evaluation results are then used to measure

## An ANN model

- Having now seen the neural network constituents and the training process, let's recap the question. What is an ANN model? What does it contain? An ANN model is represented by a set of parameters, namely the weights and biases that are obtained during training. When

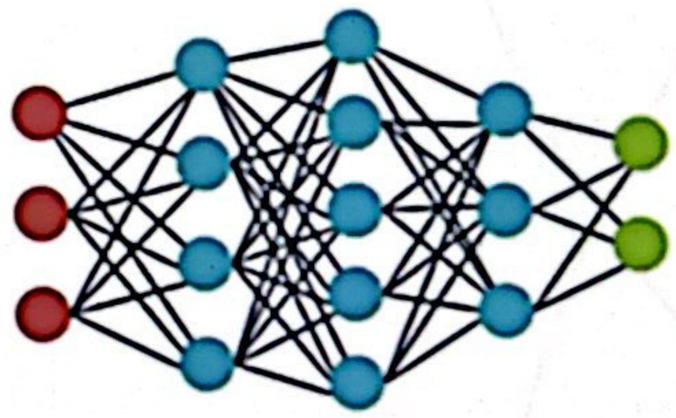
someone says that the model has X parameters, they are mentioning the total count of weights and bias values in the model. A model is also represented by a set of hyperparameters. This includes the number of layers, nodes in each layer, activation functions in each node, cost functions, optimizers and the learning rate used. It also includes the batch size and epoch values used to train the model. A model file typically contains the representation of all these values. Models can be saved to files, shared and loaded into other binaries if needed. Once you have a model, what does the prediction process look like? The prediction process is exactly the same as the forward propagation step, except that the input data used is the feature attributes for prediction and the target value is not known, so the steps involve pre-processing and preparing the inputs, passing them through the layers, computing the outputs in each node using their final weights and biases

# Prediction Process

**Preprocess and prepare inputs**

**Pass inputs to the first layer**

- **Compute Y using weights, biases, activation**
- **Pass to the next layer**



**Repeat process until output layer**

**Postprocess output for predictions**

## Reusing existing network architectures

- Having discussed the concepts of neural networks and how to train them from scratch, let's now discuss some practical aspects of building neural network models. How do we build neural networks for a use case? An interesting fact about neural networks is that most neural network implementations are not designed and built from scratch. Designing a neural network with the right number of layers and nodes is a tedious, iterative, and time consuming process. Fortunately, the neural network community is very active in sharing their work with the rest of the world. They shared their knowledge and experiments for the rest of the community to build upon. To begin with several papers are published on the architectures for neural networks that have been successfully implemented and proven. You can start off your neural network by implementing a related architecture and then fine tune it for your use case. Implementation code for these neural networks is also available in open source repositories. This can be leveraged to create a neural network for your use case. Finally, open source models are available for popular implementations. The models include all the trained parameters and hyper-parameters packaged in standardized formats that are supported by popular deep learning networks.

## Using available open-source models

- How do we select and use available open-source neural network models? We will briefly touch upon some key points in this video. Several open source models and their versions are available for use by anyone wishing to develop a neural network for their use case. These are fully trained models with updated parameters and corresponding hyper parameters. Training data and code will also be shared in many cases. Repositories like Hugging face and GitHub are popular locations where such models can be found. Universities also host models that are created from their research. These models are easy to download. They can be pretty large files though, and would need bandwidth and storage. Popular machine learning frameworks like PyTorch and TensorFlow provide capabilities to quickly load and use these models.

How do we pick the right open-source model for our use case? First, when evaluating an open-source model, understand their original purpose and use case. This helps us understand what this model is good at and whether this is applicable for our use case. It is also important to understand the type of data that this model is trained on. Data could be public data or maybe focused on a specific use case. There can also be privacy and legal concerns that needs to be looked at. Explore the model's, popularity and usage. This usually can be found by the number of downloads, forks and blocks related to this model. Review licensing requirements for the model, even though it's open source, different categories of licenses determine how they can be used, and if attribution is required. Then proceed to download the model and build fine tuning or inference pipelines around it. Test the model with training data that is specific to your use case, to ensure that the model performs well in your specific scenario. We briefly touched upon open-source neural network models, and I recommend that you explore more about them and try using one of them.