Type: MCQ

Q1. If we are not using regularization technique, then we are likely to result in (0.5)

1. Bias error
2. Variance error
3. Best fit model
4. ** Model overfitting

Q2. As model complexity increases (0.5)

1. Variance decreases and bias decreases
2. Bias increases and variance decreases
3. Bias increases
4. ** Bias decreases

Q3. If a linear regression model is underfitting, which regularization technique would you use? (0.5)

1. L1 Regularization
2. L2 Regularization
3. Weight decay
4. ** None of the above

Q4. Which Transfer Learning approach involves adapting a pre-trained model to a new task (0.5)

1. Dropout
2. Batch normalization
3. Weight initialization
4. ** None of the above

Q5. RNN(Recurrent Neural Network) is used for machine translation because (0.5)

1. It can be trained as a unsupervised machine learning algorithm on the independent data
2. ** It is applicable when the input is a sequence of data
3. It is applicable because it can be trained on the independent data
4. All of the above

Q6. Which of the following is used to increase the model capacity (0.5)

1. Increase the learning rate
2. Increase dropout probability of keeping a node
3. **Increase the number of hidden layers
4. Applying Principal Component Analysis

Q7. Which of the following propositions is TRUE about a CONV layer (0.5)

1. **The number of weights depends on the depth of the input volume.
2. The total number of parameters depends on the stride

1. The total number of parameters depends on the padding.
2. The number of weights depend on the height and width of the input image.

Q8. A scalar tensor has a rank (0.5)

1. **0
2. 1
3. 2
4. 3

Q9. A neuron with four inputs has the weight vector w=[-0.5 -0.2 -0.1 -0.1] and bias=0. The activation function is sigmoidal. If the input vector x=[4 8 5 6], then the output of the neuron will be _____ (0.5)

1. **0.9909
2. 0.0021
3. 0.1902
4. 0.0990

Q10. Which of the following statement is FALSE (0.5)

1. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set
2. Overfitting occurs when the gap between the training error and test error is too large
3. We can control whether a model is more likely to overfit or underfit by altering its capacity
4. **It is not possible for the model to have optimal capacity and yet still have a large gap between training and generalization error

Type: DES

Q11. a) Give two benefits of using convolutional layers instead of fully connected ones for visual tasks.

b) You are given a dataset of 10 × 10 grayscale images. Your goal is to build a 5-class classifier. You have to adopt one of the following two options:

i) The input is flattened into a 100-dimensional vector, followed by a fully-connected layer with 5 neurons

ii) The input is directly given to a convolutional layer with five 10 × 10 filters. Explain which one you would choose and why. (4)

Ans

a) Uses spatial context (by only assigning weights to nearby pixels)

- Translation invariance
- Have lot less parameters, since CNN's share weights

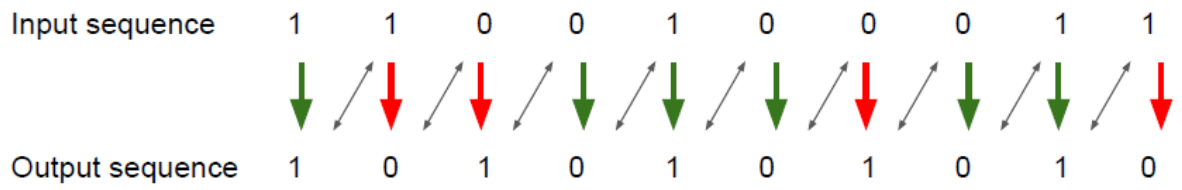
In fully connected layer, the number of parameters is : 100*5+5 = 505

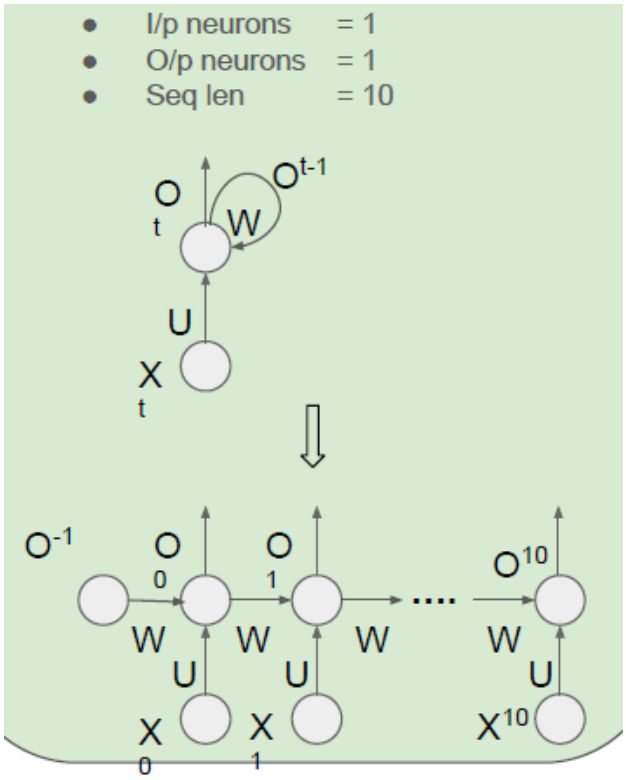
In Convolution layer, the number of parameters is: 5*100 + 5 = 505

b)

The 2 approaches are the same. But the second one seems better in terms of computational costs (no need to flatten the input). We accept the answer ”the 2 approaches are the same”.

Q12. Consider a bit reverse problem which states that “reverse a bit if the current input and previous output are same”. What kind of network is suitable to implement this problem. Assume you are given a sequence of 10 bits – 1 1 0 0 1 0 0 0 1 1 as input X to your model. Provide output for this case and pictorially illustrate the unfolded network architecture clearly marking all details. If each bit is fed at timestep t, what is the sequence length? Considering a batch size of 1, provide a neat layout of input along with necessary parameters, assuming a batch-first notation. Take hidden size as 1 and provide dimension of input, output and hidden sizes along with sizes of weights and biases. Furnish all dimensions clearly in a table format. At t=1, the output is same as the input. (4)





Data: torch.Size([10])

tensor([1., 1., 0., 0., 1., 0., 0., 0., 1., 1.])

Initial rnn state=

OrderedDict([('weight_ih_l0', tensor([[-0.6954]])), ('weight_hh_l0', tensor([[-0.2333]])), ('bias_ih_l0', tensor([-0.6354])), ('bias_hh_l0', tensor([0.9240]))])

After weight initialization, rnn state=

OrderedDict([('weight_ih_l0', tensor([[1.]])), ('weight_hh_l0', tensor([[1.]])), ('bias_ih_l0', tensor([1.])), ('bias_hh_l0', tensor([1.]))])

X: torch.Size([1, 10, 1])

tensor([[[1.],

[1.],

[0.],

[0.],

[1.],

[0.],

[0.],

[0.],

[1.],

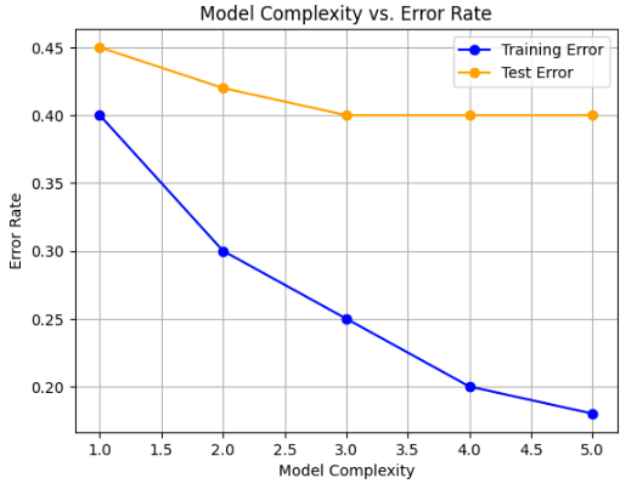
[1.]]])

Output: torch.Size([1, 10, 1])

Hidden: torch.Size([1, 1, 1])

Q13. Suppose that you have a model that provides around 80% accuracy on the training as well as on the test data. What kind of model fit is referred here? Illustrate the case by drawing a neat graph with clearly plotted points and details. How do you improve the accuracy of such a model? Explain. (3)

This is clearly a case of under-fitting because the training and test accuracy are very close. It is recommended to increase the capacity of the model by adding more units, which preferably have nonlinearity. Or, we can also increase the model complexity by fitting a higher degree predictor to increase model complexity.



Q14. Let p be the probability of keeping neurons in a dropout layer and assume we are going to implement dropout without PyTorch library support. Accidentally, it is found that the model is trained with dropout layers, but without scaling. How would you resolve this issue at test time? Justify your answer mathematically and provide Python code to clearly explain the case. (3)

Mathematically: Dropout switched off at test time - expected activation thus larger than train by factor 1/p; fix by multiplying outputs of each layer by p.

Python code:

```
# Generate a dropout mask using torch.bernoulli with a probability of 0.1

dropout_mask = torch.bernoulli(torch.full_like(x, p))  # p for keeping values

#dropout_mask = torch.bernoulli(torch.ones_like(x) * p)


# Apply dropout scaling

output = x * dropout_mask * p  # Scale by p

print("output=", output)

print(x.mean(), output.mean())
```

Q15. As you train your model, you realize that you do not have enough data. Give three data augmentation techniques that can be used to overcome the shortage of data. How do you provide transformation pipeline when there are more than one augmentation technique? Give details. (3)

Any 3 technique:

Converting Images to Tensors-.ToTensor()

Normalization - .Normalize

.Resize()

.CenterCrop()

Compose()

Transformation pipeline when there are more than one augmentation technique: Compose()

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
      mean=[0.485, 0.456, 0.406],
      std=[0.229, 0.224, 0.225])
])
```

Compose() allows to stack multiple transformations sequentially

Q16. Write the life-cycle for a deep learning model and the PyTorch API that we can use to define models, taking the example of linear regression. (3)

- The five steps in the life-cycle are as follows:

1. Prepare the Data:
2. Define the Model
3. Train the Model
4. Evaluate the Model
5. Make Predictions

Step 1:

- The first step is to load and prepare our data.
- Neural network models require numerical input data and numerical output data.
- PyTorch provides the Dataset class that we can extend and customize to load our dataset.
- the constructor of our dataset object can load our data file (e.g. a CSV file).

- We can then override
- __len__() function that can be used to get the length of the dataset (number of rows or samples), and
- __getitem__() function that is used to get a specific sample by index.

Step 2:

- Defining a model in PyTorch involves defining a class that extends the Module class.
- The constructor of our class defines the layers of the model and
- the forward() function is the override that defines how to forward propagate input through the defined layers of the model.
- Many layers are available, such as Linear for fully connected layers, Conv2d for convolutional layers, and MaxPool2d for pooling layers.
- Activation functions can also be defined as layers, such as ReLU, Softmax, and Sigmoid.

Step 3:

- The training process requires that we define a loss function and an optimization algorithm.
- Common loss functions include the following:
    - BCELoss: Binary cross-entropy loss for binary classification.
    - CrossEntropyLoss: Categorical cross-entropy loss for multi-class classification.
    - MSELoss: Mean squared loss for regression.
- Stochastic gradient descent is used for optimization, and the standard algorithm is provided by the SGD class

Ste 4:

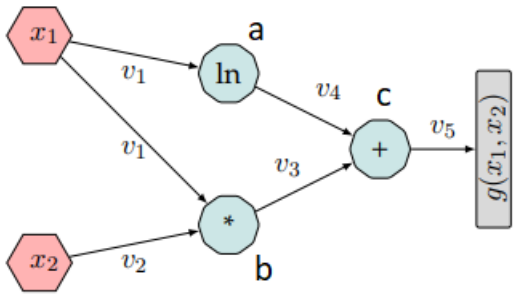
Once the model is fit, it can be evaluated on the test dataset

Step 5:

- A fit model can be used to make a prediction on new data.
- Ex: we have a single image or a single row of data and want to make a prediction.

Q17.

a) Find the function, g(x1, x2) represented by the computation graph. The v_i's represent the values that flows and a, b, c are the intermediate calculations. Nodes is colored in green indicates the operation performed, and 'ln' is for natural logarithm.



b) Find the value of the function g(x1, x2) evaluated at (x1, x2) = (e^2 , π) in the forward pass of Automatic Differentiation.

c) Find the partial derivatives for the function g(.) (3)

a = ln(x1)

b = x2 * x1

c = a + b

g(x1, x2) = ln(x1) + x2 *x1

d) Find the value of the function g(x1, x2) evaluated at (x1, x2) = (e^2 , π) in the forward pass of Automatic Differentiation.

g(x1, x2) = ln(e^2) + (e^2 * π)

= 2 + 7.3890 * 3.142

= 25.2134

e) Find the partial derivatives for the function g(.) (1+1+1)

∂C/ ∂a = 1

∂C/ ∂b = 1

∂b/ ∂x1 = x2

∂b/ ∂x2 = x1

∂a/ ∂x1 = 1/x1

∂C/ ∂x1 = ∂C/ ∂a * ∂a/ ∂x1 + ∂C/ ∂b * ∂b/ ∂x1 = 1 * 1/x1 + 1 * x2 = 1/x1 + x2

∂C/ ∂x2 = ∂C/ ∂b * ∂b/ ∂x2 = 1 * x1 = x1

Q18. Compare NumPy arrays with PyTorch tensors. (2)

- Tensors are multidimensional arrays like n-dimensional NumPy array.

- **However, tensors can be used in GPUs as well, which is not in the case of NumPy array**
- **Tensor maintain gradient information unlike Numpy**
- tensors should have the same size of columns in all dimensions
- Also, the tensors can contain only numeric data types