ECE 364: Programming Methods for Machine Learning, Spring 2025

Midterm 2 – Sample

- **You will have 75 minutes (1.25 hours) to solve all the problems. Most have multiple parts.** Don't spend too much time on questions you don't understand and focus on answering as much as you can!
- ***BUDGET YOUR TIME WISELY***. I highly recommend working on the questions you know first and the questions you need to think about second.
- *No* resources are allowed for use during the exam except a cheatsheet and scratch paper on the back of the exam. **Do not tear out the cheatsheet or the scratch paper!** It messes with the auto-scanner.
- You should write your answers *completely* in the space given for the question. We will not grade parts of any answer written outside of the designated space.
- Please *use a dark-colored pen* unless you are *absolutely* sure your pencil writing is forceful enough to be legible when scanned. We reserve the right to deduct points if we have difficulty reading the uploaded document.
- **Don't cheat.** C'mon, be cool, be honest.
- **Good luck!**

Name: _____

NetID: _____

Date: _____

1. **True/False** (30 points)

For each question, circle whether the statement is true or false.

(a) **TRUE** False In PCA, the eigenvectors of the covariance matrix correspond to the principal component directions, and the associated eigenvalues represent the amount of variance explained along those directions.

(b) True **FALSE** The K-Means algorithm is guaranteed to find the globally optimal clustering configuration for any dataset.

(c) **TRUE** False Mode collapse is a common challenge in the GAN training.

(d) True **FALSE** The different filters (or kernels) in a convolutional layer share weights to reduce the number of parameters.

(e) **TRUE** False A goal of activation function is to introduce nonlinearity

(f) **TRUE** False The forget gate in LSTM decides what information to remove from the cell state.

(g) True **FALSE** LSTM networks suffer from vanishing gradients more severely than standard RNNs.

(h) **TRUE** False Principal components are always orthogonal to each other.

(i) **TRUE** False The number of filters in a CNN layer determines the depth of the output feature map.

(j) True **FALSE** PCA is a supervised learning technique used for dimension reduction.

(k) **TRUE** False Learned positional embeddings cannot generalize to sequence lengths longer than those seen in training.

(l) True **FALSE** BERT is a left-to-right (unidirectional) language model.

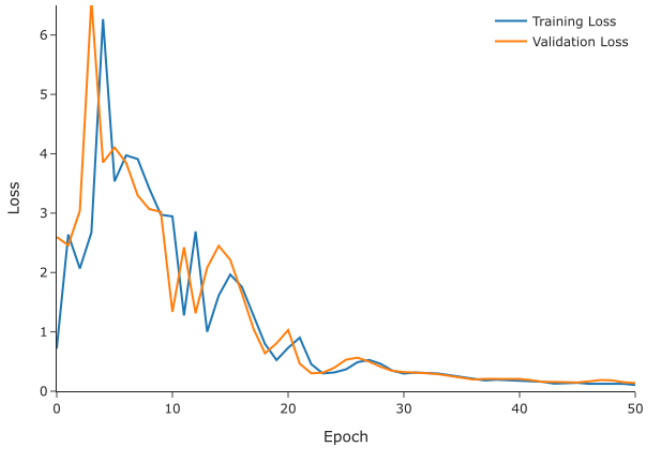
(m) True **FALSE** SSD relies on max-pooling layers to generate its multi-scale feature maps for detection.

(n) True **FALSE** In SSD, default boxes (anchors) are generated only at the final convolutional feature map.

(o) True **FALSE** The sigmoid and tanh activation functions can be used interchangably since they both have a "S"-curve shape.

2. **Debugging a Training Job** (10 points)

You have been assigned to build a classification model to detect whether emails are spam or not. After dedicating several days to the project, you have designed a promising model. But, during training, you observe the following behavior in the training and validation losses:



(a) Which of the following is a possible issue with the training? Circle your answer.

Underfitting Overfitting Learning Rate No issue

Solution:
Learning Rate

(b) Justify your answer. If you chose any option other than "No issue", suggest a possible remedy to mitigate the issue. Use no more than four sentences in total for your answer.

Solution:
The loss fluctuates significantly during the initial epochs, which is not expected. The issue may be caused by a higher learning rate than required. A possible remedy would be to reduce the learning rate. If stochastic gradient descent is used for training, increasing the batch size may also help.

3. **Neural Networks for Simple Functions** (10 points)

In this problem, you will be hand designing a simple neural network to model a specific function. Assume $x \in \mathbb{R}$ and provide appropriate weights $w_0, w_1 \in \mathbb{R}^2$. In other words, the neural network has one input neuron, 2 hidden neurons, and one output neuron.

Find $w_0, w_1 \in \mathbb{R}^2$ such that $f(x) = w_1^T \sigma(w_0 x) = x, \forall x \in \mathbb{R}$ where $\sigma = \text{ReLU}$ (note: $w_0 x$ here is a vector-scalar product: $\mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$, e.g. $[0,1]^T x = [0,x]^T$). Show why your answer is correct.

Solution:

In order to achieve this function, we can set $w_0 = [1,-1]^T, w_1 = [1,-1]^T$. We can see that this works because applying our answer to our function, we get

$$f(x) = w_1^T \sigma(w_0 x) = [1,-1]^{T^T} \sigma([1,-1]^T x)$$

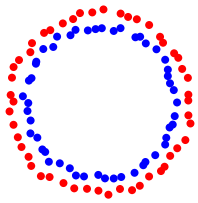
$$= [1,-1]\sigma([x,-x]^T) = \begin{cases} [1,-1][x,0]^T = x & , x > 0 \\ [1,-1][0,0]^T = 0 & , x = 0 \\ [1,-1][0,-x]^T = x & , x < 0 \end{cases}$$

$$\Rightarrow f(x) = x, \forall x \in \mathbb{R}$$

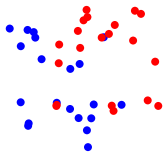
(ReLU: $\sigma(x) = max(0,x)$).

4. **K-means and GMM** (10 points)

(a) For each figure below, indicate whether it is possible for K-Means, GMM, neither, or both algorithms to produce the clustering assignments shown (as indicated by the two colors). In 1–2 sentences, briefly explain your reasoning.



(a)



(b)

Solution:

(a) Neither

(b) GMM only

(b) Which algorithm (K-Means or GMM) typically has more parameters to learn for the same number of clusters, and why?

Solution:
GMM, because for each cluster it estimates a mean, a covariance matrix (or variance in 1D), and a mixing coefficient. K-Means only estimates cluster centers.

(c) Which of the following expressions is used to compute cluster responsibilities in the E-step of the EM algorithm for GMM?

A. $\arg\min_k |x_i - \mu_k|^2$

B. $\dfrac{\pi_k, \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j, \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$

C. $\sum_{k=1}^{K} |x_i - \mu_k|^2$

D. $\sum_{i=1}^{n} |x_i - \bar{x}|^2$

Solution:
(b). The E-step computes the probability (responsibility) that each point belongs to each component using Bayes' rule.

5. **Attention is All You Need!** (15 points)

The Transformer architecture integrates attention mechanisms with multi-layer perceptrons (MLPs). The attention mechanism takes three inputs – queries (Q), keys (K), and values (V). Each has a shape of $B \times N \times d_{\text{model}}$. Here, B, N, and d_{model} represent the batch size, sequence length, and model dimension (or hidden size), respectively. Given h as the number of attention heads, the following process is used to compute attention

Step 1. For each head i, we apply learned projection matrices as

$$Q_i = QW_i^Q, K_i = KW_i^K, V_i = VW_i^V$$

where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_h}$ and $d_h = d_{\text{model}}/h$.

Step 2. For each head, attention is computed as follows

$$A_i = \text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_h}}\right) V_i$$

Step 3. Concatenate the attention across heads and apply one more linear transform

$$\text{MultiHeadAttention}(Q, K, V) = \text{concat}(A_1, A_2 \dots A_h) W^O$$

where $W_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$.

Complete the code below to implement multi-head attention. Use the provided definitions as a guide. For this question, you can ignore additional details such as attention masks, dropouts, and optimized implementations.

```
# Even though you should not need to import anything else, feel
# free to do so.
import torch
import torch.nn as nn
from torch.nn.functional import softmax



class MultiHeadAttention(nn.Module):
    def __init__(self, d_model: int, h: int) -> None:
        assert d_model % h == 0, "d_model must be divisible by h"
        self.d_model = d_model
        self.h = h
        self.d_head = self.d_model // self.h

        # Combine h weight matrices of d_head into one
        self.wq = nn.Linear(self.d_model, self.d_model, bias=False)
        self.wk = nn.Linear(self.d_model, self.d_model, bias=False)
        self.wv = nn.Linear(self.d_model, self.d_model, bias=False)
        self.wo = nn.Linear(self.d_model, self.d_model, bias=False)

        # Space to add more class variables as required
```

```
29
30      def forward(self, Q: torch.Tensor, K: torch.Tensor, V: torch.Tensor) ->
        torch.Tensor:
31          """
32          Inputs
33          ------
34          Q: B x N x d_model
35          K: B x N x d_model
36          V: B x N x d_model
37          """
38          # Complete this
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
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84          # Function end
```

Solution:

```python
# Even though you should not need to import anything else, feel
# free to do so.
import torch
import torch.nn as nn
from torch.nn.functional import softmax

class MultiHeadAttention(nn.Module):
    def __init__(self, d_model: int, h: int) -> None:
        assert d_model % h == 0, "d_model must be divisible by h"
        self.d_model = d_model
        self.h = h
        self.d_head = self.d_model // self.h

        # Combine h weight matrices of d_head into one
        self.wq = nn.Linear(self.d_model, self.d_model, bias=False)
        self.wk = nn.Linear(self.d_model, self.d_model, bias=False)
        self.wv = nn.Linear(self.d_model, self.d_model, bias=False)
        self.wo = nn.Linear(self.d_model, self.d_model, bias=False)

        # Space to add more class variables as required
        self.scaling = self.d_head**-0.5

    def forward(self, Q: torch.Tensor, K: torch.Tensor, V: torch.Tensor) -> torch.Tensor:
        """
        Inputs
        ------
        Q: B x N x d_model
        K: B x N x d_model
        V: B x N x d_model
        """
        # Complete this
        B, N, d_model = Q.size()
        query_states = self.wq(Q) * self.scaling # B x N x d_model
        key_states = self.wk(K) # B x N x d_model
        val_states = self.wv(V) # B x N x d_model

        query_states = query_states.view(B, N, self.h, self.d_head).transpose(1, 2).contiguous() # B x h x N x d_head
        key_states = key_states.view(B, N, self.h, self.d_head).transpose(1, 2).contiguous() # B x h x N x d_head
        val_states = val_states.view(B, N, self.h, self.d_head).transpose(1, 2).contiguous() # B x h x N x d_head

        query_states = query_states.view(-1, N, self.d_head) # B*h x N x d_head
        key_states = key_states.view(-1, N, self.d_head) # B*h x N x d_head
        val_states = val_states.view(-1, N, self.d_head) # B*h x N x d_head

        attn_weights = query_states @ key_states.transpose(1, 2) # B*h x N x N
        attn_weights = softmax(attn_weights, dim=-1) # B*h x N X N

        attn_outputs = attn_weights @ val_states # B*h x N x d_head
        attn_outputs = attn_outputs.view(B, self.h, N, self.d_head) # B x h x N x d_head
```

```
        attn_outputs = attn_outputs.transpose(1, 2) # B x N x h x d_head
        attn_outputs = attn_outputs.reshape(B, N, d_model) # B x N x d_model
        attn_outputs = self.wo(attn_outputs) # B x N x d_model

        return attn_outputs
        # Function end
```

6. **Output and Parameter Count** (15 points)

Consider the following network:

```
model = nn.Sequential(
    nn.Conv2d(1, 4, 3, padding=1),
    nn.MaxPool2d(2,2),
    nn.Conv2d(4, 8, 3, padding=1),
    nn.MaxPool2d(2,2),
    nn.Flatten(),
    nn.Linear(8*7*7, 10)
)
```

(a) Suppose the input is $(1, 1, 28, 28)$. What are the output shapes after each layer?

Solution:
After Conv1: $(1, 4, 28, 28)$
Pool1: $(1, 4, 14, 14)$
Conv2: $(1, 8, 14, 14)$
Pool2: $(1, 8, 7, 7)$
Flatten: $(1, 392)$
Linear: $(1, 10)$

(b) Determine the total learnable parameters in the model (including biases).

Solution:
Conv1: $4 \times 1 \times 3 \times 3 + 4 = 36 + 4 = 40$
Conv2: $8 \times 4 \times 3 \times 3 + 8 = 288 + 8 = 296$
Linear: $392 \times 10 + 10 = 3920 + 10 = 3930$
Total: $40 + 296 + 3930 = 4266$.

7. **Layer-Output Computation** (10 points)

Consider the two-channel input

$$X_1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \quad X_2 = \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix}$$

and depthwise kernels

$$K_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad K_2 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

(a) Compute the two output feature-maps of `Conv2d(in_channels=2,out_channels=2,groups=2,kernel_size=2)` (bias=0).

Solution:

Channel 1: $\begin{pmatrix} 6 & 8 \\ 12 & 14 \end{pmatrix}$, Channel 2: $\begin{pmatrix} 28 & 24 \\ 16 & 12 \end{pmatrix}$

This page is for additional scratch work!