Homework 3

SVM and Neural Networks

CMU 10-701: Machine Learning (Fall 2020)

piazza.com/cmu/fall2020/10701/home
OUT: Wednesday, Oct 07, 2020

DUE: Wednesday, Oct 21, 2020, 11:59pm
TAs: Jie Jiao, Chandreyee Bhaumik

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 on the course site for more information: https://www.cs.cmu.edu/~epxing/Class/10701-20/about.html#academic-integrity-policies
- **Late Submission Policy:** See the late submission policy here: https://www.cs.cmu.edu/~epxing/Class/10701-20/about.html#late-homework-policy
- **Submitting your work:**
  - **Gradescope:** There will be two submission slots for this homework on Gradescope: Written and 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 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 it please add an additional page a 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 question of this homework, to the Programming submission slot. Your code may

be ran 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 LaTeXusers, use ■ and ●for shaded boxes and circles, and don't change anything else.

Problem 1: Soft Margin Hyperplanes [13 points]

The soft-margin primal SVM problem is:

$$\min \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{n}\xi_i$$

subject to feasibility constraints that for all $i = 1, ..., n$:

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

Suppose instead the optimization objective was changed to $\frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{n}\xi_i^p$ with $p > 1$ while the feasibility constraints are kept the same.

1. **[10 pts]**Give the dual formulation of the problem in this general case. Be sure to show all of you work.

2. [**3 pts**] How does this more general formulation ($p > 1$) compare to the standard setting ($p = 1$) discussed in lecture? Is the general formulation more or less complex in terms of the number of optimization variables and constraints? Justify your answer.

Problem 2: Prime-Dual SVM [9 pts]

1. [**3 pts**] Suppose you are given a set of **linearly separable** labeled vectors $\{\mathbf{x}_i, y_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \in \{-1, 1\}$, $\phi : \mathbb{R}^p \to \mathbb{R}^d$, $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. Let's define the function h as:

$$h(\mathbf{w}, b) = \max_{\boldsymbol{\alpha}, \alpha_i \geq 0 \forall i} \sum_{i=1}^{N} \alpha_i \left(1 - y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b)\right) \text{(where the max is taken over } \alpha_i \geq 0 \ \forall i)$$

Determine the domain and the range of the function h.
As an example, if we define $f : \mathbb{R} \to \mathbb{R}$ as:

$$f(x) = \max_{\gamma \geq 0} x^2 + \gamma(2 - x)$$

then we can define the domain and range as:

$$f(x) = \begin{cases} x^2 & \text{if } x \geq 2 \\ +\infty & \text{otherwise} \end{cases}$$

Hint: The possible range is $\mathbb{R} \cup \{-\infty, \infty\}$, so it can include $+\infty$ or $-\infty$.
Show all steps of your work.

2. [**6 pts**] Figure 1 shows SVM decision boundaries resulting from different combinations of kernels, slack penalties, and RBF bandwidths. Match each plot from Figure 1 with one of the SVM settings.
To obtain full credit, you do not need to provide any justification, but only provide the correct pairing.

RBF kernel: $K(\mathbf{x}, \mathbf{x}') = \exp(-\kappa \|\mathbf{x} - \mathbf{x}'\|_2^2)$

1. Linear kernel, C $= 0.1$ □
2. Linear kernel, C $= 100$ □
3. RBF kernel, C $= 1$, $\kappa = 10$ □
4. RBF kernel, C $= 1$, $\kappa = 50$ □
5. RBF kernel, C $= 0.1$, $\kappa = 0.1$ □
6. RBF kernel, C $= 20$, $\kappa = 0.1$ □

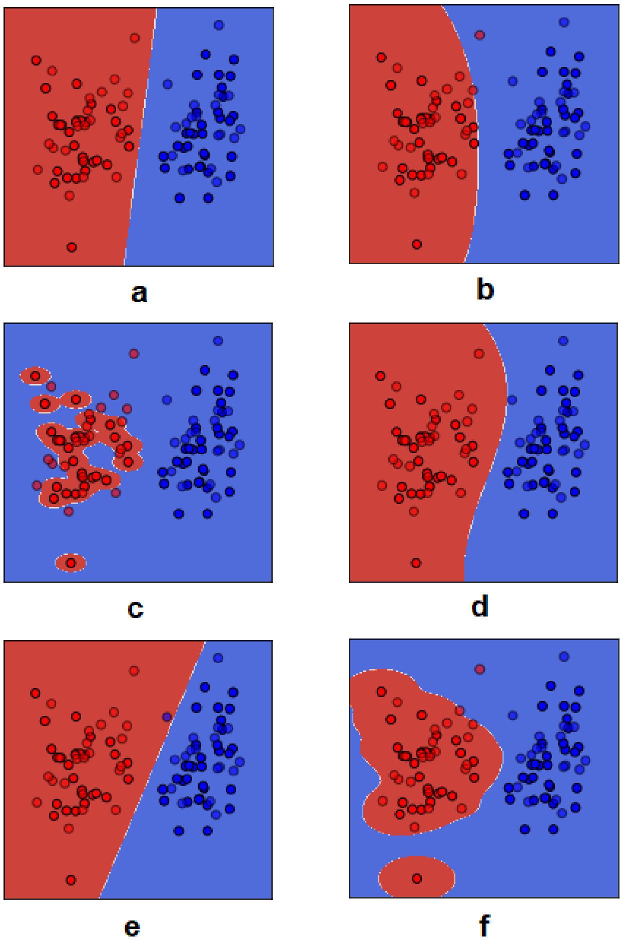

Figure 1: SVM decision boundaries learned with different parameter settings.

Problem 3: Neural Nets: Written Questions [43 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.

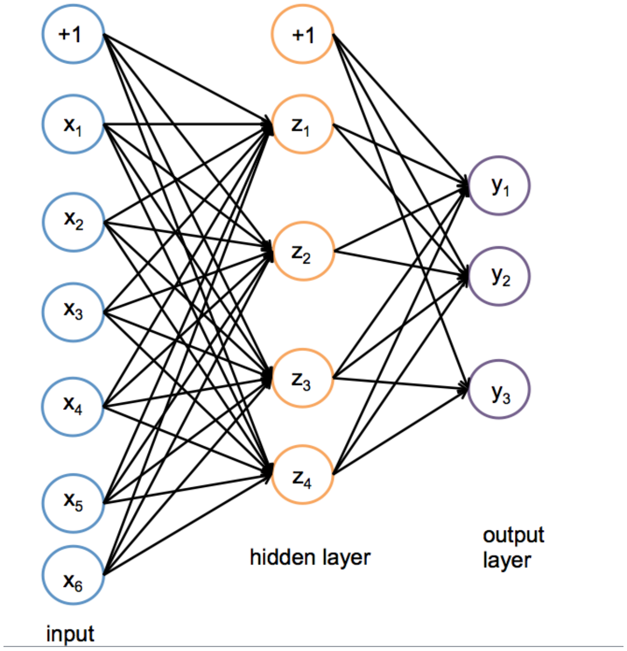

Figure 2: A One Hidden Layer Neural Network

Network Overview

Consider the neural network with one hidden layer shown in Figure 2. 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 a probability distribution $\hat{\mathbf{y}} = [y_1, y_2, y_3]^T$ 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 $\boldsymbol{\alpha}$ the matrix of weights from the inputs to the hidden layer.
2. Let $\boldsymbol{\beta}$ 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:

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

Activation at first (hidden) layer:

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

Linear combination at second (output) layer:

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

Activation at second (output) layer:

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

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:

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

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 a note 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) **[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 (3). Write your answer in terms of α and $x^{(1)}$.

(d) **[1 pt]** What is the vector $\mathbf{z}$ whose elements are made up of the entries z_j in equation (4)? 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 (5)?

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

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

The mathematics which you have to derive in this section jump significantly in difficultly, you should always be examining the shape of the matrices and vectors and making sure that you are comparing your matrix elements with calculations of individual derivatives to make sure they match (e.g. the element of the matrix $(\frac{d\ell}{d\alpha})_{2,1}$ should be equal to $\frac{d\ell}{d\alpha_{2,1}}$). Recall that ℓ is our loss function defined in equation (1.6)

(a) **[3 pts]** 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 $\hat{\mathbf{y}}, \mathbf{y}$.

(b) **[3 pts]** 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 $\frac{d\ell}{d\mathbf{b}}$ and $\mathbf{z}$.

(d) **[1 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 $\frac{d\ell}{d\alpha}$?

(e) **[1 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 $\boldsymbol{\beta}^*$

(f) **[3 pts]** What is the derivative $\frac{d\ell}{d\mathbf{a}}$ in terms of $\frac{d\ell}{d\mathbf{z}}$ 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 $y_1 = 0,\ \ y_2 = 1,\ \ y_3 = 0$.

1. [8 pts] We initialize the weights as:

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

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

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

You are given a training example $\mathbf{x}^{(1)} = [1, 1, 0, 0, 1, 1]^T$ 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?

2. [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 pts]** What is the updated value of $\beta_{2,1}$?

   (b) **[2 pts]** What is the updated weight of the hidden layer bias term applied to y_1 (i.e. $\beta_{1,0}$)?

   (c) **[2 pts]** What is the updated value of $\alpha_{3,4}$?

   (d) **[2 pts]** If we ran backpropagation on this example for a large number of iterations and then ran feed forward over the same example again, which class would we predict?

3. [8 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) **[2 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) **[4 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

Problem 4: Miscellaneous [11 pts]

1. [**2 pts**] **Select all that apply:** In a soft-margin support vector machine, decreasing the slack penalty term C causes.

   □ a smaller margin

   □ more overfitting

   □ less overfitting

   □ more robustness to outliers

2. [**2 pts**] **Select all that apply:** What is always true about the kernel matrix? (Mark all the correct choices)

   □ Symmetric

   □ All positive entries

   □ Invertibility

   □ Contains 0 as an eigenvalue

3. [**1 pt**] **Select one:** The shortest distance of point z to hyperplane $w^T x = 0$ is:

   ○ $w^T z$

   ○ $\frac{w^T z}{||z||}$

   ○ $\frac{w^T z}{||w||}$

   ○ $||w||$

4. [**1 pt**] **True or False:** For any neural network, the training error must always decrease monotonically when using SGD, provided the step size is sufficiently small (assume the step size is constant).

   ○ True

   ○ False

5. [**1 pt**] **True or False:** For any network, the validation set error must always decrease monotonically when using SGD and then later increase monotonically, provided the step size is sufficiently small.

   ○ True

   ○ False

6. **[1 pt] True or False:** Convolutional Neural Networks (CNNs) can only be used with two-dimensional inputs such as images.

   ○ True

   ○ False

7. **[1 pt]** Consider the image X and filter F given below. Let X be convolved with F using no padding and a stride of 1 to produce an output Y. What is value of j in the output Y?

$X =$

1	0	-2	3	4	1
2	9	5	6	0	-1
0	-3	1	3	4	4
6	5	2	0	6	8
-5	4	-3	1	3	-2
4	1	2	8	9	7

$F =$

-1	-1	-1
-1	8	-1
-1	-1	-1

$Y =$

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

8. **[1 pt]** Suppose you had an input feature map of size 6x4 and filter size 2x2, using no padding and a stride of 2, what would be the resulting output size?

9. **[1 pt] True or False:** A Neural Network with no activation function, with any number of layers, can be condensed to a network with just one layer.

   ○ True

   ○ False

Problem 5: Neural Net Programming [30 pts]

Figure 3: Random Images of Each of 10 digits in MNIST

Your goal in this assignment is to label images of handwritten digits (0 to 9) 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 are not allowed to use available libraries like PyTorch or TensorFLow which will make a neural network for you, you must create this model yourself.**

The MNIST dataset is comprised of 70,000 handwritten numerical digit images and their respective labels. There are 60,000 training images and 10,000 test images, all of which are 28 pixels by 28 pixels.

In this particular case, you will work with a smaller subset of MNIST that consist of 4000 images where the pixel values in each of them ranges from 0 to 1.

Dataset format For the dataset stored under `training_data_student`, you are provided with 2 files `train.csv`, `test.csv` that contain the digits and their corresponding labels (0 through 9). Each row contains 784 + 1 columns separated by commas. Columns 1 to 784 represent the pixel values and column 785 contains the corresponding label (0 to 9). Two files contain contains 3000 and 1000 instances respectively.

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 y_k of the output vector $\hat{\mathbf{y}}$ represents the probability of $\mathbf{x}$ belonging to the class k.

Following the notation from Section we have:

- For the output layer:

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

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

- For the hidden layer:

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

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

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,

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

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

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

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

Objective function

Since the output corresponds to a probabilistic distribution over the K classes, the objective function (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,

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

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).

Linear layer

1. Implement a forward function `linear_forward(x, weight, bias)`.
2. Implement a backward function `linear_backward()`. This function returns gradient of loss wrt input to the linear layer.

Sigmoid layer

1. Implement a forward function `sigmoid_forward(x)` that returns element-wise sigmoid of x. Output is of the same shape as x.
2. Implement a backward function `sigmoid_backward()`. This function returns gradient of loss wrt input to the sigmoid layer. The gradient of loss wrt output of sigmoid layer can be passed as an input to this function.

Softmax cross entropy loss layer

1. Implement a forward function `softmax_xeloss_forward(x, labels)`. The fusion of softmax and average cross entropy computation in one layer is common in popular deep learning frameworks.

2. Implement a backward function `softmax_xeloss_backward()`.

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.

Programming Submission

NOTE: Initial weights and biases are provided in the folder `params`. Do not shuffle the training data, pass through it in the order it is given. Use the following hyper-parameters:
Learning rate = 0.01
Number of epochs = 15
Width of hidden layer = 256 *(excluding bias term)*
We have provided the outputs of first five nodes of each layers (`a.txt, z.txt, b.txt`) and the updated weights for the first data point in the first epoch (`updated_alpha1.txt, updated_alpha2.txt, updated_beta1.txt, updated_beta2.txt`) 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. [**3 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. [**5 pts**] What are the values of $\beta_{k,0}$ for $k \in \{1, \ldots, K\}$ (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. [**5 pts**] List the total test loss at the end of each epoch. Report numbers till 4 places of decimal.

6. [**5 pts**] List the test accuracy at the end of each epoch. Report numbers till 4 places of decimal.

7. [**5 pts**] Run your model for 100 epochs. Plot a graph of epoch vs loss, which has the epoch number on the x-axis and the loss value on the y-axis. To this plot add two lines one representing the average training loss per epoch and one representing the average test loss per epoch. Please make sure your graph is properly labelled.

8. [**4 pts**] Run the model for 100 epochs, report the average final training loss and the test accuracy. Report numbers till 4 places of decimal.

   Training Loss:

   Test Accuracy:

9. [**1 pt**] Now we want to give you a chance to experiment with the model by exploring the hyper-parameters of a neural network. Produce three plots similar to that of the one requested in question 7, however each of which you should have changed hyper-parameters to a different values. Tell us the details of the hyper-parameter that you changed and explain how this compares to your findings in question 7. For your reference here are some hyperparameters which you can change:

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

   Some examples may be:

- Running your model with learning rate = 0.3, with 100 hidden nodes and weights initialized all to zeros.
- Running your model with everything the same except comparing three different numbers of hidden nodes.

This question is mostly for fun so you are welcome to do whatever you want provided you label your graphs and give some sort of explanation.

Experiment 1

Experiment 2

Experiment 3

1 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.).