QUESTION BANK

Multiple Choice Questions

Q1. ________ is the application of machine learning.

i. Sentimental analysis
ii. Traffic prediction
iii. Speech and face recognition
iv. All of the above

Answer: d. All of the above

Q2. From the following, _______ is not a feature of K-Nearest Neighbors (KNN).

v. KNN is simple and pretty intuitive
vi. KNN constantly evolves
vii. KNN has assumptions
viii. No Training Step

Answer: c. KNN has assumptions

Q3. Which of the following is the main goal of machine learning?

i. Enable computers to learn data
ii. To automate manual tasks
iii. To make computers intelligent
iv. To generate self-aware machines

Answer: a. Enable computer to learn data

Q4. Choose the real-world application of ML from the following.

i. Fraud detection
ii. Chatbots
iii. Digital assistants
iv. All of the above

Answer: d. All of the above

Q5. Which of the following ML algorithms can be used with unlabelled data?

i. Instance-based algorithms
ii. Regression algorithms
iii. Clustering algorithm
iv. All of the above

Answer: c. Clustering algorithms

Q6. Machine learning is a subset of _______.

i. Deep learning
ii. Data earning
iii. Artificial intelligence
iv. All of the above

Answer: c. Artificial intelligence

Q7. From the following, choose the successful applications of ML

i. Learning to recognize spoken words
ii. Learning to classify astronomical structures
iii. Learning to drive an autonomous vehicles
iv. All of the above

Answer: d. All of the above

Q8. From the following, ________ is not machine learning.

i. Rule-based interference
ii. Artificial intelligence
iii. Both a and b
iv. Neither a nor b

Answer: a. Rule-based interference

Q9. Choose a machine learning technique from the following.

i. Speech recognition and regression
ii. Genetic programming and induction learning
iii. Both a and b are correct
iv. Neither a nor b is correct

Answer: b. Genetic programming and induction learning

Q10. Which of the following elements defines the Candidate-Elimination algorithm?

i. Just a set of candidates' hypothesis
ii. Set of instances, set of candidate hypothesis
iii. Depends on the dataset
iv. Just a set of instances

Answer: b. Set of instances, set of candidate hypothesis

Q11. FIND-S algorithm ignores ______.

i. Negative
ii. Positive
iii. With positive or negative
iv. Neither positive nor negative

Answer: a. Negative

Q12. ________ is an example of stacking.

i. Voting Classifier
ii. Random Forest
iii. AdaBoost
iv. Bagged Decision Trees

Answer: a. Voting Classifier

Q13. ________ is a clustering algorithm in ML.

i. CART
ii. Expectation Maximisation
iii. Apriori
iv. Gaussian Naive Bayes

Answer: b. Expectation Maximisation

Q14. Which of the following is the most significant phase in the genetic algorithm?

i. Fitness function
ii. Selection
iii. Mutation
iv. Crossover

Answer: d. Crossover

Q15. Dimensionality reduction reduces in _______.

i. Collinearity
ii. Entropy
iii. Stochastics
iv. Performance

Answer: a. Collinearity

Q16. _______ model is a generative model used in ML

i. Naive Bayes
ii. Linear Regression
iii. Logistic Regression
iv. Support vector machines

Answer: a. Naive Bayes

Q17. Choose the invalid statement for Ensemble voting.

i. It takes non-linear combinations for learners
ii. It takes linear combinations for learners
iii. It is the easiest way to merge multiple classifiers
iv. It is also called ensembles and linear opinion pools

Answer: a. It takes non-linear combinations for learners

Q18. ML comprises learning algorithms that

i. Improve their performance
ii. Over time with experience
iii. At executing some task

iv. All of the above

Answer: d. All of the above

Q19. Data points possess negative residual if

i. The regression lines truly pass through the point
ii. They are above the regression line
iii. They are below the regression line
iv. None of the above

Answer: c. They are below the regression line

Q20. Which of the following models is trained with data only in a single batch?

i. Batch learning
ii. Offline learning
iii. Both a and b
iv. Neither a nor b

Answer: c. Both a and b

Q21. ______machine learning algorithm is associated with the idea of bagging?

i. Decision tree
ii. Classification
iii. Random forest
iv. Regression

Answer: c. Random forest

Q22. Different learning methods do not involve _______.

i. Analogy
ii. Memorization
iii. Introduction
iv. Deduction

Answer: c. Introduction

Q23. From the following, choose the evaluation metric commonly used for classification tasks in the presence of class imbalance.

i. R-squared
ii. F1-score
iii. Accuracy
iv. Mean Squared Error (MSE)

Answer: b. F1-score

Q24. ________ is not a supervised ML algorithm.

i. K-means
ii. SVM for classification problems
iii. Decision Tree
iv. Naive Bayes

Answer: a. K-means

Q25. What do we call an application of machine learning methods to large databases?

i. Big data computing
ii. Artificial intelligence
iii. Data mining
iv. Internet of Things (IoT)

Answer: c. Data mining

Q26. From the following, which is the appropriate definition of neuro software?

i. Software used by neurosurgeons
ii. Software used to examine neurons
iii. An easy and powerful neural network
iv. Both a and b

Answer: c. An easy and powerful neural network

Q27. If the ML model output does not include the target variable, the model is called ______.

i. Predictive model
ii. Descriptive model
iii. Reinforcement learning
iv. All of the above

Answer: b. Descriptive model

Q28. _______ is a supervised learning task.

i. Reinforcement learning
ii. Dimensionality reduction
iii. Clustering
iv. Classification

Answer: d. Classification

Q29. ______ algorithm is used to identify frequent itemsets in transactional databases.

i. K-Means clustering
ii. Decision Trees
iii. Support Vector Machine (SVM)
iv. Apriori algorithm

Answer: d. Apriori algorithm

Q30. Choose a valid statement with respect to bias and variance.

i. Models that underfit possess a low variance
ii. Models that overfit possess a low bias
iii. Models that underfit possess a high bias
iv. Both a and b

Answer: d. Both a and b

Short /Long Answer Questions

SHORT ANSWER QUESTION:

Q1: What is Machine Learning?

Answer:

Machine learning, at its core, is a field concerned with the development of algorithms capableof learning from data and making predictions or decisions on unseen data. This process heavily relies on various mathematical concepts and methods to achieve its goals.

Q2 : What is the main advantage of using deep learning over traditional machine learning methods for complex tasks?

Answer:
Deep learning can automatically learn hierarchical feature representations from raw data, making it highly effective for complex tasks such as image and speech recognition.

Q3: What are activation functions, and why are they critical in designing deep learning models?

Answer:
Activation functions are mathematical functions used in all modern deep neural network architectures. This occurs at the neuron level, during the process of mapping several neuron inputs into an output value being fired to neurons in the next layer.
They are crucial in deep learning models because they introduce non-linearity, which is vital to enable them to learn complex relationships and patterns in the data during training.

Q4: In the context of machine learning, regularization techniques are crucial for improving model performance and preventing overfitting. Describe and compare different regularization methods used in machine learning, including but not limited to L1 and L2 regularization, dropout, and data augmentation. For each method, discuss the underlying principles, how they modify the learning process, and their typical use cases. Provide examples of scenarios where one method might be preferred over another. Additionally, explain the potential drawbacks of each technique and how these drawbacks might be mitigated.

Answer:
Regularization techniques are essential tools in machine learning for enhancing model generalization and preventing overfitting, which occurs when a model learns noise in the training

data instead of the underlying pattern. Below, we discuss and compare several regularization methods, including L1 and L2 regularization, dropout, and data augmentation

Q5: Discuss the challenges associated with selecting an appropriate activation function for a neural network and provide insights into the factors influencing this choice.

Answer:
Selecting an appropriate activation function involves balancing considerations such as the desired output range, computational efficiency, and ability to mitigate issues like vanishing gradients. Sigmoid and tanh functions are effective for squashing outputs to a specific range but suffer from vanishing gradients, particularly in deep networks. Rectified Linear Unit (ReLU) addresses this issue by thresholding negative values, promoting sparsity, and accelerating training. However, ReLU can also suffer from dying neurons. activation is commonly used in multi-class classification tasks to produce probability distributions over the classes. The choice of activation function depends on the specific task, network architecture, and desired properties of the output.

Q6: What is Semi-supervised Machine Learning?

Answer:
Semi-supervised learning is the blend of supervised and unsupervised learning. The algorithm is trained on a mix of labeled and unlabeled data. Generally, it is utilized when we have a very small labeled dataset and a large unlabeled dataset.
In simple terms, the unsupervised algorithm is used to create clusters and by using existing labeled data to label the rest of the unlabeled data. A Semi-supervised algorithm assumes continuity assumption, cluster assumption, and manifold assumption.
It is generally used to save the cost of acquiring labeled data. For example, protein sequence classification, automatic speech recognition, and self-driving cars.

Q7: Explain the K Nearest Neighbor Algorithm.

Answer:
The K Nearest Neighbor (KNN) is a supervised learning classifier. It uses proximity to classify labels or predict the grouping of individual data points. We can use it for regression and classification. KNN algorithm is non-parametric, meaning it doesn't make an underlying assumption of data distribution.
In the KNN classifier:

- We find K-neighbors nearest to the white point. In the example below, we chose k=5.
- To find the five nearest neighbors, we calculate the Euclidean distance between the white point and the others. Then, we chose the 5 points closest to the white point.
- There are three red and two green points at K=5. Since the red has a majority, we assign a red label to it.

Q8: Which cross-validation technique would you suggest for a time-series dataset and why?

Answer:

Cross-validation is used to evaluate model performance robustly and prevent overfitting. Generally, cross-validation techniques randomly pick samples from the data and split them into train and test data sets. The number of splits is based on the K value.
For example, if the K = 5, there will be four folds for the train and one for the test. It will repeat five times to measure the model performed on separate folds.
We cannot do it with a time series dataset because it doesn't make sense to use the value from the future to forecast the value of the past. There is a temporal dependency between observations, and we can only split the data in one direction so that the values of the test dataset are after the training set.
The diagram shows that time series data k fold split is unidirectional. The blue points are the training set, the red point is the test set, and the white is unused data. As we can observe with every iteration, we are moving forward with the training set while the test set remains in front of the training set, not randomly selected.

Q9: What are the methods of reducing dimensionality?

Answer:
For dimensionality reduction, we can use feature selection or feature extraction methods.
Feature selection is a process of selecting optimal features and dropping irrelevant features. We use Filter, Wrapper, and Embedded methods to analyze feature importance and remove less important features to improve model performance.
Feature extraction transforms the space with multiple dimensions into fewer dimensions. No information is lost during the process, and it uses fewer resources to process the data. The most common extraction techniques are Linear discriminant analysis (LDA), Kernel PCA, and Quadratic discriminant analysis.

Q10: What are the assumptions of linear regression?

Answer:
Linear regression is used to understand the relation between features (X) and target (y). Before we train the model, we need to meet a few assumptions:

1. The residuals are independent
2. There is a linear relation between X independent variable and y dependent variable.
3. Constant residual variance at every level of X
4. The residuals are normally distributed.

Q11: What is the difference between supervised and unsupervised machine learning?

Answer:
Supervised learning requires training labelled data. For example, in order to do classification (a supervised learning task), you'll need to first label the data you'll use to train the model to classify data into your labelled groups. Unsupervised learning, in contrast, does not require labelling data explicitly.

Q12: Which is more important to you: model accuracy or model performance?

Answer:

Such machine learning interview questions tests your grasp of the nuances of machine learning model performance! Machine learning interview questions often look towards the details. There are models with higher accuracy that can perform worse in predictive power—how does that make sense?
Well, it has everything to do with how model accuracy is only a subset of model performance, and at that, a sometimes misleading one. For example, if you wanted to detect fraud in a massive dataset with a sample of millions, a more accurate model would most likely predict no fraud at all if only a vast minority of cases were fraud. However, this would be useless for a predictive model—a model designed to find fraud that asserted there was no fraud at all! Questions like this help you demonstrate that you understand model accuracy isn't the be-all and end-all of model performance.

LONG ANSWER QUESTIONS:

Q1: What are autoencoders? Explain the different layers of autoencoders and mention three practical usages of them?
Answer:

Autoencoders are one of the deep learning types used for unsupervised learning. There are key layers of autoencoders, which are the input layer, encoder, bottleneck hidden layer, decoder, and output.

The three layers of the autoencoder are:-

1. Encoder - Compresses the input data to an encoded representation which is typically much smaller than the input data.
2. Latent Space Representation/ Bottleneck/ Code - Compact summary of the input containing the most important features
3. Decoder - Decompresses the knowledge representation and reconstructs the data back from its encoded form. Then a loss function is used at the top to compare the input and output images. NOTE- It's a requirement that the dimensionality of the input and output be the same. Everything in the middle can be played with.

Autoencoders have a wide variety of usage in the real world. The following are some of the popular ones:

1. Transformers and Big Bird (Autoencoders is one of these components in both algorithms): Text Summarizer, Text Generator
2. Image compression
3. Nonlinear version of PCA

Q2: What is an activation function and discuss the use of an activation function? Explain three different types of activation functions?

Answer:

In mathematical terms, the activation function serves as a gate between the current neuron input and its output, going to the next level. Basically, it decides whether neurons should be activated or not. It is used to introduce non-linearity into a model.

Activation functions are added to introduce non-linearity to the network, it doesn't matter how many layers or how many neurons your net has, the output will be linear combinations of the input in the absence of activation functions. In other words, activation functions are what make a linear regression model different from a neural network. We need non-linearity, to capture more complex features and model more complex variations that simple linear models can not capture.

There are a lot of activation functions:

- Sigmoid function: f(x) = 1/(1+exp(-x))

The output value of it is between 0 and 1, we can use it for classification. It has some problems like the gradient vanishing on the extremes, also it is computationally expensive since it uses exp.

- Relu: f(x) = max(0,x)

it returns 0 if the input is negative and the value of the input if the input is positive. It solves the problem of vanishing gradient for the positive side, however, the problem is still on the negative side. It is fast because we use a linear function in it.

- Leaky ReLU:

F(x)= ax, x<0 F(x)= x, x>=0

It solves the problem of vanishing gradient on both sides by returning a value “a” on the negative side and it does the same thing as ReLU for the positive side.

- Softmax: it is usually used at the last layer for a classification problem because it returns a set of probabilities, where the sum of them is 1. Moreover, it is compatible with cross-entropy loss, which is usually the loss function for classification problems.

Q3: You are using a deep neural network for a prediction task. After training your model, you notice that it is strongly overfitting the training set and that the performance on the test isn’t good. What can you do to reduce overfitting?

To reduce overfitting in a deep neural network changes can be made in three places/stages: The input data to the network, the network architecture, and the training process:

1. The input data to the network:

- Check if all the features are available and reliable

- Check if the training sample distribution is the same as the validation and test set distribution. Because if there is a difference in validation set distribution then it is hard for the model to predict as these complex patterns are unknown to the model.
- Check for train / valid data contamination (or leakage)
- The dataset size is enough, if not try data augmentation to increase the data size
- The dataset is balanced

2. Network architecture:

- Overfitting could be due to model complexity. Question each component:
    - can fully connect layers be replaced with convolutional + pooling layers?
    - what is the justification for the number of layers and number of neurons chosen? Given how hard it is to tune these, can a pre-trained model be used?
    - Add regularization - lasso (l1), ridge (l2), elastic net (both)
- Add dropouts
- Add batch normalization

3. The training process:

- Improvements in validation losses should decide when to stop training. Use callbacks for early stopping when there are no significant changes in the validation loss and restore_best_weights.

Q4: Why should we use Batch Normalization?

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch.

Usually, a dataset is fed into the network in the form of batches where the distribution of the data differs for every batch size. By doing this, there might be chances of vanishing gradient or exploding gradient when it tries to backpropagate. In order to combat these issues, we can use BN (with irreducible error) layer mostly on the inputs to the layer before the activation function in the previous layer and after fully connected layers.

Batch Normalisation has the following effects on the Neural Network:

1. Robust Training of the deeper layers of the network.
2. Better covariate-shift proof NN Architecture.
3. Has a slight regularisation effect.
4. Centred and Controlled values of Activation.
5. Tries to Prevent exploding/vanishing gradient.
6. Faster Training/Convergence to the minimum loss function

Q5: How to know whether your model is suffering from the problem of Exploding Gradients?

By taking incremental steps towards the minimal value, the gradient descent algorithm aims to minimize the error. The weights and biases in a neural network are updated using these processes. However, at times, the steps grow excessively large, resulting in increased updates to weights and bias terms to the point where the weights overflow (or become NaN, that is, Not a Number). An exploding gradient is the result of this, and it is an unstable method.

There are some subtle signs that you may be suffering from exploding gradients during the training of your network, such as:

1. The model is unable to get traction on your training data (e g. poor loss).
2. The model is unstable, resulting in large changes in loss from update to update.
3. The model loss goes to NaN during training.

If you have these types of problems, you can dig deeper to see if you have a problem with exploding gradients. There are some less subtle signs that you can use to confirm that you have exploding gradients:

1. The model weights quickly become very large during training.
2. The model weights go to NaN values during training.
3. The error gradient values are consistently above 1.0 for each node and layer during training.

Q6: Can you name and explain a few hyperparameters used for training a neural network?

Answer:

Hyperparameters are any parameter in the model that affects the performance but is not learned from the data unlike parameters (weights and biases), the only way to change it is manually by the user.

1. Number of nodes: number of inputs in each layer.
2. Batch normalization: normalization/standardization of inputs in a layer.
3. Learning rate: the rate at which weights are updated.
4. Dropout rate: percent of nodes to drop temporarily during the forward pass.
5. Kernel: matrix to perform dot product of image array with
6. Activation function: defines how the weighted sum of inputs is transformed into outputs (e.g. tanh, sigmoid, softmax, Relu, etc)
7. Number of epochs: number of passes an algorithm has to perform for training
8. Batch size: number of samples to pass through the algorithm individually. E.g. if the dataset has 1000 records and we set a batch size of 100 then the dataset will be divided into 10 batches which will be propagated to the algorithm one after another.

9. Momentum: Momentum can be seen as a learning rate adaptation technique that adds a fraction of the past update vector to the current update vector. This helps damps oscillations and speed up progress towards the minimum.
10. Optimizers: They focus on getting the learning rate right.

- Adagrad optimizer: Adagrad uses a large learning rate for infrequent features and a smaller learning rate for frequent features.
- Other optimizers, like Adadelta, RMSProp, and Adam, make further improvements to fine-tuning the learning rate and momentum to get to the optimal weights and bias. Thus getting the learning rate right is key to well-trained models.

11. Learning Rate: Controls how much to update weights & bias (w+b) terms after training on each batch. Several helpers are used to getting the learning rate right.

Q7: Can you explain the parameter sharing concept in deep learning?

Answer: Parameter sharing is the method of sharing weights by all neurons in a particular feature map. Therefore, helps to reduce the number of parameters in the whole system, making it computationally cheap. It basically means that the same parameters will be used to represent different transformations in the system. This basically means the same matrix elements may be updated multiple times during backpropagation from varied gradients. The same set of elements will facilitate transformations at more than one layer instead of those from a single layer as conventional. This is usually done in architectures like Siamese that tend to have parallel trunks trained simultaneously. In that case, using shared weights in a few layers(usually the bottom layers) helps the model converge better. This behavior, as observed, can be attributed to more diverse feature representations learned by the system. Since neurons corresponding to the same features are triggered in varied scenarios. Helps to model to generalize better.

Note that sometimes the parameter sharing assumption may not make sense. This is especially the case when the input images to a ConvNet have some specific centered structure, where we should expect, for example, that completely different features should be learned on one side of the image than another.

One practical example is when the input is faces that have been centered in the image. You might expect that different eye-specific or hair-specific features could (and should) be learned in different spatial locations. In that case, it is common to relax the parameter sharing scheme, and instead, simply call the layer a Locally-Connected Layer.

Q8: Describe the architecture of a typical Convolutional Neural Network (CNN)?

Answer:

In a typical CNN architecture, a few convolutional layers are connected in a cascade style. Each convolutional layer is followed by a Rectified Linear Unit (ReLU) layer or other activation function, then a pooling layer*, then one or more convolutional layers (+ReLU), then another pooling layer.

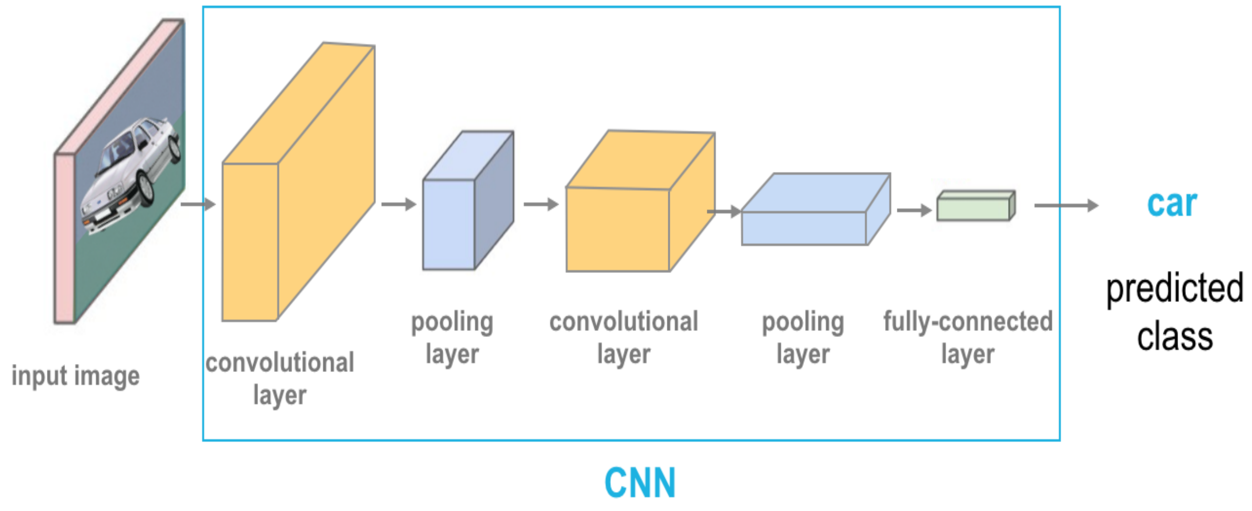
The output from each convolution layer is a set of objects called feature maps, generated by a single kernel filter. The feature maps are used to define a new input to the next layer. A common trend is to keep on increasing the number of filters as the size of the image keeps dropping as it passes through the Convolutional and Pooling layers. The size of each kernel filter is usually 3×3 kernel because it can extract the same features which extract from large kernels and faster than them.

After that, the final small image with a large number of filters(which is a 3D output from the above layers) is flattened and passed through fully connected layers. At last, we use a softmax layer with the required number of nodes for classification or use the output of the fully connected layers for some other purpose depending on the task.

The number of these layers can increase depending on the complexity of the data and when they increase you need more data. Stride, Padding, Filter size, Type of Pooling, etc all are Hyperparameters and need to be chosen (maybe based on some previously built successful models)

*Pooling: it is a way to reduce the number of features by choosing a number to represent its neighbor. And it has many types max-pooling, average pooling, and global average.

- Max pooling: it takes the max number of window 2×2 as an example and represents this window by using the max number in it then slides on the image to make the same operation.
- Average pooling: it is the same as max-pooling but takes the average of the window.



Q9: What is the Vanishing Gradient Problem in Artificial Neural Networks and How to fix it?

Answer:

The vanishing gradient problem is encountered in artificial neural networks with gradient-based learning methods and backpropagation. In these learning methods, each of the weights of the neural network receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training. Sometimes when gradients become vanishingly small, this prevents the weight to change value.

When the neural network has many hidden layers, the gradients in the earlier layers will become very low as we multiply the derivatives of each layer. As a result, learning in the earlier layers becomes very slow. **This can cause the neural network to stop learning**. This problem of vanishing gradient descent happens when training neural networks with many layers because the gradient diminishes dramatically as it propagates backward through the network.

Some ways to fix it are:

1. Use skip/residual connections.
2. Using ReLU or Leaky ReLU over sigmoid and tanh activation functions.
3. Use models that help propagate gradients to earlier time steps like in GRUs and LSTMs.

Q10: When it comes to training an artificial neural network, what could be the reason why the loss doesn't decrease in a few epochs?

Answer:

Some of the reasons why the loss doesn't decrease after a few Epochs are:

a) The model is under-fitting the training data.

b) The learning rate of the model is large.

c) The initialization is not proper (like all the weights initialized with 0 doesn't make the network learn any function)

d) The Regularization hyper-parameter is quite large.

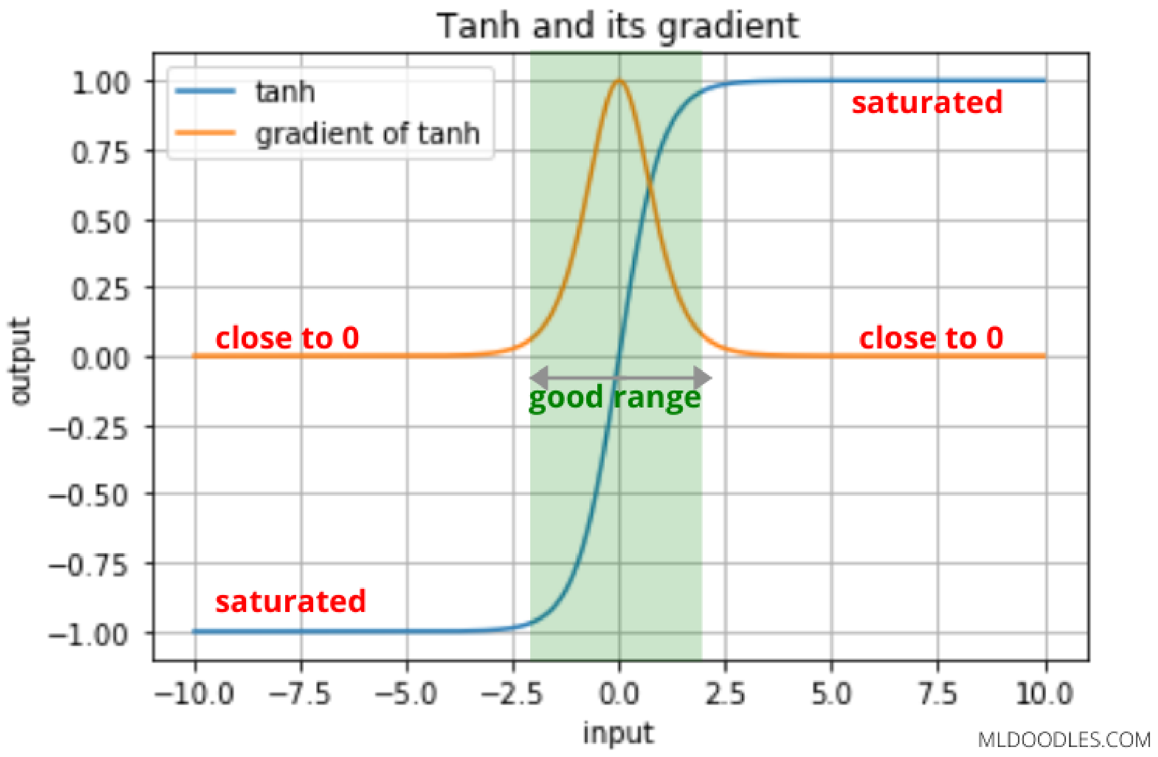
e). The classic case of vanishing gradients

Q11: Why Sigmoid or Tanh is not preferred to be used as the activation function in the hidden layer of the neural network?

Answer:

A common problem with Tanh or Sigmoid functions is that they saturate. Once saturated, the learning algorithms cannot adapt to the weights and enhance the performance of the model. Thus, Sigmoid or Tanh activation functions prevent the neural network from learning effectively leading to a vanishing gradient problem. The vanishing gradient problem can be addressed with the use of Rectified Linear Activation Function (ReLu) instead of sigmoid and

Tanh.



Q12: Discuss in what context it is recommended to use transfer learning and when it is not.

Answer:

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point for computer vision and natural language processing tasks given the vast computing and time resources required to develop neural network models on these problems and from the huge jumps in a skill that they provide on related problems.

Transfer learning is used for tasks where the data is too little to train a full-scale model from the beginning. In transfer learning, well-trained, well-constructed networks are used which have learned over large sets and can be used to boost the performance of a dataset.

Transfer Learning can be used in the following cases:

1. The downstream task has a very small amount of data available, then we can try using pre-trained model weights by switching the last layer with new layers which we will train.
2. In some cases, like in vision-related tasks, the initial layers have a common behavior of detecting edges, then a little more complex but still abstract features and so on which is common in all vision tasks, and hence a pre-trained model's initial layers can be used directly. The same thing holds for Language Models too, for example, a model trained in

a large Hindi corpus can be transferred and used for other Indo-Aryan Languages with low resources available.

Cases when transfer Learning should not be used:

1. The first and most important is the "COST". So is it cost-effective or we can have a similar performance without using it.
2. The pre-trained model has no relation to the downstream task.
3. If the latency is a big constraint (Mostly in NLP) then transfer learning is not the best option. However Now with the TensorFlow lite kind of platform and Model Distillation, Latency is not a problem anymore.

Q13: Discuss the vanishing gradient in RNN and How they can be solved.

Answer:

In Sequence to Sequence models such as RNNs, the input sentences might have long-term dependencies for example we might say "The boy who was wearing a red t-shirt, blue jeans, black shoes, and a white cap and who lives at ... and is 10 years old ...... etc, is genius" here the verb (is) in the sentence depends on the (boy) i.e if we say (The boys, ..., are genius". When training an RNN we do backward propagation both through layers and backward through time. Without focusing too much on mathematics, during backward propagation we tend to multiply gradients that are either > 1 or < 1, if the gradients are < 1 and we have about 100 steps backward in time then multiplying 100 numbers that are < 1 will result in a very very tiny gradient causing no change in the weights as we go backward in time (0.1 * 0.1 * 0.1 * .... a 100 times = 10^(-100)) such that in our previous example the word "is" doesn't affect its main dependency the word "boy" during learning the meanings of the word due to the long description in between.

Models like the Gated Recurrent Units (GRUs) and the Long short-term memory (LSTMs) were proposed, the main idea of these models is to use gates to help the network determine which information to keep and which information to discard during learning. Then Transformers were proposed depending on the self-attention mechanism to catch the dependencies between words in the sequence.

Q14: What are the main gates in LSTM and what are their tasks?

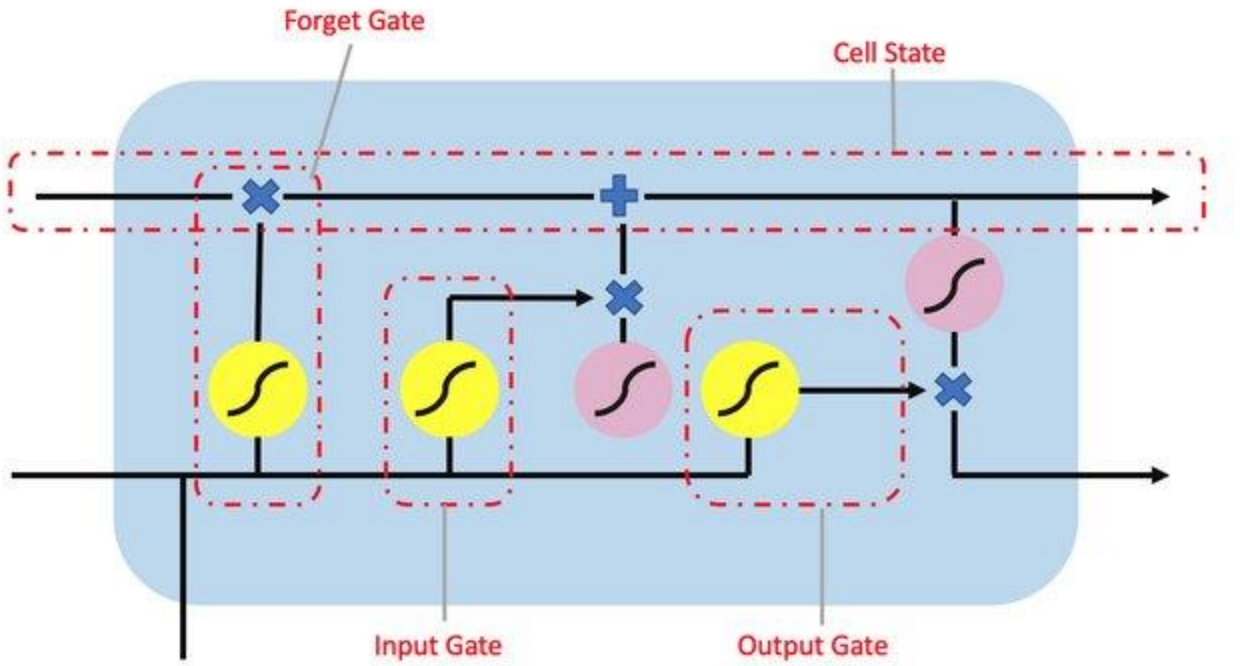
Answer: There are 3 main types of gates in a LSTM Model, as follows:

- Forget Gate
- Input/Update Gate
- Output Gate

1. Forget Gate:- It helps in deciding which data to keep or thrown out
2. Input Gate:- it helps in determining whether new data should be added in long term memory cell given by previous hidden state and new input data

3. Output Gate:- this gate gives out the new hidden state

Common things for all these gates are they all take take inputs as the current temporal state/input/word/observation and the previous hidden state output and sigmoid activation is mostly used in all of these.



Q15: Is it a good idea to use CNN to classify 1D signal?

Answer: For time-series data, where we assume temporal dependence between the values, then convolutional neural networks (CNN) are one of the possible approaches. However, the most popular approach to such data is to use recurrent neural networks (RNN), but you can alternatively use CNNs, or a hybrid approach (quasi-recurrent neural networks, QRNN).

With **CNN**, you would use sliding windows of some width, that would look at certain (learned) patterns in the data, and stack such windows on top of each other, so that higher-level windows would look for patterns within the lower-level patterns. Using such sliding windows may be helpful for finding things such as repeating patterns within the data. One drawback is that it doesn't consider the temporal or sequential aspect of the 1D signals, which can be very important for prediction.

With **RNN**, you would use a cell that takes as input the previous hidden state and current input value, to return output and another hidden form, so the information flows via the hidden states and takes into account the temporal dependencies.

QRNN layers mix both approaches.

Q16: How does L1/L2 regularization affect a neural network?

Answer:

Overfitting occurs in more complex neural network models (many layers, many neurons) and the complexity of the neural network can be reduced by using L1 and L2 regularization as well as dropout , Data augmentation and Dropout. L1 regularization forces the weight parameters to become zero. L2 regularization forces the weight parameters towards zero (but never exactly zero|| weight deccay)

Smaller weight parameters make some neurons neglectable therefore neural network becomes less complex and less overfitting.

Regularization has the following benefits:

- Reducing the variance of the model over unseen data.
- Makes it feasible to fit much more complicated models without overfitting.
- Reduces the magnitude of weights and biases.
- L1 learns sparse models that is many weights turn out to be 0.

Q17: How would you change a pre-trained neural network from classification to regression?

Answer: Using transfer learning where we can use our knowledge about one task to do another. First set of layers of a neural network are usually feature extraction layers and will be useful for all tasks with the same input distribution. So, we should replace the last fully connected layer and SoftMax responsible for classification with one neuron for regression-or fully connected-layer for correction then one neuron for regression.

We can optionally freeze the first set of layers if we have few data or to converge fast. Then we can train the network with the data we have and using the suitable loss for the regression problem, making use of the robust feature extraction -first set of layers- of a pre-trained model on huge data.

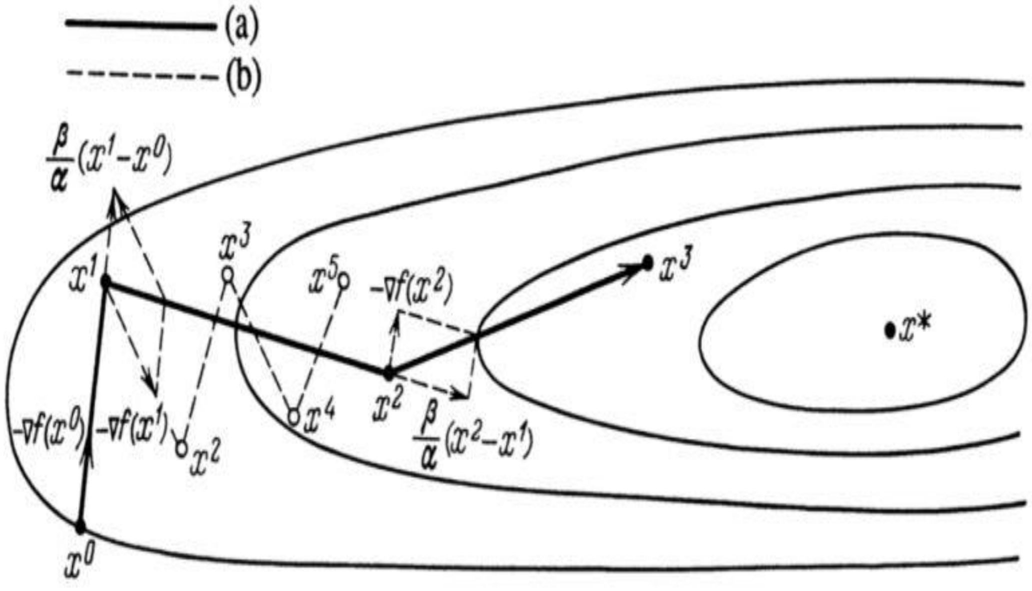
Q18: What might happen if you set the momentum hyperparameter too close to 1 (e.g., 0.9999) when using an SGD optimizer?

Answer:

If the momentum hyperparameter is set too close to 1 (e.g., 0.99999) when using an SGD optimizer, then the algorithm will likely pick up a lot of speed, hopefully moving roughly toward the global minimum, but its momentum will carry it right past the minimum.

Then it will slow down and come back, accelerate again, overshoot again, and so on. It may oscillate this way many times before converging, so overall it will take much longer to converge than with a smaller momentum value.

Also since the momentum is used to update the weights based on an "exponential moving average" of all the previous gradients instead of the current gradient only, this in some sense, combats the instability of the gradients that comes with stochastic gradient descent, the higher the momentum term, the stronger the influence of previous gradients to the current optimization step (with the more recent gradients having even stronger influence), setting a momentum term close to 1, will result in a gradient that is almost a sum of all the previous gradients basically, which might result in an exploding gradient scenario.



Q19: What are the hyperparameters that can be optimized for the batch normalization layer?

Answer: The γ and β hyperparameters for the batch normalization layer are learned end to end by the network. In batch-normalization, the outputs of the intermediate layers are normalized to have a mean of 0 and standard deviation of 1. Rescaling by γ and shifting by β helps us change the mean and standard deviation to other values.

Q20: What is the effect of dropout on the training and prediction speed of your deep learning model?

Answer: Dropout is a regularization technique, which zeroes down some weights and scales up the rest of the weights by a factor of 1/(1-p). Let's say if Dropout layer is initialized with p=0.5, that means half of the weights will zeroed down, and rest will be scaled by a factor of 2. This layer is only enabled during training and is disabled during validation and testing. Hence validation and testing is faster. The reason why it works only during training is, we want to reduce the complexity of the model so that model doesn't overfit. Once the model is trained, it doesn't make sense to keep that layer enabled.

Q21: What is the advantage of deep learning over traditional machine learning?

Answer:

Deep learning offers several advantages over traditional machine learning approaches, including:

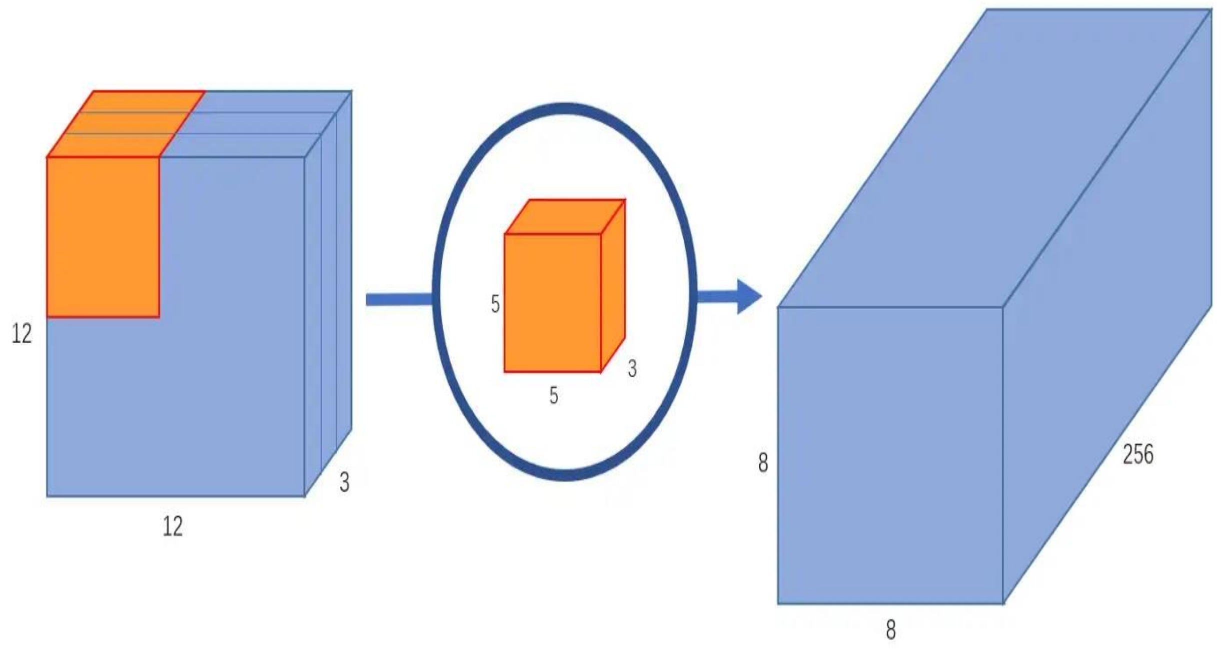
1. Ability to process large amounts of data: Deep learning models can analyze and process massive amounts of data quickly and accurately, making it ideal for tasks such as image recognition or natural language processing.
2. Automated feature extraction: In traditional machine learning, feature engineering is a crucial step in the model building process. Deep learning models, on the other hand, can automatically learn and extract features from the raw data, reducing the need for human intervention.
3. Better accuracy: Deep learning models have shown to achieve higher accuracy levels in complex tasks such as speech recognition and image classification when compared to traditional machine learning models.
4. Adaptability to new data: Deep learning models can adapt and learn from new data, making them suitable for use in dynamic and ever-changing environments.

While deep learning does have its advantages, it also has some limitations, such as requiring large amounts of data and computational resources, making it unsuitable for some applications.

Q22: What is a depthwise Separable layer and what are its advantages?

Answer:

Standard neural network Convolution layers involve a lot of multiplications that make them unsuitable for deployment.

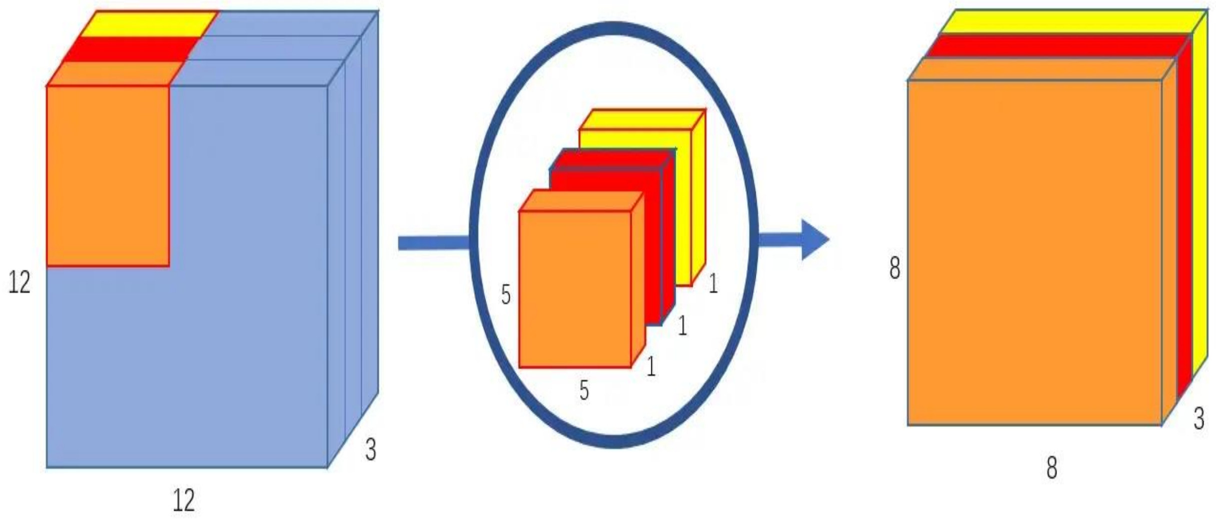


In this above scenario, we have an input image of 12x12x3 pixels and we apply a 5x5 convolution (no padding, stride = 1). We stack 256 such kernels so that we get an output of dimensions 8x8x256.

Here, there are 256 5x5x3 kernels that move 8x8 times which leads to 256x3x5x5x8x8 = 1,28,800 multiplications.

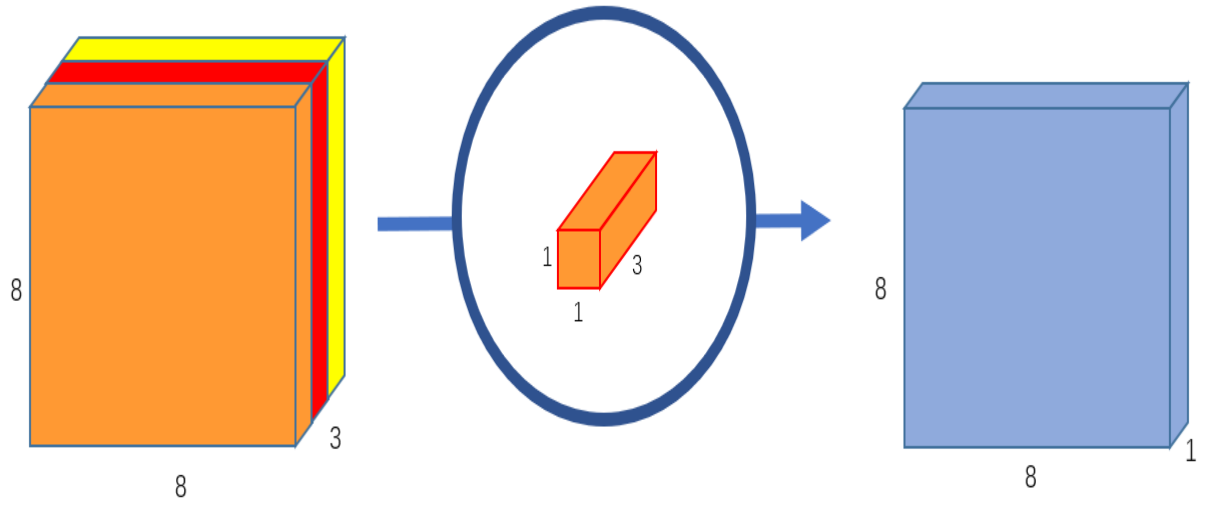
Depth wise separable convolution separates this process into two parts: a depth wise convolution and a pointwise convolution.

In depth wise convolution, we apply a kernel parallelly to each channel of the image.

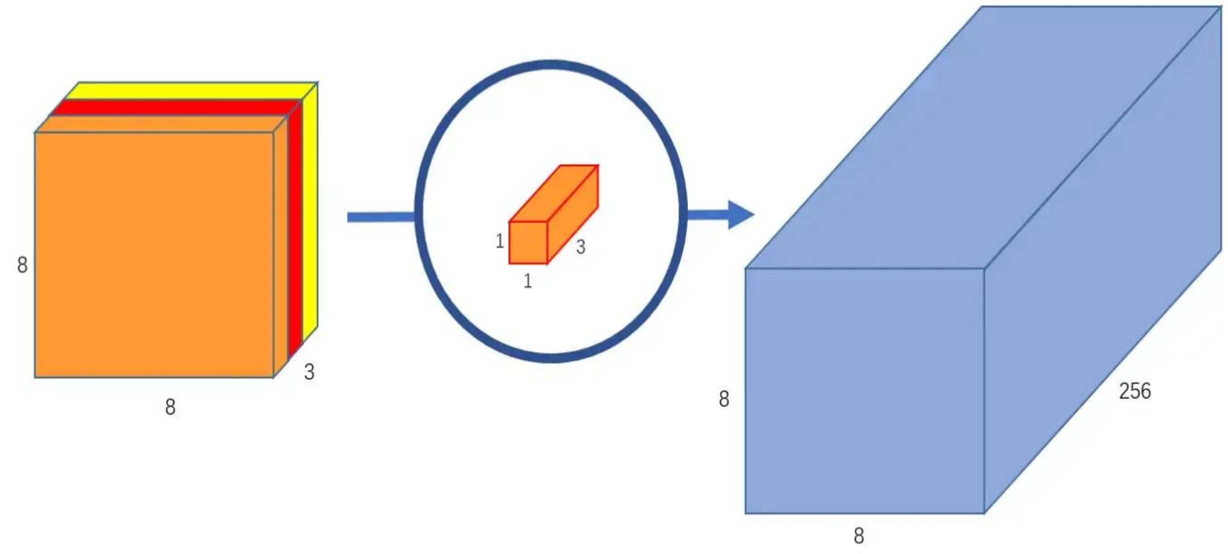


We end up getting 3 different outputs (representing 3 channels of the image) to get an 8x8x1 image. These are stacked together to form a 8x8x3 image.

Pointwise Convolution now converts this 8x8x3 image input from the depthwise convolution back to an 8x8x1 output.



Stacking 256 1x1x3 kernels give us the final output as the standard convolution.



Total Number of multiplications:

For Depthwise convolution, we have 3 5x5x1 kernels moving 8x8 times, totalling 3x5x5x8x8=4800 multiplications.

In Pointwise convolution, we have 256 1x1x3 kernels moving 8x8 times, which is a total of 256x1x1x3x8x8=49152 multiplications.

Total number of multiplications = 4800 + 49152 = 53952 multiplications which is way lower than the standard convolution case.

Reference: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

Natural Language Processing

Q23: What is transformer architecture, and why is it widely used in natural language processing tasks?

Answer: The key components of a transformer architecture are as follows:

1. Encoder: The encoder processes the input sequence, such as a sentence or a document, and transforms it into a set of representations that capture the contextual information of each input element. The encoder consists of multiple identical layers, each containing a self-attention mechanism and position-wise feed-forward neural networks. The self-attention mechanism allows the model to attend to different parts of the input sequence while encoding it.
2. Decoder: The decoder takes the encoded representations generated by the encoder and generates an output sequence. It also consists of multiple identical layers, each containing

a self-attention mechanism and additional cross-attention mechanisms. The cross-attention mechanisms enable the decoder to attend to relevant parts of the encoded input sequence when generating the output.

3. Self-Attention: Self-attention is a mechanism that allows the transformer to weigh the importance of different elements in the input sequence when generating representations. It computes attention scores between each element and every other element in the sequence, resulting in a weighted sum of the values. This process allows the model to capture dependencies and relationships between different elements in the sequence.
4. Positional Encoding: Transformers incorporate positional encoding to provide information about the order or position of elements in the input sequence. This encoding is added to the input embeddings and allows the model to understand the sequential nature of the data.
5. Feed-Forward Networks: Transformers utilize feed-forward neural networks to process the representations generated by the attention mechanisms. These networks consist of multiple layers of fully connected neural networks with activation functions, enabling non-linear transformations of the input representations.

The transformer architecture is widely used in NLP tasks due to several reasons:

Self-Attention Mechanism: Transformers leverage a self-attention mechanism that allows the model to focus on different parts of the input sequence during processing. This mechanism enables the model to capture long-range dependencies and contextual information efficiently, making it particularly effective for tasks that involve understanding and generating natural language.

Parallelization: Transformers can process the elements of a sequence in parallel, as opposed to recurrent neural networks (RNNs) that require sequential processing. This parallelization greatly accelerates training and inference, making transformers more computationally efficient.

Scalability: Transformers scale well with the length of input sequences, thanks to the self-attention mechanism. Unlike RNNs, transformers do not suffer from the vanishing or exploding gradient problem, which can hinder the modeling of long sequences. This scalability makes transformers suitable for tasks that involve long texts or documents.

Transfer Learning: Transformers have shown great success in pre-training and transfer learning. Models like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) are pre-trained on massive amounts of text data, enabling them to learn rich representations of language. These pre-trained models can then be fine-tuned on specific downstream tasks with comparatively smaller datasets, leading to better generalization and improved performance.

Contextual Understanding: Transformers excel in capturing the contextual meaning of words and sentences. By considering the entire input sequence simultaneously, transformers can generate more accurate representations that incorporate global context, allowing for better language understanding and generation.

Q24: Explain the key components of a transformer model.

Answer:

A transformer model consists of several key components that work together to process and generate representations for input sequences. The main components of a transformer model are as follows:

- Encoder: The encoder is responsible for processing the input sequence and generating representations that capture the contextual information of each element. It consists of multiple identical layers, typically stacked on top of each other. Each layer contains two sub-layers: a self-attention mechanism and a position-wise feed-forward neural network.
    - Self-Attention Mechanism: This mechanism allows the model to attend to different parts of the input sequence while encoding it. It computes attention scores between each element and every other element in the sequence, resulting in a weighted sum of values. This process allows the model to capture dependencies and relationships between different elements.
    - Position-wise Feed-Forward Neural Network: After the self-attention mechanism, a feed-forward neural network is applied to each position separately. It consists of fully connected layers with activation functions, enabling non-linear transformations of the input representations.
- Decoder: The decoder takes the encoded representations generated by the encoder and generates an output sequence. It also consists of multiple identical layers, each containing sub-layers such as self-attention, cross-attention, and position-wise feed-forward networks.
    - Self-Attention Mechanism: Similar to the encoder, the decoder uses self-attention to attend to different parts of the decoded sequence while generating the output. It allows the decoder to consider the previously generated elements in the output sequence when generating the next element.
    - Cross-Attention Mechanism: In addition to self-attention, the decoder employs cross-attention to attend to relevant parts of the encoded input sequence. It allows the decoder to align and extract information from the encoded sequence when generating the output.
- Self-Attention and Cross-Attention: These attention mechanisms are fundamental components of the transformer architecture. They enable the model to weigh the importance of different elements in the input and output sequences when generating representations. Attention scores are computed by measuring the compatibility between elements, and the weighted sum of values is used to capture contextual dependencies.
- Positional Encoding: Transformers incorporate positional encoding to provide information about the order or position of elements in the input sequence. It is added to the input embeddings and allows the model to understand the sequential nature of the data.
- Residual Connections and Layer Normalization: Transformers employ residual connections and layer normalization to facilitate the flow of information and improve gradient propagation. Residual connections enable the model to capture both high-level and low-level features, while layer normalization normalizes the inputs to each layer, improving the stability and performance of the model.

These components collectively enable the transformer model to process and generate representations for input sequences in an efficient and effective manner. The self-attention mechanisms, along with the feed-forward networks and positional encoding, allow the model to capture long-range dependencies, handle the parallel processing, and generate high-quality representations, making transformers highly successful in natural language processing tasks.

Q25: What is self-attention, and how does it work in transformers?

Answer:

Self-attention is a mechanism that allows a Transformer model to focus on relevant information and capture dependencies between words in a sentence:

Definition: Self-attention transforms the input sequence into three vectors: query, key, and value. The model then calculates attention scores by taking the dot product of the query vector for the current word and the key vectors for all the words in the input sequence. These scores indicate how much focus each word should receive.

What it does: Self-attention allows the model to weigh the importance of different words in a sentence when processing a specific word. For example, when processing the word "it" in the sentence "The animal didn't cross the street because it was too tired", self-attention enables the model to associate "it" with "animal".

Benefits: Self-attention allows the model to look at the whole context of the sequence while encoding each of the input elements. This helps the model avoid "forgetting" facts that might occur if the window of information was too large

Q26: What are the advantages of transformers over traditional sequence-to-sequence models?

Answer: Transformers have several advantages over traditional sequence-to-sequence models, such as recurrent neural networks (RNNs), when it comes to natural language processing tasks. Here are some key advantages:

- Long-range dependencies: Transformers are capable of capturing long-range dependencies in sequences more effectively compared to RNNs. This is because RNNs suffer from vanishing or exploding gradient problems when processing long sequences, which limits their ability to capture long-term dependencies. Transformers address this issue by using self-attention mechanisms that allow for capturing relationships between any two positions in a sequence, regardless of their distance.
- Parallelization: Transformers can process inputs in parallel, making them more efficient in terms of computational time compared to RNNs. In RNNs, the sequential nature of computation limits parallelization since each step depends on the previous step's output. Transformers, on the other hand, process all positions in a sequence simultaneously, enabling efficient parallelization across different positions.

- Scalability: Transformers are highly scalable and can handle larger input sequences without significantly increasing computational requirements. In RNNs, the computational complexity grows linearly with the length of the input sequence, making it challenging to process long sequences efficiently. Transformers, with their parallel processing and self-attention mechanisms, maintain a constant computational complexity, making them suitable for longer sequences.
- Global context understanding: Transformers capture global context information effectively due to their attention mechanisms. Each position in the sequence attends to all other positions, allowing for a comprehensive understanding of the entire sequence during the encoding and decoding process. This global context understanding aids in various NLP tasks, such as machine translation, where the translation of a word can depend on the entire source sentence.
- Transfer learning and fine-tuning: Transformers facilitate transfer learning and fine-tuning, which is the ability to pre-train models on large-scale datasets and then adapt them to specific downstream tasks with smaller datasets. Pretraining transformers on massive amounts of data, such as in models like BERT or GPT, helps capture rich language representations that can be fine-tuned for a wide range of NLP tasks, providing significant performance gains.

Q27: How does the attention mechanism help transformers capture long-range dependencies in sequences?

Answer: The attention mechanism in transformers plays a crucial role in capturing long-range dependencies in sequences. It allows each position in a sequence to attend to other positions, enabling the model to focus on relevant parts of the input during both the encoding and decoding stages. Here's how the attention mechanism works in transformers:

- Self-Attention: Self-attention, also known as intra-attention, is the key component of the attention mechanism in transformers. It computes the importance, or attention weight, that each position in the sequence should assign to other positions. This attention weight determines how much information a position should gather from other positions.
- Query, Key, and Value: To compute self-attention, each position in the sequence is associated with three learned vectors: query, key, and value. These vectors are derived from the input embeddings and transformed through linear transformations. The query vector is used to search for relevant information, the key vector represents the positions to which the query attends, and the value vector holds the information content of each position.
- Attention Scores: The attention mechanism calculates attention scores between the query vector of a position and the key vectors of all other positions in the sequence. The attention scores quantify the relevance or similarity between positions. They are obtained by taking the dot product between the query and key vectors and scaling it by a factor of the square root of the dimensionality of the key vectors.
- Attention Weights: The attention scores are then normalized using the softmax function to obtain attention weights. These weights determine the contribution of each position to the final representation of the current position. Positions with higher attention weights have a stronger influence on the current position's representation.

- Weighted Sum: Finally, the attention weights are used to compute a weighted sum of the value vectors. This aggregation of values gives the current position a comprehensive representation that incorporates information from all relevant positions, capturing the long-range dependencies effectively.

By allowing each position to attend to other positions, the attention mechanism provides a mechanism for information to flow across the entire sequence. This enables transformers to capture dependencies between distant positions, even in long sequences, without suffering from the limitations of vanishing or exploding gradients that affect traditional recurrent neural networks. Consequently, transformers excel in modeling complex relationships and dependencies in sequences, making them powerful tools for various tasks, including natural language processing and computer vision.

Q28: What are the limitations of transformers, and what are some potential solutions?

Answer: While transformers have revolutionized many natural language processing tasks, they do have certain limitations. Here are some notable limitations of transformers and potential solutions:

- Sequential Computation: Transformers process the entire sequence in parallel, which limits their ability to model sequential information explicitly. This can be a disadvantage when tasks require strong sequential reasoning. Potential solutions include incorporating recurrent connections into transformers or using hybrid models that combine the strengths of transformers and recurrent neural networks.
- Memory and Computational Requirements: Transformers consume more memory and computational resources compared to traditional sequence models, especially for large-scale models and long sequences. This limits their scalability and deployment on resource-constrained devices. Solutions involve developing more efficient architectures, such as sparse attention mechanisms or approximations, to reduce memory and computational requirements without sacrificing performance significantly.
- Lack of Interpretability: Transformers are often considered as black-box models, making it challenging to interpret the reasoning behind their predictions. Understanding the decision-making process of transformers is an ongoing research area. Techniques such as attention visualization, layer-wise relevance propagation, and saliency maps can provide insights into the model's attention and contribution to predictions, enhancing interpretability.
- Handling Out-of-Distribution Data: Transformers can struggle with data that significantly deviates from the distribution seen during training. They may make overconfident predictions or produce incorrect outputs when faced with out-of-distribution samples. Solutions include exploring uncertainty estimation techniques, robust training approaches, or incorporating external knowledge sources to improve generalization and handle out-of-distribution scenarios.
- Limited Contextual Understanding: Transformers rely heavily on context information to make predictions. However, they can still struggle with understanding the broader context, especially in scenarios with complex background knowledge or multi-modal data. Incorporating external knowledge bases, leveraging graph neural networks, or combining

transformers with other modalities like images or graphs can help improve contextual understanding and capture richer representations.

- Training Data Requirements: Transformers typically require large amounts of labeled data for effective training due to their high capacity. Acquiring labeled data can be expensive and time-consuming, limiting their applicability to domains with limited labeled datasets. Solutions include exploring semi-supervised learning, active learning, or transfer learning techniques to mitigate the data requirements and leverage pretraining on large-scale datasets.

Researchers and practitioners are actively working on addressing these limitations to further enhance the capabilities and applicability of transformers in various domains. As the field progresses, we can expect continued advancements and novel solutions to overcome these challenges.

Q29: How are transformers trained, and what is the role of pre-training and fine-tuning?

Answer:

Q30: What is BERT (Bidirectional Encoder Representations from Transformers), and how does it improve language understanding tasks?

Answer: BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based neural network model introduced by Google in 2018. It is designed to improve the understanding of natural language in various language processing tasks, such as question answering, sentiment analysis, named entity recognition, and more.

BERT differs from previous language models in its ability to capture the context of a word by considering both the left and right context in a sentence. Traditional language models, like the ones based on recurrent neural networks, process text in a sequential manner, making it difficult to capture the full context.

BERT, on the other hand, is a "pre-trained" model that is trained on a large corpus of unlabeled text data. During pre-training, BERT learns to predict missing words in sentences by considering the surrounding words on both sides. This bidirectional training allows BERT to capture contextual information effectively.

Once pre-training is complete, BERT is fine-tuned on specific downstream tasks. This fine-tuning involves training the model on labeled data from a particular task, such as sentiment analysis or named entity recognition. During fine-tuning, BERT adapts its pre-trained knowledge to the specific task, further improving its understanding and performance.

The key advantages of BERT include:

1. Contextual understanding: BERT can capture the contextual meaning of words by considering both the preceding and following words in a sentence, leading to better language understanding.

2. Transfer learning: BERT is pre-trained on a large corpus of unlabeled data, enabling it to learn general language representations. These pre-trained representations can then be fine-tuned for specific tasks, even with limited labeled data.
3. Versatility: BERT can be applied to a wide range of natural language processing tasks. By fine-tuning the model on specific tasks, it can achieve state-of-the-art performance in tasks such as question answering, text classification, and more.
4. Handling ambiguity: BERT's bidirectional nature helps it handle ambiguous language constructs more effectively. It can make more informed predictions by considering the context from both directions.

Q31: Describe the process of generating text using a transformer-based language model.

Answer:

Q32: What are some challenges or ethical considerations associated with large language models?

Answer:

Q33: Explain the concept of transfer learning and how it can be applied to transformers.

Answer:

Transfer learning is a machine learning technique where knowledge gained from training on one task is leveraged to improve performance on another related task. Instead of training a model from scratch on a specific task, transfer learning enables the use of pre-trained models as a starting point for new tasks.

In the context of transformers, transfer learning has been highly successful, particularly with models like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer).

Here's how transfer learning is applied to transformers:

1. Pre-training: In the pre-training phase, a transformer model is trained on a large corpus of unlabeled text data. The model is trained to predict missing words in a sentence (masked language modeling) or to predict the next word in a sequence (causal language modeling). This process enables the model to learn general language patterns, syntactic structures, and semantic relationships.
2. Fine-tuning: Once the transformer model is pre-trained, it can be fine-tuned on specific downstream tasks with smaller labeled datasets. Fine-tuning involves retraining the pre-trained model on task-specific labeled data. The model's parameters are adjusted to optimize performance on the specific task, while the pre-trained knowledge acts as a strong initialization for the fine-tuning process.

a. Task-specific architecture: During fine-tuning, the architecture of the pre-trained transformer model is often modified or extended to accommodate the specific requirements of the downstream task. For example, in sentiment analysis, an additional classification layer may be added on top of the pre-trained model to classify text sentiment.

b. Few-shot or zero-shot learning: Transfer learning with transformers allows for few-shot or even zero-shot learning scenarios. Few-shot learning refers to training a model on a small amount of labeled data, which is beneficial when data availability is limited. Zero-shot learning refers to using the pre-trained model directly on a task for which it hasn't been explicitly trained, but the model can still generate meaningful predictions based on its understanding of language.

Transfer learning with transformers offers several advantages:

1. Reduced data requirements: Pre-training on large unlabeled datasets allows the model to capture general language understanding, reducing the need for massive amounts of labeled task-specific data.
2. Improved generalization: The pre-trained model has learned rich representations of language from extensive pre-training, enabling it to generalize well to new tasks and domains.
3. Efficient training: Fine-tuning a pre-trained model requires less computational resources and training time compared to training from scratch.
4. State-of-the-art performance: Transfer learning with transformers has achieved state-of-the-art performance on a wide range of NLP tasks, including text classification, named entity recognition, question answering, machine translation, and more.

By leveraging the knowledge encoded in pre-trained transformers, transfer learning enables faster and more effective development of models for specific NLP tasks, even with limited labeled data.

Q34: What is the purpose of data augmentation in computer vision, and what techniques can be used?

Answer:

The purpose of data augmentation in computer vision is to artificially increase the size and diversity of a training dataset by applying various transformations to the original images. Data augmentation helps prevent overfitting and improves the generalization ability of deep learning models by exposing them to a broader range of variations and patterns present in the data. It also reduces the risk of the model memorizing specific examples in the training data.

By applying different augmentation techniques, the model becomes more robust and capable of handling variations in the real-world test data that may not be present in the original training set. Common data augmentation techniques include:

1. Horizontal Flipping: Flipping images horizontally, i.e., left to right, or vice versa. This is particularly useful for tasks where the orientation of objects doesn't affect their interpretation, such as object detection or image classification.

2. Vertical Flipping: Similar to horizontal flipping but flipping images from top to bottom.
3. Random Rotation: Rotating images by a random angle. This can be helpful to simulate objects at different angles and orientations.
4. Random Crop: Taking random crops from the input images. This forces the model to focus on different parts of the image and helps in handling varying object scales.
5. Scaling and Resizing: Rescaling images to different sizes or resizing them while maintaining the aspect ratio. This augmentation helps the model handle objects of varying sizes.
6. Color Jittering: Changing the brightness, contrast, saturation, and hue of the images randomly. This augmentation can help the model become more robust to changes in lighting conditions.
7. Gaussian Noise: Adding random Gaussian noise to the images, which simulates noisy environments and enhances the model's noise tolerance.
8. Elastic Transformations: Applying local deformations to the image, simulating distortions that might occur due to variations in the imaging process.
9. Cutout: Randomly masking out portions of the image with black pixels. This helps the model learn to focus on other informative parts of the image.
10. Mixup: Combining two or more images and their corresponding labels in a weighted manner to create new training examples. This encourages the model to learn from the combined patterns of multiple images.

It's important to note that the choice of data augmentation techniques depends on the specific computer vision task and the characteristics of the dataset. Additionally, augmentation should be applied only during the training phase and not during testing or evaluation to ensure that the model generalizes well to unseen data.