We at The Data Monk hold the vision to make sure everyone in the IT industry has an equal stand to work in an open domain such as analytics. Analytics is one domain where there is no formal under-graduation degree and which is achievable to anyone and everyone in the World.

We are a team of 30+ mentors who have worked in various product-based companies in India and abroad, and we have come up with this idea to provide study materials directed to help you crack any analytics interview.

Every one of us has been interviewing for at least the last 6 to 8 years for different positions like Data Scientist, Data Analysts, Business Analysts, Product Analysts, Data Engineers, and other senior roles. We understand the gap between having good knowledge and converting an interview to a top product-based company.

Rest assured that if you follow our different mediums like our blog cum questions-answer portal www.TheDataMonk.com , our youtube channel - The Data Monk, and our e-books, then you will have a very strong candidature in whichever interview you participate in.

There are many blogs that provide free study materials or questions on different analytical tools and technologies, but we concentrate mostly on the questions which are asked in an interview. We have a set of 100+ books which are available both on Amazon and on The Data Monk e-shop page

We would recommend you to explore our website, youtube channel, and e-books to understand the type of questions covered in our articles. We went for the question-answer approach both on our website as well as our e-books just because we feel that the best way to go from beginner to advance level is by practicing a lot of questions on the topic.

We have launched a series of 50 e-books on our website on all the popular as well as niche topics. Our range of material ranges from SQL, Python, and Machine Learning algorithms to ANN, CNN, PCA, etc.

We are constantly working on our product and will keep on updating it. It is very necessary to go through all the questions present in this book.

Give a rating to the book on Amazon, do provide your feedback and if you want to help us grow then please subscribe to our Youtube channel.

TENSORFLOW

Q1. What is Tensorflow?

A1. Tensorflow is a free and open source library for machine learning and deep learning in python. It is one of the most popular and widely used machine learning libraries in python used extensively for developing advanced machine learning and deep learning models.

Q2. How was tensorflow developed?

A2. Tensorflow was started by Google first as "DistBelief" as a deep learning framework by Google brain for neural networks. It was later refactored to a more robust library which became Tensorflow.

Q3. What language is Tensorflow written in?

A3. Tensorflow is mainly written using C++ and CUDA.

Q4. What all languages support the Tensorflow API?

A4. While you can use tensorflow in C++, Javascript, and Java, it is most widely used using Python as it offers much more functionality and stability.

Q5. What are the requirements needed to install and use tensorflow?

A5. These are the requirements needed to run TensorFlow 2 locally in python:

1. Python 3.6 – 3.8
2. Ubuntu 16.04 or later
3. Windows 7 or later
4. macOS 10.12.6 or later

Q6. How to install tensorflow using pip?

A6. Tensorflow 2 requires pip version 19.0 or greater. To install it, we simply rum the command:

```
$ pip install tensorflow
```

Q7. How to install tensorflow with GPU support?

A7. Tensorflow has GPU support for specific NVIDIA GPU cards that are CUDA enabled . First, make sure that the GPU you have is CUDA enabled. You can do so by checking the list on this link here : https://developer.nvidia.com/cuda-gpus

Next, make sure you have the necessary drivers install required by your particular hardware.

Once you are done with that, you need to make sure you have the following softwares installed on your system:

1. CUDA toolkit
2. CUPTI
3. cuDNN

Once you are done with installing all of these, you can pip install tensorflow with gpu support.

If you are using a conda environment, you can simply use the command:

```
conda create --name *environment_name* tensorflow-gpu
```

Where *environment_name* will be replaced by your actual environment name.

This will install all of the necessary softwares mentioned above by its own for GPU support and you will not need to install them externally.

Q8. What is a tensor?

A8. A tensor is basically a multidimensional array, that is tensors are vectors or matrices that can go up to very high dimensions. When we work in tensorflow, primarily we will be dealing with tensors only. While performing any kind of operation, at the back end the main object being manipulated will be tensors. Tensors are used to store the values of the weights of the models, the gradients during back propagation, the values during forward propagation and pretty much everything else.

Q9. What is the difference between a tensor and a numpy array?

A9. There are two primary differences between a tensorflow tensor and a numpy array:

1. A tensor is backed by accelerated memory like GPUs, which means if GPU is available using a tensor for your applications than a numpy array will result in a significant improvement in speed.
2. A tensor is immutable, while numpy arrays are mutable.

Q10. Write a python program to declare a simple single scalar tensor using tensorflow.

A10. First let us import the library:

```
import tensorflow as tf
```

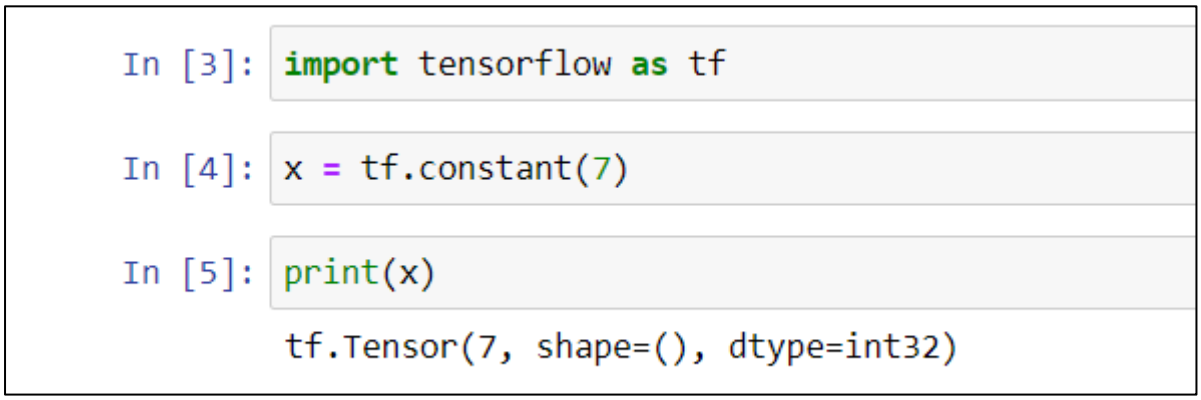
Let us now declare a single scalar tensor using the tensorflow function tf.constant:

```
x = tf.constant(7)
```

Finally let us print the tensor:

```
print(x)
```

Output:



Q11. Write a python program to declare a simple single scalar tensor using tensorflow with a shape [1,1].

A11. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a single scalar tensor using the tensorflow function tf.constant and pass the argument for shape:

```
x = tf.constant(7, shape = [1.1])
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [3]: import tensorflow as tf

In [6]: x = tf.constant(7, shape = [1,1])

In [7]: print(x)
        tf.Tensor([[7]], shape=(1, 1), dtype=int32)
```

As you can see, now the tensor has a shape of (1, 1)

Q12. Write a python program to declare a simple single scalar tensor using tensorflow with a shape [1,1] and dtype equal to float.

A12. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a single scalar tensor using the tensorflow function tf.constant and pass the argument for shape as well as for the dtype:

```
x = tf.constant(7, shape = [1.1], dtype = tf.float64)
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [3]: import tensorflow as tf

In [8]: x = tf.constant(7, shape = [1,1], dtype = tf.float64)

In [9]: print(x)
        tf.Tensor([[7.]], shape=(1, 1), dtype=float64)
```

As you can see, the tensor now has a shape of (1, 1) as well as the dtype this time has become float as well.

Q13. Write a python program to declare a one dimensional tensor or a vector tensor using tensorflow.

A13. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a vector tensor by using the function tf.constant and pass a list inside for a vector tensor:

```
x = tf.constant([7, 14, 21])
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [1]: import tensorflow as tf

In [2]: x = tf.constant([7, 14, 21])

In [3]: print(x)
        tf.Tensor([ 7 14 21], shape=(3,), dtype=int32)
```

As you can see, the tensor has 3 values this time like an array.

Q14. Write a python program to declare a two dimensional tensor or a matrix tensor using tensorflow.

A14. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a matrix tensor by using the function tf.constant and pass a two dimensional list inside for a matrix tensor:

```
x = tf.constant([[7, 14, 21], [6, 12, 18]])
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [1]: import tensorflow as tf

In [4]: x = tf.constant([[7, 14, 21], [6, 12, 18]])

In [5]: print(x)
        tf.Tensor(
        [[ 7 14 21]
         [ 6 12 18]], shape=(2, 3), dtype=int32)
```

As you can see, the tensor here is like a matrix and is two dimensional and has the shape equal to (2, 3)

Q15. Write a python program to declare a two dimensional tensor or a matrix tensor using tensorflow by passing in a python array.

A15. First let us import the library:

```
import tensorflow as tf
```

Let us now declare the python array that we want to convert into a tensor:

```
arr = [[1, 2, 3], [4, 5, 6]]
```

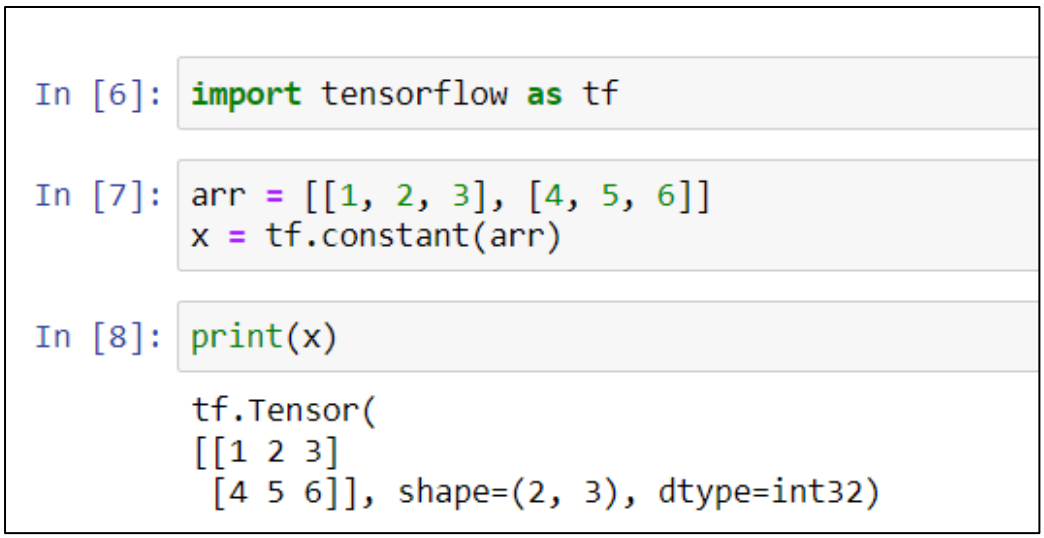
Let us now declare a matrix tensor by using the function tf.constant and pass the python array inside:

```
x = tf.constant(arr)
```

Finally let us print the tensor:

```
print(x)
```

Output:



As you can see, the python array has been converted into a two dimensional tensor.

Q16. Write a python program to demonstrate the addition of two two dimensional tensors.

A16. First let us import the library:

```
import tensorflow as tf
```

Let us now declare the two tensors that we will perform the addition on

```
x1 = tf.constant([[1, 2], [3, 4]])
x2 = tf.constant([[4, 3], [2, 1]])
```

Let us now perform the addition of these two tensors:

```
sum = tf.add(x1, x2)
```

Finally let us print the sum tensor:

```
print(sum)
```

Output:

```
In [2]: import tensorflow as tf

In [3]: x1 = tf.constant([[1, 2], [3, 4]])
        x2 = tf.constant([[4, 3], [2, 1]])

In [4]: sum = tf.add(x1, x2)

In [5]: print(sum)

        tf.Tensor(
        [[5 5]
         [5 5]], shape=(2, 2), dtype=int32)
```

Q17. Write a python program to demonstrate the element wise multiplication of two two dimensional tensors.

A17. First let us import the library:

```
import tensorflow as tf
```

Let us now declare the two tensors that we will perform the element wise multiplication on

```
x1 = tf.constant([[1, 2], [3, 4]])
x2 = tf.constant([[4, 3], [2, 1]])
```

Let us now perform the multiplication of these two tensors:

```
product = tf.multiply(x1, x2)
```

Finally let us print the product tensor:

```
print(product)
```

Output:

```
In [2]: import tensorflow as tf

In [3]: x1 = tf.constant([[1, 2], [3, 4]])
        x2 = tf.constant([[4, 3], [2, 1]])

In [6]: product = tf.multiply(x1, x2)

In [7]: print(product)

        tf.Tensor(
        [[4 6]
         [6 4]], shape=(2, 2), dtype=int32)
```

Q18. Write a python program to demonstrate the matrix multiplication of two two dimensional tensors.

A18. First let us import the library:

```
import tensorflow as tf
```

Let us now declare the two tensors that we will perform the matrix multiplication on

```
x1 = tf.constant([[1, 2], [3, 4]])
x2 = tf.constant([[4, 3], [2, 1]])
```

Let us now perform the multiplication of these two tensors:

```
product = tf.matmul(x1, x2)
```

Finally let us print the product tensor:

```
print(product)
```

Output:

```
In [2]: import tensorflow as tf

In [3]: x1 = tf.constant([[1, 2], [3, 4]])
        x2 = tf.constant([[4, 3], [2, 1]])

In [8]: product = tf.matmul(x1, x2)

In [9]: print(product)

        tf.Tensor(
        [[ 8  5]
         [20 13]], shape=(2, 2), dtype=int32)
```

Q19. Write a python program to declare a ones only tensor in tensorflow.

A19. First let us import the library:

```
import tensorflow as tf
```

Let us now declare the ones tensor with a shape (3, 4):

```
x = tf.ones((3,4))
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [2]: import tensorflow as tf

In [3]: x = tf.ones((3,4))

In [4]: print(x)
        tf.Tensor(
        [[1. 1. 1. 1.]
         [1. 1. 1. 1.]
         [1. 1. 1. 1.]], shape=(3, 4), dtype=float32)
```

Q20. Write a python program to declare a zeroes only tensor in tensorflow.

A20. First let us import the library:

```
import tensorflow as tf
```

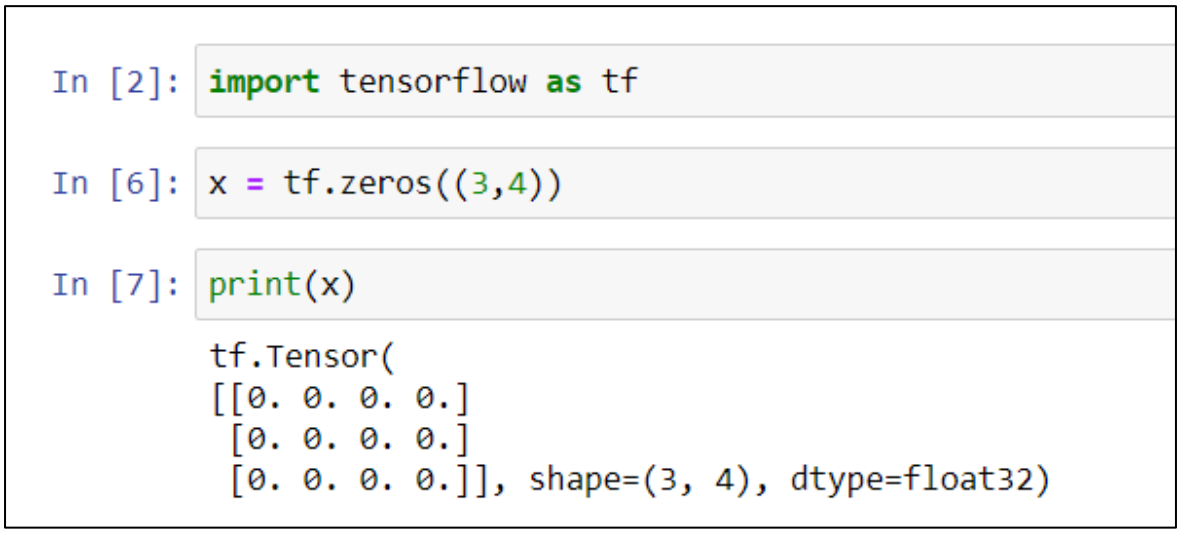
Let us now declare the zeroes tensor with a shape (3, 4):

```
x = tf.zeros((3,4))
```

Finally let us print the tensor:

```
print(x)
```

Output:



Q21. Write a python program to print the first 3 elements of a tensor.

A21. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a vector tensor:

```
x = tf.constant([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Finally let us print the first three elements of the tensor using indexing:

```
print(x[:3])
```

Output:

```
In [2]: import tensorflow as tf

In [8]: x = tf.constant([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [9]: print(x[:3])
        tf.Tensor([1 2 3], shape=(3,), dtype=int32)
```

Q22. Write a python program to print the last 4 elements of a tensor.

A22. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a vector tensor:

```
x = tf.constant([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Finally let us print the last four elements of the tensor using indexing:

```
print(x[-4:])
```

Output:

```
In [2]: import tensorflow as tf

In [8]: x = tf.constant([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [11]: print(x[-4:])
         tf.Tensor([6 7 8 9], shape=(4,), dtype=int32)
```

Q23. Write a python program to declare an identity matrix tensor in tensorflow of shape (4, 4).

A23. First let us import the library:

```
import tensorflow as tf
```

Let us now declare the identity matrix tensor of shape (4, 4):

```
x = tf.eye(4)
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [2]: import tensorflow as tf

In [14]: x = tf.eye(4)

In [15]: print(x)
tf.Tensor(
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]], shape=(4, 4), dtype=float32)
```

Q24. Write a python program to declare a tensor with random values from a normal distribution of mean equal to zero and a standard deviation of one.

A24. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a tensor with random values from a normal distribution of mean 0 and standard deviation 1.

```
x = tf.random.normal((3,3), mean = 0, stddev = 1)
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [2]: import tensorflow as tf

In [16]: x = tf.random.normal((3,3), mean = 0, stddev = 1)

In [17]: print(x)

tf.Tensor(
[[-0.37314793 -0.40658426  1.136344  ]
 [-2.0146627   1.8065737   0.77292585]
 [-0.24714753 -1.3135073   0.41982093]], shape=(3, 3), dtype=float32)
```

Q25. Write a python program to declare a tensor with random values from a uniform distribution between zero and one.

A25. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a tensor with random values from a uniform distribution between 0 and 1.

```
x = tf.random.uniform((3,3), minval = 0, maxval = 1)
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [2]: import tensorflow as tf

In [18]: x = tf.random.uniform((3,3), minval = 0, maxval = 1)

In [19]: print(x)

tf.Tensor(
[[0.6542227  0.6162466  0.56471395]
 [0.93637645 0.00852966 0.88141406]
 [0.5202075  0.84322214 0.4234531 ]], shape=(3, 3), dtype=float32)
```

Q26. Write a python program to declare a tensor with values from 1 to 25 using the range function.

A26. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a tensor with values from 1 to 25 using the range function:

```
x = tf.range(25)
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [2]: import tensorflow as tf

In [20]: x = tf.range(25)

In [21]: print(x)

tf.Tensor(
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24], shape=(25,), dtype=int32)
```

Q27. Write a python program to declare a tensor with only the odd values between 1 and 15 using the range function.

A27. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a tensor with only odd values from 1 to 15 using the range function:

```
x = tf.range(start = 1, limit = 15, delta = 2)
```

The delta value is used to set the number of values skipped in the range interval.

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [2]: import tensorflow as tf

In [22]: x = tf.range(start = 1, limit = 15, delta = 2)

In [23]: print(x)
         tf.Tensor([ 1  3  5  7  9 11 13], shape=(7,), dtype=int32)
```

Q28. Write a python program to reshape a tensor of shape (16, 1) to shape (4, 4).

A28. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a vector tensor with 16 values:

```
x = tf.range(16)
```

Now let us reshape this tensor to shape (4, 4) using the function tf.reshape():

```
x = tf.reshape(x , (4, 4))
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [2]: import tensorflow as tf

In [26]: x = tf.range(16)

In [27]: print(x)
         tf.Tensor([ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15], shape=(16,), dtype=int32)

In [28]: x = tf.reshape(x , (4, 4))

In [29]: print(x)
         tf.Tensor(
         [[ 0  1  2  3]
          [ 4  5  6  7]
          [ 8  9 10 11]
          [12 13 14 15]], shape=(4, 4), dtype=int32)
```

Q29. Write a python program to create a transpose of a matrix tensor.

A29. First let us import the library:

```
import tensorflow as tf
```

Let us now declare a vector tensor with 16 values:

```
x = tf.range(16)
```

Now let us reshape this tensor to shape (4, 4) using the function tf.reshape():

```
x = tf.reshape(x , (4, 4))
```

Let us now create the transpose tensor for this tensor:

```
x = tf.transpose(x)
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [2]: import tensorflow as tf

In [26]: x = tf.range(16)

In [27]: print(x)
tf.Tensor([ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15], shape=(16,), dtype=int32)

In [28]: x = tf.reshape(x , (4, 4))

In [29]: print(x)
tf.Tensor(
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]], shape=(4, 4), dtype=int32)

In [31]: x = tf.transpose(x)

In [32]: print(x)
tf.Tensor(
[[ 0  4  8 12]
 [ 1  5  9 13]
 [ 2  6 10 14]
 [ 3  7 11 15]], shape=(4, 4), dtype=int32)
```

Q30. Write a python program to declare a tensor of strings.

A30. First let us import the library:

```
import tensorflow as tf
```

Let us now declare our tensor with strings:

```
x = tf.constant(["The", "Data", "Monk"])
```

Finally let us print the tensor:

```
print(x)
```

Output:

```
In [2]: import tensorflow as tf

In [33]: x = tf.constant(["The", "Data", "Monk"])

In [34]: print(x)
         tf.Tensor([b'The' b'Data' b'Monk'], shape=(3,), dtype=string)
```

Q31. What is keras?

A31. Keras is a higher level API that is intergrated with tensorflow. It is used to create neural networks while using tensorflow and has a number of functions that help to develop and evaluate your deep learning model's training and performance.

Q32. What is a neural network?

A32. A neural network is an advanced machine learning algorithm where in you group a bunch of neurons in a layer have them perform tasks like classification, regression etc.

Q33. How does a neural network work?

A33. Suppose you have a simple classification task given two input variables x1 and x2, and you have the prediction value y that can be 0 or 1.

So, you can have a simple neural network with 2 layers that will take x1 and x2 as input, forward propagate those values through the network, and then output the prediction value y.

Q34. What does MNIST database stand for?

A34. MNIST stands for "Modified National Institute of Standards and Technology" database.

Q35. What is the MNIST dataset?

A35. The MNIST dataset is a dataset of 60,000 images of handwritten digits from 0 to 9. Each image is of the size 28 x 28 pixels. Each image is also a grayscale so that means that the image has only one channel. It is one of the most popular and widely used datasets to train and test basic machine learning or deep learning models.

Q36. What is the input shape of the MNIST dataset?

A36. Since each image has an image size of 28 x 28 pixels, and all images are grayscale which means that they have only one channel, the MNIST dataset images have an input shape of (28, 28, 1).

Q37. What are training and test sets for a neural network?

A37. While creating a neural network, we often have two different datasets known as training and test data sets. The training data set is the one that is used to train the neural network model. The test data set is used to evaluate the performance of the model as this data has never been seen by the neural network model and is a good choice to test how well the model has generalized for any particular given task.

Q38. Write a python program to import the MNIST dataset using tensorflow and observe the shapes.

A38. First, let us import the libraries we need:

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
```

Let us now load the MNIST dataset to create the train and test data sets:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Let us print the shape of the training data set:

```
print(x_train.shape)
```

Let us print the shape of the training data set labels:

```
print(y_train.shape)
```

Output:

```
In [3]: import tensorflow as tf
        from tensorflow.keras.datasets import mnist

In [4]: (x_train, y_train), (x_test, y_test) = mnist.load_data()

        Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
        11493376/11490434 [==============================] - 0s 0us/step

In [5]: print(x_train.shape)

        (60000, 28, 28)

In [6]: print(y_train.shape)

        (60000,)
```

As you can see, the training data set has 60,000 images with image size of 28 by 28 pixels.

Q39. What is normalization?

A39. Normalization is a method during the preparation of data in order to scale all of the data to a common scale so as to get all the values in a common range.

Q40. Why is normalization done?

A40. Normalization is done for the following reasons:

1. It reduces the processing and training time and resources as the values are usually scaled down by a significant proportion.
2. It makes sure that the network is unbiased. As in it makes the contribution of each feature be equal such that if any feature has a higher numerical value it will not have a greater contribution in influencing the models result.
3. It reduces internal covariance shift.

Q41. Write a python program to prepare the MNIST dataset for training for a vanilla neural network and perform normalization on the dataset.

A41. First, let us import the libraries we need:

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
```

Let us now load the MNIST dataset to create the train and test data sets:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Next, we need to reshape the data sets. A vanilla neural network can only process vectors and not multi-dimensional inputs. So, we need to convert our 28 by 28 pixels image and unroll it into a single vector of 784 numbers.

Also, all pixel values are in the range of 0 to 255. So, to normalize the data, we simply divide all values by 255 to bring the range from 0 to 1:

```
x_train = x_train.reshape(-1, 28*28) / 255
x_test = x_test.reshape(-1, 28*28) / 255
```

Q42. What are the two different APIs provided by keras to create neural networks?

A42. We can create neural networks in keras by using two different methods.

1. By using the Sequential API.
2. By using the Functional API.

Q43. What is the Sequential API in keras to create neural networks?

A43. The Sequential API in keras is a simple way to create layer by layer models of neural networks. But it has limited flexibility, we cannot use Sequential API to create complex or advanced models with features like sharing layers or having multiple inputs and outputs.

Q44. What is the Functional API in keras to create neural networks?

A44. The Functional API in keras can be used to create advanced models that can be tweaked to extreme customization. We can define models with advanced features like connecting one layer to any other layer, multiple inputs outputs etc.

Q45. How should we decide when to use the Sequential API and when to use the Functional API for creating neural networks in keras?

A45. For simple, straightforward networks we should use the Sequential API as it is much easier to create networks with it. For advanced models with any complex custom architecture, we should use the Functional API, as we can define our model with extreme flexibility using the functional API.

Q46. Write a python program to create a simple dense vanilla neural network using the Sequential API in keras for the MNIST data set.

A46. First, let us import the libraries we need:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

Let us now create a simple neural network using the Sequential API in keras. To do so, we simply define our model by adding layer after layer like this:

```
model = keras.Sequential([
        layers.Dense(512, activation = 'relu'),
        layers.Dense(256, activation = 'relu'),
        layers.Dense(10),
])
```

As you can see, we have created a neural network with three dense layers, the first one with 512 neurons and activation function equal to the ReLU function, the second layer with 256 neurons and the activation function equal to the ReLU function again, and the last layer with 10 neurons as there are 10 class labels in the training set and no output function as we want the output to be in the range 0 to 9.

Q47. What is a loss function for a neural network?

A47. A loss function is a differentiable function that compares the value of the predicted value and the real value and calculates the loss value. The objective is to minimize the value of the loss function so that the predictions performed are closer to the actual values.

Q48. What is an optimizer for a neural network?

A48. An optimizer is a method that is used during training of a neural network to boost the training process so as to make it faster and better.

Q49. What is a training metric for a neutral network?

A49. A training metric for a neutral network is simply a metric that is used to keep track of the performance of the neural network while training. It can be thought of as a loss function but that is simply used to observe the training performance it is not actually used to train the neural network.

Q50. Write a python program to compile the model created in the previous question for the MNIST dataset.

A50. To compile the given model, we use the compile function to decide parameters like the loss function, the optimizers and the training metrics:

```
model.compile(
    loss = keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    optimizer = keras.optimizers.Adam(learning_rate = 0.001),
    metrics = ["accuracy"],
)
```

Q51. What does the parameter "epochs" mean in the training of a neural network?

A51. Epochs simply means the number of steps or loops the training will go on for. For example, 5 epochs means that the model will be trained a total of 5 times as in the backpropagation and weight training process will be repeated 5 times.

A small number of epochs may result in the model not getting trained very well, while a very high number of epochs may result in the model overfitting the data and not generalizing well on datasets other than the given training data set.

Q52. What does the parameter "batch size" mean in the training of a neural network?

A52. While training a neural network, it usually takes up a lot of time if we were to train the network on each single example one by one, updating the weights for every single example every time. Instead, it is much faster to take our training examples in sets or batches and train the network on these batches instead of single examples so as to speed up training while considering every single example as well.

Q53. Write a python program to train the model created in the previous questions on the MNIST data set.

A53. To train the created model on the MNIST dataset, we simply use the fit method by keras:

```
model.fit(x_train, y_train, batch_size = 128, epochs = 5)
```

As you can see, we use the model.fit method and pass in the arguments:

1. x_train: The training data set
2. y_train: The labels for the training data set

Along with the batch size, epochs and many other optional arguments if you want to.

The above code might take some time to execute depending upon the dataset, the parameters, the model, and your hardware. After it is done executing, the output should look something like this.

Output:

```
In [4]: from tensorflow import keras
        from tensorflow.keras import layers

In [5]: model = keras.Sequential([
                layers.Dense(512, activation = 'relu'),
                layers.Dense(256, activation = 'relu'),
                layers.Dense(10),
        ])

In [6]: model.compile(
            loss = keras.losses.SparseCategoricalCrossentropy(from_logits = False),
            optimizer = keras.optimizers.Adam(learning_rate = 0.001),
            metrics = ["accuracy"],
        )

In [7]: model.fit(x_train, y_train, batch_size = 128, epochs = 5)
        Epoch 1/5
        469/469 [==============================] - 11s 22ms/step - loss: 2.3209 - accuracy: 0.3079
        Epoch 2/5
        469/469 [==============================] - 10s 21ms/step - loss: 2.3013 - accuracy: 0.2737
        Epoch 3/5
        469/469 [==============================] - 10s 21ms/step - loss: 2.2986 - accuracy: 0.2844
        Epoch 4/5
        469/469 [==============================] - 10s 21ms/step - loss: 2.2954 - accuracy: 0.2902
        Epoch 5/5
        469/469 [==============================] - 10s 21ms/step - loss: 2.2837 - accuracy: 0.3025
Out[7]: <tensorflow.python.keras.callbacks.History at 0x7f59cda376a0>
```

As you can see, during training, you can see the model performance for each epoch, along with the loss function value and the time needed to execute.

Q54. Write a python program to evaluate the performance of the model trained in the previous question on the MNIST test data set.

A54. To evaluate the performance of a model, we can use the evaluate() function given by keras:

```
model.evaluate(x_test, y_test)
```

Output:

```
In [8]: model.evaluate(x_test, y_test)
        313/313 [==============================] - 2s 4ms/step - loss: 2.2796 - accuracy: 0.3115
Out[8]: [2.279564142227173, 0.31150001287460327]
```

As you can see, this method returns two values, the first being the loss function followed by the metrics that we specified when we compiled our model.

Q55. Write a python program to perform predictions using the model that was trained in the previous questions.

A55. To perform predictions using a trained model, we simply use to predict() function and pass in the data that is to be used to perform predictions in it:

```
preds = model.predict(x_test)
```

Let us check out the shapes of the predictions:

First let us print the total length of the predictions array:

```
print(len(preds))
```

Then let us print the length of a single vector in the predictions array:

```
print(len(preds[1]))
```

Output:

```
In [10]: preds = model.predict(x_test)

In [16]: print(len(preds))
         10000

In [18]: print(len(preds[1]))
         10
```

As you can see, the total number of examples in the predictions array is the same as the number of examples in the test set.

Also, the length of a single vector in the predictions array is equal to 10, which is correct as we have 10 class labels as well.

Q56. Write a python program to print the summary of a given model in keras.

A56. To see the summary of any built model in keras, we can simply use the function summary() to look at a detailed summary of our model:

```
model.summary()
```

Output:

```
In [19]: model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
```

As you can see, we can view a detailed summary of our model. We can look at all the layers in the model, with their output shapes. The summary also tells us the number of parameters in each layer, as well as the total of trainable and untrainable parameters present in our model.

Q57. Write a python program to create the same neural network as above but using the Functional API instead of the Sequential API. Also name the individual layers.

A57. To make a model using the functional API, we work using inputs and outputs. Basically we have to define each layer and its function separately, by defining the input, output, and the type of function that layer does sequentially like this:

```
inputs = keras.Input(shape = (784))
x = layers.Dense(512, activation = 'relu')(inputs)
x = layers.Dense(256, activation = 'relu')(x)
outputs = layers.Dense(10)(x)
```

As you can see, we have specified the input first, and later we have defined each layer by passing in the input as the previous layer. It is understandable now how

we can create advanced models using Functional API as we have such extreme customization available. We can define a layer in any way we want.

To have the layers named, we simply add another argument to the layer as name:

```
inputs = keras.Input(shape = (784))
x = layers.Dense(512, activation = 'relu', name = 'layer_number_one')(inputs)
x = layers.Dense(256, activation = 'relu', name = 'layer_number_two')(x)
outputs = layers.Dense(10)(x)
```

To create the model, we use the keras function model() and simply pass in the input and output layers:

```
model = keras.Model(inputs = inputs, outputs = outputs)
```

We compile the model the same way we did using Sequential API:

```
model.compile(
    loss = keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    optimizer = keras.optimizers.Adam(learning_rate = 0.001),
    metrics = ["accuracy"],
)
```

The training, evaluation and prediction is done the same way too:

```
model.fit(x_train, y_train, batch_size = 128, epochs = 5)
model.evaluate(x_test, y_test)
preds = model.predict(x_test)
```

Let us look at this models summary:

```
model.summary()
```

Output:

```
In [20]: model.summary()
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 784)]             0
_________________________________________________________________
layer_number_one (Dense)     (None, 512)               401920
_________________________________________________________________
layer_number_two (Dense)     (None, 256)               131328
_________________________________________________________________
dense_9 (Dense)              (None, 10)                2570
=================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
```

As you can see, it is the same model as we created using the Sequential API.

Also notice that the layers have been named to what we named them.

Q58. Write a python program to print the weights of a given layer in a neural network in keras.

A58. To print the weights of any layer in a neural network, let us say in this case the weights of the second layer of our model, we can use the command:

```
print(model.layers[2].weights)
```

This will return two arrays as output, one will be the weights matrice and one will be the bias array.

Output:

```
In [14]: print(model.layers[2].weights)

[<tf.Variable 'layer_number_two/kernel:0' shape=(512, 256) dtype=float32, numpy=
array([[ 0.00264426, -0.00845679,  0.04483547, ...,  0.01250772,
         0.00982956,  0.06287971],
       [-0.03759135, -0.02116814, -0.01595067, ..., -0.00164443,
        -0.07147584, -0.04212614],
       [-0.01298618,  0.02677501, -0.07217494, ..., -0.03491977,
        -0.02012195, -0.07357954],
       ...,
       [ 0.06431062, -0.04355209, -0.09032113, ...,  0.0370977 ,
         0.07270182,  0.08992044],
       [-0.04300422, -0.03901668,  0.06688859, ...,  0.04472803,
         0.04304439, -0.04219202],
       [ 0.06963155, -0.00698028, -0.0383897 , ..., -0.0574399 ,
         0.02878764,  0.0597105 ]], dtype=float32)>, <tf.Variable 'layer_number_two/bias:0' shape=(256,) dtype=float32, numpy=
array([-3.79727513e-04,  2.80785770e-03, -4.27893782e-03, -9.92004480e-03,
       -7.69847445e-03,  2.37121210e-02, -3.71797476e-03,  8.53351259e-04,
       -8.95366352e-03, -2.57727178e-03,  1.17317867e-02, -9.64223687e-03,
       -7.38561340e-03,  2.52677724e-02, -1.35674216e-02, -8.02221522e-03,
        1.77295450e-02,  2.84363702e-03, -3.78372753e-03, -5.48718637e-03,
       -6.63001137e-03,  1.31556643e-02, -7.81824626e-03, -1.05678812e-02,
        5.96905407e-03,  1.78836305e-02, -6.40073139e-03,  1.40263001e-02,
       -1.16162915e-02,  2.50423094e-04, -3.92036140e-03, -3.61178489e-03,
        6.65404228e-03, -4.84846625e-03, -2.97292857e-03, -8.00761743e-04,
       -5.51185198e-03, -6.29652059e-03, -4.67024604e-03, -1.55043753e-03,
       -7.30547635e-03, -1.88424985e-03,  1.87372565e-02,  3.34769045e-03,
        1.52609693e-02,  2.39739176e-02, -5.98324090e-03, -3.31243686e-03,
        2.44235434e-02, -3.73458350e-03, -2.61687487e-03,  1.46855200e-02,
       -5.98935643e-03, -5.99449268e-03,  1.17057664e-02, -3.94705869e-03,
       -5.53110288e-03, -6.24365523e-04,  3.58476024e-03, -1.09915491e-02,
        6.44832291e-03, -3.25635960e-03,  1.08361188e-02, -3.23038269e-03,
       -2.88832583e-03,  6.08773786e-04, -1.06991257e-03, -4.74919099e-03,
        3.77186341e-03, -7.23566348e-03, -7.12394016e-03, -5.96329058e-03,
        7.97388516e-03, -3.17730708e-03, -7.85770267e-03,  1.44834640e-02,
       -6.71596546e-03,  2.69036684e-02, -6.68575056e-03, -3.84201901e-03,
```

Q59. Write a python program to print the weights of a layer of a neural network by calling it by the layer name given to it.

A59. We can print the weights of a layer of a neural network by using the name given to it in the following way:

```
print(model.get_layer('layer_number_two').weights)
```

This will return the same output as above.

Output:

```
In [17]: print(model.get_layer('layer_number_two').weights)

[<tf.Variable 'layer_number_two/kernel:0' shape=(512, 256) dtype=float32, numpy=
array([[ 0.00264426, -0.00845679,  0.04483547, ...,  0.01250772,
         0.00982956,  0.06287971],
       [-0.03759135, -0.02116814, -0.01595067, ..., -0.00164443,
        -0.07147584, -0.04212614],
       [-0.01298618,  0.02677501, -0.07217494, ..., -0.03491977,
        -0.02012195, -0.07357954],
       ...,
       [ 0.06431062, -0.04355209, -0.09032113, ...,  0.0370977 ,
         0.07270182,  0.08992044],
       [-0.04300422, -0.03901668,  0.06688859, ...,  0.04472803,
         0.04304439, -0.04219202],
       [ 0.06963155, -0.00698028, -0.0383897 , ..., -0.0574399 ,
         0.02878764,  0.0597105 ]], dtype=float32)>, <tf.Variable 'layer_number_two/bias:0' shape=(256,) dtype=float32, numpy=
array([-3.79727513e-04,  2.80785770e-03, -4.27893782e-03, -9.92004480e-03,
       -7.69847445e-03,  2.37121210e-02, -3.71797476e-03,  8.53351259e-04,
       -8.95366352e-03, -2.57727178e-03,  1.17317867e-02, -9.64223687e-03,
       -7.38561340e-03,  2.52677724e-02, -1.35674216e-02, -8.02221522e-03,
        1.77295450e-02,  2.84363702e-03, -3.78372753e-03, -5.48718637e-03,
       -6.63001137e-03,  1.31556643e-02, -7.81824626e-03, -1.05678812e-02,
        5.96905407e-03,  1.78836305e-02, -6.40073139e-03,  1.40263001e-02,
       -1.16162915e-02,  2.50423094e-04, -3.92036140e-03, -3.61178489e-03,
        6.65404228e-03, -4.84846625e-03, -2.97292857e-03, -8.00761743e-04,
       -5.51185198e-03, -6.29652059e-03, -4.67024604e-03, -1.55043753e-03,
       -7.30547635e-03, -1.88424985e-03,  1.87372565e-02,  3.34769045e-03,
        1.52609693e-02,  2.39739176e-02, -5.98324090e-03, -3.31243686e-03,
        2.44235434e-02, -3.73458350e-03, -2.61687487e-03,  1.46855200e-02,
       -5.98935643e-03, -5.99449268e-03,  1.17057664e-02, -3.94705869e-03,
       -5.53110288e-03, -6.24365523e-04,  3.58476024e-03, -1.09915491e-02,
        6.44832291e-03, -3.25635960e-03,  1.08361188e-02, -3.23038269e-03,
       -2.88832583e-03,  6.08773786e-04, -1.06991257e-03, -4.74919099e-03,
        3.77186341e-03, -7.23566348e-03, -7.12394016e-03, -5.96329058e-03,
        7.97388516e-03, -3.17730708e-03, -7.85770267e-03,  1.44834640e-02,
       -6.71596546e-03,  2.69036684e-02, -6.68575056e-03, -3.84201901e-03,
```

Q60. Write a python program to print the outputs of any given layer in a neural network in keras.

A60. To see the outputs of a certain layer of a neural network in keras, we can define a new model that has the same inputs but the outputs are upto that specific layer only, and then we can perform predictions on that new model using the same outputs to find out the outputs of that layer.

First, let us define the new model:

```
model_2 = keras.Model(inputs = model.inputs, outputs = model.layers[2].output)
```

Let us look at this models summary:

```
model.summary()
```

Output:

```
In [19]: model_2 = keras.Model(inputs = model.inputs, outputs = model.layers[2].output)

In [20]: model_2.summary()
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 784)]             0
_________________________________________________________________
layer_number_one (Dense)     (None, 512)               401920
_________________________________________________________________
layer_number_two (Dense)     (None, 256)               131328
=================================================================
Total params: 533,248
Trainable params: 533,248
Non-trainable params: 0
_________________________________________________________________
```

As you can see, this model is the same as our last model except that it has excluded the last layer.

To see its outputs, we simply perform predictions the same way we did for other models:

```
preds = model_2.predict(x_test)
```

Let us look at the shapes of our predictions:

```
print(preds.shape)
```

Output:

```
In [23]: preds = model_2.predict(x_test)

In [24]: print(preds.shape)
         (10000, 256)
```

As you can see, the output vector has 256 values, same as the length of the second layer in our neural network.

Q61. What are Convolutional Neural Networks?

A61. Convolutional Neural Networks are a special kind of Neural Networks usually used in image recognition and other computer vision applications. They use a special kind of matrix operation known as Convolution that helps identify features in a matrix rather than a row vector.

Q62. Write a python program to create a Convolutional Neural Network in keras for the MNIST dataset.

A62. First, let us import the libraries required:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras import layers
```

Let us prepare our data:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1) / 255
x_test = x_test.reshape(-1, 28, 28, 1) / 255
```

Note that we have not unrolled the data set here and kept it as a 2-D matrix. We also had to add another dimension as for convolution here we need to have 4 dimensions for our input tensor.

Let us create our model:

```
model = keras.Sequential([
    keras.Input(shape = (28, 28, 1)),
    layers.Conv2D(32, 1, padding = 'valid', activation = 'relu'),
    layers.MaxPooling2D(pool_size = (2, 2)),
    layers.Conv2D(64, 1, activation = 'relu'),
    layers.MaxPooling2D(),
```

```
    layers.Conv2D(128, 1, activation = 'relu'),
    layers.Flatten(),
    layers.Dense(64, activation = 'relu'),
    layers.Dense(10),
])
```

Let us look at our model summary:

```
print(model.summary())
```

Output:

```
In [4]: print(model.summary())
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        64
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        2112
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 128)         8320
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 64)                401472
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 412,618
Trainable params: 412,618
Non-trainable params: 0
_________________________________________________________________
None
```

Next let us compile our model:

```
model.compile(
    loss = keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    optimizer = keras.optimizers.Adam(learning_rate = 0.001),
```

```
    metrics = ["accuracy"],
)
```

Let us train the model now:

```
model.fit(x_train, y_train, batch_size = 128, epochs = 5)
```

Note that this will take much longer to train than our previous model as this model is much bigger.

Finally, let us look at the predictions of our model:

```
preds = model.predict(x_test)
print(preds.shape)
```

Output:

```
In [5]: model.compile(
            loss = keras.losses.SparseCategoricalCrossentropy(from_logits = False),
            optimizer = keras.optimizers.Adam(learning_rate = 0.001),
            metrics = ["accuracy"],
        )

In [6]: model.fit(x_train, y_train, batch_size = 128, epochs = 5)

        Epoch 1/5
        469/469 [==============================] - 75s 159ms/step - loss: 2.5159 - accuracy: 0.3170
        Epoch 2/5
        469/469 [==============================] - 73s 156ms/step - loss: 2.3026 - accuracy: 0.3083
        Epoch 3/5
        469/469 [==============================] - 70s 150ms/step - loss: 2.3026 - accuracy: 0.3083
        Epoch 4/5
        469/469 [==============================] - 70s 150ms/step - loss: 2.3026 - accuracy: 0.3083
        Epoch 5/5
        469/469 [==============================] - 71s 151ms/step - loss: 2.3026 - accuracy: 0.3083

Out[6]: <tensorflow.python.keras.callbacks.History at 0x7fbab6523a20>

In [9]: preds = model.predict(x_test)

In [10]: print(preds.shape)

         (10000, 10)
```

Q63. What is a Recurrent Neural Network?

A63. A recurrent neural network is a type of neural network that is used for sequence models. It uses a type of architecture where in it considers not just the input at the current time step but also the hidden state of the previous time step to calculate the output at the current time step. There are different types of RNNs, like a simple RNN, LSTMs, GRUs.

Q64. Write a python program to create a LSTM model for the MNIST dataset in keras.

A64. First, let us import the libraries required:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras import layers
```

Let us prepare our data:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255
x_test = x_test / 255
```

Note that we have not unrolled the data set here and kept it as a 2-D matrix.

Let us create our model:

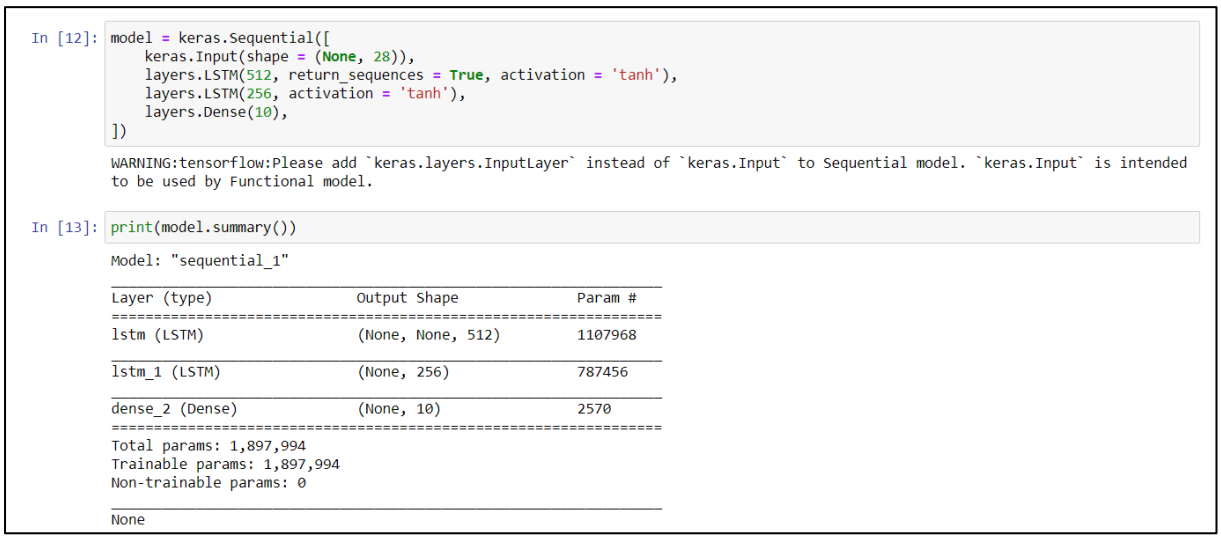
```
model = keras.Sequential([
    keras.Input(shape = (None, 28)),
    layers.LSTM(512, return_sequences = True, activation = 'tanh'),
    layers.LSTM(256, activation = 'tanh'),
    layers.Dense(10),
])
```

Note that our input shape here is simply a vector of 28. This means that the model will be looking at each single row pixels of the image as an input at each time step.

Let us look at our model summary:

```
print(model.summary())
```

Output:



```
In [12]: model = keras.Sequential([
             keras.Input(shape = (None, 28)),
             layers.LSTM(512, return_sequences = True, activation = 'tanh'),
             layers.LSTM(256, activation = 'tanh'),
             layers.Dense(10),
         ])

         WARNING:tensorflow:Please add `keras.layers.InputLayer` instead of `keras.Input` to Sequential model. `keras.Input` is intended
         to be used by Functional model.

In [13]: print(model.summary())

         Model: "sequential_1"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         lstm (LSTM)                  (None, None, 512)         1107968
         _________________________________________________________________
         lstm_1 (LSTM)                (None, 256)               787456
         _________________________________________________________________
         dense_2 (Dense)              (None, 10)                2570
         =================================================================
         Total params: 1,897,994
         Trainable params: 1,897,994
         Non-trainable params: 0
         _________________________________________________________________
         None
```

Next let us compile our model:

```
model.compile(
    loss = keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    optimizer = keras.optimizers.Adam(learning_rate = 0.001),
    metrics = ["accuracy"],
)
```

Let us train the model now:

```
model.fit(x_train, y_train, batch_size = 128, epochs = 5)
```

Note that this will take even longer to train than the convolutional model as this model is much bigger.

Finally, let us look at the predictions of our model:

```
preds = model.predict(x_test)
print(preds.shape)
```

Output:

```
In [17]: preds = model.predict(x_test)

In [18]: print(preds.shape)
         (10000, 10)
```

Q65. What is subclassing in keras?

A65. In keras, we can use classes and subclasses to define extremely custom models according to our requirement. We can use subclasses to define each single step of a layer, customizing every single function and working of the layer.

Q66. Write a python program demonstrating custom layers using subclasses using keras for the same MNIST problem.

A66. First, let us import the libraries required:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras import layers
```

Let us prepare our data:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1) / 255
x_test = x_test.reshape(-1, 28, 28, 1) / 255
```

Here we have 4 dimensions because we will be defining a custom layer that will use the CNN architecture.

Let us define our custom layer using subclasses now:

```
class custom_layer(layers.Layer):
    def __init__(self, out_channels, kernel_size=3):
        super(custom_layer, self).__init__()
        self.conv = layers.Conv2D(out_channels, kernel_size, padding="same")

    def call(self, input_tensor, training=False):
        x = self.conv(input_tensor)
        x = tf.nn.relu(x)
        return x
```

As you can see, you can define any kind of a procedure or function in the classes to create a layer.

Let us create a model using this layer now:

```
model = keras.Sequential([
    custom_layer(32),
    custom_layer(64),
    custom_layer(128),
    layers.Flatten(),
    layers.Dense(10),
])
```

Let us look at our model summary:

```
print(model.summary())
```

Output:

```
In [13]: print(model.summary())

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
custom_layer_3 (custom_layer (None, 28, 28, 32)        320
_________________________________________________________________
custom_layer_4 (custom_layer (None, 28, 28, 64)        18496
_________________________________________________________________
custom_layer_5 (custom_layer (None, 28, 28, 128)       73856
_________________________________________________________________
flatten_1 (Flatten)          (None, 100352)            0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1003530
=================================================================
Total params: 1,096,202
Trainable params: 1,096,202
Non-trainable params: 0
_________________________________________________________________
None
```

Next let us compile our model:

```
model.compile(
    loss = keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    optimizer = keras.optimizers.Adam(learning_rate = 0.001),
    metrics = ["accuracy"],
)
```

Let us train the model now:

```
model.fit(x_train, y_train, batch_size = 128, epochs = 5)
```

Finally, let us look at the predictions of our model:

```
preds = model.predict(x_test)
print(preds.shape)
```

Output:

```
In [14]: preds = model.predict(x_test)

In [15]: print(preds.shape)
         (10000, 10)
```

Q67. Write a python program to save a created model in tensorflow.

A67. To save a created model in tensorflow, we simply use the save function:

```
model.save("file_path")
```

Q68. Write a python program to save the weights of a created model in tensorflow.

A68. To save the weights of a created model in tensorflow, we simply use the save weights function:

```
model.save_weights("file_path")
```

Q69. Write a python program to load a saved model in tensorflow.

A69. To load a saved model in tensorflow, we simply use the load model function:

```
model = keras.models.load_model("saved_model_path")
```

Q70. Write a python program to load the weights of a saved model in tensorflow.

A70. To load the weights of a saved model in tensorflow, we simply use the load weights function:

```
model.load_weights("saved_model_path")
```