CIS519: Applied Machine Learning **Fall 2020**

Homework 4

Handed Out: November 17, 2020 *Due: December 3, 2020 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:
  - Notebook on Colab can be found here: `https://colab.research.google.com/drive/1AWUhNJGRJpjzfwj9SIJXjaEIQYYTMWGj?usp=sharing` You will need to `File` → `Save a copy in Drive` in order to obtain a copy you can modify in your Colab drive.
  - Latex template and data can be found here: `https://www.seas.upenn.edu/~cis519/fall2020/assets/HW/HW4/hw4-materials.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 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.[2]

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

[2] 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.

To setup a 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.

To obtain a copy of the notebook for this assignment, follow this link: `https://colab.research.google.com/drive/1AWUhNJGRJpjzfwj9SIJXjaEIQYYTMWGj?usp=sharing`. In order to take advantage of Google Colab's GPU access, **we will run this homework on Google Colab**.

1.2 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 32×32 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 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.

1.3 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[3]
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.

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:

[3]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.

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.

1.4 Extra Credit: Image Normalization [5 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.

1.5 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

2 Document Classification [40 Points]

In this problem, you will develop Naive Bayes-based models to do text classification. The task of text classification to is label a piece of text with one class out of a predefined set of classes.

2.1 Dataset

The dataset you will use for the experiments will be a sentiment analysis dataset from IMDb. Each document is a review of a movie and has been labeled as either a positive or negative review. We have provided for you training, validation, and testing splits in `data/train.jsonl`, `data/valid.jsonl`, and `data/test.jsonl`. We have also provided the Python code to load the documents into memory. Each document has been pre-processed with tokenization[4] and lemmatization.[5]

2.2 Document Representation [5 points]

You will experiment with three different ways to represent the documents. "Representation" means how you convert the raw text of a document to a feature vector. Similar to Homework 2, you will use a sparse representation of the feature vectors which is based on a dictionary that maps from the feature name to the feature value. For each representation, there will be exactly 1 feature per unique token in the vocabulary, but the value of that feature will change.

Vocabulary Creation The vocabulary is the set of tokens that you will use to compute features. There is some nuance to how to select which tokens from the dataset should be part of the vocabulary. If you use all of the tokens in the training data, there could be a token that appears for the first time in the test set, and it's not clear what to do with that token. A common solution to this problem is to map infrequent words in the training data to a token called `<unk>`, compute the features with the `<unk>` tokens, then map the novel test words to `<unk>`. You will do exactly that.

For each of the experiments below, you should create the vocabulary by taking all of the tokens in the training data which appear more than once plus a special `<unk>` token. You

[4]We say "token" instead of "word" because it is more general and covers symbols like punctuation which we treat as their own tokens separate from the surrounding words. "Tokenization" is splitting raw text into individual tokens.

[5]`https://en.wikipedia.org/wiki/Lemmatisation`

should still estimate probabilities for the `<unk>` token. At test time, any unknown words should be treated as `<unk>`s.

Representations We will use $f_d(v)$ to denote the value for feature v and document d.

The three ways that you should represent each document (and the corresponding Python functions to implement) are described below.

Note that we have already implemented the first two, (1) Binary Bag-of-Words and (2) Count Bag-of-Words for you. It is up to you to implement TF-IDF.

1. **Binary Bag-of-Words** Each document should be represented with binary features, one for each token in the vocabulary.

$$f_d(v) = \begin{cases} 1 & \text{if } v \in d \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

   Since you are using a sparse representation, you do not need to remember which tokens are not present in each document, only those which *are* present.

2. **Count Bag-of-Words** Instead of having a binary feature for each token, you should keep count of how many times the token appears in the document, a quantity known as *term frequency* and denoted $tf(d, v)$.

$$f_d(v) = tf(d, v) \tag{2}$$

3. **TF-IDF Model** The final representation will use the TF-IDF score of each token. The TF-IDF score combines how frequently the word appears in the document as well as how infrequently that word appears in the document collection as a whole. First, the inverse document frequency (IDF) of a token is defined as

$$idf(v) = \log \frac{|\mathcal{D}|}{|\{d : v \in d, d \in \mathcal{D}\}|} \tag{3}$$

   Here, $\mathcal{D}$ is the set of documents[6] and the denominator is the number of documents that the token v appears in. Use the `numpy.log()` function to compute the log. Then, the TF-IDF feature you should use is

$$f_d(v) = tf(d, v) \times idf(v) \tag{4}$$

2.3 Naive Bayes Experiment [35 Points]

In the first experiment, you will build three Naive Bayes classifiers, each one using one of the above document representations, and compare their performances on the test dataset.

Recall that the prediction rule for Naive Bayes is

$$\arg\max_y P(d \mid y) \cdot P(y) \tag{5}$$

[6] You should use only the training documents to compute the IDF score. (A previous version of this assignment said you should use the training, validation, and testing. It is OK if you do either of these two options.)

and so you need to compute those two quantities. The simpler of the two quantities $P(y)$ can be computed by counting the number of times each label appears divided by the total number of instances.

$$P(y) = \frac{|\{d : y_d = y\}|}{|\mathcal{D}|} \tag{6}$$

The other quantity can be computed as follows. Under the Naive Bayes assumption, $P(d \mid y)$ is defined as

$$P(d \mid y) = \prod_{v \in d} P(v \mid y) \tag{7}$$

For this question, you will use the features to compute $P(v \mid y)$ as follows:

$$P(v \mid y) = \frac{\sum_{d \in \mathcal{D} : y_d = y} f_d(v)}{\sum_{w \in \mathcal{V}} \sum_{d \in \mathcal{D} : y_d = y} f_d(w)} \tag{8}$$

The summands in the numerator and inner summand of the denominator are over all of the documents in the training data that have label y. The outer summand of the denominator is over all of the words in the vocabulary.

Finally, as you saw in class, you will implement Laplace smoothing on $P(v \mid y)$ as follows:

$$P(v \mid y) = \frac{k + \sum_{d \in \mathcal{D} : y_d = y} f_d(v)}{k \cdot |\mathcal{V}| + \sum_{w \in \mathcal{V}} \sum_{d \in \mathcal{D} : y_d = y} f_d(w)} \tag{9}$$

where k is a hyperparameter which controls the strength of the smoothing. This is the final equation which you should implement to build your Naive Bayes classifier.

Implementation Details When you are computing the values for the prediction rule (Equation 5), you need to multiply many probabilities together, which may result in underflow. Instead, you should implement the classifier in log-space. That is, instead of computing with probabilities $P(v \mid y)$, you should compute with $\log P(v \mid y)$ and use the following equation as the prediction rule:

$$\begin{aligned} \arg\max_y P(d \mid y) \cdot P(y) &= \prod_{v \in d} P(v \mid y) \cdot P(y) && (10) \\ &= \sum_{v \in d} \log P(v \mid y) + \log P(y) && (11) \end{aligned}$$

Note that there is a term in the summand for each token in the document, so if a token appears twice, you will add the corresponding term twice.

What to Report For each of the three document representations, run a hyperparameter sweep over the smoothing parameter $k \in \{0.001, 0.01, 0.1, 1.0, 10.0\}$ using the training and validation data. Then, using the best value of k for each representation, report and compare the test performance. Use accuracy to evaluate the models. Note that the most complicated document representation may not get the best results.

Answer the following question: As the value of k goes to infinity, what will happen to $P(y \mid d)$? Describe this both in terms of the mathematical equation and in words.

3 Theory [20 points]

3.1 Multivariate Exponential naïve Bayes [15 points]

In this question, we consider the problem of classifying piazza posts (Y) into two categories: student posts (A), and instructor posts (B). For every post, we have two attributes: number of words (X_1), and number of mathematical symbols (X_2). We assume that each attribute (X_i, $i = 1, 2$) is related to a post category (A/B) via an Exponential distribution[7] with a particular mean ($\lambda_{A;i}^{-1}/\lambda_{B;i}^{-1}$). That is

$$P(X_i = x \mid Y = A) = e^{-x\lambda_{A;i}}\lambda_{A;i} \quad \text{and} \quad P(X_i = x \mid Y = B) = e^{-x\lambda_{B;i}}\lambda_{B;i} \text{ for } i = 1, 2$$

X_1	X_2	Y
1	4	A
3	9	A
4	6	A
7	3	B
2	6	B
3	0	B
6	5	B

Table 1: Dataset for Exponential naïve Bayes

Assume that the given data in Table 1 is generated by a Exponential naïve Bayes model. You will use this data to develop a naïve Bayes predictor over the Exponential distribution.

$P(Y{=}A) =$	$P(Y{=}B) =$
$\lambda_{A;1} =$	$\lambda_{B;1} =$
$\lambda_{A;2} =$	$\lambda_{B;2} =$

Table 2: Parameters for Exponential naïve Bayes

1. [**5 points**] Compute the prior probabilities ($P(Y = A)$ and $P(Y = B)$). Use maximum likelihood estimation to find the corresponding λ parameter values. Fill in the results in Table 2. Please show all the intermediate steps and results.
2. [**5 points**] Based on the parameter values from Table 2, compute

$$\frac{P(X_1{=}3, X_2{=}5|Y{=}A)}{P(X_1{=}3, X_2{=}5|Y{=}B)}$$

Write the full expression. You do not need to simplify.

[7]http://en.wikipedia.org/wiki/Exponential_distribution

3. [**2.5 points**] Write the decision rule for the Exponential Naive Bayes predictor based on the previous equation. The answer we are looking for uses an if statement based on the value of the previous ratio and an additional probability term.
4. [**2.5 points**] Using the parameter values you estimated in the previous steps and the decision rule, what will the classifier predict as the value of Y, given the data point: X_1=3, X_2=5?

3.2 Coin Toss [5 points]

Consider the following way to generate a series of Heads and Tails. For each element in the series, first a coin is tossed. If it comes up as a T, it is shown to the user. On the other hand, if the coin toss comes up as an H, then the coin is tossed the second time, and the outcome of this toss is shown to the user.
Assume that the probability of a coin toss coming up as an H is p (and hence, the probability of it coming up as a T is $1 - p$). Suppose you see a sequence $TTHTHHTHTT$ generated based on the scheme given above. What is the most likely value of p?
(Assume a Bernoulli model to compute probability of the coin toss sequence.)

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 4 - Code": This is the assignment where you should upload your a python version of your Jupyter Notebook. Note that for this homework, we are asking you to submit *all* of your final code. Submit your implementations of `FeedForward`, `Convolutional`, `TFIDFFeaturizer`, `compute_idf`, `get_vocabulary`, `train_naive_bayes`, and `predict_naive_bayes` for unit testing. Also submit any other code you used to run experiments, generate plots, and run hyperparameter sweeps, etc., but comment it out. (The only non-commented-out sections should be the import statements and functions listed in the previous sentence.) Please name your file `hw4.py`
- "Homework 4 - PDF": This is the assignment where you should upload your writeup as a PDF.