TOP 50
DEEP LEARNING
INTERVIEW QUESTIONS
YOU MUST PREPARE IN
2022

Q1) What is the difference between Deep Learning and Machine Learning?

Ans1:

Conceptually, Deep Learning is quite similar to Supervised Machine Learning, in which Data Scientists use labeled data to train a model using an algorithm and then use this model to predict labels of new data. Differences between Deep Learning and ML are

Factors	Deep Learning	Machine Learning
Hardware Requirement	Requires GPU to train properly	Works on CPU as well
Feature Engineering	Facilitates automatic feature extraction	It does not facilitate automatic feature extraction
Accuracy	Provides high accuracy	Gives lesser accuracy
Data Requirement	Requires a large amount of data to understand the patterns completely	Does not require a very large dataset to understand patterns in the data
Training Time	Large number of hyperparameters and complex mathematical vector operations make training slow	ML algorithms require relatively less training time.

Q2) What are Artificial Neural Networks (ANN)?

Ans2:

ANN"s were developed in an attempt to help computers simulate the way a human brain works, using a network of neurons to process information. It helps computers learn things and make decisions in a human-like manner. An ANN consists of a few hundred to millions of neurons (also called nodes), which are divided into layers that are interconnected to form a complex network. It consists of 3 different layers:

- Input Layer: The layer which receives the inputs from the training datasets.
- Hidden Layers: It follows the input layer. There can be one or more hidden layers. It facilitates   forward and backward passes and also helps in minimizing the error with each pass.
- Output Layer: It outputs a probability, which is used to assign a class to the set of input.
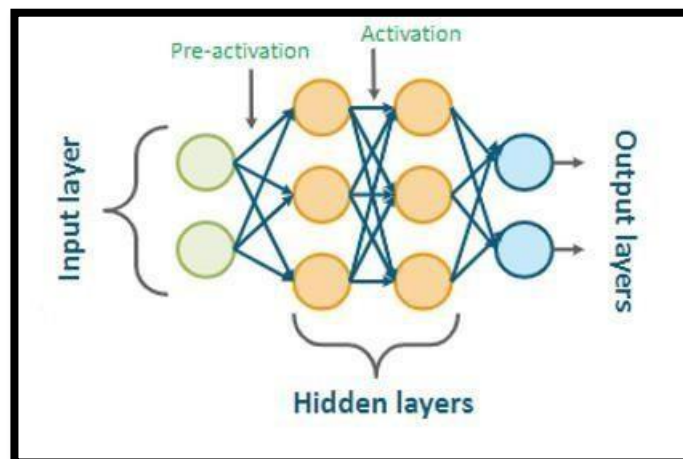
Q3: What are the typical applications where neural networks are being used in the real world?

Ans3:

Some applications in real-world where neural networks are being used are:

- **Security:** Detection of bombs in suitcases
- **Financial Risk Analysis**: Predicting stock prices (helping investors make informed decisions)
- **Weather Prediction:** Forecasting weather patterns
- **Cybersecurity:** Identifying fraudulent credit card transactions
- **Healthcare:** Predicting the risk of heart attacks from ECG output waves

Q4: Explain one real world application where ANN can be used?

Ans4:

Traveling salesman problem: A salesman has to cover all the cities in a given area. He wants to figure out the shortest possible path to travel to all the cities. He uses neural networks to solve this problem. A neural network algorithm like genetic algorithm starts with a random orientation of the network. The algorithm chooses a city in a random manner and finds the nearest city. This process continues several times and after every iteration, the shape of the network changes and the network converges to a ring around all the cities. The algorithm aims to minimize the length of this ring with eachiteration.

Q5: What is Multilayer Perceptron (MLP)? How does it overcome the shortcomings of Single Layer perceptron? MLP's are feedforward ANN's that consist of multiple hidden layers and generate a set of outputs from a set of inputs. MLP uses backpropagation as a supervised learning technique. It has 3 layers:

Ans5:

- **Input layer:** The input nodes provide information from the outside world to the network and are together referred to as the "Input Layer". No computation is performed in any of the input nodes, they just pass on the information to the hidden nodes.

- **Hidden layers:** The hidden nodes perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a "Hidden Layer".

- **Output layer:** The output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world.

A single-layer perceptron cannot deal with non-linearly separable classes. MLP"s overcome this limitation by:

- Using a non-linear activation like logistic sigmoid, ReLU or tanh in its hidden layers and output layer, which help it to understand the non-linearities in the data.

- Using multiple hidden layers and nodes, which help it to understand complex patterns in the data better.

Q6: Which is the simplest type of Artificial neural network? What is its limitation?

Ans6:

Single-layer perceptron is the simplest ANN. It is a single layer, binary linear classifier. It classifies objects and groups by finding a linear separation boundary between different classes. It has only one node. Below is a basic representation of a perceptron:
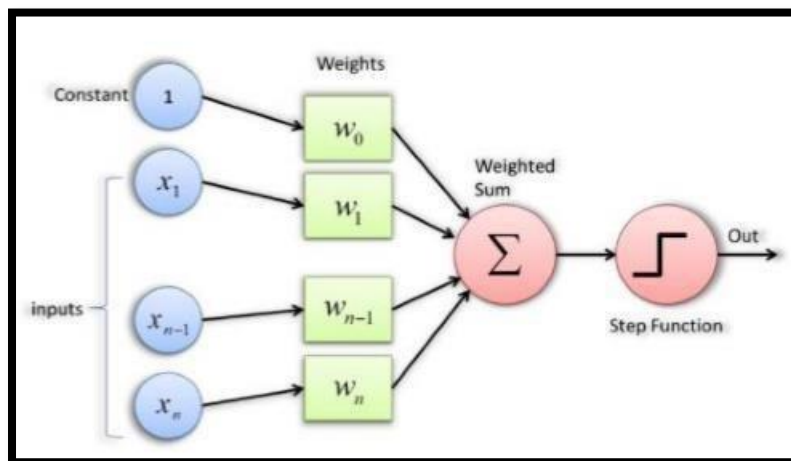

It consists of:

- A set of **input values** with their associated **weights** and **biases**
- A **pre-activation function**, calculated by a sum of products of inputs and their associated weights. A bias is also added to this to provide every node with a trainable constant value in addition to normal inputs
- The **activation function** is a step function. If the weighted sum exceeds the predefined threshold, then it assigns one class, else the other.

It updates the values of weights and bias matrices based on the loss (difference between actual and predicted values) and the model learns progressively with each iteration.

Its **limitation** is that it is only able to classify a linearly separable set of inputs. It cannot deal with classification problems where the classes cannot be separated by a linear separation boundary.

Linear Problems · Non-Linear Problems
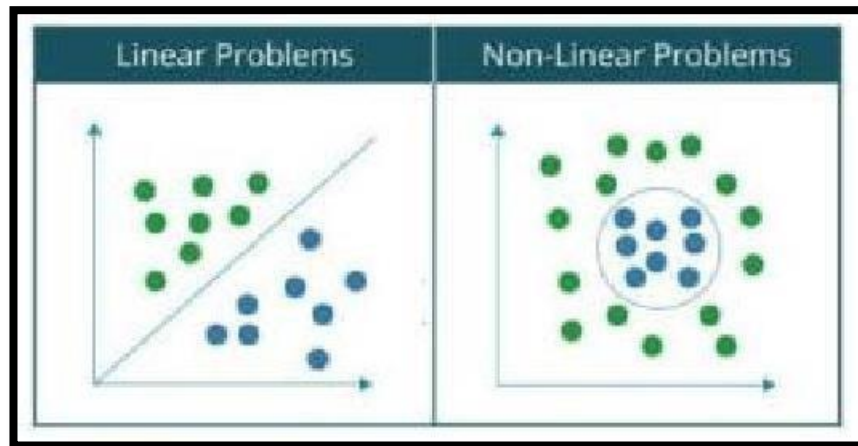
Q7: What is an Activation Function? Explain it with respect to a single node?

Ans7:

The figure shows a single neuron (or node), which receives inputs from nodes of the previous layer, and a bias to add a trainable parameter. Each input to the neuron is associated with a weight, which is assigned based on its relative importance with respect to other inputs. The node computes the weighted sum of the inputs and the bias, and applies a function „f" to the computed sum. This function „f" is called activation function (or non-linearity).

The purpose of the activation function is to introduce non-linearity to the output of the neuron (This is important as the most real-world data is non-linear, and we want the neuron to learn these nonlinearities).
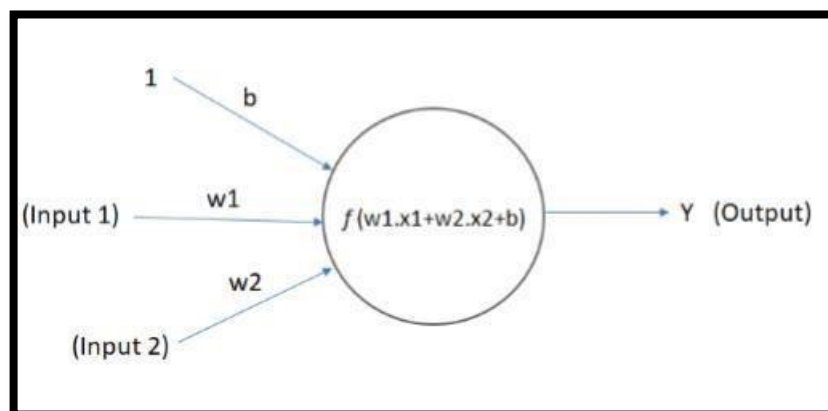
Q8: Why is SoftMax Activation function primarily used in the output layer of NN?

Ans8:

When we are using a neural network to solve a "multiclass" classification problem with „K" number of categories/ classes, the SoftMax function is used at the output layer to calculate a probability value

for each class. It outputs values in the range (0,1) for each output class, such that the sum of outputs of all classes is equal to 1.

Advantages of using SoftMax function are:

1) The properties of SoftMax (all output values in the range (0, 1) and sum up to 1.0) make it suitable for a probabilistic interpretation which is very useful in ML.

2) SoftMax normalization is a way of reducing the influence of extreme values or outliers in the data without removing data points from the set.

Q9: What are the different types of activation functions used in neural networks?

Ans9:

Some of the commonly used activation functions are:

- Sigmoid Function: It takes a single real-valued input and squashes it between 0 and 1.
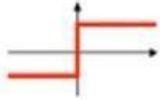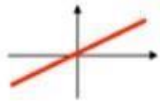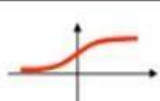
σ(x)= 1/(1+e-x)

- ReLU (Rectified linear unit): It converts negative values to 0 and retains positive ones.

f(x) = max(0,x)

- tanh: It takes a real value and squashes it between [-1,1]

tanh(x)= 2σ(2x)-1

Below is a graphical representation of some well-known activation functions:

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \dfrac{1}{1+e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer NN	

Q10: How does using the tanh activation function help the neural network to converge faster than logistic sigmoid?

Ans10:

The global minima of the loss function can be achieved much faster i.e, in a fewer number of epochs when using tanh activation function. It outputs values between -1 and 1, which helps the weight vectors to change directions much faster as opposed to logistic sigmoid which outputs values between 0 and 1.

Q11: What are the various techniques of Data Normalization and how does it help in boosting the training of a neural network?

Ans11:

Data Normalization in Deep Learning is often applied as a part of data pre-processing. The goal of normalization is to change the values of numeric columns in the dataset to a common scale. Normalizing the data generally speeds up learning and leads to faster convergence. 3 main methods used for normalizing data are:

1) **Rescaling (min-max scaling):** It transforms data to a scale of [0,1].
   $xnorm = (x-xmin)/(xmax-xmin)$

2) **Standardization:** The data is normalized to a Z-score(standard score).
   $xnorm=(x-\mu)/\sigma$

3) **Scaling to unit length:**
   $xnorm= x/ \|x\|$, where $\|x\|$ is the Euclidean length of the feature vector.

Normalization helps in boosting the training of a neural network by:

- Ensuring a feature has both positive and negative values which makes it easier for the weight vectors to change directions. This helps in making the learning flexible and faster by reducing the number of epochs required to reach the minima of the loss function

- Normalization ensures that the magnitude of the values a feature assumes fall within a similar range. The network regards all input features to a similar extent, irrespective of the magnitude of the values they hold.

Q12: What is the cost/ loss function? Explain how it is used to improve the performance of the neural network.
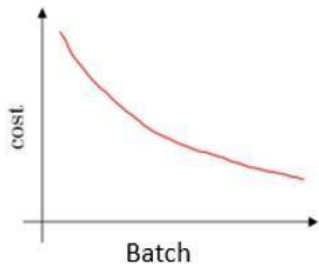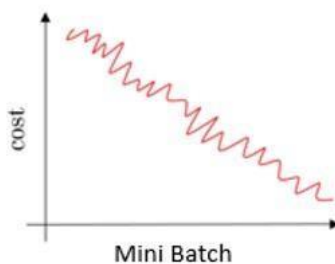
Ans12:

Loss/ cost function is a performance indicator of the neural network. It is a measure of the accuracy of the neural network with respect to a given training sample and an expected output. While training neural networks, the primary goal is to minimize the cost function by making appropriate changes to the trainable parameters of the model (weights and bias values). The steps followed to achieve this goal are:

- For each iteration of the neural network, we calculate the values of the loss function partial derivatives with respect to the trainable hyperparameters of our model

- The values of the weights are then adjusted so that the model moves in a direction opposite to the direction of increasing slope of the cost function **(using Gradient Descent)**

- With each successive epoch, we head closer to the minima of the cost function

- Training stops when the model loss reaches the minima of the loss function

Q13: What is the difference between Batch Gradient Descent and Stochastic Gradient Descent?

Ans13:

Batch Gradient Descent	Stochastic Gradient Descent
It considers all the training examples at once to take a single step.	We consider just one training example at a time to calculate the gradient and update the parameters.
Not suitable for very large datasets as the process becomes very slow as it considers the entire data for each iteration.	Suitable for very large datasets as it considers only one training example in each iteration.
Since we use all the data for computing the gradients, we move somewhat directly towards the optimum value.	Since we use just one training example at a time, the cost will not necessarily decrease with each iteration but keeps fluctuating up and down with each successive iteration.
Example of how the cost function decreases with each epoch using Batch Gradient descent: 	Example of how the cost function decreases with each epoch using Stochastic Gradient descent:

Q14: Which optimization algorithm is used in neural networks to automatically update the values of the model parameters in order to minimize the loss function?

Ans14:

Gradient Descent is an optimization algorithm, used to find the optimal set of parameters for a function, for which the function attains its global minimum. For neural networks, this function is the cost function. To achieve this objective, the algorithm follows the following steps iteratively:

- Initialize random weight and bias.

- Pass an input through the network and get values from the output layer.

- Calculate the error between the actual value and the predicted value.

- Go to each neuron which contributes to the error and then change its respective values to reduce the error.

- Reiterate until you find the best weights of the network.
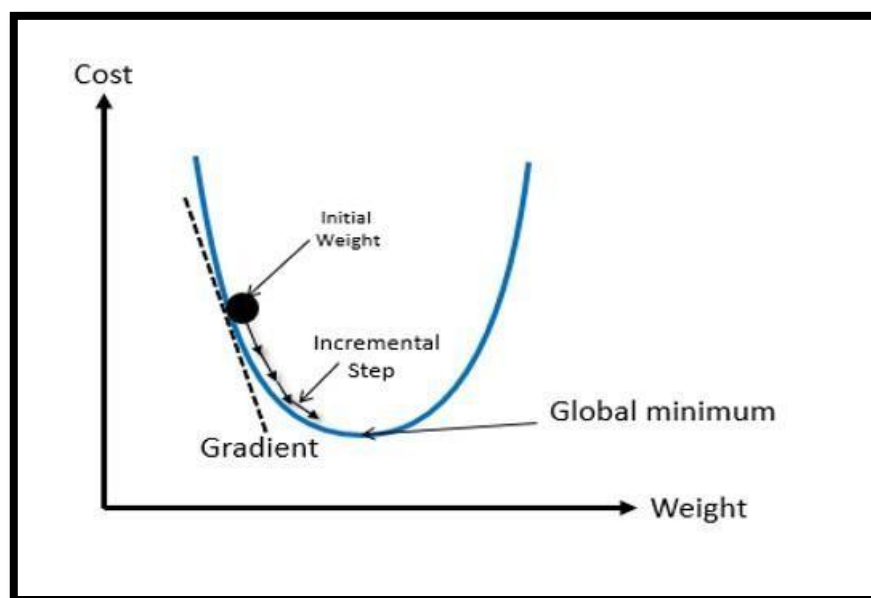
Q15: What are the different ways in which Gradient Descent is used to attain a global minimum of cost function?

Ans15:

Let us understand this with an example. Suppose there is a man who wants to trek down a valley. At each step, he takes a step forward so as to get closer to the bottom(global minima in this case). He takes the next step based on his current position and stops when he reaches the bottom, which is his aim. There are different ways in which the man(weights) can reach the bottom. The commonly used ones are:

- **Batch Gradient Descent**: Calculate the gradients for the whole dataset and perform just one update at each iteration.

- **Stochastic Gradient Descent:** Uses only a single training example to calculate the gradient and update parameters.

- **Mini Batch Gradient Descent:** Mini-batch gradient is a variation of stochastic gradient descent where instead of single training example, mini-batch of samples is used. It"s one of the most popular optimization algorithms.

Q16: How does mini-batch Gradient Descent perform better than Batch Gradient Descent and SGD?

Ans16:

In mini-batch gradient descent, we utilize the advantages of both Batch Gradient Descent and SGD. Batch Gradient Descent can be used to obtain smoother curves and converge directly to the minima. SGD can be used for huge datasets as it converges faster, but we cannot use vectorized operations as it uses just a single example at a time. This makes the computations much slower. To tackle this problem, a mixture of Batch Gradient Descent and SGD is used.

Working: We use samples of the training data at a time (batches). For example, if the dataset consists of 10000 examples, and we select a batch size of 1000 examples at a time(called mini-batch). After creating mini-batches of fixed size, we perform the following steps in one epoch:

- Randomly pick a mini-batch from the training data

- Feed it to the NN

- Calculate the mean gradient for that batch

- Use the calculated mean gradient to update the weights

- Repeat the above steps for different samples/batches The figure below makes it more clear. Notice how Batch gradient (blue) descent moves directly towards the center without many fluctuations, SGD (purple) moves towards the center with a lot of fluctuations and mini-batch gradient descent (green) moves towards the center with lesser fluctuations than SGD.
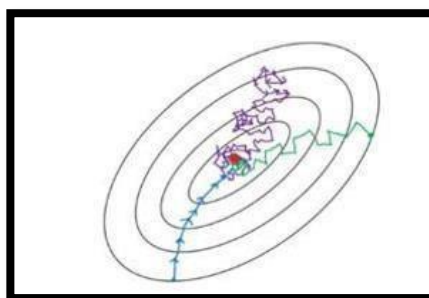
Q17: What is Backpropagation algorithm? What are its drawbacks?

Ans17:

The process by which an MLP learns is called Backpropagation. It repeatedly adjusts the weights of the connections in the neural network so as to minimize a measure of the difference between the actual output vector and the desired output vector with each successive iteration. BackProp is like "learning from mistakes". It is a supervised learning algorithm and follows these steps:

- Initially, all weights and biases are randomly assigned

- The input is fed to the net and ANN is activated

- The output is compared with the desired output and the error is calculated

- This error is propagated backward to the previous layers and the weights and biases are adjusted following gradient descent method

- This process is repeated until the error is below a predefined threshold

The drawbacks of using backpropagation are:

- **Local minima problem:** The algorithm always adjusts the weights so as to decrease the error. In this process, it might get stuck at a local minimum where the gradient will be zero, and it will stop the training process.

- **Network paralysis:** Occurs when the weights are adjusted to very large values during training, which can force most of the units to operate at extreme values, in a region where the derivative of the activation function is very small.

Q18: What is the learning rate?

Ans18:

Learning rate is a hyperparameter that defines how quickly the model moves towards the optimal set of weights and biases to achieve minimal cost.

In Gradient Descent, the updated value of weights is given by:

$\Delta W = -\square.(\partial L/\partial W)$

$W_{Updated} = W_{old} + \Delta W$

Here,

$\square$: learning rate

$(\partial L/\partial W)$: the slope of the loss function

Q19: What is an optimal value for learning rate? What are the effects of setting the learning rate to be too high or too low?

Ans19:

For Gradient Descent to perform well, it is important to set the learning rate to an appropriate value. If the learning rate is very large, you will skip the optimal solution and if it is too small you will need too many iterations to converge to the best values. An optimum learning rate is one that"s low enough so that the network converges to something useful but high enough so that it can be trained within a reasonable amount of time.

The graph below shows the effect of various learning rates on the cost function convergence:
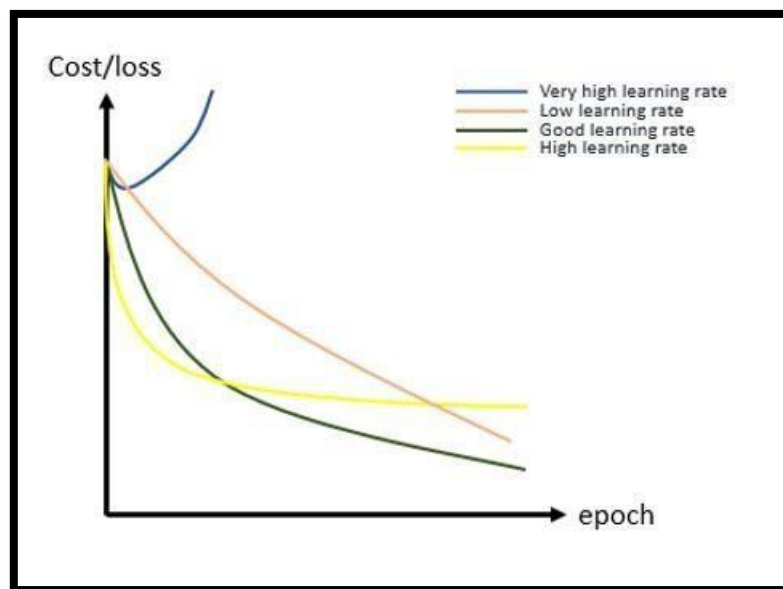

Very high or very low learning rates can lead to waste of time and resources. A lower learning rate implies more training time, which results in increasing GPU costs. A higher learning rate would result in a model that is not able to predict anything accurately.

Q20: Traditional deep learning neural networks are not able to deal with sequential data where the current value has some dependency on the values that come before it i.e, when there is a sense of ordering in the data. Which algorithm is used to deal with ordered data?

Ans20:

Traditional neural networks treat each input example independently i.e, the inputs are not related to each other and there is no sense of ordering in the data. They lose their power in applications like time series forecasting, connected handwriting recognition and speech recognition. RNN is the go-to algorithm in such cases.

An RNN (Recurrent Neural Network) is a generalization of a Feedforward NN with an additional internal "memory". RNN"s can use their internal "state" memory to process sequences of inputs. In other neural networks, all the inputs are independent of each other while in RNN,l the inputs are related to each other. The following is a diagrammatic representation of how an RNN works:
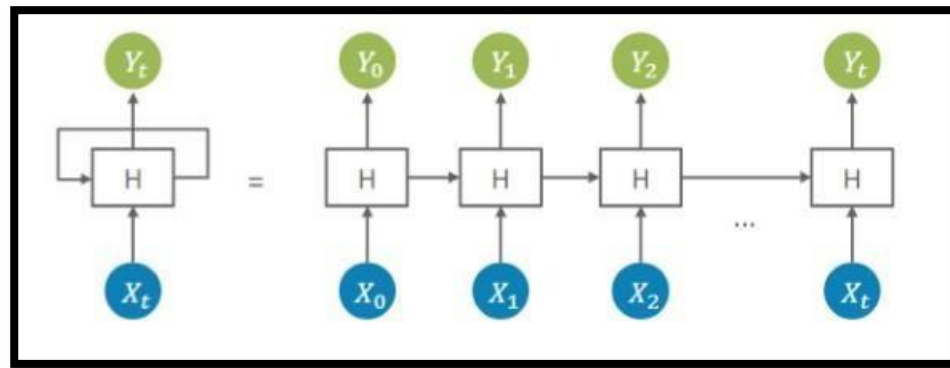
The formula for the current state can be represented as: h t = f(ht-1 , xt)

The steps followed in RNN are:

- First, it takes the X(0) from the sequence of input and generates h(0) output.
- h(0) combined with X(1) is the input for the next step. So, h(0) and X(1) are the inputs for the next step.
- Similarly, h(1) combined with X(2) is the input for the next step and so on. This way, it keeps remembering the context while training.

Q21: How does the problem of vanishing and exploding gradients affect the performance of an RNN?

Ans21:

While training an RNN, the slope can at times become either too small or too large. This makes the training process difficult. When the slope becomes too small, the problem is known as a "Vanishing Gradient." and when the slope grows exponentially instead of decaying, it"s referred to as an "Exploding Gradient."

Gradient problems lead to unacceptably long training time, poor performance, and low accuracy.

Q22: RNN is unable to learn long term dependencies in the data. What is used to combat this problem?

Ans22:

LSTM (Long Term Short Memory) has a default behavior of remembering information for long periods. It is a special kind of RNN capable of learning long-term dependencies. It resolves the problem of vanishing gradients associated with RNN"s. It is well suited to predict time series problems with unknown durations. It trains the model using backpropagation and uses 3 gates, an **input gate**, a **forget gate** and an **output gate**.

Q23: What is the difference between Feedforward neural network and backpropagation?

Ans23:

A Feed-Forward Neural Network is a type of Neural Network architecture where the connections are "fed forward", i.e. do not form cycles. The term "Feed-Forward" is also used when you input something at the input layer and it travels from input to hidden and from hidden to the output layer.

Backpropagation is a training algorithm consisting of 2 steps:

- Feed-Forward the values.

- Calculate the error and propagate it back to the earlier layers.

To be precise, forward-propagation is part of the backpropagation algorithm but comes before back-propagating.

Q24: What is the difference between Feedforward and Recurrent Neural Networks?

Ans24:

Feedforward Neural Network	Recurrent Neural Network
Signals travel only in one direction, from input towards the output.	Signals travel in both directions making it a looped network.
Considers only current input to generate the output of a layer.	Considers current input as well as previously used inputs for generating the output of a layer.
Cannot memorize previous inputs as it does not have any internal memory (eg: CNN)	Its internal memory enables it to memorize past data.

Q25: What are the techniques by which you can prevent a neural network from overfitting?

Ans25:

Some of the popular methods of avoiding overfitting while training neural networks are:

- **L1 and L2 regularizations:** Regularization involves adding an extra element to the loss function, which punishes our model for being too complex. In simple words, for using too high values in the weight matrix. By this method, we attempt to limit its flexibility and also encourage it to build solutions based on multiple features. Two popular versions of this method are Least Absolute Deviations (LAD or L1) and Least Square Errors (LS or L2).

    L1 reduces the weights associated with less important features to zero, thereby completely removing their effect. It is effectively an in-built mechanism for automatic feature selection. In **most**

cases, L1 is preferred over L2, as it does not perform well on datasets with a large number of outliers.

- **Dropout**: In this method, every unit of our neural network (except the output layer) is given a probability „p" of being ignored in the calculations. The hyperparameter „p" is called the dropout rate, and is usually set to 0.2. In each iteration, we randomly select the neurons that we drop based on the value of „p". As a result, each time we work with a smaller neural network and we are able to prevent overfitting by using a fewer number of features at each iteration.

- **Early Stopping:** Many times it is observed that till a certain number of epochs, the error on the training set and the cross-validation set decreases, but after that the error on the cross-validation set starts to increase, while that of the training set still decreases. This is where overfitting starts, and it is advisable to stop the training process after a certain number of epochs. Thus, early stopping means to stop training the model once it starts overfitting.

Q26: Many programmers prefer Deeper Networks over shallow ones. But is it always advisable to use deeper networks over shallow ones?

Ans26:

A shallow network is one that has a fewer number of hidden layers. Deeper the network (more the number of nodes and hidden layers), the better the model is able to learn. But, using a very large number of nodes can result in the model following the data too closely, resulting in overfitting. The number of hidden layers to be used, and consequently, the number of nodes depends on the size and complexity of the dataset. For example, if the classes are linearly separable or we can say if the classes do not intersect each other much, a shallow network performs better as it provides a relatively generalized output. So the optimal "depth" of the neural network largely depends on the data and there can be no general rule that deeper networks can outperform shallow ones.

Q27: How is data represented in Deep Learning? List the various types with examples?

Ans27:

A tensor is a standard way of representing data in deep learning. It is an N-dimensional array of data. For example, a tensor holding just a single scalar value is a 0-dimension tensor. A vector is a 1dimensional tensor whereas a matrix is a 2-dimensional tensor.

Tensor of dimension[1]

Tensor of dimensions[2]

Tensor of dimensions[3]
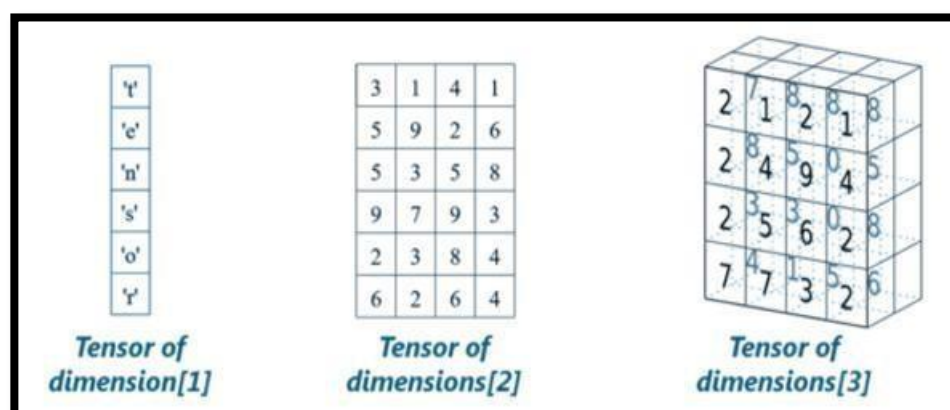
The different types of tensors commonly used in Deep Learning are:

Constant: A constant is a tensor whose value cannot be changed and remains the same. Example of constant tensors are:

a=tf.constant(5), scalar 0-dimension tensor

v=tf.constant([4,6,8]) , 2-dimension tensor

mat=tf.constant([[1,2,3],[4,5,6],[5,7,9]]) , 3.dimensional tensor

Variable: A tensor whose value can be changed as and when the program runs. It can be used to add trainable parameters to the model.

w=tf.Variable([-0.3], tf.float32)

b=tf.Variable([0.3], tf.float32)

Variables need to be initialized first in order to use them to make computations in a session. It is initialized using the **global_variables_initializer()** function.

Placeholder: It is used to provide external inputs. It is a promise to provide a value later when the session is run.

A=tf.placeholder(tf.float32)

Q28: What is a computational graph? How is it executed?

Ans28:

Tensorflow core programs consist of 2 discrete sections:

- Building a computational graph
- Running a computational graph

Building a Computational Graph: A Computational Graph can be thought of as a network of nodes, with each node known as an operation. Below is an example of a computational graph.
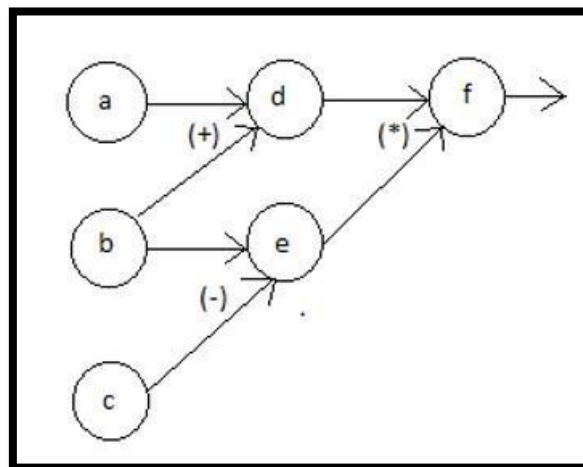
It can be implemented as:

a=tf.constant(5.0, tf.float32)

b=tf.constant(9.0, tf.float32)

c=tf.constant(7.0, tf.float32)

d=tf.add(a,b)

e=tf.subtract(b,c)

f=tf.multiply(d,e)

We have built a computational graph. Now we have to execute it in a session.

Running a Computational Graph: To evaluate a graph, we need to run it within a session. A session encapsulates the control and state of the TensorFlow runtime. It can be implemented as:

```
with tf.Session as sess:
    print(sess.run(f))
```

This will print the output as: 28

Q29: Which is the go-to algorithm for Image recognition and classification problems? What are the advantages it offers over traditional neural networks?

Ans 29:

Convolutional Neural Networks (CNN) is the go-to algorithm for any image-related problems. It can also be applied to recommender systems, NLP and more. The advantages that CNN offers over other Deep Learning methods are:

- CNN is more efficient in terms of memory and complexity

- Convolution and pooling layers of CNN help in reducing the number of parameters to tune.

- CNN is much faster than traditional NN as it extracts important information before feeding the image into fully connected layers, reducing the number of nodes.

Q30: What is the architecture of CNN? Explain the working of each layer in CNN?

Ans30:
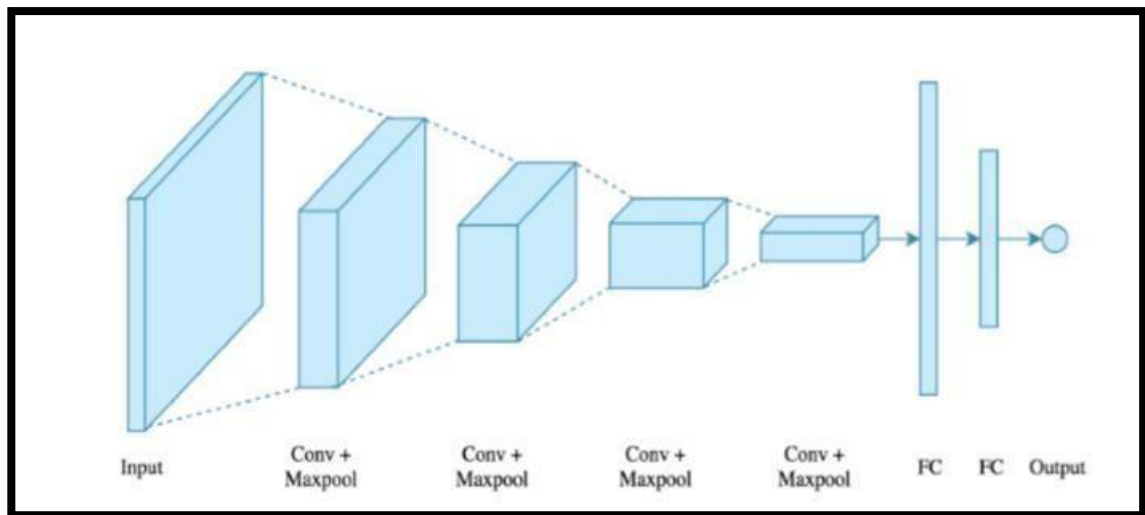
We pass an input image through a series of convolution and pooling operations, followed by a number of fully connected layers. The convolution and pooling operations are used to extract the features of interest and reduce the dimensionality of the image before it is passed to the fully connected layers which perform classification.

Input: The input image (an n-D matrix where each element of the matrix contains information about a pixel). The values of the matrix represent the pixel intensities. In most cases, an image is represented as a 3D matrix with dimensions of height, width and depth, where depth corresponds to color channels (RGB).

Convolution layer: Convolution is a mathematical operation that is used to merge two sets of information. The result of applying a convolution operation is called feature map. We can also use multiple filters to extract different types of information. We then stack all these different feature maps obtained to get the result of the convolution layer.

Pooling: Convolution is followed by Pooling, to reduce the dimensionality. Pooling helps to reduce the number of parameters, which reduces the training time and helps to combat overfitting. Pooling helps in downsampling the feature map, while keeping important information. The most common types of pooling are max pooling and average pooling.

Fully Connected Layers: We pass the output of the final pooling layer to a network of fully connected layers. We flatten the output of the final pooling layer to make it a 1-D vector which is the input to the fully connected layer. These fully connected layers work in the same way as traditional ANN"s. At the output, we generally use Softmax activation function, which is the most preferred for multiclass classification problems.

Q31: What are the advantages of using convolutional layers over fully connected layers?

Ans31:

The convolution layers are the powerhouse and the most important step in CNN. Convolution + Pooling layers perform feature extraction. For example, given an input image of a cat, it detects features such as 2 eyes, whiskers, small ears, short legs, short tail etc. The fully connected layers then act as a classifier on top of these features and assign a probability of the image being a cat.

Thus, convolution layers enable automatic feature detection and extraction. The convolution layers learn these meaningful features by building on top of each other. The first few layers detect edges, the next layers combine them to detect shapes, and the following layers merge this information to infer that this is the eye.

Another advantage is that it drastically reduces the dimension of the input vector to the fully connected layer, also reducing the number of nodes, thereby reducing the number of matrix multiplications resulting in a reduction in the training time.

Q32: Many times, we do not have a sufficient number of images for CNN to be applied. What can you do when you have limited training data?

Ans32:

Data Augmentation is the process of generating additional training data from the current set. If we have too few training instances, the model has poor regularization, leading to overfitting. Data Augmentation enriches or "augments" the training data by generating new images via random transformations of existing ones. This way we artificially boost the size of the training set, providing more information and thereby reducing overfitting. In this way, it can also be considered as a regularization technique. Data augmentation can boost the size of the training set by even 50 times the original. It"s a very powerful technique that is used in every single image-based deep learning model.

Q33: Given an image of size (n☐n☐d), filter of size (f☐f☐d), (nf) is the number of filters used, padding 'p', stride of 's', what will be the dimension of the output image?

Ans33:

Input dimension: $(n x n x d)$

Filter size: $(f x f x d)$

Number of filters: n_f

Padding: p

Stride length: s

The size of the output image will be: **$[\{(n+2p-f)/s\} +1] \times [\{(n+2p-f)/s\} +1] \times n_f$**

Q34: We at times tend to lose information at the borders of an image after a convolution operation. How would you deal with this? When we perform convolution operation, the pixels at the borders of the image are used fewer times, as compared to central pixels. We do not focus much on the corners. Also, when we apply convolution operation, the size of the image shrinks. This results in a loss of information.

Ans34:

To combat this, we can pad the image with an additional border, i.e., we add one pixel all around the edges. There are 2 common choices of padding:

- Valid: It means no padding. The output image size will shrink.
- Same: Here we apply padding, so that the output image size is same as the input image size, i.e., $(n+2p-f+1)=n$
  $\Rightarrow p=(f-1)/2$

In this way, we do not lose too much information and the image size does not shrink either.

Q35: How does pooling help in filtering only important features and reducing training time?

Ans35:

Suppose we have an image, where the object of interest has pixels of higher intensity compared to the background. In this case, the pixels of lower intensities are not of interest, and filtering them out will largely reduce the number of nodes and consequently the number of computations and the training time.

For such cases, we can use max pooling which considers only the pixel with maximum value in the window defined by it. Suppose the size of the image is 28x28, then the size of input vector to the fully connected layers would be an array of size 784. But if we apply max pooling with window of size 2x2 and a stride of 2, the image dimension will be reduced to 14x14, and the input vector will now be an array of size 196 (which is 1/4th of the size without pooling). This boosts the training time and also reduces the number of parameters to tune.

Q36: What are Autoencoders?

Ans36:

Autoencoders are feedforward neural networks where the input is the same as the output. They compress the input into a lower-dimensional code (also called latent space representation) and then reconstruct the output from this latent space representation. The code is a "compression" of the input.

The three components of autoencoder are:

- Encoder: Compresses the input and produces the code
- Code: a compact summary of the input
- Decoder: Reconstructs the input using the code
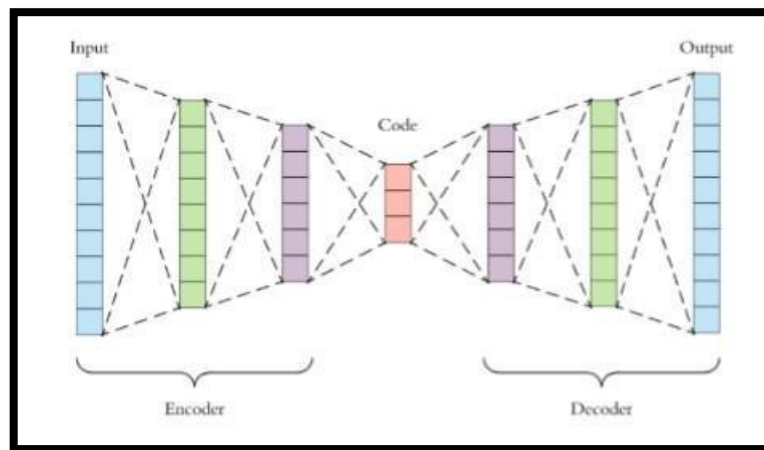
The input passes through the encoder, which is a fully connected ANN to produce the code. The decoder also has a similar ANN structure, with an architecture that is the mirror image of the encoder (this is not a requirement but is generally the case). The only requirement is that the dimensionality of the output and input should be the same.

The goal is to get an output that is identical to the input. Since the code layer has fewer number of nodes than the input, it is said to be undercomplete. In this way, it won't be able to simply copy the input to the output but will instead learn the important features.

Q37: List the various types of scenarios where autoencoders can be applied?

Ans37:

Image Coloring: Autoencoders are used for converting any grayscale image into a colored image. Depending on what is in the picture, it makes it possible to tell what the color should be.

Feature variation: It extracts only the important features of an image and generates the output by filtering out any noise or unnecessary interruption.

Dimensionality Reduction: It helps in providing a similar image with a reduced pixel value, wherein the reconstructed image is the same as our input but with reduced dimensions.

Denoising Image: The input seen by the autoencoder is a stochastically corrupted version of the raw input. A denoising autoencoder is trained to reconstruct the original input from the noisy version.

Q38: What is Bottleneck in Autoencoder and why is it used?

Ans38:

The layer between the encoder and decoder i.e., the code is also known as Bottleneck. This is a well-designed approach to decide which aspects of observed data are relevant information and what aspects can be discarded.
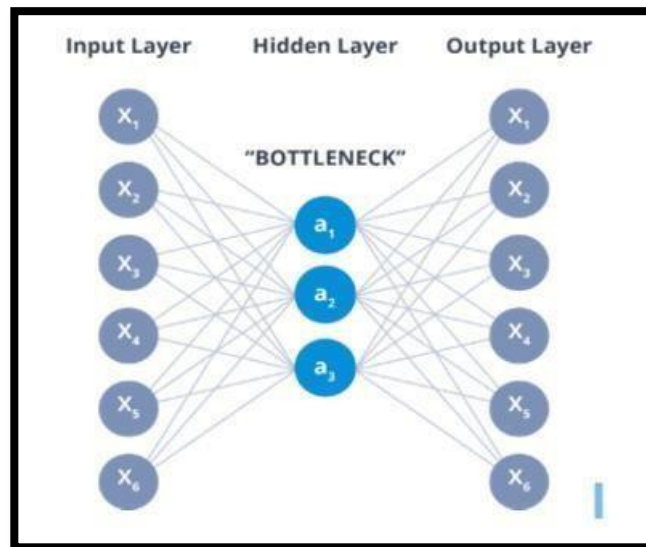
It does this by balancing two criteria:

- Compactness of representation, measured as the compressibility.

- It retains some behaviorally relevant variables from the input.

Q39: What is a Boltzmann Machine?

Ans39:

These are Stochastic (or non-deterministic) Generative deep learning models. It has only 2 types of nodes: hidden and visible. They have no output nodes (which gives them this non-deterministic feature). Unlike other traditional deep learning methods, Boltzmann machines have connections between the input nodes as well. All nodes are connected to all other nodes irrespective of whether they are input or hidden nodes. This allows them to share information and self-generate subsequent data. When the input is provided, they are able to capture all the patterns and correlations among the data.

Therefore, they are called Deep Generative models and fall under the class of Unsupervised Deep Learning.
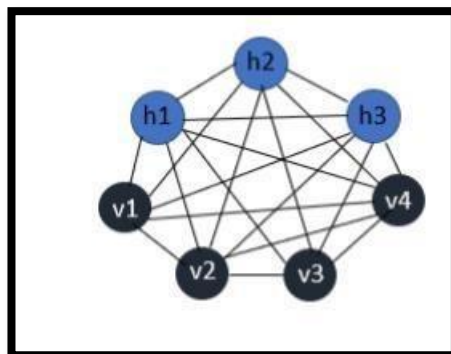
Q40: What are the differences between Boltzmann Machine and Restricted Boltzmann Machines?

Ans40:

Restricted Boltzmann Machines are a special type of Boltzmann Machine, with the restriction that every node in the visible layer is connected to every node in the hidden layer but no two nodes of the same layer are interconnected. This makes them less complicated and easier to implement as compared to Boltzmann Machines. They have the ability to learn a probability distribution over its inputs. They have generative capabilities and can be used for dimensionality reduction, classification, regression, collaborative filtering, feature learning, and topic modeling.
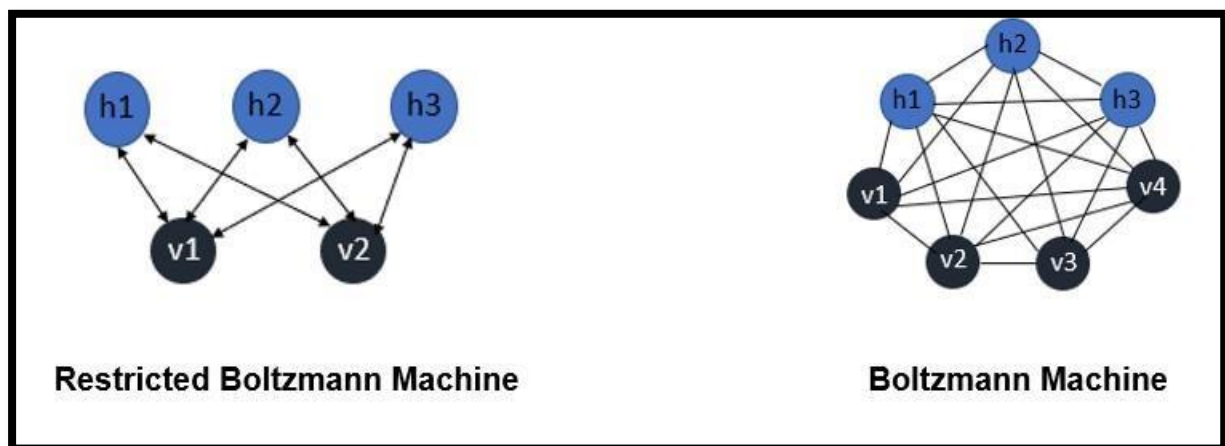
Q41: What is the limitation of CNN in object detection and which algorithm is used to overcome those limitations?

Ans41:

Suppose you are in a hurry and you can''t find your room keys as you have misplaced them in your room. CNN can only tell whether the keys are there in the room or not, but it cannot assist in finding the keys. The limitations of CNN are:

- CNN is used only to identify the class of an image, and not the spatial location of the object in the image
- It cannot be used to identify multiple objects in the image.

For multiple object detection using CNN, you will first have to divide the image into a large number of small images(regions) and look for the objects in those images. Even this fails if the object has different sizes in different images i.e, in some cases, an object might cover the entire image (dividing the image in this case divides the object, and its presence cannot be detected), and in some cases, the object covers only a small portion in the image. CNN loses its power in such cases where objects in the image can have different aspect ratios and spatial locations.

This is where **RCNN** (region-based CNN) comes to the rescue. It outputs an image with each identified object surrounded by a distinct bounding box and a certain level of precision.

Q42) What is the architecture of RCNN?

Ans42:
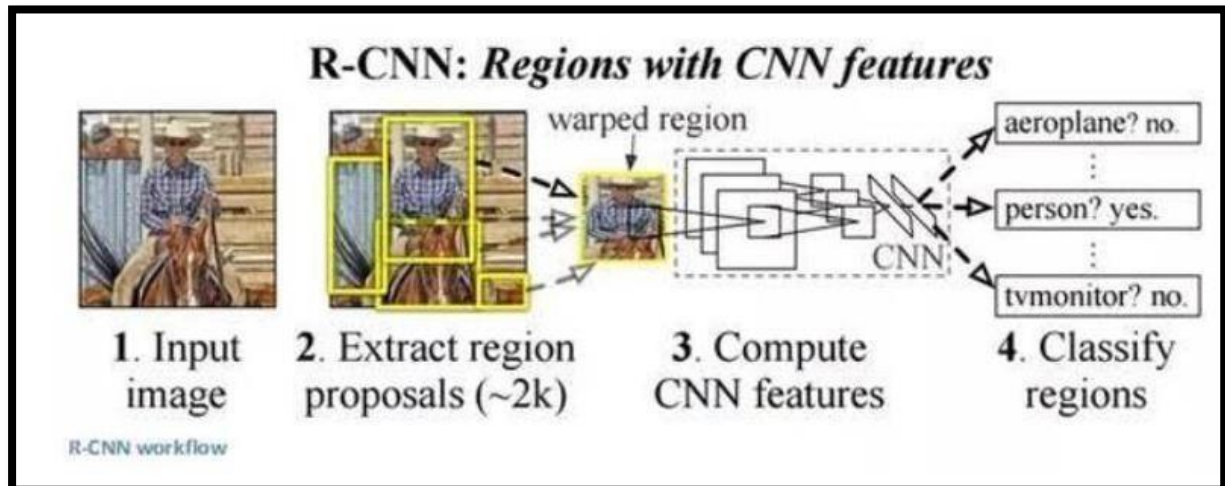
The steps followed by RCNN algorithm are:

- It first takes an image as input and extracts the ROI (regions of interest) using some proposal method eg: selective search

- All these regions are then reshaped as per the size of the input of the pre-trained CNN, and each region is passed through the ConvNet

- CNN then extracts the features for each region and then SVM is used to classify objects and backgrounds. For each class, we train one binary SVM separately

- In the final step, a Bounding Box regression (Bbox regression) is used to generate a bounding box for each identified image

Q43) How does RCNN identify the regions of interest in an image?

Ans43:

RCNN is used to identify a number of objects in the image along with their spatial locations. It outputs an image with each identified object surrounded by a distinct bounding box and a certain level of precision. RCNN uses selective search to extract these boxes from an image (these boxes are out regions of interest i.e., regions in the image that can contain an object).

There are basically four distinct regions that form an object: varying colors, texture, scale and enclosure. Selective search identifies those patterns in an image and based on that it proposes various regions. The steps followed are:

- It first takes an image as input and generates sub-segmentations so that we have multiple regions from this image
- It then combines similar regions to form larger regions. It combines regions based on color, texture, size and shape compatibility)
- Finally, these regions then produce the ROI (regions of interest i.e., the final object locations

Q44: What is the problem of using RCNN with very large datasets? What are the modifications of RCNN which make the computations faster?

Ans44:

Training an RCNN is computationally slow and expensive because of the following reasons:

- Extracting nearly 2000 regions for each image using selective search itself is a time-consuming process

- Training a CNN for each of these 2000 regions. Suppose we have N images, then the number of features for CNN will become 2000xN, which will be very large

- RCNN required 3 different models, a pre-trained CNN, a Linear SVM classifier and a regression model to tighten the bounding boxes for each identified region.

All these steps combine to make RCNN very slow and it takes around 30-40 seconds to make predictions for each new image. This makes the model cumbersome and practically impossible to build when presented with a big dataset.

The modifications of RCNN are:

- **Fast RCNN:** It uses a single model which extracts features from the regions, divides them into different regions, and returns the boundary boxes for the identified classes simultaneously
- **Faster RCNN:** It used RPN (Region Proposal Network) instead of selective search for identifying the regions of interest in the image)

Q45: Many times, while training a neural network, optimization becomes very difficult if the network gets stuck at a local minima. How would you try to overcome this situation? (asked in Amazon)

Ans45:
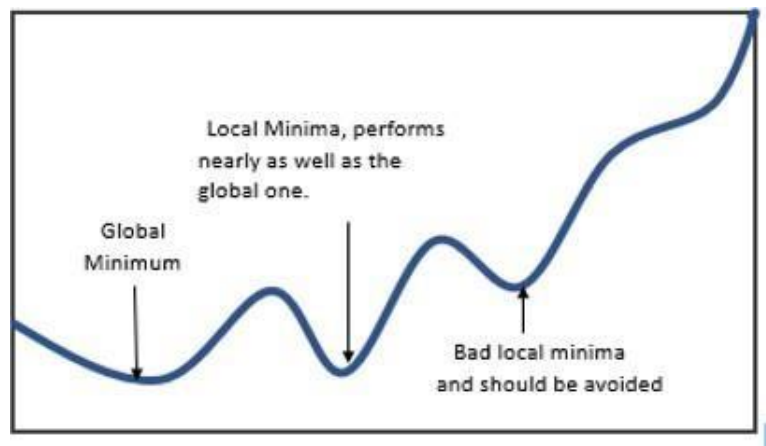
If the network is stuck at a bad local minimum (like saddle points, which is surrounded by flat regions), then it needs to be optimized. In such cases, we can tune our parameters to make our model perform better:

- **Increasing the learning rate:** If the learning rate set is too small, then it has a higher probability of being stuck at a local minima

- **Increasing the number of hidden layers:** Increasing the number of nodes will help to approximate the function better

- **Trying different combinations of activation functions**

- **Trying different optimization algorithms** like ADAM"s optimizer and RMSProp instead of the traditional gradient descent

Q46: What will happen if the activation function will be removed from a neural network? (Google interview question)

Ans46:

Without the activation function (which introduces non-linearity), the weights and bias would simply do a linear transformation. A linear equation is simple to solve but it cannot be used with complex problems. A neural network without an activation function is essentially just a linear regression. The activation function does the non-linear transformation to the input, making the network capable of learning the nonlinearities in the data. Linear transformations would never be able to perform complicated tasks like language translations and image classifications.

Q47: What do we need to use GPUs to run Deep Learning Models?

Ans47:

GPU"s help to reduce the training time of the neural networks. Deep Neural networks can contain hundreds of hidden layers and nodes and can have millions of training parameters and consequently a very large number of matrix multiplication operations. Using GPU"s allows us to perform all these operations parallelly at the same time instead of using CPU"s, which can perform only one operation at a time.

Q48: What is the difference between generative and discriminative algorithms?

Ans48:

Discriminative algorithms try to classify input data i.e, given the features of an instance of data, they predict a label or category to which that data belongs. For example, given an input email, a discriminative algorithm would predict whether the email is spam or not spam.

They can assign a label to a set of input features. If labels are represented as „Y" and features as „X", then they find $p(Y \mid X)$ (probability of Y given X), which in this case would translate to "the probability that an email is spam given the words it contains". Discriminative algorithms map features to labels.

On the other hand, the question a generative algorithm tries to answer is: Assuming that an email is spam, how likely are these features? While discriminative models model the relation between y and x, generative models care about "how you get x." They capture $p(X \mid Y)$ (the probability of x given y), or the probability of features given a label or category. While discriminative models learn the boundary between classes, generative models model the distribution of individual classes.

Q49: How do Generative Adversarial Networks work?

Ans49:

One neural network, the generator, generates new data instances, while another neural network, the discriminator, evaluates them for authenticity, i.e. the discriminator decides whether each instance of data that it reviews belongs to the actual training dataset or not. Let"s say we want to generate handwritten numerals, and we already have a dataset of handwritten digits from the real world. The goal of the discriminator, when shown an instance from the true dataset, is to recognize those that are authentic.

The generator creates new, synthetic images that it passes to the discriminator. It does so in the hopes that they will be deemed authentic, even though they are fake. The goal of the generator is to generate passable hand-written digits (to lie without being caught). The goal of the discriminator is to identify images coming from the generator as fake.
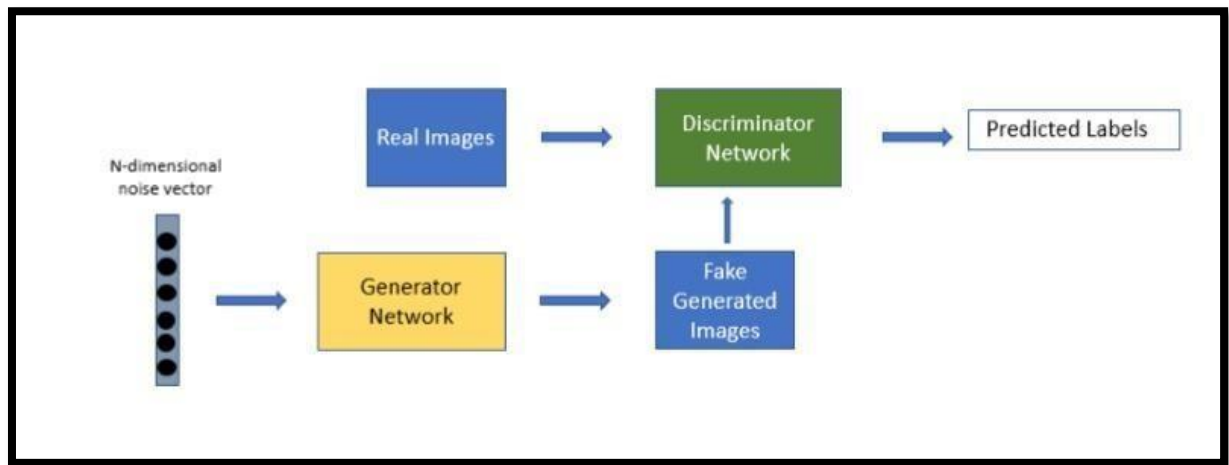
Q50: What are GAN's used for?

Ans50:

Given a training set, GAN technique learns to generate new data with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that can look authentic to human observers, having many realistic characteristics. Some of the applications of GAN"s are:

- GANs can be used to create photos of imaginary fashion models, with no need to hire a model, photographer, makeup artist.

- GAN"s can be used as a method of up-scaling low-resolution 2D textures in old video games by recreating them in 4k or higher resolutions via image training.

- GANs can be used to show how an individual's appearance might change with age.