

HOMework 3

REGULARIZATION AND NEURAL NETWORKS

CMU 10-701: MACHINE LEARNING (SPRING 2026)

<https://www.piazza.com/cmu/spring2026/10701>

OUT: Friday, Feb 20, 2026

DUE: Tuesday, Mar 10th, 2026, 11:59pm

START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., “Jane explained to me what is asked in Question 2.1”). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section in our course syllabus for more information: <https://www.cs.cmu.edu/~10701-s26#Syllabus>
- **Late Submission Policy:** See the late submission policy here: <https://www.cs.cmu.edu/~10701-s26/#Syllabus>
- **Submitting your work to Gradescope:** There will be two submission slots for this homework on Gradescope, the Written and the Programming:
 - For the written problems such as short answer, multiple choice, derivations, proofs, or plots, we will be using the written submission slot. Please use the provided template. The best way to format your homework is by using the Latex template released in the handout and writing your solutions in Latex. However submissions can be handwritten onto the template, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Each derivation/proof should be completed in the boxes provided below the question, **you should not move or change the sizes of these boxes** as Gradescope is expecting your solved homework PDF to match the template on Gradescope. If you find you need more space than the box provides you should consider cutting your solution down to its relevant parts, if you see no way to do this, please add an additional page at the end of the homework and guide us there with a ‘See page xx for the rest of the solution’.
 - You are also required to upload your code, which you wrote to solve the final questions of this homework, to the Programming submission slot. Your code may be run by TAs so please make sure it is in a workable state.

Regrade requests can be made after the homework grades are released, however this gives the TA the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted.

For multiple choice or select all that apply questions, shade in the box or circle in the template document corresponding to the correct answer(s) for each of the questions. For \LaTeX users, use \blacksquare and \bullet for shaded boxes and circles, and don’t change anything else. If an answer box is included for showing work, **you must show your work!**

1 Regularization [11 Points]

1. Lasso is another form of regularized linear regression, where the L1 norm of the parameter vector is penalized. It is used in an attempt to get a sparse parameter vector where features of little “importance” are assigned zero weight. But how does L1 regularization encourage sparse parameters? In the following questions, you are going to examine this.

Let \mathbf{X} denote an $n \times d$ matrix where rows are training points, \mathbf{y} denote an $n \times 1$ vector of their corresponding targets, \mathbf{w} denote a $d \times 1$ parameter vector and \mathbf{w}^* denote the optimal parameter vector. To make the analysis easier we will consider the special case where the training data is whitened (i.e., $X^\top X = I$). For lasso regression, the optimal parameter vector is given by

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} J_\lambda(\mathbf{w}),$$

where $J_\lambda(\mathbf{w})$ is the function we want to minimize, which is given by

$$J_\lambda(\mathbf{w}) = \frac{1}{2} \|\mathbf{y} - X\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1,$$

for some $\lambda > 0$. Note that the L_1 norm of a vector $\mathbf{w} = [w_1, \dots, w_d]^\top \in \mathbb{R}^d$ is defined as $\|\mathbf{w}\|_1 = |w_1| + \dots + |w_d|$.

In [1a](#) and [1b](#), we will show that whitening the training data nicely decouples the features, making \mathbf{w}_i^* determined solely by the i th feature and the output, independent of all the other features. To show this, we begin by writing $J_\lambda(\mathbf{w})$ as

$$\begin{aligned} J_\lambda(\mathbf{w}) &= \frac{1}{2} (\mathbf{y} - X\mathbf{w})^\top (\mathbf{y} - X\mathbf{w}) + \lambda \|\mathbf{w}\|_1 \\ &= \frac{1}{2} (\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top X\mathbf{w} + \mathbf{w}^\top X^\top X\mathbf{w}) + \lambda \|\mathbf{w}\|_1 \\ &= \frac{1}{2} \mathbf{y}^\top \mathbf{y} + \sum_{i=1}^d \left(-\mathbf{y}^\top X_{\cdot i} \mathbf{w}_i + \frac{1}{2} \mathbf{w}_i^2 + \lambda |\mathbf{w}_i| \right) \end{aligned}$$

where $X_{\cdot i}$ is the i th column of X .

- (a) **[3 Points]** Assume that $w_i^* > 0$, what is the value of w_i^* in this case?

- (b) **[3 Points]** Assume that $w_i^* < 0$, what is the value of w_i^* in this case?

- (c) **[5 Points]** From **1a** and **1b**, what is the condition for w_i^* to be 0? How can you interpret that condition?

Now let's return to ridge regression where the regularization term is replaced by $\frac{1}{2}\lambda\|\mathbf{w}\|_2^2$. The corresponding optimization problem regarding \mathbf{w}_i is given by,

$$-y^T \mathbf{X}_{.i} \mathbf{w}_i + \frac{1}{2} \mathbf{w}_i^2 + \frac{1}{2} \lambda \mathbf{w}_i^2$$

Taking the derivative and equating to 0, we have:

$$\mathbf{w}_i^* = \frac{y^T \mathbf{X}_{.i}}{1 + \lambda}$$

which is equal to 0 iff $y^T \mathbf{X}_{.i}$ is precisely zero (or λ goes to infinity). Hence, we can argue that lasso regression encourages sparsity relative to ridge regression!

2 Neural Nets: Written Questions [29 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.

Note: Questions 1-7 will be autograded, but please also enter your answers in the written PDF. You will be submitting the files: `base_experiment.py`, `custom_functions.py` and `custom_modules.py` on Gradescope.

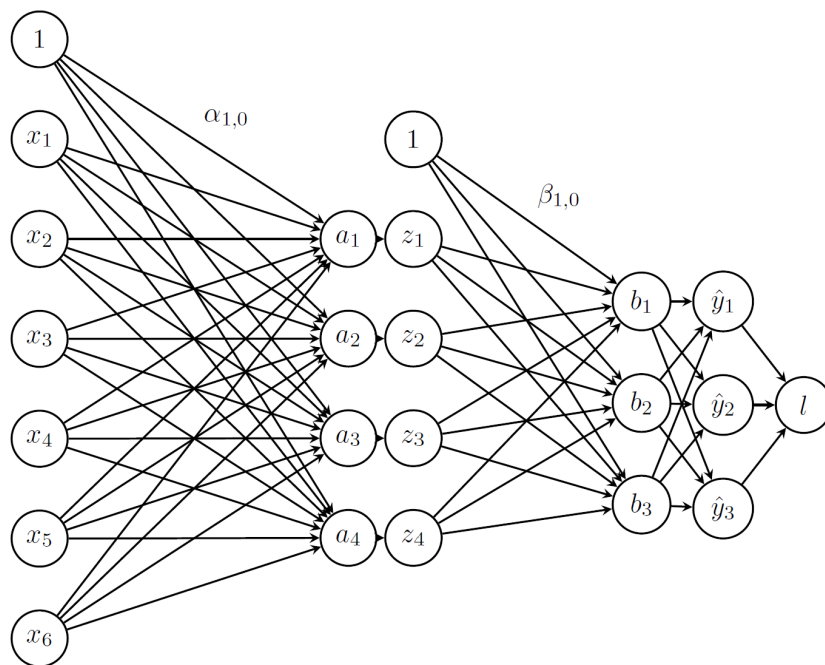


Figure 1: A One Hidden Layer Neural Network

Network Overview

Consider the neural network with one hidden layer shown in Figure 1. The input layer consists of 6 features $\mathbf{x} = [x_1, \dots, x_6]^T$, the hidden layer has 4 nodes $\mathbf{z} = [z_1, \dots, z_4]^T$, 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 α be the matrix of weights from the inputs to the hidden layer.
2. Let β 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 $\beta_{k,j}$ 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:

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

Linear combination at first (hidden) layer:

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

Activation at first (hidden) layer:

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

Linear combination at second (output) layer:

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

Activation at second (output) layer:

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

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):

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

1. **[6 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:

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

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

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.

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

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

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) **[1 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)

- (b) **[1 pt]** How many output neurons will our neural network have?

- (c) **[1 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 α and $\mathbf{x}^{(1)}$.

- (d) **[1 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} .

- (e) **[1 pt] Select one:** We cannot take the matrix multiplication of our weights β and our vector \mathbf{z} since they are not compatible shapes. Which of the following would allow us to take the matrix multiplication of β and \mathbf{z} such that the entries of the vector $\mathbf{b} = \beta\mathbf{z}$ are equivalent to the values of b_k in equation (4)?

- ☐ Remove the last column of β
- ☐ 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 β
- ☐ Append a row of 1's to be the first row of β

○ Take the transpose of β

- (f) [1 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 .

2. [7 pts] We will now derive the matrix and vector forms for the backpropagation algorithm.

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

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

- (a) [1 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 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 .

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

- (c) [1 pt] What is the derivative $\frac{d\ell}{d\beta}$? Your answer should be in terms of $\frac{d\ell}{db}$ and \mathbf{z} .

- (d) [1 pt] Explain in one short sentence why must we go back to using the matrix β^* (The matrix β without the first column of ones) when calculating the matrix $\frac{d\ell}{d\alpha}$?

- (e) [1 pt] What is the derivative $\frac{d\ell}{dz}$? Your answer should be in terms of $\frac{d\ell}{d\mathbf{b}}$ and β^*

- (f) [1 pt] What is the derivative $\frac{d\ell}{da}$ in terms of $\frac{d\ell}{dz}$ and \mathbf{z}

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

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. [8 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) [1 pt] What is the value of a_1 ?

- (b) [1 pt] What is the value of z_1 ?

- (c) [1 pt] What is the value of a_3 ?

(d) **[1 pt]** What is the value of z_3 ?

(e) **[1 pt]** What is the value of b_2 ?

(f) **[1 pt]** What is the value of \hat{y}_2 ?

(g) **[1 pt]** Which class value we would predict on this training example?

(h) **[1 pt]** What is the value of the total loss on this training example?

4. **[3 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) **[1 pt]** What is the updated value of $\beta_{2,1}$?

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

- (c) [1 pt] What is the updated value of $\alpha_{3,4}$?

5. [5 pts] Let us now introduce regularization into our neural network. For this question, we will incorporate L2 regularization into our loss function $\ell(\hat{\mathbf{y}}, \mathbf{y})$, with the parameter λ controlling the weight given to the regularization term.

- (a) [1 pt] Write the expression for the regularized loss function of our network after adding L2 regularization (**Hint:** Remember that bias terms should not be regularized!)

- (b) [1 pts] Compute the regularized loss for training example $\mathbf{x}^{(1)}$ (assume $\lambda = 0.01$ and use the weights before backpropagation)

- (c) [1 pts] For a network which uses the regularized loss function, write the gradient update equation for $\alpha_{j,i}$. You may use $\frac{\partial \ell(\hat{\mathbf{y}}, \mathbf{y})}{\partial \alpha_{j,i}}$ to denote the gradient update w.r.t non-regularized loss and η to denote the learning rate.

(d) **[2 pts]** Based on your observations from previous questions, **select all statements which are true:**

- ☐ The non-regularized loss is always higher than the regularized loss
- ☐ As weights become larger, the regularized loss increases faster than non-regularized loss
- ☐ On adding regularization to the loss function, gradient updates for the network become larger
- ☐ When using large initial weights, weight values decrease more rapidly for a network which uses regularized loss
- ☐ None of the above

3 Neural Net Programming [40 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:

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

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

- For the hidden layer:

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

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

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. $\boldsymbol{\alpha}_{:,0}$ and $\boldsymbol{\beta}_{:,0}$) hold the bias parameters. With these considerations we have,

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

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

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

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

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**,

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

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 $x \rightarrow 0$. 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 through 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, otherwise the answers won't match.

Programming Submission

NOTE: Initial weights and biases are provided in the file `weights.pt`. Use the following hyper-parameters:

- Learning rate = 0.01
- Number of epochs = 15
- Width of hidden layer = 256 (excluding bias)
- (Q1-Q6 only) Batch size = 1

We have provided the outputs of first five nodes of each layer and the updated weights for the first data point in the first epoch in the folder `check`. Use these to verify your implementation.

1. [1 pt] **Linear** : What is the value of a_{10} for first data point in first epoch?

2. [1 pt] **Sigmoid** : What is the value of z_{20} for first data point in first epoch?

3. [1 pt] What is the predicted class for the first data point x_1 after the first forward pass in the first epoch. Remember that our y values go from 0 to 9.

4. [4 pts] What are the values of $\beta_{k,0}$ for $k \in \{1, \dots, K\}$ (bias terms of the second layer) at the end of third epoch? Report numbers till 4 places of decimal and in the correct order.

5. [4 pts] List the average test loss at the end of each epoch. Report numbers till 4 places of decimal.

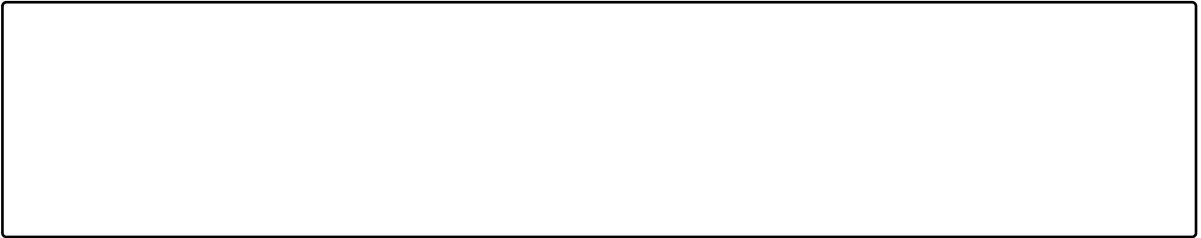
6. [4 pts] List the test accuracy at the end of each epoch. Report numbers till 4 places of decimal.

7. [2 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.

8. [4 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.



9. [2 pts] For each class, display the first data point in the test dataset that was wrongly classified.



10. **[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.

11. **[2 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)

12. **[2 pts]** Based on the plots in the part 10 of the programming question, what do you think is the best next step given the trends of different batch sizes? (Explain in 1-2 lines)

13. [5 pts] Now we want to give you a chance to experiment with the model by exploring the hyperparameters 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,
- describe the hyperparameter you varied
 - list the values you set the hyperparameter to
 - detail the setting of all the other hyperparameters, and
 - 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.

4 Collaboration Questions

1. (a) Did you receive any help whatsoever from anyone in solving this assignment?
(b) If you answered 'yes', give full details (e.g. "Jane Doe explained to me what is asked in Question 3.4")

2. (a) Did you give any help whatsoever to anyone in solving this assignment?
(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")

3. (a) Did you find or come across code that implements any part of this assignment?
(b) If you answered 'yes', give full details (book & page, URL & location within the page, etc.).