DESCRIPTIVE QUESTIONS-  
  
Q1. What is RNN?  
Ans:

RNN stands for Recurrent Neural Network. It's a type of artificial neural network designed to recognize patterns in sequences of data, such as text, time series, or spatial data. Unlike feedforward neural networks, which process input data in a single direction without any feedback loops, RNNs have connections that form directed cycles, allowing them to exhibit dynamic temporal behavior.

The key characteristic of RNNs is their ability to maintain a memory of previous inputs through hidden states. This memory enables RNNs to perform tasks that depend on past information, making them well-suited for tasks like language modeling, speech recognition, machine translation, and time series prediction.

However, traditional RNNs often suffer from the vanishing gradient problem, where gradients diminish as they propagate through time, leading to difficulties in learning long-term dependencies. To address this issue, several variants of RNNs have been developed, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), which incorporate mechanisms to better capture and retain long-range dependencies in sequences.

Q2. Explain LSTM?   
Ans: LSTM stands for Long Short-Term Memory, and it's a type of recurrent neural network (RNN) architecture designed to overcome the limitations of traditional RNNs in learning long-term dependencies in sequential data.

The key innovation of LSTM networks lies in their ability to selectively remember or forget information over time, enabling them to capture relevant dependencies over long sequences. This is accomplished through the use of a specialized memory cell, which maintains a constant internal state and interacts with three gating mechanisms:

Forget Gate: This gate determines which information from the previous cell state should be discarded or forgotten. It takes input from the previous hidden state and the current input and outputs a forget vector, which is multiplied element-wise with the previous cell state. This allows the LSTM to learn when to forget or retain information from past time steps.

Input Gate: The input gate decides which new information should be stored in the memory cell. It consists of two components: the input gate itself, which regulates how much of the new information should be added to the memory cell, and the update gate, which controls the updates to the cell state. The input gate is responsible for deciding which values will be updated in the cell state based on the current input and previous hidden state.

Output Gate: The output gate determines what information from the current cell state should be output to the next hidden state. It selectively activates parts of the cell state to produce the output based on the input and the previous hidden state.

By dynamically adjusting these gating mechanisms, LSTM networks can effectively capture and propagate relevant information over long sequences while mitigating the vanishing gradient problem often encountered in traditional RNNs. As a result, LSTMs have become widely used in various applications such as natural language processing, speech recognition, and time series forecasting.

Q3. What is GRU models?  
Ans: GRU stands for Gated Recurrent Unit, and it's another variant of recurrent neural network (RNN) architecture, similar to LSTM (Long Short-Term Memory), designed to address the challenges of capturing long-term dependencies in sequential data.

Like LSTM, GRU also incorporates gating mechanisms to control the flow of information through the network and mitigate the vanishing gradient problem. However, GRU is somewhat simpler than LSTM, as it combines the forget and input gates into a single update gate, resulting in fewer parameters and potentially faster training.

The key components of a GRU unit include:

Update Gate: This gate determines how much of the previous hidden state should be retained and how much of the new information should be incorporated into the current state. It combines aspects of both the forget gate and input gate in LSTM.

Reset Gate: The reset gate controls how much of the previous hidden state should be ignored in the calculation of the new hidden state, allowing the model to adaptively forget irrelevant information from earlier time steps.

By adjusting the values of the update gate and reset gate, GRU models can selectively update their internal state and effectively capture dependencies across different time steps in sequential data. GRUs are particularly popular in scenarios where computational resources are limited or where simpler models are preferred due to their faster training times and comparable performance to LSTM networks. They are commonly used in tasks such as natural language processing, machine translation, and time series analysis.

Q4. Create LSTM layers in keras-

Ans:

import tensorflow as tf

# Define LSTM model

model = tf.keras.Sequential([

tf.keras.layers.LSTM(64, input\_shape=(10, 1)),

tf.keras.layers.Dense(1)

])

Q5. Create GRU layers in keras-  
Ans:

import tensorflow as tf

# Define LSTM model

model = tf.keras.Sequential([

tf.keras.layers.GRU(64, input\_shape=(10, 1)),

tf.keras.layers.Dense(1)

])

Q6. What is an LSTM layer, and why is it used in deep learning?

Answer: An LSTM layer is a type of recurrent neural network (RNN) layer designed to capture long-term dependencies in sequential data. It is used in deep learning to model and make predictions on time series data, natural language processing tasks, and other sequential data applications.

Q7. Explain the vanishing gradient problem. How does LSTM address this issue compared to traditional RNNs?

Answer: The vanishing gradient problem occurs when gradients become too small during backpropagation, making it challenging to update the weights of deep neural networks. LSTMs address this by using a gating mechanism that allows them to learn when to update and forget information, mitigating the vanishing gradient problem better than traditional RNNs.

Q8. What are the key components of an LSTM cell?

Answer: An LSTM cell consists of three key components: the input gate, the forget gate, and the output gate. These gates control the flow of information into and out of the cell, and the cell state stores the information over time.

Q9. How does the input gate work in an LSTM cell?

Answer: The input gate controls what information to add to the cell state. It uses a sigmoid activation function to determine which values should be updated and a tanh activation function to create a candidate update.

Q10.Explain the role of the forget gate in an LSTM cell?

Answer: The forget gate decides which information from the previous cell state should be retained and which should be discarded. It uses a sigmoid activation function to produce values between 0 and 1 for each component of the cell state.

Q11. What is the purpose of the output gate in an LSTM cell?

Answer: The output gate determines which parts of the cell state should be exposed as the hidden state. It uses a sigmoid activation function and a tanh activation function to produce the output hidden state.

Q12. How is the cell state updated in an LSTM cell?

Answer: The cell state is updated by combining the result of the forget gate (which decides what to forget from the previous cell state) with the result of the input gate (which decides what to add). This updated cell state is then used in the next time step.

Q13. What is the difference between the hidden state and the cell state in an LSTM?

Answer: The hidden state is the output of an LSTM cell at a specific time step and contains information relevant to making predictions. The cell state represents the internal memory of the LSTM and can store information over long sequences.

Q14. How is dropout used in LSTM networks?

Answer: Dropout is a regularization technique applied to LSTM networks by randomly setting a fraction of the hidden units to zero during training. It helps prevent overfitting and encourages the network to learn robust representations.

MULTIPLE-CHOICE QUESTIONS-

Q15. Can you provide an example of a practical LSTM implementation, such as text generation?

Answer: a)

a) In text generation, an LSTM can be trained to predict the next word in a sentence given the previous words. This involves preprocessing text data, creating input sequences, encoding words as vectors, training the LSTM model, and using it to generate text based on a seed sentence.

b) LSTM can be utilized in stock market prediction, analyzing historical stock prices to forecast future trends.

c) An LSTM can be applied in image recognition tasks, processing sequential data from image pixels to recognize patterns and objects.

d) LSTM can be employed in weather forecasting, analyzing past weather data to predict future atmospheric conditions.

Q16. Can you show an example of how you would implement an LSTM in Python using a deep learning library like TensorFlow or Keras?

Answer: a)

a)

from keras.models import Sequential

from keras.layers import LSTM, Dense

# Initialize the model

model = Sequential()

# Add LSTM layer with 50 units and 'relu' activation function

model.add(LSTM(50, activation='relu', input\_shape=(None, 1)))

# Add output layer with 1 unit (for regression task)

model.add(Dense(1))

# Compile the model with mean squared error loss and adam optimizer

model.compile(loss='mean\_squared\_error', optimizer='adam')

# Fit the model to the data for 100 epochs

model.fit(X\_train, y\_train, epochs=100, verbose=0)

b)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

# Initialize the model

model = Sequential()

# Add LSTM layer with 100 units and 'tanh' activation function

model.add(LSTM(100, activation='tanh', input\_shape=(None, 1)))

# Add output layer with 1 unit (for regression task)

model.add(Dense(1))

# Compile the model with mean absolute error loss and rmsprop optimizer

model.compile(loss='mean\_absolute\_error', optimizer='rmsprop')

# Fit the model to the data for 50 epochs

model.fit(X\_train, y\_train, epochs=50, verbose=0)

c)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

# Initialize the model

model = Sequential()

# Add LSTM layer with 32 units and 'tanh' activation function

model.add(LSTM(32, activation='tanh', input\_shape=(None, 1)))

# Add output layer with 1 unit (for regression task)

model.add(Dense(1))

# Compile the model with mean absolute percentage error loss and sgd optimizer

model.compile(loss='mean\_absolute\_percentage\_error', optimizer='sgd')

# Fit the model to the data for 90 epochs

model.fit(X\_train, y\_train, epochs=90, verbose=0)

d) from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

# Initialize the model

model = Sequential()

# Add LSTM layer with 32 units and 'tanh' activation function

model.add(LSTM(32, activation='tanh', input\_shape=(None, 1)))

# Add output layer with 1 unit (for regression task)

model.add(Dense(1))

# Compile the model with mean absolute percentage error loss and sgd optimizer

model.compile(loss='mean\_absolute\_percentage\_error', optimizer='sgd')

# Fit the model to the data for 90 epochs

model.fit(X\_train, y\_train, epochs=90, verbose=0)

Q17. What’s the difference between LSTMs and GRUs? Which one should be used more often?

Answer: a)

a) LSTMs and GRUs are both types of recurrent neural networks (RNNs) that are used for processing sequential data. The main difference between the two is that LSTMs have a memory cell that helps them remember information for longer periods of time, while GRUs do not have this memory cell. Because of this, LSTMs are better suited for tasks that require remembering information over a long period of time, while GRUs are better for tasks that do not require remembering information for such a long time.

b) LSTMs and GRUs differ in their gating mechanisms; LSTMs have three gates (input, forget, and output gates), while GRUs have two gates (reset and update gates).

c) LSTMs and GRUs have similar capabilities in processing sequential data, but LSTMs tend to perform better in tasks requiring long-term memory retention due to their explicit memory cell.

d) LSTMs and GRUs are identical in functionality and are interchangeable in most scenarios, with no significant difference in performance observed across various tasks.

Q18. How would you Compile the Keras GRU RNN

Answer: a)

a)

model.compile(

loss=keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

optimizer="sgd",

metrics=["accuracy"],

)

b)

model.compile(

loss=keras.losses.BinaryCrossentropy(from\_logits=True),

optimizer="adam",

metrics=["accuracy"],

)

c)

model.compile(

loss=keras.losses.MeanSquaredError(),

optimizer="rmsprop",

metrics=["mae"],

)

d)

model.compile(

loss=keras.losses.CategoricalCrossentropy(from\_logits=True),

optimizer="adagrad",

metrics=["accuracy"],

)

Q19. What is Update Gate?

Answer: a)

a) Think of the update gate as the chef’s decision on the proportion of the current mixture in the bowl (previous memory) to retain, and the proportion of the new ingredients (new candidate state) to add.

If the update gate value is close to 1, it means the chef wants to use more of the current mixture, and less of the new ingredients.

If the update gate value is close to 0, the chef wants to use more new ingredients and less of the current mixture.

In this way, the update gate helps balance the old and new information when updating the overall memory.

At first, this was confusing to me, as I expected a higher update gate value to mean that the chef wants to use more of the new ingredients, but it was the contrary.

b) The update gate serves as a threshold for discarding irrelevant information, helping to maintain the integrity of the model's memory by removing noise from the input.

c) The update gate functions as a control mechanism to regulate the flow of information into the memory cell, preventing sudden spikes or drops in memory content.

d) The update gate acts as a filter, determining the relevance of new information by comparing it with the existing memory content, allowing only significant updates to be integrated into the memory cell.

Q20. How to check if you have GPU or not using torch?

Answer: a)

a) To check if you have a GPU installed and correctly configured with PyTorch (backend library), run the code below:

import torch

print(torch.cuda.is\_available())

This function returns True if you have a GPU installed and correctly configured, and False otherwise.

b) Use the command nvidia-smi in the terminal to check if your system recognizes an NVIDIA GPU and its status.

c) Check the output of the command lspci | grep -i vga in the terminal to see if your system detects a VGA-compatible controller, which could indicate the presence of a GPU.

d) Execute cat /proc/cpuinfo in the terminal and look for entries related to GPU devices to determine if your system has a GPU.

Q21. How To Train GRU In Python?

Answer: a)

a) from neuralforecast import NeuralForecast

from neuralforecast.models import GRU

from neuralforecast.losses.pytorch import DistributionLoss

models = [GRU(h=h,

loss=DistributionLoss(distribution='Normal', level=[90]),

max\_steps=100,

encoder\_n\_layers=2,

encoder\_hidden\_size=200,

context\_size=10,

encoder\_dropout=0.5,

decoder\_hidden\_size=200,

decoder\_layers=2,

learning\_rate=1e-3,

scaler\_type='standard',

futr\_exog\_list=['onpromotion'])]

model = NeuralForecast(models=models, freq='D')

model.fit(train)

b)

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import GRU, Dense

model = Sequential()

model.add(GRU(100, input\_shape=(None, n\_features)))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

model.fit(X\_train, y\_train, epochs=100, verbose=0)

c)

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import GRU, Dense

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = Sequential()

model.add(GRU(50, input\_shape=(X\_train.shape[1], X\_train.shape[2])))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test), verbose=0)

d)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import GRU, Dense

model = Sequential()

model.add(GRU(128, return\_sequences=True, input\_shape=(n\_steps, n\_features)))

model.add(GRU(128))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

model.fit(X\_train, y\_train, epochs=100, verbose=0)

Q22. How To Split Time Series Data For Validation?

Answer: You should never use random or k-fold validation for time series.

That would cause data leakage, as you would be using future data to train your model.

In practice, you can’t take random samples from the future to train your model, so you can’t use them here.

To avoid this issue, we will use a simple time series split between past and future.

A career tip: knowing how to do time series validation correctly is a skill that will set you apart from many data scientists (even experienced ones!).

Our training set will be all the data between 2013 and 2016 and our validation set will be the first 3 months of 2017.

train = data2.loc[data2['ds'] < '2017-01-01']

valid = data2.loc[(data2['ds'] >= '2017-01-01') & (data2['ds'] < '2017-04-01')]

h = valid['ds'].nunique()

b) To split time series data for validation, you can use the train\_test\_split function from scikit-learn library. This function automatically handles the splitting based on the provided test size or train size parameters.

from sklearn.model\_selection import train\_test\_split

train, valid = train\_test\_split(data, test\_size=0.2, shuffle=False)

c) Time series data validation can be achieved by dividing the data into chronological segments, where the first segment is used for training and the subsequent segments are used for validation. This ensures that the model is evaluated on unseen future data.

train = data[:800]

valid = data[800:]

d) Time series data can be split for validation using cross-validation techniques such as k-fold cross-validation, where the data is divided into k consecutive folds. However, special care must be taken to ensure that the folds are non-overlapping and preserve the temporal order of the data.

from sklearn.model\_selection import TimeSeriesSplit

tscv = TimeSeriesSplit(n\_splits=5)

for train\_index, test\_index in tscv.split(data):

train, valid = data[train\_index], data[test\_index]

Q23. How To Prepare Time Series Data For The GRU?

Answer:

Let’s use the very practical example of sales forecasting) in this tutorial.

We will use real sales data from the Favorita store chain, from Ecuador.

We have sales data from 2013 to 2017 for multiple stores and product categories.

For this tutorial I will use only the data from one store and two product categories.

You can use as many categories, SKUs, stores, etc as you want.

Dataset Link: https://www.kaggle.com/competitions/store-sales-time-series-forecasting/

import pandas as pd

import numpy as np

path = 'train.csv'

data = pd.read\_csv(path, index\_col='id', parse\_dates=['date'])

data2 = data.loc[(data['store\_nbr'] == 1) & (data['family'].isin(['MEATS', 'PERSONAL CARE'])

Q23. How To Prepare Time Series Data For The GRU?

Answer: a)

a) Let’s use the very practical example of sales forecasting) in this tutorial.

We will use real sales data from the Favorita store chain, from Ecuador.

We have sales data from 2013 to 2017 for multiple stores and product categories.

For this tutorial I will use only the data from one store and two product categories.

You can use as many categories, SKUs, stores, etc as you want.

Dataset Link: https://www.kaggle.com/competitions/store-sales-time-series-forecasting/

import pandas as pd

import numpy as np

path = 'train.csv'

data = pd.read\_csv(path, index\_col='id', parse\_dates=['date'])

data2 = data.loc[(data['store\_nbr'] == 1) & (data['family'].isin(['MEATS', 'PERSONAL CARE']))

b) To prepare time series data for GRU, first, ensure your dataset is loaded into a pandas DataFrame. Then, filter the data based on the store and product categories you are interested in using.

import pandas as pd

import numpy as np

path = 'train.csv'

data = pd.read\_csv(path, index\_col='id', parse\_dates=['date'])

store\_data = data[data['store\_nbr'] == 1]

selected\_categories = ['MEATS', 'PERSONAL CARE']

filtered\_data = store\_data[store\_data['family'].isin(selected\_categories)]

c) In order to prepare time series data for GRU, you need to ensure that the dataset is sorted chronologically by the time index. Additionally, it's important to handle missing values appropriately, either by imputing them or removing them from the dataset.

import pandas as pd

import numpy as np

path = 'train.csv'

data = pd.read\_csv(path, index\_col='date', parse\_dates=['date'])

data.sort\_index(inplace=True)

d) To prepare time series data for GRU, it's crucial to first perform feature engineering to extract relevant features such as lagged values, moving averages, or seasonal components. Once the features are extracted, you can split the data into input sequences and target values, which can then be fed into the GRU model for training.

import pandas as pd

import numpy as np

path = 'train.csv'

data = pd.read\_csv(path, index\_col='id', parse\_dates=['date'])

Q24. How to use model.fit in keras?

Answer: a)

a)

model.fit(feature\_train, label\_train, batch\_size=512, epochs=10, validation\_data = (feature\_test, label\_test))

b)

modelfeature\_train, label\_train, batch\_size=256, epochs=15, validation\_split=0.1, shuffle=True)

c)

model.fit\_train(feature\_train, label\_train, batch\_size=64, epochs=20, validation\_split=0.3, verbose=1)

d)

fit(feature\_train, label\_train, batch\_size=32, epochs=8, validation\_data=(feature\_test, label\_test), shuffle=False)

Q25. Create model in Keras which should have Sequential layer, LSTM layer, Dropout layers in it.

Answer: a)

a)

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(feature\_train.shape[1],1)))

model.add(Dropout(0.2))

model.add(LSTM(100, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(1, activation = "linear"))

b)

from keras.models import Sequential

from keras.layers import LSTM, Dropout, Dense

model = Sequential()

model.add(GRU(64, return\_sequences=True, input\_shape=(feature\_train.shape[1],1)))

model.add(Dropout(0.3))

model.add(GRU(128))

model.add(Dropout(0.3))

model.add(Dense(1, activation='sigmoid'))

c)

from tensorflow.keras.models import Sequential

from tensorflow.keras.models import LSTM, Dropout, Dense

model = Sequential()

model.add(LSTM(32, return\_sequences=True, input\_shape=(feature\_train.shape[1],1)))

model.add(Dropout(0.25))

model.add(LSTM(64))

model.add(Dropout(0.25))

model.add(Dense(1, activation='relu'))

d)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dropout, Dense

model = Sequential()

model.add(LSTM(128, return\_sequences=True, input\_shape=(feature\_train.shape[1],1)))

model.add(Dropout(0.5))

model.add(LSTM(256))

model.add(Dropout())

model.add(Dense(1, activation='cosine'))