ECE 364: Programming Methods for Machine Learning, Spring 2025

Midterm 1 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 multi-page 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 take off points if we have difficulty reading the uploaded document.
- Unless otherwise stated, assume $\mathsf{P} \neq \mathsf{NP}$.
- Assume that whenever the word "reduction" is used, we mean a (not necessarily polynomial-time) *mapping/many-one* reduction.
- You can only refer to the cheat sheet content as a black box.
- **Don't cheat.** If we catch you, you will get an F in the course.
- **Good luck!**

Name: _____

NetID: _____

Date: _____

1. **Tensor manipulations** (4 points)

For each of the code segments, answer the following questions.

(a)
```
import torch
a = torch.tensor([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
b = a[0::2 , ::5]
d = a>4
c = a[d]
```

i. (1 point) What will be the shape of b?

Solution:
$[2, 1]$ vector

ii. (1 point) What does c contain?

Solution:
$[5, 6, 7, 8, 9, 10, 11]$

(b)
```
import torch
a = torch.tensor([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
e = a.clone().view([6,-1])
f = torch.tensor([0,1,0])
l = a.multiply_(f.unsqueeze(-1))
```

i. (1 point) What does e contain?

Solution:

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \\ 8 & 9 \\ 10 & 11 \end{bmatrix}$$

Something

ii. (1 point) What does a contain?

Solution:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 4 & 5 & 6 & 7 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Total for Question 1: 8

2. **Striding along** (1 point)

Consider the following code segment:

```
import torch
a = torch.arange(1000).view(10,10,10)
b = ?
```

Slice up a in a way that will cause b to have a separate data space from a.

Solution:

A lot of potential solutions here but you basically just got to make sure the computer can think of a size and stride scheme to index the data from a. Grabbing slices randomly works well enough:

$b = a[[4, 3, 7, 2, 9], :, :]$

3. **Matrix Calculus** (1 point)

Given:

$$f(x) = x^T A^T A x$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times n}$

Find the solution to the following partial derivative:

$$\frac{\partial f}{\partial x} =$$

You must explain why your answer is correct for full credit.

Solution:

Well this is very similar to what we have seen in the homework but instead of A, we have $A^T A$ but the question wants you to show your work so that is what we are going to do:

We want to compute the derivative:

$$f(x) = x^T A^T A x$$

Step 1: Express the Function in Summation Form Expanding using summation notation:

$$f(x) = \sum_i \sum_j x_i (A^T A)_{ij} x_j.$$

Step 2: Compute the Partial Derivative We differentiate $f(x)$ with respect to x_k:

$$\frac{\partial f}{\partial x_k} = \sum_i \sum_j \frac{\partial}{\partial x_k} \left(x_i (A^T A)_{ij} x_j \right).$$

Since $(A^TA)_{ij}$ is a constant coefficient independent of x, we apply the derivative:

$$\frac{\partial}{\partial x_k}\left(x_i(A^TA)_{ij}x_j\right) = (A^TA)_{ij}\frac{\partial}{\partial x_k}(x_ix_j).$$

Using the derivative of a product of variables:

$$\frac{\partial}{\partial x_k}(x_ix_j) = \begin{cases} x_j, & i = k, \\ x_i, & j = k, \\ 0, & \text{otherwise.} \end{cases}$$

Thus, substituting back:

$$\sum_j (A^TA)_{kj}x_j + \sum_i (A^TA)_{ik}x_i.$$

Since A^TA is symmetric ($(A^TA)_{kj} = (A^TA)_{jk}$), both sums are equal, and we get:

$$\frac{\partial f}{\partial x_k} = 2\sum_j (A^TA)_{kj}x_j.$$

Step 3: Convert Back to Matrix Form Rewriting in matrix notation, we recognize the summation as the k-th component of A^TAx, so:

$$\nabla_x f = 2A^TAx.$$

Thus, we have proven from first principles that:

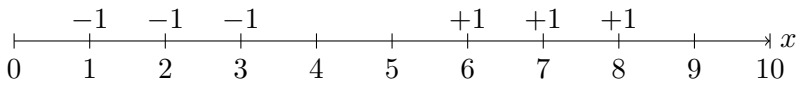
$$\frac{d}{dx}(x^TA^TAx) = 2A^TAx.$$

Note: this is a bit more difficult than the level of matrix derivative I'll ask on the exam but make sure you can do this problem, the HW, and lectures problems easily and you should be fine for the actual exam.

4. **Support Vector Machines** (1 point)

Support vectors are the critical data points in an SVM that lie closest to the decision boundary. They determine both the optimal separating boundary and the margin width between classes. Removing these points from the training set would shift the boundary, potentially altering the classification results.

Consider a SVM that separates the following 1D points into two classes (-1 and $+1$) along the real number line:



With points:

- **Class -1:** 1, 2, 3
- **Class $+1$:** 6, 7, 8

(a) (1 point) **Drawing the Boundary:** Sketch the decision boundary for an SVM on the provided number line. Choose the decision boundary that maximizes the margin and minimizes the loss. Additionally, sketch the margin boundaries for each class (these are the lines that pass through the closest data points from each class, positioned half a margin away from the decision boundary). Label each line clearly. You should draw a total of three lines. Include the approximate equation for each line.

Solution:
Class -1 boundary: vertical line at $x = 3$. Class $+1$ boundary: vertical line at $x = 6$. Decision boundary: vertical line at $x = \frac{3+6}{2} = 4.5$.

(b) (1 point) **Identifying Support Vectors:** For each class (-1 and $+1$) in the provided data, identify the support vectors and list them below.

Solution:
Class -1: $x = 3$, Class $+1$: $x = 6$.

(c) (1 point) **Margin Calculation:** Calculate the margin (total width between class boundary lines) for your decision boundary.

Solution:
margin $= 2 * |4.5 - 3| = 3$ or margin $= 2 * |6 - 4.5| = 3$.

5. **Linear Classification** (1 point)

Let g be the logical **OR** function, defined on the feature space $\{+1, -1\}^2$, which maps:

- $g(+1, +1) = +1$
- $g(-1, +1) = +1$
- $g(+1, -1) = +1$
- $g(-1, -1) = -1$.

Given a linear classifier $h(x) = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$, where $\boldsymbol{w} \in \mathbb{R}^{1\times 2}$, $\boldsymbol{x} \in \mathbb{R}^{2\times 1}$, and $b \in \mathbb{R}^{1\times 1}$, give a valid $(\boldsymbol{w}, b)$ pair that matches the ground truth g. Let $\text{sign}(z) = +1$ for $z \geq 0$ and -1 otherwise. Give your solution and show that it is valid.

Solution:

Let $\boldsymbol{w} = \begin{bmatrix} +1 & +1 \end{bmatrix}$ and $b = +1$. Then the ($\boldsymbol{w}$, b) pair would match the ground truth g for a linear classifier $h(x) = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$. We can check by doing the following calculations, seeing that this mapping leads to:

$$h(+1,+1) = \text{sign}(\begin{bmatrix} +1 & +1 \end{bmatrix} \cdot \begin{bmatrix} +1 \\ +1 \end{bmatrix} + 1) = \text{sign}(+3) = +1 = g(+1,+1)$$

$$h(-1,+1) = \text{sign}(\begin{bmatrix} +1 & +1 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ +1 \end{bmatrix} + 1) = \text{sign}(+1) = +1 = g(-1,+1)$$

$$h(+1,-1) = \text{sign}(\begin{bmatrix} +1 & +1 \end{bmatrix} \cdot \begin{bmatrix} +1 \\ -1 \end{bmatrix} + 1) = \text{sign}(+1) = +1 = g(+1,-1)$$

$$h(-1,-1) = \text{sign}(\begin{bmatrix} +1 & +1 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ -1 \end{bmatrix} + 1) = \text{sign}(-1) = -1 = g(-1,-1)$$

and matches the mapping of g where g is the logical **OR** function defined on the feature space $\{+1,-1\}^2$.

6. **Gradient Descent** (15 points)

Consider the following function

$$f(x,y) = \frac{1}{1+e^{x-y}}$$

(a) Determine the gradient $\nabla f(x,y)$.

Solution:

$$\frac{df}{dx} = \frac{-e^{x-y}}{(1+e^{x-y})^2} = f(x,y)\cdot(f(x,y)-1)$$
$$\frac{df}{dy} = \frac{e^{x-y}}{(1+e^{x-y})^2} = f(x,y)\cdot(1-f(x,y))$$

Hence,

$$\nabla f(x,y) = \begin{bmatrix} f(x,y)\cdot(f(x,y)-1) \\ f(x,y)\cdot(1-f(x,y)) \end{bmatrix}$$

(b) Let the starting point for gradient descent at $k=0$ be $\left(x^{(0)}, y^{(0)}\right) = (0,0)$ and the step size be $\alpha = 4$. Apply gradient descent to obtain the values of x and y iterations $k=1$ and 2.

Solution:

Gradient descent update can be written as

$$\left(x^{(k)}, y^{(k)}\right) = \left(x^{(k-1)}, y^{(k-1)}\right) - \alpha \nabla f(x,y)\,|_{\left(x^{(k-1)}, y^{(k-1)}\right)}$$

At $k=1$, we have

$$f(0,0) = \frac{1}{2}$$
$$\nabla f(x,y)|_{(0,0)} = \begin{bmatrix} -\frac{1}{4} & \frac{1}{4} \end{bmatrix}^T$$
$$\left(x^{(1)}, y^{(1)}\right) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - 4\begin{bmatrix} -\frac{1}{4} \\ \frac{1}{4} \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

At $k=2$, we have

$$f(1,-1) = \frac{1}{1+e^2}$$
$$\nabla f(x,y)|_{(1,1)} = \begin{bmatrix} \frac{-e^2}{1+e^2} & \frac{e^2}{1+e^2} \end{bmatrix}^T$$
$$\left(x^{(2)}, y^{(2)}\right) = \begin{bmatrix} 1 \\ -1 \end{bmatrix} - 4\begin{bmatrix} \frac{-e^2}{1+e^2} \\ \frac{e^2}{1+e^2} \end{bmatrix} = \begin{bmatrix} \frac{1+4e^2}{1+e^2} \\ -\frac{1+4e^2}{1+e^2} \end{bmatrix}$$

7. **Dataloaders and Optimizers** (20 points)

(a) Consider a dataset of type `torch.utils.data.Dataset` named `my_dataset`. We want to create two separate dataloaders for the training and validation sets of the dataset. Each batch must have 64 examples. The validation set is created by randomly selecting 512 examples from the dataset, and the remaining examples are used for training. Complete the below function to implement the dataloaders.

```
# Even though you should not need to import anything else, feel
# free to do so.
import random
import torch.utils.data import DataLoader, SubsetRandomSampler

def split_dataset(my_dataset):
    # complete this function
    train_loader = None
    val_loader = None
    return train_loader, val_loader
```

Solution:

```
import random
import torch.utils.data import DataLoader, SequentialSampler

def split_dataset(my_dataset):
    # Step 1: Select 512 random indices for validation set
    val_indices = random.sample(range(len(my_dataset)), 512)
    # Step 2: Use the remaining indices for training set
    train_indices = set(range(len(my_dataset))) - set(val_indices)
    # Step 3: Create samplers
    train_sampler = SubsetRandomSampler(train_indices)
    val_sampler = SubsetRandomSampler(val_indices)
    # Step 4: Create dataloaders
    train_loader = DataLoader(
        dataset=my_dataset,
        batch_size=64,
        sampler=train_sampler,
    )
    val_loader = DataLoader(
        dataset=my_dataset,
        batch_size=64,
        sampler=val_sampler,
    )
    return train_loader, val_loader
```

(b) We aim to train `Model` using `SimpleDataset`. The dataloaders provide a dictionary containing the input and target, where the input has a shape of $B \times M$ and the target has a shape of N. The Model expects an input of shape $B \times M$ and produces an output of shape B. Here, B and M correspond to the batch size and the number of features, respectively. We will optimize the model using stochastic gradient descent (SGD) with a learning rate of 0.01 and a momentum of 0.9 and will use mean squared error loss. Complete the following code snippet.

```
import torch
from lib.model import Model
from lib.data import SimpleDataset

model = Model()
dataset = SimpleDataset()
train_loader, _ = split_dataset(dataset)
criterion = # Complete this
optim = # Complete this

for epoch in range(10):
    for batch in train_loader:
        inp, tgt = batch["input"], batch["target"]
        # Complete this
```

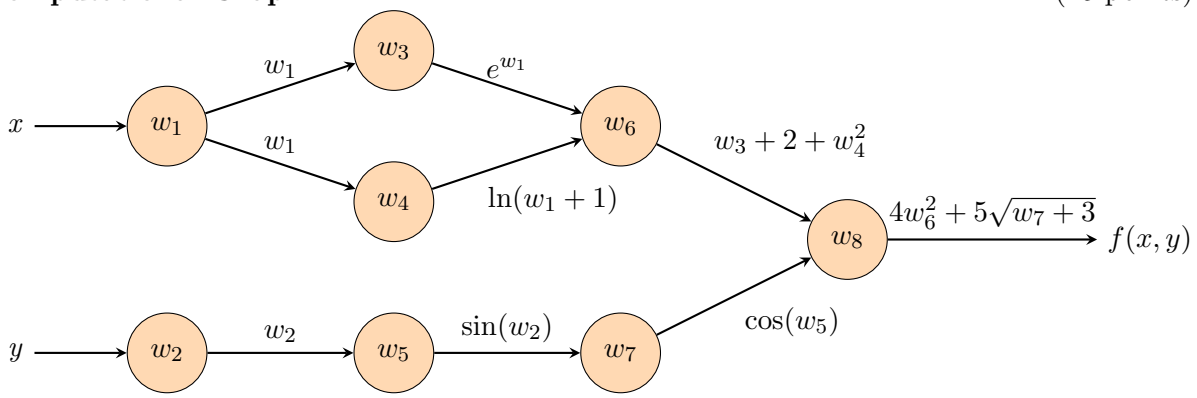
Solution:

```
import torch
from lib.model import Model
from lib.data import SimpleDataset

model = Model()
dataset = SimpleDataset()
train_loader, _ = split_dataset(dataset)
criterion = nn.MSELoss()
optim = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

for epoch in range(10):
    for batch in train_loader:
        inp, tgt = batch["input"], batch["target"]
        optim.zero_grad()
        output = model(inp)
        loss = criterion(output, tgt)
        loss.backward()
        optim.step()
```

8. **Computational Graph** (15 points)



(a) Determine the function $f(x, y)$ represented by the above computational graph.

Solution:

$$
\begin{aligned}
w_1 &= x, \quad w_2 = y, \\
w_3 &= e^{w_1} = e^x, \\
w_4 &= \ln(w_1 + 1) = \ln(x + 1), \\
w_5 &= \sin(w_2) = \sin(y), \\
w_6 &= w_3 + 2 + w_4^2 = e^x + 2 + \Big[\ln(x + 1)\Big]^2, \\
w_7 &= \cos(w_5) = \cos(\sin(y)), \\
f(x, y) = w_8 &= 4w_6^2 + 5\sqrt{w_7 + 3} = 4 * \left(e^x + 2 + [\ln(x + 1)]^2\right)^2 + 5\sqrt{\cos(\sin(y)) + 3}
\end{aligned}
$$

(b) Compute the partial derivatives of each successor node with respect to its predecessors.

Solution:

$$
\begin{aligned}
\frac{\partial w_6}{\partial w_3} &= 1 & \frac{\partial w_6}{\partial w_4} &= 2w_4 \\
\frac{\partial w_7}{\partial w_5} &= -\sin(w_5) & \frac{\partial w_5}{\partial w_2} &= \cos(w_2) \\
\frac{\partial w_8}{\partial w_6} &= 8w_6 & \frac{\partial w_8}{\partial w_7} &= \frac{5}{2\sqrt{w_7 + 3}} \\
\frac{\partial w_3}{\partial w_1} &= e^{w_1} & \frac{\partial w_4}{\partial w_1} &= \frac{1}{w_1 + 1}
\end{aligned}
$$

(c) Compute the adjoints at each node $\bar{w}_i = \frac{\partial f}{\partial w_i}$.

Solution:

$$
\begin{aligned}
\bar{w_8} &= 1 \\
\bar{w_6} &= \bar{w_8}\frac{\partial w_8}{\partial w_6} = 8w_6 \\
\bar{w_7} &= \bar{w_8}\frac{\partial w_8}{\partial w_7} = \frac{5}{2\sqrt{w_7+3}} \\
\bar{w_5} &= \bar{w_7}\frac{\partial w_7}{\partial w_5} = -\sin(w_5)\cdot\frac{5}{2\sqrt{w_7+3}} \\
\bar{w_4} &= \bar{w_6}\frac{\partial w_6}{\partial w_4} = 8w_6\cdot 2w_4 \\
\bar{w_3} &= \bar{w_6}\frac{\partial w_6}{\partial w_3} = 8w_6 \\
\bar{w_2} &= \bar{w_5}\frac{\partial w_5}{\partial w_2} = -\sin(w_5)\frac{5}{2\sqrt{w_7+3}}\cos(w_2) \\
\bar{w_1} &= \bar{w_3}\frac{\partial w_3}{\partial w_1} + \bar{w_4}\frac{\partial w_4}{\partial w_1} \\
&= 8w_6e^{w_1} + (16w_6w_4)\frac{1}{w_1+1}
\end{aligned}
$$

Solution:
We verified the correctness of our solution by verifying it using the PyTorch automatic differentiator. Feel free to play around with the code!

```
import torch
import numpy as np

x = torch.tensor([1.5], requires_grad=True) # make sure gradients are computed when backpropagation is called
y = torch.tensor([np.pi/3], requires_grad=True)

w1 = x
w2 = y
w3 = torch.exp(w1)
w4 = torch.log(w1+1)
w5 = torch.sin(w2)
w6 = w3 + 2 + w4**2
w7 = torch.cos(w5)
w8=4*(w6**2) + 5*torch.sqrt(w7+3)
f = w8

# manual gradients
with torch.no_grad():
    # adjoints
    w8bar = 1
    w7bar = 5/(2*torch.sqrt(w7+3))
    w6bar = 8*w6
    w5bar = -1*torch.sin(w5)*5/(2*torch.sqrt(w7+3))
    w4bar = 8*w6*2*w4
```

```
        w3bar = 8*w6
        w2bar = -1*torch.sin(w5)*5/(2*torch.sqrt(w7+3))*torch.cos(w2)
        w1bar = 8*w6*torch.exp(w1) + (16*w6*w4)/(w1+1)

    # automatic gradients via backpropagation
    w3.retain_grad(), w4.retain_grad(), w5.retain_grad(), w6.retain_grad()
    , w7.retain_grad(), w8.retain_grad() # making sure PyTorch populates
    all gradients
    f.backward() # initiate backpropagation from f as the seed node

    print("Making sure the overall equation is correct: ")
    f_manual = 4*(torch.exp(x)+2+(torch.log(x+1))**2)**2 + 5*torch.sqrt(
    torch.cos(torch.sin(y))+3)
    print('f: Manual = {}, PyTorch = {}'.format(f_manual, f))

    print('Comparing our calculations to PyTorch Autograd:')
    print('w1: Manual = {}, PyTorch = {}'.format(w1bar, w1.grad))
    print('w2: Manual = {}, PyTorch = {}'.format(w2bar, w2.grad))
    print('w3: Manual = {}, PyTorch = {}'.format(w3bar, w3.grad))
    print('w4: Manual = {}, PyTorch = {}'.format(w4bar, w4.grad))
    print('w5: Manual = {}, PyTorch = {}'.format(w5bar, w5.grad))
    print('w6: Manual = {}, PyTorch = {}'.format(w6bar, w6.grad))
    print('w7: Manual = {}, PyTorch = {}'.format(w7bar, w7.grad))
    print('w8: Manual = {}, PyTorch = {}'.format(w8bar, w8.grad))

```