Theory Questions :

1. What is TensorFlow 2.0, and how is it different from TensorFlow 1.x?

TensorFlow 2.0 is a major upgrade of TensorFlow that emphasizes simplicity and ease of use. Key differences from 1.x:

- Eager execution is enabled by default (like PyTorch)
- Unified tf.keras high-level API
- Removal of deprecated functions and redundancy
- Better support for custom training loops

2 How do you install TensorFlow 2.0?

pip install tensorflow

3. What is the primary function of tf.function in TensorFlow 2.0?

tf.function is used to convert a Python function into a TensorFlow graph, improving performance and enabling optimization.

4. What is the purpose of the Model class in TensorFlow 2.0?

The Model class (from tf.keras.Model) is used to define custom neural networks, enabling you to override the call() method and add custom behavior.

5 How do you create a neural network using TensorFlow 2.0?

Use the Sequential API:

```
model = tf.keras.Sequential([
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(10)
])
```

6. What is the importance of Tensor space in TensorFlow?

Tensor space refers to the **multidimensional array structure** where computations are performed. It allows efficient mathematical operations on scalars, vectors, matrices, and higher-dimensional data.

7. How can TensorBoard be integrated with TensorFlow 2.0?

TensorBoard can be integrated using callbacks:

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs")

model.fit(..., callbacks=[tensorboard_callback])

Then run:

tensorboard --logdir=./logs

8. What is the purpose of TensorFlow Playground?

TensorFlow Playground is a web-based visualization tool to experiment with neural networks in a browser, useful for educational purposes.

9. What is Netron, and how is it useful for deep learning models?

Netron is a model visualizer that allows you to inspect the architecture of deep learning models (e.g., TensorFlow, PyTorch, ONNX) in a graphical interface.

10. What is the difference between TensorFlow and PyTorch?

Feature	**TensorFlow 2.0**	**PyTorch**
Execution	Static (Graph, but supports eager)	Dynamic (eager by default)
Debugging	Harder	Easier (Pythonic)
Community	Broad (Google)	Growing (Meta/Facebook)
API	tf.keras	Pythonic OOP

11. How do you install PyTorch?

Basic CPU version:

pip install torch torchvision

12. What is the basic structure of a PyTorch neural network?

- Subclass nn.Module
- Define layers in __init__
- Define forward pass in forward

Example:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc = nn.Linear(10, 1)

    def forward(self, x):
        return self.fc(x)
```

13. What is the significance of tensors in PyTorch?

Tensors are the core data structures in PyTorch used for storing data and performing computations, just like NumPy arrays but with GPU acceleration.

14. What is the difference between torch.Tensor and torch.cuda.Tensor in PyTorch?

- torch.Tensor: Resides on **CPU**
- torch.cuda.Tensor: Resides on **GPU**

Convert with:

tensor.cuda()

tensor.cpu()

15. What is the purpose of the torch.optim module in PyTorch?

torch.optim provides optimization algorithms like SGD, Adam, RMSprop to minimize loss functions and update model weights.

16. What are some common activation functions used in neural networks?

- **ReLU**: Rectified Linear Unit (most common)
- **Sigmoid**: Output between 0 and 1 (binary classification)
- **Tanh**: Output between -1 and 1
- **Softmax**: For multi-class classification

17. What is the difference between torch.nn.Module and torch.nn.Sequential in PyTorch?

- nn.Module: For custom models, flexible
- nn.Sequential: For simple feedforward models, layers in sequence

18. How can you monitor training progress in TensorFlow 2.0?

- Use TensorBoard for real-time visualization
- Use callbacks like ModelCheckpoint, EarlyStopping
- Monitor loss, accuracy during model.fit

19. How does the Keras API fit into TensorFlow 2.0?

Keras is fully integrated into TensorFlow 2.0 as tf.keras, serving as the high-level API for building, training, and evaluating models.

20. What is an example of a deep learning project that can be implemented using TensorFlow 2.0?

Image Classification using CNNs on CIFAR-10 or MNIST dataset, using tf.keras, with TensorBoard integration and deployment via TensorFlow Lite.

21. What is the main advantage of using pre-trained models in TensorFlow and PyTorch?

- Faster training
- Improved performance
- Works well with small datasets
- Enables transfer learning

Models: ResNet, VGG, BERT, MobileNet, etc.

0.1 1) How do you install and verify that TensorFlow 2.0 was installed successfully

[ ]:
```
!pip install tensorflow
import tensorflow as tf
print(tf.__version__)  # Should be 2.x.x
```

0.2 2) How can you define a simple function in TensorFlow 2.0 to perform addition

[4]:
```
import tensorflow as tf

@tf.function
def add(a, b):
    return a + b

print(add(tf.constant(5), tf.constant(3)))  # Output: 8
```

```
tf.Tensor(8, shape=(), dtype=int32)
```

0.3 3) How can you create a simple neural network in TensorFlow 2.0 with one hidden layer

[6]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import warnings
warnings.filterwarnings('ignore')
model = Sequential([
    Dense(16, activation='relu', input_shape=(4,)),  # Hidden layer
    Dense(3, activation='softmax')                   # Output layer
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
```

```
Layer (type)                         Output Shape                ␣
↪Param #

dense (Dense)                        (None, 16)                  ␣
↪ 80

dense_1 (Dense)                      (None, 3)                   ␣
↪ 51


Total params: 131 (524.00 B)

Trainable params: 131 (524.00 B)

Non-trainable params: 0 (0.00 B)
```

0.4 4) How can you visualize the training progress using TensorFlow and Matplotlib

[1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load and preprocess the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Normalize the data
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split into train and validation
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Build the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='relu', input_shape=(4,)),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

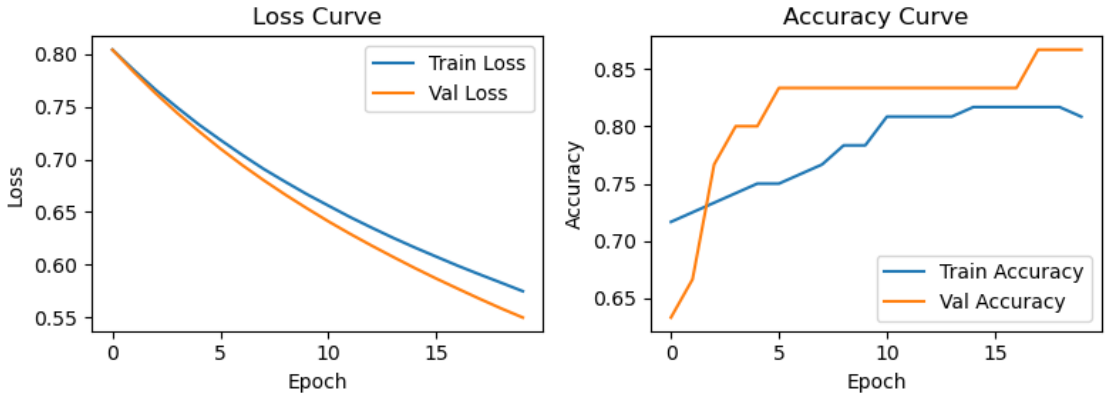
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model and capture history
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val))

# Plot loss and accuracy
plt.figure(figsize=(8, 3))

# Loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Loss Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Accuracy Curve')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

```
C:\Users\kumar\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
4/4             3s 203ms/step -
accuracy: 0.7106 - loss: 0.8142 - val_accuracy: 0.6333 - val_loss: 0.8041
Epoch 2/20
4/4             0s 53ms/step -
accuracy: 0.7317 - loss: 0.7665 - val_accuracy: 0.6667 - val_loss: 0.7828
Epoch 3/20
4/4             0s 61ms/step -
accuracy: 0.7496 - loss: 0.7652 - val_accuracy: 0.7667 - val_loss: 0.7627
Epoch 4/20
4/4             0s 56ms/step -
```

```
accuracy: 0.7727 - loss: 0.7167 - val_accuracy: 0.8000 - val_loss: 0.7440
Epoch 5/20
4/4             0s 54ms/step -
accuracy: 0.7323 - loss: 0.7493 - val_accuracy: 0.8000 - val_loss: 0.7268
Epoch 6/20
4/4             0s 56ms/step -
accuracy: 0.7604 - loss: 0.7139 - val_accuracy: 0.8333 - val_loss: 0.7102
Epoch 7/20
4/4             0s 54ms/step -
accuracy: 0.7815 - loss: 0.6912 - val_accuracy: 0.8333 - val_loss: 0.6947
Epoch 8/20
4/4             0s 51ms/step -
accuracy: 0.7796 - loss: 0.6924 - val_accuracy: 0.8333 - val_loss: 0.6802
Epoch 9/20
4/4             0s 59ms/step -
accuracy: 0.7883 - loss: 0.6542 - val_accuracy: 0.8333 - val_loss: 0.6665
Epoch 10/20
4/4             0s 62ms/step -
accuracy: 0.7654 - loss: 0.6698 - val_accuracy: 0.8333 - val_loss: 0.6536
Epoch 11/20
4/4             0s 54ms/step -
accuracy: 0.7931 - loss: 0.6686 - val_accuracy: 0.8333 - val_loss: 0.6411
Epoch 12/20
4/4             0s 55ms/step -
accuracy: 0.8244 - loss: 0.6246 - val_accuracy: 0.8333 - val_loss: 0.6293
Epoch 13/20
4/4             0s 63ms/step -
accuracy: 0.8223 - loss: 0.6294 - val_accuracy: 0.8333 - val_loss: 0.6181
Epoch 14/20
4/4             0s 53ms/step -
accuracy: 0.7869 - loss: 0.6433 - val_accuracy: 0.8333 - val_loss: 0.6073
Epoch 15/20
4/4             0s 49ms/step -
accuracy: 0.8204 - loss: 0.5989 - val_accuracy: 0.8333 - val_loss: 0.5968
Epoch 16/20
4/4             0s 44ms/step -
accuracy: 0.8204 - loss: 0.6110 - val_accuracy: 0.8333 - val_loss: 0.5867
Epoch 17/20
4/4             0s 41ms/step -
accuracy: 0.8173 - loss: 0.6041 - val_accuracy: 0.8333 - val_loss: 0.5770
Epoch 18/20
4/4             0s 65ms/step -
accuracy: 0.8277 - loss: 0.5942 - val_accuracy: 0.8667 - val_loss: 0.5675
Epoch 19/20
4/4             0s 59ms/step -
accuracy: 0.8423 - loss: 0.5779 - val_accuracy: 0.8667 - val_loss: 0.5584
Epoch 20/20
4/4             0s 60ms/step -
```

```
accuracy: 0.7942 - loss: 0.5799 - val_accuracy: 0.8667 - val_loss: 0.5497
```



0.5 5) How do you install PyTorch and verify the PyTorch installation

[10]:
```
model.save("my_model.keras")  # Recommended


from tensorflow.keras.models import load_model

model = load_model('my_model.keras')
```

0.6 6) How do you create a simple neural network in PyTorch

[5]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# 1. Data Transformation and Loading
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.
 ↪5,), (0.5,))])
train_data = datasets.MNIST(root='./data', train=True, transform=transform,␣
 ↪download=True)
test_data = datasets.MNIST(root='./data', train=False, transform=transform)

train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)

# 2. Define the Neural Network
class SimpleNN(nn.Module):
```

```
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# 3. Initialize Model, Loss Function, Optimizer
model = SimpleNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 4. Train the Model
epochs = 5
for epoch in range(epochs):
    running_loss = 0.0
    for images, labels in train_loader:
        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss/len(train_loader):.
 ↪4f}")

# 5. Evaluate on Test Data
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Test Accuracy: {100 * correct / total:.2f}%")
```

100%|

```
 | 9.91M/9.91M [03:29<00:00, 47.3kB/s]
100%|
 | 28.9k/28.9k [00:00<00:00, 98.4kB/s]
100%|
 | 1.65M/1.65M [00:24<00:00, 67.8kB/s]
100%|
 | 4.54k/4.54k [00:00<00:00, 29.9kB/s]
```

```
Epoch [1/5], Loss: 0.3897
Epoch [2/5], Loss: 0.1825
Epoch [3/5], Loss: 0.1326
Epoch [4/5], Loss: 0.1056
Epoch [5/5], Loss: 0.0936
Test Accuracy: 96.87%
```

0.7 7) How do you define a loss function and optimizer in PyTorch

[6]:
```python
import torch.nn as nn
import torch.nn.functional as F

class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(4, 16)
        self.fc2 = nn.Linear(16, 3)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        return F.log_softmax(self.fc2(x), dim=1)

model = SimpleNN()
print(model)
```

```
SimpleNN(
  (fc1): Linear(in_features=4, out_features=16, bias=True)
  (fc2): Linear(in_features=16, out_features=3, bias=True)
)
```

0.8 8) How do you implement a custom loss function in PyTorch

[9]:
```python
import torch

def custom_mse_loss(output, target):
    return torch.mean((output - target) ** 2)

# Example usage
output = torch.tensor([0.9, 0.2, 0.1], requires_grad=True)
target = torch.tensor([1.0, 0.0, 0.0])
```

```
loss = custom_mse_loss(output, target)
print(loss)
```

```
tensor(0.0200, grad_fn=<MeanBackward0>)
```

0.9 9) How do you save and load a TensorFlow model?

[ ]:
```
model.save('my_model.keras')

from tensorflow.keras.models import load_model

model = load_model('my_model.keras')
```

[ ]:
```

```