Name - Chirag N
Class - 4MScDS-A
Reg no - 23122013

Whats happening in this piece of code?

```
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = np.array(x_train, np.float32).reshape([-1, num_features]) / 255.
x_test = np.array(x_test, np.float32).reshape([-1, num_features]) / 255.

x_train, x_test = x_train / 255., x_test / 255.
```

Firstly, the dataset is loaded
Now, we are using np.array function to create an array of size similar to that of x_train s1ze which will have images pixel stored in unsigned 8 bit integer format which is mapped to float32 data type. And then the 28*28(784) elements are flattened to form a single line of elements or inputs which we are gonna feed it to the nn. Here, -1 is used to specify the size of the dimension.
MNIST images have pixel value ranging between 0 and 255 so we now normalize it by dividing it with 255 so that the numbers lie between 0 and 1.

Whats happening in this piece of code? Basically data pipeline

```
def update_train_data(batch_size):
    train_data = tf.data.Dataset.from_tensor_slices((x_train, y_train))
    train_data =
train_data.repeat().shuffle(5000).batch(batch_size).prefetch(1)
    return train_data

train_data = update_train_data(batch_size)
```

This function converts the array into tensorflow dataset.
This function **tf.data.Dataset.from_tensor_slices** especially convert the array into a dataset. Later the dataset is repeated shuffled and divided into different batches of same size. .repeat() function is used to repeat the dataset.

Ex. **Original Dataset**: [1, 2, 3, 4, 5], **Repeated Dataset**: [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, ...] (repeats indefinitely)

.shuffle() is used to shuffle the repeat data.
Ex. **Shuffled Dataset**: [4, 1, 3, 5, 2] (after shuffle)
Repeated Shuffled Dataset: [4, 1, 3, 5, 2, 4, 1, 3, 5, 2, ...]

Whats happening in this piece of code?

```
def create_model(activation=tf.nn.relu):
    class NeuralNet(Model):
        def __init__(self):
            super(NeuralNet, self).__init__()
            self.fc1 = layers.Dense(n_hidden_1, activation=activation)
            self.fc2 = layers.Dense(n_hidden_2, activation=activation)
            self.out = layers.Dense(num_classes)

        def call(self, x, is_training=False):
            x = self.fc1(x)
            x = self.fc2(x)
            x = self.out(x)
            return x

    return NeuralNet()
```

Initializing the model with 2 hidden layers and one output layer.
N_hidden_1 and n_hidden_2 are the number of neurons in each hidden layer.

“Call” function defines the forward pass. It receives the input data and tells how the input data is transformed by each layer of the model.

The self.dense function applies linear combination and the activation function is linear by default which ,eans nothing.

Whats happening in this piece of code?

```
def cross_entropy_loss(x, y):
    y = tf.cast(y, tf.int64)
    loss = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,
logits=x)
    return tf.reduce_mean(loss)
```

X and y are predicted and true value respectively.
Tf.cast is used to convert one dta type to another. Here, the true values are converted into int because **nn.sparse_softmax_cross_entropy_with_logits** function expects integer function.
tf.nn.sparse_softmax_cross_entropy_with_logits is a TensorFlow function that helps you measure how well your model's predictions match the actual results. Basically it checks if the model prediction matches with the result or not. **Soft_max is used to convert predictions into probabilities.**

Whats happening in this piece of code?

```
def accuracy(y_pred, y_true):
      y_pred = tf.nn.softmax(y_pred)
      correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.cast(y_true,
tf.int64))
      return tf.reduce_mean(tf.cast(correct_prediction, tf.float32),
axis=-1)
```

This basically checks the accuracy here.
Soft_max is used to convert predictions into probabilities.

tf.argmax(y_pred, 1): This finds the class with the highest probability for each sample. For example, if the probabilities are [0.7, 0.2, 0.1], tf.argmax will pick the class with the highest value, which is 0 in this case (assuming class 0 has the highest probability).
tf.cast(y_true, tf.int64): This makes sure the true labels are in the right format (integers).
tf.equal(...): This checks if the predicted class is the same as the true class for each sample. It gives True if they match and False if they don't.

tf.cast(correct_prediction, tf.float32): Converts the True/False values into 1.0/0.0. True becomes 1.0 and False becomes 0.0.
tf.reduce_mean(...): Averages these values. This gives the fraction of correct predictions out of all predictions, which is the accuracy.

Whats happening in this piece of code?

```
optimizer = tf.optimizers.SGD(learning_rate)

def run_optimization(x, y):
    with tf.GradientTape() as g:
        pred = neural_net(x, is_training=True)
        loss = cross_entropy_loss(pred, y)
    trainable_variables = neural_net.trainable_variables
    gradients = g.gradient(loss, trainable_variables)
    optimizer.apply_gradients(zip(gradients, trainable_variables))
```

tf.GradientTape(): A tool to track operations on tensors so you can later compute gradients.
apply_gradients(): Updates model weights using computed gradients.

Whats happening in this piece of code?

```
while True:
    print("\nMenu:")
    print("1. Change learning rate")
    print("2. Change training steps")
    print("3. Change batch size")
    print("4. Change activation function")
    print("5. Start training")
    print("6. Visualize predictions")
    print("7. Upload and predict")
    print("8. Exit")

    choice = input("Enter your choice: ")

    if choice == '1':
        learning_rate = float(input("Enter new learning rate: "))
        optimizer.learning_rate = learning_rate
    elif choice == '2':
        training_steps = int(input("Enter new training steps: "))
    elif choice == '3':
        batch_size = int(input("Enter new batch size: "))
        train_data = update_train_data(batch_size)
    elif choice == '4':
        print("\nAvailable activation functions:")
        print("1. relu")
        print("2. sigmoid")
        print("3. tanh")
        print("4. linear")

        activation_choice = input("Enter your choice: ")
        if activation_choice == '1':
            activation = tf.nn.relu
        elif activation_choice == '2':
            activation = tf.nn.sigmoid
        elif activation_choice == '3':
            activation = tf.nn.tanh
        elif activation_choice == '4':
            activation = None
        else:
            print("Invalid choice. Using default activation (relu).")
            activation = tf.nn.relu

        neural_net = create_model(activation)
    elif choice == '5':
        train_model()
    elif choice == '6':
        visualize_predictions()
    elif choice == '7':
        upload_and_predict()
    elif choice == '8':
        break
    else:
        print("Invalid choice.")
```

Menu driven part.