DLP - PA-9

1) Which of the following does not fall under computer vision task? *1 point*

- (a) Segmentation of a car/bus on a road. → obj. recogn
- (b) Auto correction of text in an email. → NLP
- (c) Text summarizing of an image. → CV + NLP
- (d) Pose estimation of a video. → Capturing the area & poses

2) Mark the different techniques in computer vision for generating a 3D structure of an object. *1 point*

- (a) Depth estimation using a single image.
- (b) Structure from motion using a video camera.
- (c) Capturing different pictures of the same object with different illumination.
- (d) Using two cameras and estimating the depth of the image.

when a image is taken from different angles, then it creates 3d model of that object.
Eg. Taj Mahal

3) For an RGB image with dimensions 300x300 from a mobile phone, how many parameters would need to be handled by a traditional Multi-Layer Perceptron (MLP)?

$3 \times 300 \times 300 = 270{,}000$ params

4) What are the features of a CNN? *1 point*

- [x] (a) Parameter sharing.
- [ ] (b) Full connectivity → this was present in MLP ⇒ leading to parameter explosion problems
- [x] (c) Local connectivity.
- [x] (d) Weight sharing.

5) Consider a 4x4 input matrix:

$$\begin{bmatrix} 1 & 3 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

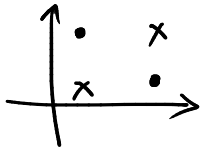
What will be the output matrix after applying an average pooling with a stride of 1?

stride = 1

$$\begin{bmatrix} 1 & 3 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \xrightarrow{\text{avg. pool}} \begin{bmatrix} 3.75 & 4.5 & 5.25 \\ 7.5 & 8.5 & 9.5 \\ 11.5 & 12.5 & 13.5 \end{bmatrix}$$

6) Mark the gate(s) which are not solvable by a normal perceptron? *1 point*

- (a) NOR.
- (b) NAND.
- (c) AND.
- (d) XNOR. / XOR



} one line can't seperate these points.

7) *For a CNN network with 10 input nodes and 5 output nodes with a local connectivity of 3 nodes, what do you think should be the runtime and storage requirements without parameter sharing?* *1 point*

- (a) Runtime: 15 operations, Storage: 15 parameters.
- (b) Runtime: 50 operations, Storage: 15 parameters.
- (c) Runtime: 30 operations, Storage: 30 parameters.
- (d) Runtime: 10 operations, Storage: 5 parameters.

$m = 10$, $n = 5$, $k = 3$

MLNN-LC (no PS)

Storage = $k \times n = 15$

runtime = $O(k \times n) = 15$ ops

8) For a CNN network with 10 input nodes and 5 output nodes with a local connectivity of 3 nodes, what do you think should be the runtime and storage requirements with parameter sharing ? *1 point*

- (a) Runtime: 15 operations, Storage: 15 parameters.
- (b) Runtime: 5 operations, Storage: 3 parameters.
- (c) Runtime: 30 operations, Storage: 30 parameters.
- (d) Runtime: 15 operations, Storage: 3 parameters.

MLNN-LC-PS

storage = $k = 3$

runtime = $O(k \times n) = 15$

9) Why is the ReLU (Rectified Linear Unit) preferred over the Sigmoid activation function in CNNs? **1 point**

- (a) ReLU is computationally cheaper compared to the Sigmoid function than tanh as well.
- (b) ReLU helps mitigate the vanishing gradient problem, which is common with Sigmoid. ✓ max (a, 0)
- (c) Sigmoid outputs are bounded between 0 and 1, which can limit learning in deeper networks. ✓ (Tanh)
- (d) ReLU introduces sparsity in activations, which can improve computational efficiency. ✓
- (e) All of the above.

10) For a CNN layer, suppose the output after the convolutional and pooling operations is a feature map of shape (16,16, 32) (height = 16, width = 16, depth = 32). If this feature map is flattened and connected to a fully connected to a fully connected layer with 128 output nodes, how many parameters (weights and biases) will this layer have? **1 point**

feature map = 16 × 16 × 32
FC = 128
wts = 16 × 16 × 32 × 128 = 1048576
bias = 128
with wts + bias = 1048704

- (a) 131,072
- (b) 1,048,576 ✓
- (c) 65,536
- (d) 65,664

11) In a CNN, why is the Softmax activation function typically used in the output layer? **1 point**

- (a) To convert the output feature maps into class probabilities.
- (b) To ensure that the outputs of the CNN represent probabilities that sum to 1 across all classes. ✓ → sigmoid activation gives the output btw. 0 & 1 as probabilities. Σ of all probab. will be 1 for all classes.
- (c) To allow multi-class classification by emphasizing the highest probability class.
- (d) To improve computational efficiency in training the CNN.

12) In a CNN, what is the primary distinction between the feature learning layers (convolutional and pooling layers) and the classification layers (fully connected layers)? **1 point**

- (a) Feature learning layers reduce the size of the input, while classification layers increase it.
- (b) Feature learning layers (CONV & POOL layers) extract spatial and hierarchical features while classification layers (FC) map these features to specific output categories. ✓
- (c) Feature learning layers use activation functions, while classification layers do not.
- (d) Feature learning layers perform probabilistic operations, while classification layers perform linear transformations.

13) Consider a CNN layer with the following parameters: **1 point**
- **Input size**: 32×32×3 (height × width × depth),
- **Filter size**: 5×5,
- **Number of filters**: 16,
- **Stride**: 2,
- **Padding**: 1.

What will be the output size of the feature map?

$$\text{output ht./width} = \frac{\text{input} - \text{filter} + (2 \times \text{Padding})}{\text{stride}} + 1$$

$$= \frac{32 - 5 + 2}{2} + 1 = \frac{29}{2} + 1 = 14.5 + 1 = 15.5 \approx 16$$

depth = no. of filters = 16

- (a) 16 ×16×16 ✓
- (b) 32×32×16
- (c) 16 ×16×32
- (d) 32×32×3

14) You are given an image of dimensions 227 × 227 × 3 representing its height, width, and color channels. You want to flatten it into a 1D array using Python. **1 point**
Which of the following code snippets will correctly achieve this transformation?

MXNXD → MND×1

(A) import numpy as np
img = np.random.rand(227, 227, 3)
flat_img = img.flatten() ✓

(B) import numpy as np
img = np.random.rand(227, 227, 3)
flat_img = img.reshape(-1) ✓

(C) import numpy as np
img = np.random.rand(227, 227, 3)
flat_img = img.reshape((227*227*3,))

(D) import numpy as np
img = np.random.rand(227, 227, 3)
flat_img = img.flatten((227, 227, 3)), not there

Which option(s) are correct?

- (a) (A) and (B).
- (b) (B) and (C).
- (c) (A), (B), and (C). ✓
- (d) Only (D).

15) In a CNN, parameter sharing ensures that the same set of weights is applied across different regions of the input, enabling spatial feature detection. Consider a ***1 point***
$7 \times 7 \times 3$ input image passed through a convolutional layer with the following characteristics:

4 filters
Kernel size of 3×3
Stride 1
No padding
Which of the following PyTorch code snippets correctly demonstrates parameter sharing in this convolutional layer?

A, B, and C all implements the PS network

a.

```
import torch
import torch.nn as nn
class CNN(nn.Module):
  def __init__(self):
    super(CNN, self).__init__()
    self.conv = nn.Conv2d(in_channels=3, out_channels=4,
                kernel_size=(3, 3), stride=1, padding=0)
  def forward(self, x):
    return self.conv(x)

model = CNN()
x = torch.randn(1, 3, 7, 7)  # Batch size 1, 3 channels, 7x7 image
output = model(x)
print(output.shape)
```

b.

```
import torch
import torch.nn as nn
class CNN(nn.Module):
  def __init__(self):
    super(CNN, self).__init__()
    self.conv = nn.Conv2d(in_channels=3, out_channels=4,
                kernel_size=(3, 3), stride=1, padding=1)
  def forward(self, x):
    return self.conv(x)

model = CNN()
x = torch.randn(1, 3, 7, 7)  # Batch size 1, 3 channels, 7x7 image
output = model(x)
print(output.shape)
```

c.

```
import torch
import torch.nn as nn
class CNN(nn.Module):
  def __init__(self):
    super(CNN, self).__init__()
    self.conv = nn.Conv2d(in_channels=3, out_channels=4,
                kernel_size=(3, 3), stride=1, padding=0)
  def forward(self, x):
    # Manually perform convolution
    weight = self.conv.weight
    bias = self.conv.bias
    return nn.functional.conv2d(x, weight, bias=bias,
                stride=1, padding=0)

model = CNN()
x = torch.randn(1, 3, 7, 7)  # Batch size 1, 3 channels, 7x7 image
output = model(x)
```

16.

```
import torch
import torch.nn as nn
input_tensor = torch.tensor([[[[1, 2, 3, 4, 5, 6],
                               [7, 8, 9, 10, 11, 12],
                               [13, 14, 15, 16, 17, 18],
                               [19, 20, 21, 22, 23, 24],
                               [25, 26, 27, 28, 29, 30],
                               [31, 32, 33, 34, 35, 36]]]], dtype=torch.float32)
conv_layer = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(3, 3), stride=(1, 1), padding=0)
output_tensor = conv_layer(input_tensor)
print("Output shape:", output_tensor.shape)
```

What will be the output?

- (a) The input tensor size remains 6 × 6 because no padding is applied.
- (b) The output size reduces to 4 ×4 because of the 3×3 filter and the stride of 1.
- (c) The convolution operation cannot be performed because of incompatible dimensions.
- (d) The output size becomes 6 ×6 because the filter size matches the input size.

17) If the padding is changed to 'padding='same'', what happens next? *1 point*

- (a) The input image size remains 6 ×6, and the output size also stays 6 ×6.
- (b) The output size reduces to 4 ×4 because of the 3×3 filter and the stride of 1, but with extra padding.
- (c) The output size increases to 8 × 8 due to the padding.
- (d) The convolution operation cannot be performed because the padding is invalid.

if padding remains same, then the convolutions act won't take place.

18) The following code applies a custom pooling operation with a pool size of 2×2 and a stride of 2×2 using PyTorch: *1 point*

```
import torch
import torch.nn.functional as F
input_tensor = torch.tensor([[[[1, 2, 3, 4],
                               [5, 6, 7, 8],
                               [9, 10, 11, 12],
                               [13, 14, 15, 16]]]], dtype=torch.float32)
def min_pool2d(input_tensor, kernel_size, stride):
    return -F.max_pool2d(-input_tensor, kernel_size=kernel_size, stride=stride)
output_tensor = min_pool2d(input_tensor, kernel_size=2, stride=2)
print("Output after pooling:")
print(output_tensor)
```

What will be the output?

- (a) The output size is 4 ×4, and the minimum value in each 2 ×2 block is selected.
- (b) The output size is 2 ×2, and the minimum value in each 2 ×2 block is selected.

max pool 2d takes maximum value

min pool 2d takes min. value

19) For the given below code snippet: *1 point*

```
import torch
import torch.nn as nn
input_tensor = torch.tensor([[[[1, 2, 3, 4, 5, 6, 7, 8],
                               [9, 10, 11, 12, 13, 14, 15, 16],
                               [17, 18, 19, 20, 21, 22, 23, 24],
                               [25, 26, 27, 28, 29, 30, 31, 32],
                               [33, 34, 35, 36, 37, 38, 39, 40],
                               [41, 42, 43, 44, 45, 46, 47, 48],
                               [49, 50, 51, 52, 53, 54, 55, 56],
                               [57, 58, 59, 60, 61, 62, 63, 64]]]], dtype=torch.float32)
conv_layer = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(3, 3), stride=(2, 2), padding=0)
output_tensor = conv_layer(input_tensor)
print("Output shape:", output_tensor.shape)
```

What will be the output?

stride = 2
kernel size = 3×3

4×4 output because 3×3 size & stride = 2.

20) For the given below code snippet: *1 point*

```
import torch
import torch.nn as nn
import torch.optim as optim

class DeepNN(nn.Module):
    def __init__(self):
        super(DeepNN, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(100, 128),
            nn.Sigmoid(),
            nn.Linear(128, 64),
            nn.Sigmoid(),
            nn.Linear(64, 32),
            nn.Sigmoid(),
            nn.Linear(32, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.layers(x)

model = DeepNN()
input_tensor = torch.randn(10, 100)  # Batch size of 10, 100 input features
labels = torch.ones(10, 1)  # Target values

criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

output = model(input_tensor)
loss = criterion(output, labels)
loss.backward()
print("Loss:", loss.item())
```

FC → S → FC → S

Slow convergence ⇒ wts. in early layers barely update

Which of the following issues is most likely to occur during training with the above model?

- ( ) (a) The model will face the exploding gradient problem because sigmoid produces large outputs.
- (✓) (b) The model will experience slow convergence or stagnation due to the vanishing gradient problem caused by sigmoid activation.
- ( ) (c) The model will underfit because sigmoid activation does not allow backpropagation.
- ( ) (d) The optimizer will fail because sigmoid activation does not work with gradient descent.

training will get stagnant ⇒ gradients will approach 0.

you can replace sigmoid with ReLu to mitigate vanishing gradient problem.

21) For the given below code snippet: *1 point*

```
import torch
import torch.nn as nn
import torch.optim as optim

X = torch.tensor([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=torch.float32) # Inputs
y = torch.tensor([[1], [0], [0], [1]], dtype=torch.float32) # Outputs (XNOR)

class SimpleMLP(nn.Module):
    def __init__(self):
        super(SimpleMLP, self).__init__()
        self.fc = nn.Linear(2, 1)
        self.sigmoid = nn.Sigmoid()

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

model = SimpleMLP()
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

for epoch in range(1000):
    optimizer.zero_grad()
    outputs = model(X)
    loss = criterion(outputs, y)
    loss.backward()
    optimizer.step()

predictions = model(X).round()
print("Predicted outputs:")
print(predictions)
```

What do you think will be the output of the model?

$\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}_{4\times 2}$ $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

Linear perceptrons ⇒ ∴, can't learn the XNOR gate ↳ so output can't be [1,0,0,1]

it won't fail as well, as the Adam optimizer works fine. The model will work fine, but the results are not optimized!

output will be like around [0.5, 0.5, 0.5, 0.5]