PyTorch & TorchVision-Interview-Questions-Answers

1. What is PyTorch?

→ Open-source deep learning library with dynamic computation graph, used for building and training neural networks.

2. What is TorchVision?

→ PyTorch library for computer vision; provides datasets, transforms, and pretrained models.

3. Difference between TensorFlow and PyTorch?

→ PyTorch uses dynamic computation graphs; TensorFlow uses static (though TF 2.x supports eager execution).

4. What is a Tensor?

→ Multi-dimensional array similar to NumPy array; can run on GPU.

5. How to move tensors to GPU?

→ `tensor.to(device)` or `tensor.cuda()` .

6. What is `requires_grad=True` ?

→ Tells PyTorch to track operations for gradient computation during backpropagation.

7. What is `.backward()` ?

→ Computes gradients of tensor w.r.t. loss for all parameters.

8. What is `torch.nn.Module` ?

→ Base class for all neural networks in PyTorch; contains layers and forward method.

9. Difference between `torch.nn.functional` and `torch.nn.Module` layers?

→ `functional` is stateless functions; `Module` layers have weights and can be trained.

10. What is an optimizer in PyTorch?

→ Updates model weights based on gradients (SGD, Adam, RMSprop).

11. How to define a simple ANN in PyTorch?

→ Subclass `nn.Module` and define layers in `__init__` and forward pass in `forward()`.

12. How to compute parameters of layers?

→ Linear layer: `(input_features * output_features) + bias`.

13. How does CNN work in PyTorch?

→ Use `nn.Conv2d` for convolution → `nn.MaxPool2d` for pooling → `nn.Flatten` → `nn.Linear`.

14. CNN parameter formula

→ `(kernel_h * kernel_w * in_channels + 1) * out_channels`.

15. RNN basics in PyTorch

→ `nn.RNN`, `nn.LSTM`, `nn.GRU` handle sequential data; maintains hidden states across time steps.

16. LSTM vs GRU

→ LSTM → input, forget, output gates; GRU → simpler, combines gates.

17. Transfer learning workflow

→ Load pretrained model from `torchvision.models` → freeze layers → modify classifier → train on new dataset.

18. How to use pretrained ResNet?

→ `from torchvision.models import resnet50; model = resnet50(pretrained=True)`

19. Difference between `model.eval()` and `model.train()`

→ `train()` → enables dropout & batchnorm training; `eval()` → disables them for evaluation.

20. How to normalize images?

→ Use `torchvision.transforms.Normalize(mean, std)`.

21. What is DataLoader?

→ Efficiently loads data in batches, supports shuffling, multiprocessing.

22. Difference between Dataset and DataLoader

→ Dataset → stores data & labels; DataLoader → creates batches, shuffles, parallel loading.

23. How to save and load models?

→ `torch.save(model.state_dict(), path)` and `model.load_state_dict(torch.load(path))` .

24. How to implement Dropout?

→ `nn.Dropout(p=0.5)` in layers; reduces overfitting.

25. How to handle GPU & CPU device placement?

→ `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")` → `model.to(device)` .

26. Create a tensor

```
import torch
x = torch.randn(3,4, requires_grad=True)
```

27. Move tensor to GPU

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
x = x.to(device)
```

28. Define simple ANN

```
import torch.nn as nn
class ANN(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(10, 16)
        self.fc2 = nn.Linear(16, 1)
    def forward(self, x):
        x = torch.relu(self.fc1(x))
        return torch.sigmoid(self.fc2(x))
```

29. Define CNN

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.pool = nn.MaxPool2d(2,2)
        self.fc1 = nn.Linear(32*15*15, 10)
    def forward(self,x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = torch.flatten(x,1)
        return self.fc1(x)
```

30. Define LSTM

```
class LSTMModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.lstm = nn.LSTM(input_size=10, hidden_size=50, batch_firs
t=True)
        self.fc = nn.Linear(50, 1)
    def forward(self,x):
        out, _ = self.lstm(x)
        out = out[:,-1,:]
        return torch.sigmoid(self.fc(out))
```

31. Loss & optimizer

```
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

32. Training loop

```
for epoch in range(5):
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()
```

33. Evaluate model

```
model.eval()
with torch.no_grad():
    y_pred = (model(X_test) > 0.5).float()
```

34. DataLoader example

```
from torch.utils.data import DataLoader, TensorDataset
dataset = TensorDataset(X_train, y_train)
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

35. TorchVision transforms

```
from torchvision import transforms
transform = transforms.Compose([transforms.Resize((64,64)),
                                transforms.ToTensor(),
                                transforms.Normalize((0.5,0.5,0.5),
(0.5,0.5,0.5))])
```

36. Load CIFAR10 dataset

```
from torchvision.datasets import CIFAR10
train_set = CIFAR10(root='./data', train=True, download=True, transfo
rm=transform)
train_loader = DataLoader(train_set, batch_size=32, shuffle=True)
```

37. Freeze pretrained layers

```
from torchvision.models import resnet18
model = resnet18(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
```

38. Modify classifier

```
import torch.nn as nn
model.fc = nn.Linear(model.fc.in_features, 10)
```

39. Forward pass example

```
sample = torch.randn(1,3,64,64)
output = model(sample)
```

40. Count trainable parameters

```
sum(p.numel() for p in model.parameters() if p.requires_grad)
```

41. Apply Dropout

```
nn.Dropout(p=0.5)
```

42. Apply BatchNorm

```
nn.BatchNorm2d(32)
```

43. Gradient clipping

```
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
```

44. Save & load model

```
torch.save(model.state_dict(), 'model.pth')
model.load_state_dict(torch.load('model.pth'))
```

45. Set train/eval mode

```
model.train()  # training mode
model.eval()   # evaluation mode
```

46. Tensor operations

```
x = torch.randn(3,4)
y = torch.randn(3,4)
z = x + y
```

47. Reshape tensor

```
x = x.view(-1, 12)  # flatten 3x4 to 1x12
```

48. Convert NumPy to tensor

```
import numpy as np
a = np.array([1,2,3])
t = torch.from_numpy(a)
```

49. GPU check

```
torch.cuda.is_available()
torch.cuda.get_device_name(0)
```