Homework 3

Neural Networks

UTK COSC 522: Machine Learning (Fall 2025)

OUT: Sunday, Nov 2nd, 2025

DUE: Sun, Dec 7th, 2025, 11:59pm, **Early Deadline: Sun, Nov 30th, 2025, 11:59pm**

Homework Instructions

- **Collaboration policy: Taking Responsibility for Your Career**

  The skills you learn in this course are tools for your future, and this policy is designed to help you sharpen them. You are responsible for your own learning and career, and this framework ensures you get the most from every assignment.

  We encourage collaboration, but it must be done *right*. *First*, make a genuine effort to solve problems on your own. This independent effort is where the most critical learning occurs. *Afterward*, you may discuss strategies with peers or consult resources to clarify concepts—the goal is to deepen your own understanding, not to simply get an answer.

  Finally, like any professional, you must stand behind your own work. Your submitted solution must be written entirely by you, from scratch, without collaborators present. This proves you have truly mastered the material. To maintain academic and professional integrity, you must also cite any person or resource that contributed to your understanding along the way.

- **Late Submission Policy:** See the late homework policy here.

- **Submitting your work:**

  - **Canvas:** For this homework, you will submit a **single .zip file** to Canvas. Please name your file in the format **YourName_NetID_HW3.zip**. This zip file must contain exactly two files: your written solutions in a single PDF and your Python code.

    Your **single PDF file** must contain your answers to all questions, including written problems (proofs, short answers, plots) and the empirical questions from the programming section. You must use the provided homework template. Using the LaTeX version to typeset is strongly recommended, but you may also submit a scan of the template with legible handwriting; **illegible answers will not be graded**. Please complete all answers within the provided boxes.

    The second file in your zip archive must be your completed programming files. Please ensure your code is runnable, as we may execute it to verify your results.

    Regrade requests can be made after homework grades are released. Please be

aware that a regrade request allows the TA to review your entire assignment, which may result in points being deducted if new errors are found.

1 Neural Nets: Written Questions [50 pts]

Note: We strongly encourage you to do the written part of this homework before the programming, as it will help you gain familiarity with the calculations you will have to code up in the programming section. We suggest that for each of these problems, you write out the equation required to calculate each value in terms of the variables we created (a_j, z_j, b_k, etc.) before you calculate the numerical value.

Note: For all questions which require numerical answers, round up your final answers to four decimal places. For integers, you may drop trailing zeros.

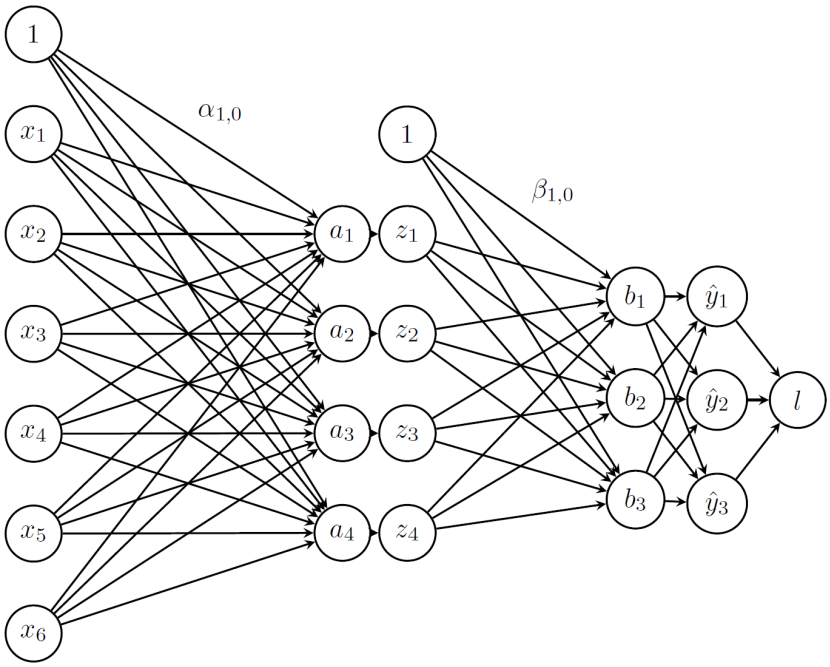

Figure 1: A One Hidden Layer Neural Network

Network Overview

Consider the neural network with one hidden layer shown in Figure 1. The input layer consists of 6 features $\mathbf{x} = [x_1, ..., x_6]^T$, the hidden layer has 4 nodes $\mathbf{z} = [z_1, ..., z_4]^T$, and the output layer is $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \hat{y}_3]^T$ that sums to one over 3 classes. We also add a bias to the input, $x_0 = 1$ and the hidden layer $z_0 = 1$, both of which are fixed to 1.

We adopt the following notation:

1. Let $\boldsymbol{\alpha}$ be the matrix of weights from the inputs to the hidden layer.
2. Let $\boldsymbol{\beta}$ be the matrix of weights from the hidden layer to the output layer.
3. Let $\alpha_{j,i}$ represent the weight going *to* the node z_j in the hidden layer *from* the node x_i in the input layer (e.g. $\alpha_{1,2}$ is the weight from x_2 to z_1)

4. Let $\boldsymbol{\beta}_{k,j}$ represent the weight going *to* the node y_k in the output layer *from* the node z_j in the hidden layer.
5. We will use a *sigmoid activation function* (σ) for the hidden layer and a *softmax* for the output layer.

Network Details

Equivalently, we define each of the following.

The input:

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6]^T \tag{1}$$

Linear combination at first (hidden) layer:

$$a_j = \alpha_{j,0} + \sum_{i=1}^{6} \alpha_{j,i} x_i, \quad j \in \{1, \dots, 4\} \tag{2}$$

Activation at first (hidden) layer:

$$z_j = \sigma(a_j) = \frac{1}{1 + \exp(-a_j)}, \quad j \in \{1, \dots, 4\} \tag{3}$$

Linear combination at second (output) layer:

$$b_k = \beta_{k,0} + \sum_{j=1}^{4} \beta_{k,j} z_j, \quad k \in \{1, \dots, 3\} \tag{4}$$

Activation at second (output) layer:

$$\hat{y}_k = \frac{\exp(b_k)}{\sum_{l=1}^{3} \exp(b_l)}, \quad k \in \{1, \dots, 3\} \tag{5}$$

Note that the linear combination equations can be written equivalently as the product of the weight matrix with the input vector. We can even fold in the bias term α_0 by thinking of $x_0 = 1$, and fold in β_0 by thinking of $z_0 = 1$.

Loss

We will use cross entropy loss, $\ell(\hat{\mathbf{y}}, \mathbf{y})$. If $\mathbf{y}$ represents our target (true) output, which will be a one-hot vector representing the correct class, and $\hat{\mathbf{y}}$ represents the output of the network, the loss is calculated as (note that the log terms are in base e):

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{3} y_k \log(\hat{y}_k) \tag{6}$$

1. **[12 pts]** In the following questions you will derive the matrix and vector forms of the previous equations which define our neural network. These are what you should hope to program in order to avoid excessive loops and large run times.
When working these out it is important to keep track of the vector and matrix dimensions in order for you to easily identify what is and isn't a valid multiplication. Suppose you are given a training example: $\mathbf{x}^{(1)} = [x_1, x_2, x_3, x_4, x_5, x_6]^T$ with **label class 2**, so $\mathbf{y}^{(1)} = [0, 1, 0]^T$. We initialize the network weights as:

$$\boldsymbol{\alpha}^* = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4} & \alpha_{1,5} & \alpha_{1,6} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} & \alpha_{2,5} & \alpha_{2,6} \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} & \alpha_{3,5} & \alpha_{3,6} \\ \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \alpha_{4,4} & \alpha_{4,5} & \alpha_{4,6} \end{bmatrix}$$

$$\boldsymbol{\beta}^* = \begin{bmatrix} \beta_{1,1} & \beta_{1,2} & \beta_{1,3} & \beta_{1,4} \\ \beta_{2,1} & \beta_{2,2} & \beta_{2,3} & \beta_{2,4} \\ \beta_{3,1} & \beta_{3,2} & \beta_{3,3} & \beta_{3,4} \end{bmatrix}$$

We want to also consider the bias term and the weights on the bias terms ($\alpha_{j,0}$ and $\beta_{k,0}$). To account for these we can add a new column to the beginning of our initial weight matrices.

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4} & \alpha_{1,5} & \alpha_{1,6} \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} & \alpha_{2,5} & \alpha_{2,6} \\ \alpha_{3,0} & \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} & \alpha_{3,5} & \alpha_{3,6} \\ \alpha_{4,0} & \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,0} & \alpha_{4,4} & \alpha_{4,5} & \alpha_{4,6} \end{bmatrix}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_{1,0} & \beta_{1,1} & \beta_{1,2} & \beta_{1,3} & \beta_{1,4} \\ \beta_{2,0} & \beta_{2,1} & \beta_{2,2} & \beta_{2,3} & \beta_{2,4} \\ \beta_{3,0} & \beta_{3,1} & \beta_{3,2} & \beta_{3,3} & \beta_{3,4} \end{bmatrix}$$

And we can set our first value of our input vectors to always be 1 ($x_0^{(i)} = 1$), so our input becomes:

$$\mathbf{x}^{(1)} = [1, x_1, x_2, x_3, x_4, x_5, x_6]^T$$

(a) **[2 pt]** By examining the shapes of the initial weight matrices, how many neurons do we have in the first hidden layer of the neural network? (Not including the bias neuron)

6

(b) **[2 pt]** How many output neurons will our neural network have?

3

(c) **[2 pt]** What is the vector $\mathbf{a}$ whose elements are made up of the entries a_j in equation (2). Write your answer in terms of $\boldsymbol{\alpha}$ and $\mathbf{x}^{(1)}$.

$\boldsymbol{\alpha}\mathbf{x}^{(1)}$

(d) **[2 pt]** What is the vector $\mathbf{z}$ whose elements are made up of the entries z_j in equation (3)? Write your answer in terms of $\mathbf{a}$.

$\sigma(\mathbf{a})$

(e) **[2 pt] Select one:** We cannot take the matrix multiplication of our weights $\boldsymbol{\beta}$ and our vector $\mathbf{z}$ since they are not compatible shapes. Which of the following would allow us to take the matrix multiplication of $\boldsymbol{\beta}$ and $\mathbf{z}$ such that the entries of the vector $\mathbf{b} = \boldsymbol{\beta}\mathbf{z}$ are equivalent to the values of b_k in equation (4)?

- ○ Remove the last column of $\boldsymbol{\beta}$
- ○ Remove the first row of $\mathbf{z}$
- ● Append a value of 1 to be the first entry of $\mathbf{z}$
- ○ Append an additional column of 1's to be the first column of $\boldsymbol{\beta}$
- ○ Append a row of 1's to be the first row of $\boldsymbol{\beta}$
- ○ Take the transpose of $\boldsymbol{\beta}$

(f) **[2 pt]** What are the entries of the output vector $\hat{\mathbf{y}}$? Your answer should be written in terms of b_1, b_2, b_3.

$\hat{\mathbf{y}} = [\frac{exp(b_1)}{\sum_{i=1}^{3} exp(b_i)}, \frac{exp(b_2)}{\sum_{i=1}^{3} exp(b_i)}, \frac{exp(b_3)}{\sum_{i=1}^{3} exp(b_i)}]$

2. **[14 pts]** We will now derive the matrix and vector forms for the backpropagation algorithm.

$$\frac{d\ell}{d\boldsymbol{\alpha}} = \begin{bmatrix} \frac{dJ}{d\alpha_{10}} & \frac{dJ}{d\alpha_{11}} & \cdots & \frac{dJ}{d\alpha_{1M}} \\ \frac{dJ}{d\alpha_{20}} & \frac{dJ}{d\alpha_{21}} & \cdots & \frac{dJ}{d\alpha_{2M}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dJ}{d\alpha_{D0}} & \frac{dJ}{d\alpha_{D1}} & \cdots & \frac{dJ}{d\alpha_{DM}} \end{bmatrix}$$

Recall that ℓ is our loss function defined in equation (6)

(a) **[2 pt]** The derivative of the softmax function with respect to b_k is as follows:

$$\frac{d\hat{y}_l}{db_k} = \hat{y}_l(\mathbb{I}[k = l] - \hat{y}_k)$$

where $\mathbb{I}[k = l]$ is an indicator function such that if $k = l$ then it it returns value 1 and 0 otherwise. Using this, write the derivative $\frac{d\ell}{db_k}$ in a smart way such that you do not need this indicator function. Write your solutions in terms of $\hat{y}_k, y_k$.

$\frac{d\hat{y}_l}{db_k} = \hat{y}_k - y_k$

(b) **[2 pt]** What are the elements of the vector $\frac{d\ell}{db}$? (Recall that $\mathbf{y}^{(1)} = [0, 1, 0]^T$)

$[\hat{y}_1, \hat{y}_2 - 1, \hat{y}_3]$

(c) **[2 pt]** What is the derivative $\frac{d\ell}{d\beta}$? Your answer should be in terms of $\frac{d\ell}{d\mathbf{b}}$ and $\mathbf{z}$.

$\frac{dl}{d\mathbf{b}}\mathbf{z}^T$

(d) **[2 pt]** Explain in one short sentance why must we go back to using the matrix $\boldsymbol{\beta}^*$ (The matrix $\boldsymbol{\beta}$ without the first column of ones) when calculating the matrix $\frac{d\ell}{d\alpha}$?

The bias term is disconnected from any neurons in our hidden layer, therefore it should not be included in computing the gradient for our weight matrix connecting the input and hidden layer.

(e) **[2 pt]** What is the derivative $\frac{d\ell}{d\mathbf{z}}$? Your answer should be in terms of $\frac{d\ell}{d\mathbf{b}}$ and $\boldsymbol{\beta}^*$

$\frac{dl}{d\mathbf{b}} = \beta^{*T}\frac{dl}{d\mathbf{b}}$

(f) **[2 pt]** What is the derivative $\frac{d\ell}{d\mathbf{a}}$ in terms of $\frac{d\ell}{d\mathbf{z}}$ and $\mathbf{z}$

$\frac{dl}{da} = \frac{dl}{dz} \cdot z \cdot (1 - z)$

(g) **[2 pt]** What is the matrix $\frac{d\ell}{d\alpha}$? Your answer should be in terms of $\frac{d\ell}{d\mathbf{a}}$ and $x^{(1)}$.

$\frac{dl}{d\alpha} = \frac{dl}{da}(\mathbf{x}^{(1)})^T$

Prediction

When doing prediction, we will predict the argmax of the output layer. For example, if $\hat{\mathbf{y}}$ is such that $\hat{y}_1 = 0.3,\ \hat{y}_2 = 0.2,\ \hat{y}_3 = 0.5$ we would predict class 3 for the input $\mathbf{x}$. If the true class from the training data $\mathbf{x}$ was 2 we would have a one-hot vector $\mathbf{y}$ with values $y_1 = 0,\ \ y_2 = 1,\ \ y_3 = 0$.

3. **[16 pts]** We initialize the weights as:

$$\boldsymbol{\alpha}^* = \begin{bmatrix} 2 & 1 & -1 & -1 & 0 & -2 \\ 0 & 1 & 0 & -1 & 1 & 3 \\ -1 & 2 & 1 & 3 & 1 & -1 \\ 1 & 3 & 4 & 2 & -1 & 2 \end{bmatrix}$$

$$\boldsymbol{\beta}^* = \begin{bmatrix} 2 & -2 & 2 & 1 \\ 3 & -1 & 1 & 2 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

And weights on the bias terms ($\alpha_{j,0}$ and $\beta_{j,0}$) are initialized to 1.

You are given a training example $\mathbf{x}^{(1)} = [1, 0, 1, 0, 1, 0]^T$ with label class 2, so $\mathbf{y}^{(1)} = [0, 1, 0]^T$. Using the initial weights, run the feed forward of the network over this training example (without rounding during the calculation) and then answer the following questions.

(a) **[2 pt]** What is the value of a_1?

2

(b) **[2 pt]** What is the value of z_1?

0.8808

(c) **[2 pt]** What is the value of a_3?

2

(d) **[2 pt]** What is the value of z_3?

0.8808

(e) **[2 pt]** What is the value of b_2?

22

(f) **[2 pt]** What is the value of $\hat{y}_2$?

0.8588

(g) **[2 pt]** Which class value we would predict on this training example?

2

(h) **[2 pt]** What is the value of the total loss on this training example?

0.1522

4. **[8 pts]** Now use the results of the previous question to run backpropagation over the network and update the weights. Use the learning rate $\eta = 1$.

   Do your backpropagation calculations without any rounding then answer the following questions: (in your final responses round to four decimal places)

   (a) **[2 pt]** What is the updated value of $\beta_{2,1}$?

   -0.0083

   (b) **[3 pt]** What is the updated weight of the hidden layer bias term applied to y_1 (i.e. $\beta_{1,0}$)?

   1.1412

(c) [**3 pt**] What is the updated value of $\alpha_{3,4}$?

2

2 Neural Net Programming [50 pts]

Figure 2: Random Images of Each of 10 classes in Fashion-MNIST

Your goal in this assignment is to label images of fashion articles by implementing a neural network from scratch. You will implement all of the functions needed to initialize, train, evaluate, and make predictions with the network. **Important: You must use PyTorch to complete this assignment. However, you are not allowed to directly use builtin PyTorch modules (e.g., nn.Linear, nn.Sigmoid, nn.Softmax, etc...) for the module implementations. Same for the corresponding forward() and backward() function implementations.)**

The Fashion-MNIST dataset is comprised of 70,000 images of fashion articles and their respective labels. There are 60,000 training images and 10,000 test images, all of which are 28 pixels by 28 pixels. The images belong to the following 10 categories – [T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot].

Dataset format In the file `nn_implementation_code/base_experiment.py` there are lines for downloading the dataset and converting it to the `torch.data.Dataset` format. More information about `torch.data.Dataset` and `torch.data.Dataloader` can be found in this tutorial; you may use these classes to operate with data and batches.

Model Definition

Preliminaries

In this section, you will implement a single-hidden-layer neural network with a sigmoid activation function for the hidden layer, and a softmax on the output layer. For this particular problem, the input vectors $\mathbf{x}$ are of length $M = 28 \times 28$, the hidden layer $\mathbf{z}$ consist of D

hidden units, and the output layer $\hat{\mathbf{y}}$ represents a probability distribution over the $K = 10$ classes. In other words, each element $\hat{y}_k$ of the output vector $\hat{\mathbf{y}}$ represents the probability of $\mathbf{x}$ belonging to the class k.

Following the notation from the written section we have:

- For the output layer:

$$\hat{y}_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}, \quad k \in \{1, \ldots, K\}, \quad \text{(softmax activation)}$$

$$b_k = \beta_{k,0} + \sum_{j=1}^{D} \beta_{kj} z_j, \quad k \in \{1, \ldots, K\}, \quad \text{(pre-activation)}$$

- For the hidden layer:

$$z_j = \frac{1}{1 + \exp(-a_j)}, \quad j \in \{1, \ldots, D\}, \quad (\sigma - \text{activation})$$

$$a_j = \alpha_{j,0} + \sum_{i=1}^{M} \alpha_{ji} x_i, \quad j \in \{1, \ldots, D\}, \quad \text{(pre-activation)}$$

It is possible to compactly express this model by assuming that $x_0 = 1$ is a bias feature on the input and that $z_0 = 1$ is also fixed. In this way, we have two parameter matrices $\boldsymbol{\alpha} \in \mathbb{R}^{D \times (M+1)}$ and $\boldsymbol{\beta} \in \mathbb{R}^{K \times (D+1)}$. The extra 0th column of each matrix (i.e. $\boldsymbol{\alpha}_{\cdot,0}$ and $\boldsymbol{\beta}_{\cdot,0}$) hold the bias parameters. With these considerations we have,

$$\hat{y}_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}, \quad k \in \{1, \ldots, K\}$$

$$b_k = \sum_{j=0}^{D} \beta_{kj} z_j, \quad k \in \{1, \ldots, K\}$$

$$z_j = \frac{1}{1 + \exp(-a_j)}, \quad j \in \{1, \ldots, D\}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \quad j \in \{1, \ldots, D\}$$

Objective function

Since the output corresponds to a probabilistic distribution over the K classes, the objective (cost) function we will use for training our neural network is the average cross entropy,

$$J(\boldsymbol{\alpha}, \boldsymbol{\beta}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} y_k^{(n)} \log(\hat{y}_k^{(n)}) \tag{7}$$

over the training dataset,

$$\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}, \quad \text{for } n \in \{1, \dots N\}$$

In Equation (7), J is a function of the model parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ because $\hat{y}_k^{(n)}$ is implicitly a function of $\mathbf{x}^{(n)}$, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$ since it is the output of the neural network applied to $\mathbf{x}^{(n)}$. As before, $\hat{y}_k^{(n)}$ and $y_k^{(n)}$ present the k-th component of $\hat{\mathbf{y}}^{(n)}$ and $\mathbf{y}^{(n)}$ respectively.

To train the network, you should optimize this objective function using stochastic gradient descent (SGD), where the gradient of the parameters for each training example is computed via backpropagation, though typically you should shuffle your data during SGD you are **not** to do so here, and instead you are to train through the dataset in the order it is given.

Implementation

To proceed, you are provided with the following guide. This is recommended but absolutely not required.

Defining layers

Just to recap, your network architecture should look like the following: `Linear`→ `Sigmoid` → `Linear` → `Softmax`. The size of the input is 784. The first linear layer can have 785 nodes to include bias term. The final (output) layer should have 10 nodes (one corresponding to each integer from 0 - 9). The hidden layer will have D = 256 nodes (excluding bias term).

Again, you are not allowed to directly use built-in PyTorch modules (e.g., nn.Linear, nn.Sigmoid, nn.Softmax, etc...) for the module implementations. Instead, you should complete the methods defined for you all in `custom_functions.py` to implement your own versions of these layers.

Be aware of computing issues! $\log(x)$ is problematic when $x \to 0$. Similarly $exp(x)$ may overflow when it is huge. Think of using log to avoid some exponential calculations and dividing both numerator and denominator by a large value to avoid overflowing:

$$\frac{e^{x_i}}{\sum e^{x_j}} = \frac{e^{x_i - b}}{\sum e^{x_j - b}}$$

Pseudo-code for Training Loop

```
For epoch in epochs:
        for x, y in train_x, train_y:
                Pass x through all the forward layers and get the loss
                Pass the output though all the backward layers
                Update weights and biases
        compute the average training loss for the epoch
        compute test loss
        compute test accuracy
```

You can follow the steps mentioned above or approach it any other way.

Important Tips

- The training loss you report, should not be aggregated throughout an epoch, rather you should compute the loss on the whole training set at the end of each epoch.
- When performing mini-batch gradient descent, you should take the mean of the loss within a batch before applying the update.
- Do NOT shuffle the training or test data, as we simply want more stability when looking at different student submissions.

Programming Submission

NOTE: Use the following hyper-parameters:

- **Learning rate = 0.01**
- **Number of epochs = 15**
- **Width of hidden layer = 256 (excluding bias)**
- **(Q1-Q2 only) Batch size = 1**

For initializing the network's weights, we will use the **Xavier (Glorot) Uniform Initialization** strategy. This is a common and effective method for networks using `sigmoid` activations, as it helps keep the signal variance stable during forward and backward passes. You can find the official PyTorch documentation for this function here: `torch.nn.init.xavier_uniform_`. All biases will be initialized to zero.

Below is a code example demonstrating how to initialize weight and bias tensors.

```
# import initialization function and define layer dimensions
import torch.nn.init as init
n_inputs = 784
n_hidden = 256

# Create empty tensors
W1 = torch.empty(n_inputs, n_hidden)
b1 = torch.empty(n_hidden)
# Apply Xavier uniform initialization for weight matrix and set the bias to zero
init.xavier_uniform_(W1)
init.zeros_(b1)
```

Based on the information above, answer the questions below:

1. [**6 pts**] List the average test loss at the end of each epoch. Report numbers till 4 places of decimal.

```
Epoch 1/15, Test Loss: 0.4668
Epoch 2/15, Test Loss: 0.4271
Epoch 3/15, Test Loss: 0.4058
Epoch 4/15, Test Loss: 0.3918
Epoch 5/15, Test Loss: 0.3813
Epoch 6/15, Test Loss: 0.3731
Epoch 7/15, Test Loss: 0.3665
Epoch 8/15, Test Loss: 0.3612
Epoch 9/15, Test Loss: 0.3569
Epoch 10/15, Test Loss: 0.3534
Epoch 11/15, Test Loss: 0.3507
Epoch 12/15, Test Loss: 0.3485
Epoch 13/15, Test Loss: 0.3470
Epoch 14/15, Test Loss: 0.3460
Epoch 15/15, Test Loss: 0.3453
```

2. [**6 pts**] List the test accuracy at the end of each epoch. Report numbers till 4 places of decimal.

```
Epoch 1/15, Test Accuracy: 83.15%
Epoch 2/15, Test Accuracy: 84.8%
Epoch 3/15, Test Accuracy: 85.63%
Epoch 4/15, Test Accuracy: 86.17%
Epoch 5/15, Test Accuracy: 86.56%
Epoch 6/15, Test Accuracy: 86.88%
Epoch 7/15, Test Accuracy: 87.13%
Epoch 8/15, Test Accuracy: 87.33%
Epoch 9/15, Test Accuracy: 87.51%
Epoch 10/15, Test Accuracy: 87.69%
Epoch 11/15, Test Accuracy: 87.7%
Epoch 12/15, Test Accuracy: 87.79%
Epoch 13/15, Test Accuracy: 87.83%
Epoch 14/15, Test Accuracy: 88.01%
Epoch 15/15, Test Accuracy: 88.04%
```

3. [**7 pts**] For the experiments until now, we were using stochastic gradient descent (SGD) i.e. batch size = 1. We will now train our model on batches of data i.e. a mini-batch gradient descent. Run the model for 50 epochs with a batch size of 5, and report the average final training loss and the test accuracy. Report numbers till 4 places of decimal.

```
Training Loss: 0.2331
```

```
Test Accuracy: 0.3357
```

4. [**7 pts**] For the model trained for 50 epochs, display the confusion matrix for both the training and test sets. In the plot, you should have *true labels* on the y-axis and *predicted labels* on the x-axis. For example, the second entry in the fourth row should show the count for the number of times class 3 was wrongly predicted as 1.

Figure 3: Confusion Matrix After Training

5. [**3 pts**] For each class, display the first data point in the test dataset that was wrongly classified.

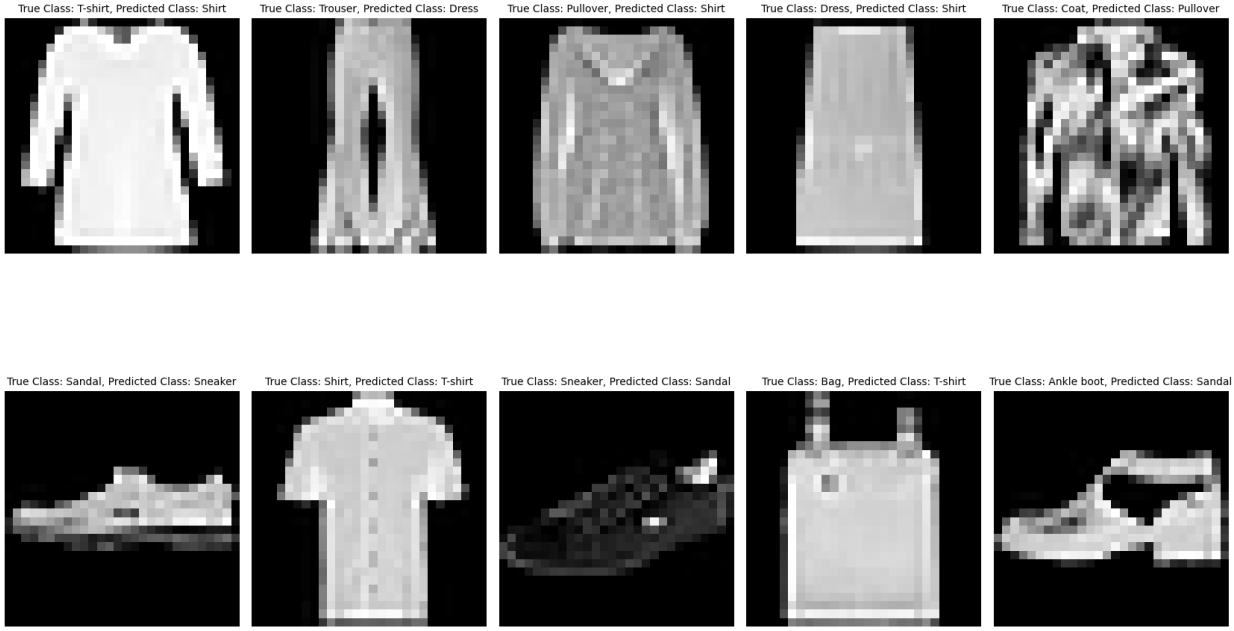

Figure 4: First Misclassification per Class

6. **[8 pts]** In this section, we will experiment with different batch sizes. For each batch size in [10, 50, 100], run your model for 50 epochs. Plot two graphs of epoch vs loss, one each for training and test loss, which have the epoch number on the x-axis and the loss value on the y-axis. Please make sure your graphs are properly labeled and include a legend showing the color for each batch size.

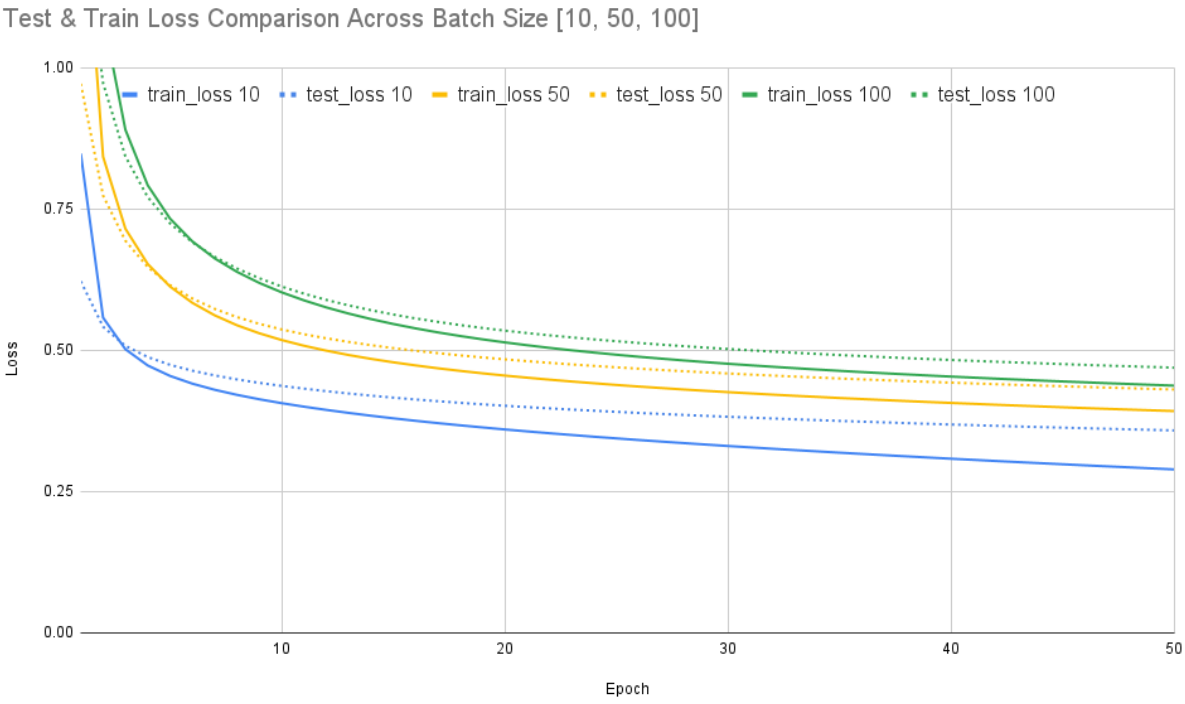

Figure 5: Experiment Varying Batch Size

7. **[6 pts]** Based on the plots in the previous part, what do you observe about the effect of increasing batch sizes? How does this relate to the learning rate? (Explain in 1-2 lines)

Larger batch sizes show slower convergence, which makes sense as we're averaging out the gradients over a larger number of data pointers. Naively I would assume that a larger batch size could tolerate a higher learning rate, as the smoothing factor of a large batch size would ensure the updates are generally moving in the right direction.

8. **[7 pts]** Now we want to give you a chance to experiment with the model by exploring the hyper-parameters of a neural network. Produce a pair of plots similar to that of the one requested in the batch size experiment, by varying a different hyperparameter in the network. Experiment with at least 3 different values for the hyperparameter that you decide to modify. For your experiment, you must,

   (a) describe the hyperparameter you varied

   (b) list the values you set the hyperparameter to

   (c) detail the setting of all the other hyperparameters, and

   (d) analyze your results in terms of the hyperparameter that you varied.

   For your reference here are some hyperparameters which you can change:

   - Learning Rate
   - Width of Hidden Layer
   - Different weight initialization

   You may choose any reasonable batch size and number of epochs for this question, just make sure to specify all design decisions you make. This question is for learning so feel free to experiment with whatever hyperparameter you wish to, as long you can provide a reasonable explanation for the observed behavior in the graphs.

I varied the hidden layer size.
The hyperparameter was varied between [8,16,32,64,128,256,512].
Learning rate was 0.01, epochs was 50, batch size was 10.
This experiment shows how hidden layer says plays an important role in the performance of a neural network. The default choice of 256 seems to be a good fit for this dataset, as going beyond to 512 offers little benefit. However, lowering the hidden layer size below 256 shows smaller drop offs until we reach 16, which shows a clear reduction in performance continuing down to 8 as well. The observed behavior makes sense as fashion mnist requires learning the complex patterns of 10 different clothing items, which cannot be easily distilled down to 8 or even 16 features in our hidden layer. The larger our hidden layer the more complex our network becomes, allowing it to recognize more individual features. For this same reason we also do not want to have too large of a hidden layer as it could drastically increase the complexity of our solution, which will likely decrease the generalization of our network.

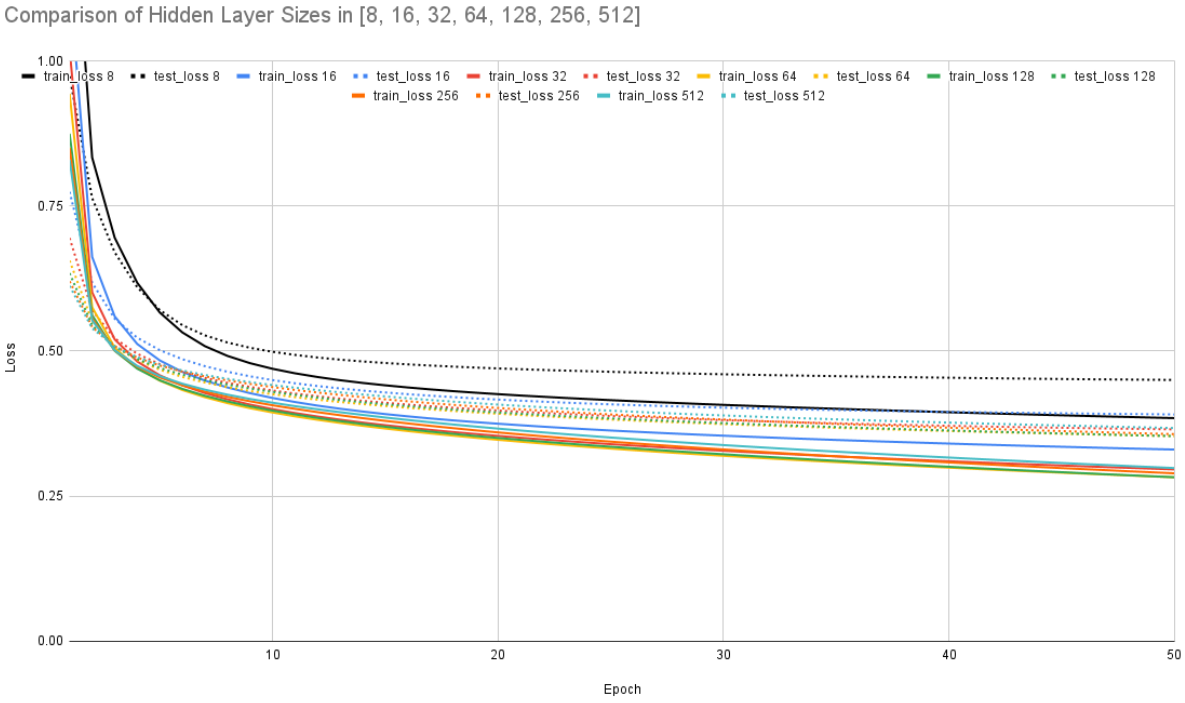

Figure 6: Experiment Varying Batch Size

3 Collaboration Questions

1. (a) Did you receive any help whatsoever from anyone in solving this assignment? **SOL:** No.

   (b) If you answered 'yes', give full details (e.g. "Jane Doe explained to me what is asked in Question 3.4")

   SOL:

2. (a) Did you give any help whatsoever to anyone in solving this assignment? **SOL:** No.

   (b) If you answered 'yes', give full details (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2")

   SOL:

3. (a) Did you find or come across code that implements any part of this assignment? **SOL:** No.

   (b) If you answered 'yes', give full details (book & page, URL & location within the page, etc.).

   SOL: