Programming Assignment 1

CSE 151B: Deep Learning

Fall 2020

Instructions

Due on October 16th, 2020

1. Please submit your assignment on Gradescope. The instructions for this will be coming soon. There are two components to this assignment: mathematical solutions/proofs with English explanations (Part I). For the programming assignment portion of the homework (Part II), you will be writing a report in a conference paper format for this assignment, reporting your findings. All parts of the assignments must be typeset, including figures. You must use NeurIPS format for your report (link below). We strongly recommend that you use some dialect of TEXor LATEX. You may also use Word if you so choose. The link below has both LATEXand Word NeurIPS formats. Figures may be generated with Excel or Python, so long as they are computer generated. **We will not be accepting any handwritten work - this includes the "written part."** NeurIPS templates in LATEXand Word are available from the 2015 NeurIPS format site. The page limits mentioned there don't apply.

2. For the group report, include an informative title, author list, and an abstract. The abstract should summarize briefly what you did, and the best percent correct you got on each problem. The report should be well-organized with an introduction, background (if you review previous work), methods, results, and discussion for each programming part. Figures should be near where they are referenced, there should be informative captions on figures, clearly specified axes and figure keys, etc. A details of what to include in the the report along with rubric can be found at the end of the document.

3. You are expected to use Python (usually with NumPy). You also need to submit all of the source code files and a *readme.txt* file that includes detailed instructions on how to run your code.

   You should write clean code with consistent format, as well as explanatory comments, as this code may be reused in the future.

4. Using any off-the-shelf code is strictly prohibited.

5. ***If you end up dropping the class and your teammate does not, you are expected to help your teammate anyway!*** Please don't leave your teammate(s) without your assistance. Being on a team means just that: teamwork! When you join a team, you have made a commitment. Please honor it.

6. Any form of copying, plagiarizing, grabbing code from the web, having someone else write your code for you, etc., is cheating. We expect you all to do your own work, and when you are on a team, to pull your weight. Team members who do not contribute will not receive the same scores as those who do. Discussions of course materials and homework solutions are encouraged, but you should write the final solutions alone. Books, notes, and Internet resources can be consulted, but not copied from. Working together on homework must follow the spirit of the **Gilligan's Island Rule** (Dymond, 1986): No notes can be made (or recording of any kind) during a discussion, and you must watch one hour of Gilligan's Island or something equally insipid before writing anything down. Suspected cheating has been and will be reported to the UCSD Academic Integrity office.

Part I

Problems to be solved and turned in individually

For this part we will *not* be accepting handwritten reports. Please use latex or word for your report. MathType is a handy tool for equations in Word. The free version (MathType Lite) has everything you need.

1. **Perceptrons (12 points)**

   Recall the perceptron activation rule:

$$\hat{y} = \begin{cases} 1 & \text{if } \sum_{i=0}^{d} w_i x_i \geqslant 0 \\ 0 & \text{else} \end{cases}$$

   Here, we have written it so that w_0 is the bias (the opposite of the threshold), and so $x_0 = 1$.

   (a) (2 pts) Assuming $d = 2$, derive the equation for the line that is the decision boundary (the solid black line in Figure 1). Write it in slope-intercept form, i.e., $y(x) = mx + b$, where m and b are written in terms of the weights.

   (b) (3 pts) Prove that the distance from the decision boundary to the origin (as shown in Figure 1) is given by:

$$l = \frac{-w_0}{\|w\|}$$

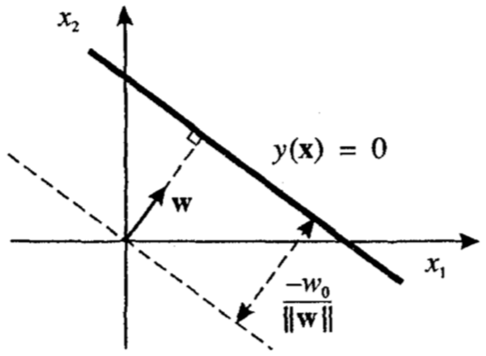

Figure 1: In part (b), you need to prove the length of the dashed line is what is shown here. *You have to assume that the weight vector is in the first quadrant, as shown.*

   In class, we showed how to learn the "OR" function using the perceptron learning rule. Now we want to learn the "NAND" function using four patterns, as shown in Table 1.

Input	Output
0 0	1
0 1	1
1 0	1
1 1	0

Table 1: The "NAND" function

(c) (1 pt) Write down the perceptron learning rule as an update equation.

(d) (4pts) Draw the rest of this table, as we did in class, as the network learns NAND. Initialize w_0, w_1, and w_2 to be 0 and fix the learning rate to 1. Add one row for each randomly selected pattern (training example) for the perceptron to learn. Stop when the learning converges. Make sure you show the final learned weights and bias. You may pick a "random" order to make the learning converge faster, if you can (you may not need all of these rows).

x_1	x_2	Output	Teacher	w_0	w_1	w_2

(e) (2 pts) Is the solution unique? Why or why not? Justify your answer.

2. **Logistic Regression (5 points)**

Logistic regression is a binary classification method. Intuitively, logistic regression can be conceptualized as a single neuron reading in a d-dimensional input vector $x \in \mathbb{R}^d$ and producing an output $\hat{y}$ between 0 and 1 that is the system's estimate of the conditional probability that the input is in the target category, given the input. The "neuron" is parameterized by a weight vector $w \in \mathbb{R}^{d+1}$, where w_0 represents the bias term (a weight from a unit that has a constant value of 1).

Consider the following model parametrized by the vector w:

$$\hat{y} = P(\mathcal{C}_1|x) = \frac{1}{1+\exp(-w^\top x)} = g(w^\top x) \tag{1}$$

$$P(\mathcal{C}_0|x) = 1 - P(\mathcal{C}_1|x) = 1 - \hat{y}, \tag{2}$$

where we assume that x has been augmented by a leading 1 to represent the bias input. With the model so defined, we now define the Cross-Entropy cost function, equation 3, the quantity we want to minimize over our training examples:

$$E(w) = -\sum_{n=1}^{N} \{y^n \ln(\hat{y}^n) + (1-y^n)\ln(1-\hat{y}^n)\}. \tag{3}$$

Here, $y^n \in \{0,1\}$ is the label or teaching signal for example n ($y^n = 1$ represents $x^n \in \mathcal{C}_1$). We minimize this cost function via gradient descent.

To do so, we need to derive the gradient of the cost function with respect to the parameters w_j. Assuming we use the logistic activation function g as in equation 1, prove that this gradient is:

$$-\frac{\partial E(w)}{\partial w_j} = \sum_{n=1}^{N} (y^n - \hat{y}^n) x_j^n \tag{4}$$

Part II
Programming Assignment

Requirements: Write your own Logistic Regression and Softmax Regression classifiers using Python, following the instructions below.
You **are allowed** to use the following Python libraries and packages: NumPy, PIL, matplotlib, os, and random.
You **are not allowed** to use any SciPy implementations or logistic or softmax regression or any other high-level machine learning packages (including, but not limited to TensorFlow, PyTorch, Keras, **scikit-learn**, etc.).

For this assignment, a "team" is defined as two or three people.

1 Dataset

For the first assignment, we will use logistic regression/softmax classifier to detect different classes of fashion objects from Fashion-MNIST dataset. The detailed description of the dataset can be found here:
https://github.com/zalandoresearch/fashion-mnist.
To load training and testing data, first clone the github repository:
`git clone https://github.com/zalandoresearch/fashion-mnist.git fashion_mnist_dataset`
Then run:

```
from fashion_mnist_dataset.utils import mnist_reader
X_train, y_train = mnist_reader.load_mnist('fashion_mnist_dataset/data/fashion', kind='train')
X_test, y_test = mnist_reader.load_mnist('fashion_mnist_dataset/data/fashion', kind='t10k')
```

We will use X_train and y_train for training and validation, and X_test and y_test for testing. As a first step, we will have to min-max normalize the dataset (both X_train and X_test), and then one hot encode the labels (both y_train and y_test). It is also a good idea to shuffle the dataset before using.

2 Cross Validation Procedure

We will need a validation dataset to estimate a model's performance on unseen data (without using test data), to tune its hyperparameters (e.g. learning rate, batch size, etc.) and to prevent overfitting. We will use the method of *k-fold cross-validation* to separate a portion from training data as validation.

For each problem below, you should divide the train data (X_train) into k mutually exclusive sets. Each set should contain a representative sample with respect to the problem. For example, we will have you recognize T-shirt and Ankle boot. In this case, it would not be good to have one of the sets contain all T-shirts or all Ankle boots. Basically, you want to have roughly equal amounts of each category in each of the k sets (a trivial way to do is to shuffle dataset once before splitting). Don't worry if they don't divide up *perfectly* equally; but make sure they are mutually exclusive! For this assignment, we will fix k to 10.

Repeat the following k times:

- Choose a set to be holdout (or validation). Then, train your model on the remaining $k-1$ of the sets. Psuedocode for this is in the Training algorithm below.
- The loss (Equation 3 for logistic and 7 for softmax) on the holdout set should go down as you train the model, even though the examples in the holdout set are not being used to change the weights. However, at some point, the holdout loss should start to rise. This means the model is overfitting and you should stop training. This is called *early stopping*.
- It is also possible that the holdout set loss never goes up over the M epochs (one pass through all of the training data is called an *epoch*). So you should also have some limit on the number of training steps. In our case, we will set M to 100.
- After early stopping or M epochs, record the model's performance (accuracy) on this holdout set.

After repeating step 1 to 4 for all remaining holdouts, calculate their average accuracy. We will use this average accuracy to determine a best set of hyperparameters. For example, say you get an average accuracy of x% in the

```
procedure TRAINING PROCEDURE
    Perform PCA on the whole **train**;
    folds = $k$ mutex split of training data;
    for fold = 1 to $k$ do
        **val_set** ← folds[fold];
        **train_set** ← remaining folds;
        Project **train_set** and **val_set** onto top $p$ **train** PC's
        for epoch = 1 to $M$ do
            train the model with **train_set**, and test the performance on **val_set** every epoch
            record **train_set, val_set** loss for plotting and accuracy on **val_set** for hyperparameter tuning
            save the best model based on **val_set** performance
    plot the training and validation loss curves
    Project **test** onto top $p$ **train** PC's and use the best model to report accuracy on **test**
```

10 validation sets for learning rate of 0.01 and batch size of 512, using 10-fold cross validaiton. Now, change the hyperparameters (e.g. 0.01 to 0.001 and 512 to 128) and observe if accuracy dropped or increased. If the accuracy increases, store the hyperparameter. In this fashion, try a different combination of hyperparameters and record the one which yields the best average validation accuracy. This process is known as 'hyperparameter tuning.'

After getting the best model using hyperparameter tuning, we will run the trained model on unseen test set and record the performance. Notice that we have used test set only once.

3 Principal Components Analysis

As the dimensions of our images is large (784), we ask you to perform dimensionality reduction on the images first, by using a linear dimensionality reduction technique called Principal Components Analysis, or PCA. Since there is no concept of overfitting for PCA, we can perform it on the whole training data before k-fold validation. But of course, the number of principal components (p) to use is a hyperparameter which we will determine after observing the performance on validation sets. Note that you will have to use principal components that was learned on training data to project the testing data to top p components (see Training algrorithm).

There are several ways to perform PCA. For example, if a data is mean centered, the principal components are the eigenvectors of its covariance matrix (extracted through Eigen Value Decomposition) or the transpose of its right Singular vectors (extracted through Singular Value Decomposition). You can choose any method you like but the implementation must use routines available in Numpy only.

For report, you will have to show two different types of figure. For the first type of figure, we will show the reconstructed images. For this, pick any image randomly (784 dimensions), reduce its dimension using top p principal components (by multiplying with $784 \times p$ matrix) and then project them back (by multiplying with the transpose of that matrix). Show a collage of reconstructed images for $p = 2, 10, 50, 100, 200, 784$. In this collage also include the original image. This should be a single figure in your report. Do you see any loss of information? If so, for which dimensions and why?

For the second type of figure, we will show the principal components themselves. Note that each PC has 784 dimensions and so we can easily visualize it by reshaping to 28 by 28 without any modification. Show the visualization of top 10 PCs after performing PCA on the three different sets of images (i.e. T-shirt and Ankle boot, Pullover and Coat, All 10 classes). Do you see any patterns or shapes? If so, provide an explanation for why you might see this. For your convenience, we have provided a figure (see below) which shows how these images might look like when PCA is performed on T-shirt and Ankle boot dataset.

4 Logistic Regression

Here, we build and evaluate classifiers on the data. We will experiment with discerning between two classes of data, and with multi-class classification. Now, without using any high-level machine learning libraries, implement logistic regression. Here, you'll be using *stochastic gradient descent*, and will only need one logistic output unit. (Think about why we only need one if we're classifying two classes?)

Here we ask you to perform logistic regression on two different sets of classes. The first set contains the images of T-shirt vs Ankle Boot, and the second set will contain Pullovers and Coats. Train a logistic regression model

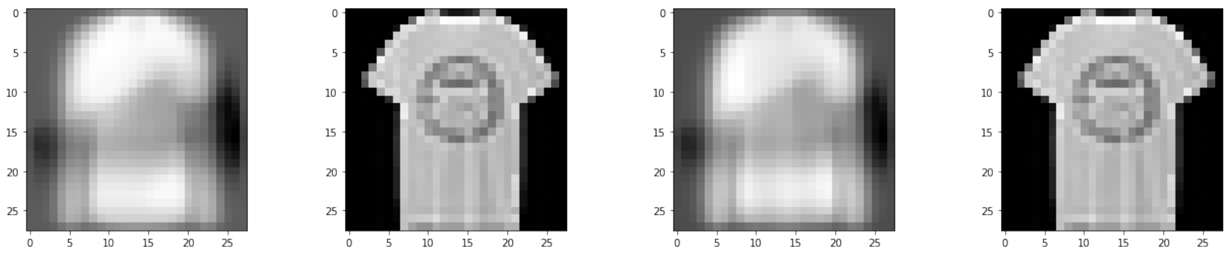

Figure 2: (a) Visualization of the first principal component, (b) A randomly sampled image, (c) Reconstructed image using first two principal components ($p = 2$), (d) Reconstructed image using all principal components.

Algorithm 1 Stochastic Gradient Descent

```
1: procedure STOCHASTIC GRADIENT DESCENT
2:     $w \leftarrow 0$
3:     for t = 1 to $M$ do                                    ▷ Here, t is one epoch.
4:         randomize the order of the indices into the training set
5:         for j = 1 to $N$, in steps of B do     ▷ Here, N is number of examples and B is the batch size
6:             start = j
7:             end = j+B
8:             $w_{t+1} = w_t - learning\_rate * \sum_{n=start}^{end} \nabla E^n(w)$
9:     return $w$
```

on these two datasets. following the training procedure mentioned above. Here your hyperparameters are learning rate, batch size and number of principal components. After getting final model, you should have these: losses for all training folds, losses for all validation folds, and test accuracy. The expected accuracy for the first set is over 90% and on the second dataset is over 70%. Draw a plot of the average loss (divide the loss by number of examples) for the 100 training epochs for the training set and the holdout set. These training and holdout curves should be easily distinguishable in your plots by using different colors or solid and dashed lines. **Make sure that your graph is well-labeled** (i.e., x and y axes labeled with what they are) with a key, and with a figure caption that clearly states what is being shown in the graph. This should be the case for any graph you plot.

Report all the different combinations of hyperparatemers considered, the one that worked best, test accuracy, and provide a discussion for the results. Also discuss if you see any performance gap between the two different sets of images. If you see any gap, provide an explanation. In particular, why is the second set harder than the first?

5 Softmax Regression

Softmax regression is the generalization of logistic regression for multiple (c) classes. Now given an input x^n, softmax regression will output a vector $\hat{y}^n$, where each element, $\hat{y}^n_k$ represents the probability that x^n is in class k.

$$\hat{y}^n_k = \frac{exp(a^n_k)}{\sum_{k'} exp(a^n_{k'})} \tag{5}$$

$$a^n_k = w_k^T x^n \tag{6}$$

Here, a^n_k is called the *net input* to output unit $\hat{y}_k$. Equation 5 is called the *softmax activation function*, and it is a generalization of the logistic activation function. For softmax regression, we use a *one hot encoding* of the targets. That is, the targets are a c-dimensional vector, where the k^{th} element for example n (written y^n_k) is 1 if the input is from category k, and 0 otherwise. Note each output has its own weight vector w_k. With our model defined, we now define the *cross-entropy* cost function for multiple categories in Equation 7:

$$E = -\sum_n \sum_{k=1}^{c} y^n_k \ln \hat{y}^n_k \tag{7}$$

Again, taking the average of this over the number of training examples normalizes this loss over different training set sizes. Also averaging over the number of categories c makes it independent of the number of categories. Please take the average over both when reporting results. Surprisingly, it turns out that the learning rule for softmax regression is basically the same as the one for logistic regression! The gradient is:

$$-\frac{\partial E^n(w)}{\partial w_{jk}} = (y_k^n - \hat{y}_k^n)x_j^n \tag{8}$$

where w_{jk} is the weight from the j^{th} input to the k^{th} output.

Now, we'll modify our network to classify all the classes in the Fashion-MNIST dataset. To achieve multi-class classification, we'll need more output units, and use the gradient derived for Softmax Regression. As a sanity check, make sure your outputs are all positive and sum to 1. You will be using one-hot encoding of the targets here. When choosing the category for evaluating percent correct, you just choose the maximum output.

Similar to what you did in logistic regression, here, we ask you to perform the same experiment for softmax regression. For report, you have to include the similar items. The expected accuracy for this experiment should be over 80%. In addition, you will need to visualize the weights of the model. Notice that we have reduced the dimensions of the data before training, and so the weights will have $p \times 10$ dimension. To visualize, we will need to project them back to original dimensions using the transpose of the principal components. This will give you a 784×10 matrix and now, you can reshape each of the 10 columns to 28×28 for visualization. Discuss anything interesting you find. Do you see any pattern in the weights?

6 Project Report Outline with Rubric (25 Points)

1. Title (1pt): The title *has to be informative* and not generic like '151B PA1 Report'. Also you have to include author's list.

2. Abstract (1pt): A short description of what you did. Please mention any key findings, interesting insights, and final results (i.e., percent correct, *not* loss numbers)

3. Introduction (1pt): Similar to abstract, but longer with more details.

4. Related Work (1pt): Cite any paper/slides for the methods you have used for this project. For example, as we will use PCA, Logistic and Softmax regression, you can cite any papers or slides from lecture that introduced/used them.

5. Dataset (1+1+1 pts): This should have these parts: a brief description of data (in one or two lines), a figure which shows one randomly sampled example from each class, any pre-processing on the dataset (e.g. normalizing, converting class labels to one hot vectors, etc.), statistics for the different splits of the dataset (for example, you can include a table which shows the size of train and test data for the two classes - T-shirt and Ankle boot).

6. PCA (1+1+1+1 pts): A figure which shows reconstructed images for different PCs and discussion for why you see this, three figures showing top 10 PCs for the three different sets of classes (i.e., (1) T-shirt and Ankle boot, (2) Pullover and Coat, and finally, (3) all 10 classes). Discuss the differences, and explain why the differences are what they are.

7. Logistic Regression (1+3+3 pts): It will contain a short description of logistic regression model, and the results. In the results, repeat these for the two different sets of classes considered (i.e. T-shirt and Ankle boot, Pullover and Coat): 1. A plot which shows the curves for training and validation loss, 2. Test performance, 3. Discussion of the results. All plots and performance mentioned should be on the set of hyperparameters that gave the best results on validation set. Clearly label the plots. For discussion, mention the all different combination of hyperparameters you considered for tuning, what worked and what did not, and any interesting findings/insights.

8. Softmax Regression (1+3+2+1 pts): Same as logistic regression. Additionally, visualize the weights and provide a discussion of any findings from the visualization.

9. Team contributions: A short paragraph from *each* team member with what they contributed to the project - team members won't necessarily get the same grade if someone slacked off!

10. For the source code - the most points will be given for clean, well-documented code.