| **CIS519: Applied Machine Learning** | **Fall 2019** |
|---|---|
| Homework 3 | |
| *Handed Out: November 4, 2019* | *Due: November 18, 2019 at 11:59pm* |

Version 1

- Feel free to talk to other members of the class in doing the homework. I am more concerned that you learn how to solve the problem than that you demonstrate that you solved it entirely on your own. You should, however, **write down your solution yourself**. Please include at the top of your document the list of people you consulted with in the course of working on the homework.
- While we encourage discussion within and outside the class, cheating and copying code is strictly not allowed. Copied code will result in the entire assignment being discarded at the very least.
- Please use Piazza if you have questions about the homework. Also, please come to the TAs recitations and to the office hours.
- Handwritten solutions are not allowed. All solutions must be typeset in Latex. Consult the class' website if you need guidance on using Latex. If you don't have a lot of experience with Latex (or even if you do), we recommend using Overleaf (`https://www.overleaf.com`) to write your solutions. You will submit your solutions as a single pdf file (in addition to the package with your code; see instructions in the body of the assignment).
- The homework is due at 11:59 PM on the due date. We will be using Gradescope for collecting the homework assignments. You should have been automatically added to Gradescope. If not, please ask a TA for assistance. Please do **not** hand in a hard copy of your write-up. Post on Piazza and contact the TAs if you are having technical difficulties in submitting the assignment.
- Here are some resources you will need for this assignment
  - The Jupyter Notebook template on Colab: `https://drive.google.com/open?id=1mQXv_Gc3P16iBEObJFyzRA4fByy3BDGQ`
  - Latex materials: `https://www.seas.upenn.edu/~cis519/fall2019/assets/HW/HW3/hw3-material.zip`

# 1 Neural Networks [40 points]

In this problem, you will implement two different neural network architectures to do image classification with the CIFAR-10 image dataset.[1] The goal of this problem is to better understand how to build different neural network architectures for image data and implement a multi-class classifier.

## 1.1 Dataset

The CIFAR-10 dataset contains tens of thousands of images that have been classified in ten different categories, such as "airplane," "cat", and "horse." Each image is $32 \times 32$ pixels large and is represented uses the RGB format. Therefore, each image is represented as a $(3 \times 32 \times 32)$ matrix where each value is a number between 0 and 255 (inclusive). Each of the three channels corresponds to the red, green, or blue channels of the image.

The dataset has been prepared for you to download. The Python template automatically downloads the data and loads it into memory for you, so you only need to execute the

[1] `https://www.cs.toronto.edu/~kriz/cifar.html`

respective cells. The images are stored as a $(N \times 3 \times 32 \times 32)$ matrix, where $N$ is the number of images. The respective labels are stored as a vector of size $N$ with an integer in the range 0 to 9 where each integer represents a class label. You will use the `DataLoader` PyTorch library to break the data into batches of 64 images. This has already been implemented for you in the template code.

## 1.2 Experiments [20 points results, 20 points implementation]

You will implement two different neural network architectures and train each of them with stochastic gradient descent with several different learning rates to find the best one. You should use the `torch.nn.CrossEntropyLoss` loss function and train for 200 epochs. For each network, you should compare each of the learning rates by plotting the following data:

1. The average loss per training instance on each epoch[2]
2. The average loss per validation instance on each epoch
3. The accuracy on the validation dataset on each epoch

Then for each network, pick the learning rate which had the highest final validation accuracy and compute the accuracy on the test set for that model. Report this accuracy.

**FeedForward [5 points]** The first network you will implement is a two-layer feed-forward neural network with a hidden layer of size 1000. Each layer should be implemented using the `torch.nn.Linear` module. The input matrix will be sized $(B \times 3 \times 32 \times 32)$ where $B$ is the batch size (64 in our case). To pass the data through the `Linear` module, you need to reshape the matrix to be size $(B \times 3072)$ with the `reshape` method. Then the data can be passed through the first layer which will output a $(B \times 1000)$ matrix. You should apply a ReLU activation on this matrix then pass it through the second linear module to get a $(B \times 10)$ matrix which corresponds the model's score for each of the 10 classes.

In summary, this is the order that the data should be processed:

| Layer/Function | Hyperparameters |
|---|---|
| `reshape` | n/a |
| `Linear` | Input size = 3072, Output size = 1000 |
| `torch.relu` | n/a |
| `Linear` | Input size = 1000, Output size = 10 |

You should try learning rates $\{0.0001, 0.00005, 0.00001\}$. Your implementation of this network will be unit tested.

[2]There are two ways to compute this. The first is to train for 1 full epoch, then recompute the losses for every training batch *without* updating the parameters. The second is to estimate this value by computing it while you are training. For example, compute the loss for 1 batch, add this loss value to a variable which will compute the total training loss, backpropagate the loss from the batch, then move on to the next batch. While the latter method does compute the true training loss exactly, you can choose to implement either method.

**Convolutional [15 points]** The second network is based on a series of convolution operations followed by several feed-forward layers. The architecture should be the following:

| Layer/Function | Hyperparameters |
|---|---|
| `Conv2d` | in channels = 3, out channels 7, kernel size 3, stride = 1, padding = 0 |
| `MaxPool2d` | kernel size = 2, stride = 2 |
| `Conv2d` | in channels = 7, out channels = 16, kernel size = 3, stride = 1, padding = 0 |
| `torch.relu` | n/a |
| `reshape` | n/a |
| `Linear` | input size = 2704, output size = 130 |
| `torch.relu` | n/a |
| `Linear` | input size = 130, output size = 72 |
| `torch.relu` | n/a |
| `Linear` | input size = 72, output size = 10 |
| `torch.sigmoid` | n/a |

You should try learning rates $\{0.01, 0.001, 0.0001\}$. Your implementation of this network will be unit tested.

## 1.3 Extra Credit: Image Normalization [10 points]

Pixel values for images are in the range from 0 to 255. Large input values can sometimes make neural network training unstable, so pixel values are often normalized before training.

One way to normalize images is as follows. Assume you have your training, validation, and test images in tensors $\mathbf{X}_{\text{train}}$, $\mathbf{X}_{\text{valid}}$, $\mathbf{X}_{\text{test}}$, which are of size $(N_{\text{train}} \times 3 \times 32 \times 32)$, $(N_{\text{valid}} \times 3 \times 32 \times 32)$, and $(N_{\text{test}} \times 3 \times 32 \times 32)$. Compute the mean and the standard deviations of the 3 channels on the training data. Then normalize the training, validation, and testing data based on those values. For example, if `mu_0` and `std_0` are the means and standard deviations of the first training channel, you can normalize the first channels of the train, validation, and test as follows:

```
X_train[:, 0] = (X_train[:, 0] - mu_0) / std_0
X_valid[:, 0] = (X_valid[:, 0] - mu_0) / std_0
X_test[:, 0] = (X_test[:, 0] - mu_0) / std_0
```

For extra credit, repeat the above experiments with normalized images. For both the FeedForward and the Convolutional networks, you may have to try different learning rates than you used for the non-normalized images. Report what learning rates you used, show the same plots as in the original experiment, and discuss whether or not you were able to get improved performance with either the FeedForward or Convolutional models by normalizing the images.

## 1.4 GPU Usage

Neural networks take a notoriously long time to train (even longer than the Perceptron-based models in Homework 2). The training time can be significantly reduced by using

computation on the GPU rather than the CPU. Luckily, Google Colab allows for limited free access to a GPU. We strongly suggest you use Google Colab for this assignment to get access to the GPU.[3]

To setup the Jupyter Notebook with the GPU access, follow these steps:

1. Create a new notebook at `https://colab.research.google.com`. This will create a Jupyter Notebook that is saved on your Google Drive. When you execute a cell on Colab, the code will be running on a cloud server instead of your local laptop/computer.
2. In the top menu bar, select "Runtime" then "Change runtime type."
3. Select "Python 3" and then "GPU" from the dropdown menus

Now the notebook is configured to use the GPU, however you will need to ensure that the PyTorch code also uses the GPU. When you create a tensor, you must move it from the CPU to the GPU (this has already been done for you in the template code):

```
X = torch.LongTensor([[1, 2, 3], [4, 5, 6]])  # CPU
X = X.cuda()  # Move to GPU
```

Then you also need to make sure that your model's parameters are on the GPU, which can be done like this:

```
network = FeedForward()  # Initialize a model
network.cuda()  # Moves the parameters to the GPU
```

The template code also does this, but you need to make sure you also do this for the code which you will write.

## 1.5 What to Report

You should include the following information in your report for each of the two network architectures

1. The average training loss plot
2. The average validation loss plot
3. The validation accuracy plot
4. The final validation accuracy for each learning rate
5. The test accuracy for the best learning rate

Additionally, you should upload your implementations of `FeedForward` and `Convolutional` to the Gradescope assignment "Homework 3 - Code" in a file named `hw3.py` for unit testing. (Make sure to include `import torch` in your file.) The unit tests will check to make sure each network has the correct data members, that each of the data members is the correct size, and that your `forward()` implementation matches ours.

[3]For comparison, the `FeedForward` and `Convolutional` models take 1 and 2 minutes to train with the GPU and 5 and 6 minutes on the CPU.

| $i$ | Label | Hypothesis 1 | | | | Hypothesis 2 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $D_0$ | $f_1 \equiv [x >]$ $\epsilon =$ | $f_2 \equiv [y >]$ $\epsilon =$ | $h_1 \equiv [\,]$ | $D_1$ | $f_1 \equiv [x >]$ $\epsilon =$ | $f_2 \equiv [y >]$ $\epsilon =$ | $h_2 \equiv [\,]$ |
| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) | (10) |
| *1* | + | | | | | | | | |
| *2* | + | | | | | | | | |
| *3* | + | | | | | | | | |
| *4* | + | | | | | | | | |
| *5* | + | | | | | | | | |
| *6* | − | | | | | | | | |
| *7* | − | | | | | | | | |
| *8* | − | | | | | | | | |
| *9* | − | | | | | | | | |
| *10* | − | | | | | | | | |

Table 1: Table for Boosting results

# 2 Boosting [30 points]

In this problem, you will manually run the AdaBoost algorithm for two iterations on the following dataset:

| $i$ | $x$ | $y$ | Label |
|---|---|---|---|
| *1* | 2 | 0 | + |
| *2* | 8 | 7 | + |
| *3* | 4 | 9 | + |
| *4* | 6 | 5 | + |
| *5* | 7 | 6 | + |
| *6* | 9 | 3 | − |
| *7* | 1 | 1 | − |
| *8* | 3 | 8 | − |
| *9* | 5 | 4 | − |
| *10* | 0 | 2 | − |

Each example $(x, y) \in \mathcal{R}^2$ has a positive or negative label. The instances are numbered under column $i$.

You will use two rounds of AdaBoost to learn a hypothesis for this data set. In each round, AdaBoost chooses a weak learner that minimizes the error $\epsilon$. As weak learners, use hypotheses of the form (a) $f_1 \equiv [x > \theta_x]$ or (b) $f_2 \equiv [y > \theta_y]$, for some integers $\theta_x, \theta_y$ (either one of the two forms, not a disjunction of the two). Each $\theta$ can be an integer between 0 and 9 inclusive (for this dataset, lower than 0 and higher than 9 will not change the hypothesis since all of the features values are between 0 and 9). To make the computation easier, use log base 2 and wherever Adaboost has $e^x$, use $2^x$ instead.

Fill in Table 1 by following these instructions:

1. [**1 point**] Start the first round with a uniform distribution $D_0$. Place the value for $D_0$ for each example in the third column of Table 1.
2. [**10 points**] Find the values for $\theta_x$ and $\theta_y$ which have the lowest errors. Put the values you find in the top boxes in columns 4 and 5 (e.g. write $x > 9$ if $\theta_x = 9$ had the lowest error for any $x$ value) and their corresponding errors $\epsilon$. If there are ties, you can pick a value arbitrarily. Then, fill in the rest of columns 4 and 5 with $+/-$ based on how $f_1$ and $f_2$ would label the instances.
3. [**1 point**] In column 6, pick the function which had the lowest error (e.g. write $h_1 \equiv f_1$ if $f_1$ has the lowest error), then copy that hypothesis' predictions in column 6.
4. [**6 points**] Now compute $D_1$ for each example and write the new values in column 7.
5. [**10 points**] Repeat the above process for hypothesis 2 using the new $D_1$ values.
6. [**2 points**] Write down the final hypothesis produced by AdaBoost. You can write the answer in terms of $h_1$ and $h_2$, but include the actual values for $\alpha_1$ and $\alpha_2$.

**What to submit:** Fill out Table 1 as explained and give the final hypothesis, $H_{final}$.

# 3 SVMs [30 points]

In this problem you will manually find the margin of a hard SVM and answer questions about the solution you found.

You have been provided with a set of 6 labeled examples $D$ in two-dimensional space in Figure 1, $D = \{(\mathbf{x}^{(1)}, y^{(1)}), ..., (\mathbf{x}^{(6)}, y^{(6)})\}$, $\mathbf{x}^{(i)} \in \mathbb{R}^2$ and $y^{(i)} \in \{1, -1\}, i = 1, 2, ..., 6$.

(a) [5 points] We want to find a linear classifier where examples $\mathbf{x}$ are positive if and only if $\mathbf{w} \cdot \mathbf{x} + \theta \geq 0$.

1. [1 point] Find a simple solution $(\mathbf{w}, \theta)$ that can separate the positive and negative examples given. (We are looking for an answer where the values for $\mathbf{w}$ and $\theta$ are integers.) Remember that the vector which defines a line is perpendicular to that line.

   Define $\mathbf{w} =$

   Define $\theta =$

2. [4 points] Recall the Hard SVM formulation:

$$\mathbf{min}_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 \tag{1}$$

$$\text{s.t } y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + \theta) \geq 1, \forall (\mathbf{x}^{(i)}, y^{(i)}) \in D \tag{2}$$

   What would the solution be if you solve this optimization problem? (Note: you don't actually need to solve the optimization problem; we expect you to use a simple geometric argument to derive the same solution SVM optimization would result in). Explain how you came to this solution.

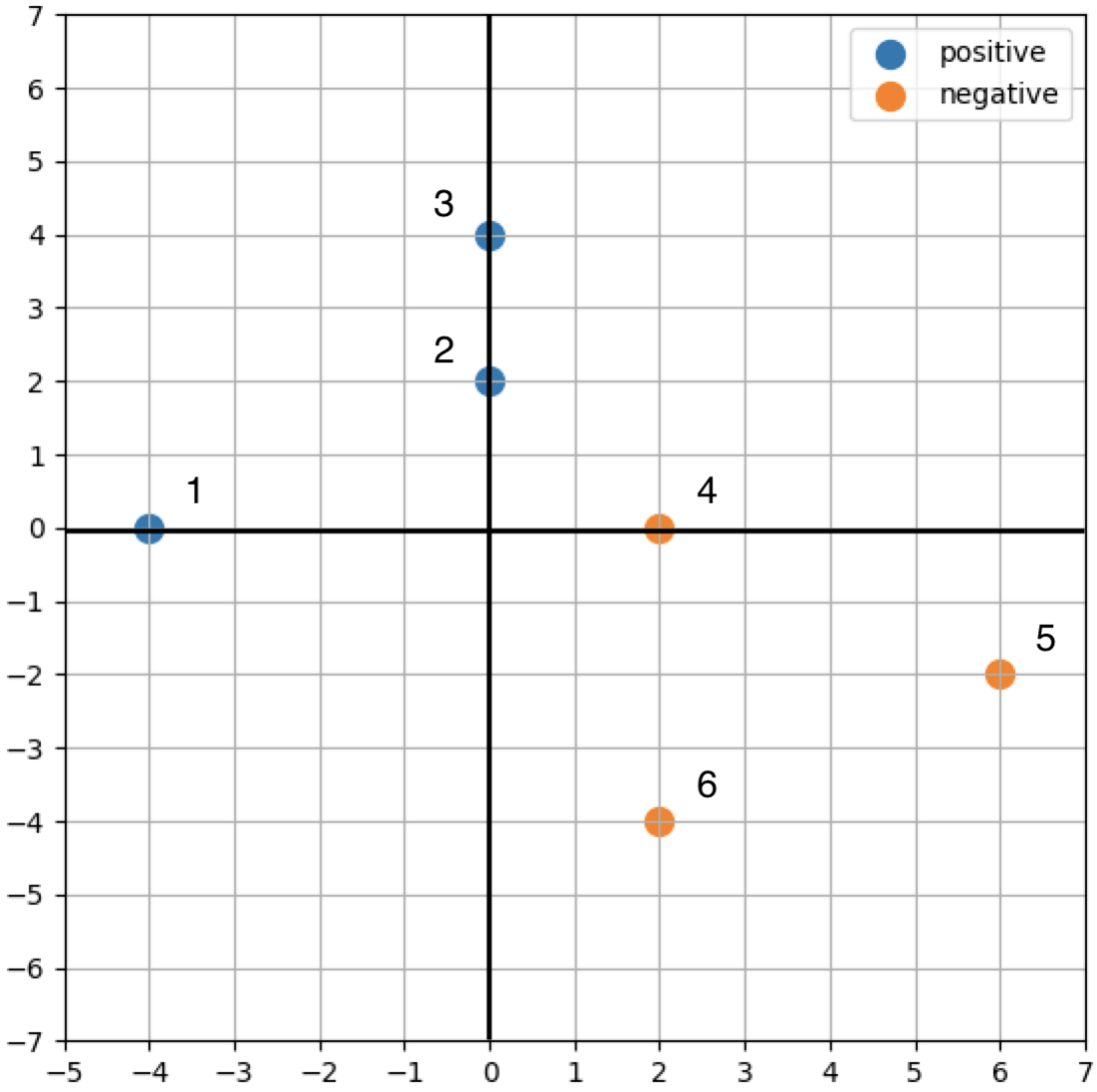


Figure 1: Training examples for SVM

Define $\mathbf{w} =$

Define $\theta =$

(b) [15 points] Recall the dual representation of SVM. There exists coefficients $\alpha_i > 0$ such that:

$$\mathbf{w}^* = \sum_{i \in I} \alpha_i y^{(i)} \mathbf{x}^{(i)} \tag{3}$$

where $I$ is the set of indices of the support vectors.

1. [5 points] Identify support vectors from the six examples given.

   Define $I =$ _____

2. [5 points] For the support vectors you have identified, find $\alpha_i$ such that the dual representation of $\mathbf{w}^*$ is equal to the primal one you found in (a)-2.

   Define $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_{|I|}\} =$ _____

3. [5 points] Compute the value of the hard SVM objective function for the optimal solution you found.

   *Objective function value* = _____

(c) [10 points] Recall the objective function for soft representation of SVM.

$$\textbf{min } \frac{1}{2}||\mathbf{w}||^2 + C\sum_{j=1}^{m}\xi_i \tag{4}$$

$$\text{s.t } y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)} + \theta) \geq 1 - \xi_i, \xi_i \geq 0, \forall(\mathbf{x}^{(i)}, y^{(i)}) \in D \tag{5}$$

where $m$ is the number of examples. Here $C$ is an important parameter. For which **trivial** value of $C$, the solution to this optimization problem gives the hyperplane that **you have found in (a)-2**? Comment on the impact on the margin and support vectors when we use $C = \infty$, $C = 1$, and $C = 0$.

# Submission Instructions

We will be using Gradescope to turn in both the Python code and writeup pdfs. You should have been automatically added to Gradescope. If you do not have access, please ask the TA staff on Piazza.

For this homework assignment, there are two Gradescope assignments:

- "Homework 3 - Code": This is the assignment where you should upload a PDF of your Jupyter Notebook and your implementations of `FeedForward` and `Convolutional` for unit testing. To generate a PDF from Colab, go to "File" then "Print" and select PDF.
- "Homework 3 - PDF": This is the assignment where you should upload your writeup as a PDF.