HW 3 (Total 200 points) Due: Wed, 5/14 @ 5:00 pm

Problem 1 (30 pts). MNIST classification

```
import torch.nn as nn

model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(784, 64),
    nn.ReLU(),
    nn.Linear(64, 10)
)

loss_fn = nn.CrossEntropyLoss()
```

Correct label: `y = torch.tensor([3])`

or $y = (0,0,0,1,0,0,0,0,0,0)$

Untrained net: $\hat{y}$ = random vector (10 components)

A. Your softmax classifier outputs the vector $\hat{y} = [0.1, 0.1, 0.0, 0.4, 0.05, 0.05, 0.05, 0.05, 0.1, 0.1]$ for an image of the digit '3'. Compute the cross-entropy loss.

-1 * log(0.4) = -0.397

B. (Refresher) Why is the cross-entropy loss used for MNIST classification? Why is MSE a poor substitute? As a specific example, assume $\hat{y}_3 = 0.9$ (for the correct label $y_3 = 1$), and compute ∂ MSE/$\partial \hat{y}_k$ vs. ∂ CE/$\partial \hat{y}_k$. Which loss gives the larger gradient signal when the network is already confident, and why is that good?

C. (PyTorch quiz) In the pseudocode above, why is there no `nn.Softmax( )` layer between the final linear layer and `CrossEntropyLoss( )`? Shouldn't classification produce probabilities adding up to 1?

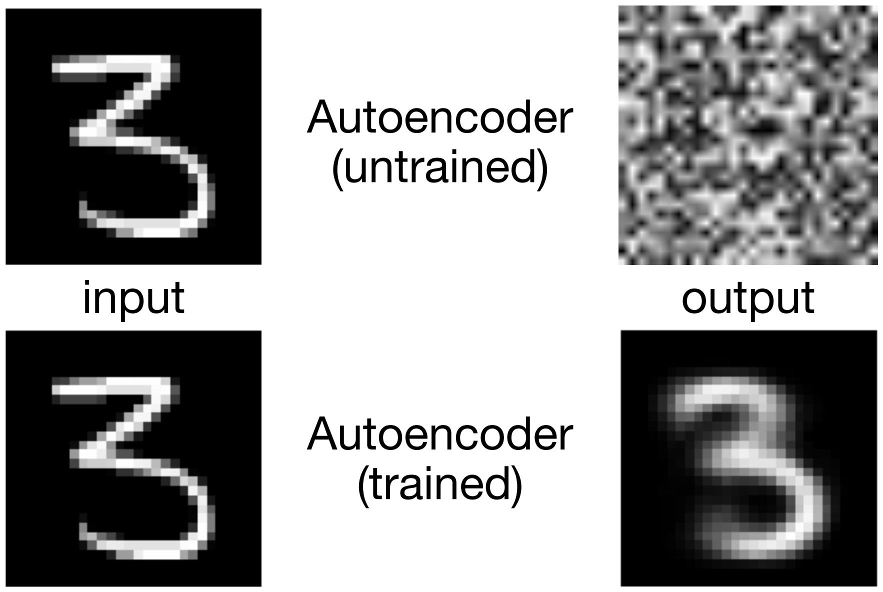
You must complete problem 1 without the use of AI tools

B.
Cross entropy loss penalizes confident wrong predictions more heavily.
Using the example given, I see that cross entropy shows LARGER gradient signal.
That is good because it allows for better learning of the classification.

C.
the CrossEntropyLoss() function in Pytorch has the Softmax functiionality already embedded in it.

HW 3

Prob 2 (20 pts). How to train an Autoencoder for MNIST reconstruction



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

model = Autoencoder()  # define this class separately

optimizer = optim.Adam(model.parameters(), lr=0.001)
loss_fn = _nn.MSELoss()__________                # (a)

for epoch in range(10):
    model.__train__()                            # (b)
    for images, _ in train_loader:
        x = images.view(images.size(0), ___-1___)   # (c)
        x_hat = model(x)
        loss = loss_fn(_x_hat__, __x____)        # (d)
        loss.backward()                          # (e)
        optimizer.__step___()                    # (f)
        optimizer.zero_grad()                    # (g)
```

A. Fill in the blanks (a) through (g)

B. What is this model learning and why the loss function (you chose) makes sense?

You must complete problem 2 without the use of AI tools

I use an MSE loss function because it is not a classification task but is instead reconstruction task. I need to be able to quantify how far off the the mdoel/s predictions are vs where the actual pixels should be. For this purpose MSE is best as it average the distance between predicted vs actual pixels.

This model is learning compressed representation of the numbers in MNIST dataset. It can map the original image 28 x 28 pixel on to just a few number (i.e 2 number)

HW 3

Problem 3 (35 pts). PyTorch refresher (look up manual; No AI use)

A. Which sub-package hosts `MNIST`? Fill in: `from` torchvisions `import` datasets

B. Where does `Adam` live? `from` torch.optim `import` Adam

C. Which class is responsible for shuffling and batching data during training?

```
from torch.utils.data import DataLoader
```

D. What happens if `loss.backward()` is missing? This function computes the loss gradient every input parameter. Without this the model fail to learn as the parameters will have no update during optimzer.step()

E. Why is it important to call `optimizer.zero_grad()` in every iteration? optimzer.zero_grad() clears the computed gradient for every parameter in the parameter. This prevents accumulation of gradients from previous passes and ensures that model trains on relevant/fresh gradients.

F. What happens to Dropout and batch normalization during `model.train()vs model.eval()?`

Batch Normalization uses the learned running statistics in model.eval, whereas, it used batch-specific stats in model.train.
Dropout is enabled is mode.train but is automatically disabled in mode.eval()

G. During training, your model achieves low loss and high accuracy. But during evaluation on the test set, performance is much worse. One possible cause is forgetting to switch the model to evaluation mode. What line of code should you add before running inference on the test set?

mode.eval() should be called before running the model on the test set

You must complete problem 3 without the use of AI tools

HW 3

Problem 4 (85 pts): Colab notebook

https://colab.research.google.com/drive/1Z_dWneqkI3Qnc37Or25-nRNrNItEKJWw?usp=sharing

For problem 4, AI-assisted coding is permitted.

HW 3

Problem 5 (30 pts): Reasoning problem. Do not use AI.

You have a dataset which would benefit from deep learning. The data consists of 8 features measured across 50 different time points (so each sample is a 50x8 matrix) and the goal is to predict which class the sample belongs to (represented by a 1 or a 0).

A) Why might a Multilayer Perceptron (MLP) be a poor choice for modeling this dataset?

B) When would a CNN be most effective at modeling this dataset? When would it be the least effective?

C) Name two other model families that would be a good choice for this dataset. For each, explain what about the model makes it useful for this type of analysis.

B.
A CNN would be most effective when trying to detect features from an image such as a person smiling or frowning etc. Also, positional invariant problems can be solved using a CNN, such as detecting a dog in a picture no matter where it appears.

It would be least effective if the images contain a pattern that changes with time over the 50 images. Such as temperature rising shown over the 50 images. CNNs struggle with capturing long-range dependencies across the entire input

C.
LSTM is well-suited for as task like this. It excels in time series analysis because it can learn and predict sequential patterns. because it is designed to process entire sequences of data.

GRU could also be used, as they are another type of RNNs just like LTSM. GRU models excel when trained on sequential data. It is not as superior in performance as LTSM but it does not have vanishing gradient problem like conventional neural networks.

You must complete problem 5 without the use of AI tools

A.
An MLP lacks the ability to extract spatial information from an image. A dog appearing at the bottom of the image vs the same dog appearing at the top of the image will be learned differently by the MLP. Also, in an MLP a pixel corresponds to 1 perceptron, which leads to an unmanageable amount of parameters for large images.