Homework 1

18739 Spring 2020

Due: Thursday, Feburary 6, 2020 at 10:20am PST / 1:20pm EST

Academic Integrity Policy

This is an individual homework. Discussions are encouraged but you should write down your own code and answers. No collaboration on code development is allowed. We refer to CMU's Policy on Cheating and Plagiarism (https://www.cmu.edu/policies/). Any code included in your submission will be examined under Similarity Check automatically.

Late Day Policy

Your solutions should be uploaded to Gradescope (https://www.gradescope.com/) by the deadline. You have 8 late days in total to spend during the whole course but no more than 3 days can be applied to any one homework.

Written Exercises

If a homework contains exercises with written answers you will need to include a `.pdf` file containing your solutions in the zip mentioned below. Please indicate your Andrew ID and name on the first page. We will not accept hard-copies and do not hand-write your answers. Latex (Overleaf: https://www.overleaf.com) and Markdown are two recommended languages to generate the clean layout but you are free to use other software.

Coding Exercises

Submit the coding portions of homeworks to Gradescope according to the following procedure:

1. Compress your `.py` and other requested files into a zip file called `submission.zip` (No other name is allowed otherwise Gradescope will fail to grade your submission). Do NOT put all files into a folder first and compress the folder. **Create the zip file directly on the top of all required files**. Each homework will specify the `.py` and other files expected to be included in the zip.
2. Submit `submission.zip` to Gradescope. Your code implementation will be auto-graded by Gradescope and you have an infinite number of submissions before the deadline but only the last submission will count towards the grading. Allow the server to take some time (around 2 min) for execution, which means that do not submit until the last moment when everyone is competing for resources.

---

1 Introduction

In this homework, you will be implementing a simple logistic regression in Python to classify images of hand-written digits in the *MNIST* dataset. In this dataset, each input image is of size 28×28 and reshaped into a size 784 vector. The output is an integer from 0 to 9 representing the image class. **You will be solving the same problem at three levels of implementation**: First, you will be using *Keras*, which is a high-level neural-network API. Secondly, you will be using *TensorFlow*, which is a popular deep learning library in Python, and finally, you will be implementing the the algorithm from ground up by deriving the gradient formulas yourself and using only the numeric library `numpy`.

Starter code is provided for loading the data. You can find the data file is split into training set, validation set, and testing set. The training set is used to train a model and the validation set is used to tune the hyper-parameters used in the model. The test set is only used to report a final score/accuracy. In this problem, we have only two hyper-parameters: learning rate and batch size. Learning rate determines the size of step to take for each gradient update. Batch size determines the amount of data to use in one gradient update step. Number of epochs (1 epoch means going through all of the training data set once) is usually determined using an early-stopping mechanism, but in this homework you can treat it as a fixed parameter and specify it yourself.

Optional

You can use *Jupyter Notebook* to debug your homework. It is a web application where you can edit/run code, write text/equations, and visualize plots, all in one place. You can consult the documentation at http://jupyter.org/. Section 3 provides detailed installation instructions. **But your submission is not required to contain a jupyter notebook**. Please refer to Section 6.3 for the requirement of submission.

2 Background

For this homework you will need to review the slides and background reading on logistic regression, softmax, cross-entropy loss, and gradient descent.

Logistic regression is a probabilistic, linear classifier defined by

- a weight matrix $\mathbf{W}$
- a bias vector $\mathbf{b}$
- scoring function: given input instance $\mathbf{x}$, the score for class j is $\mathbf{s}_j \stackrel{\text{def}}{=} \mathbf{W}_j^T \mathbf{x} + \mathbf{b}_j$,
- softmax normalization: given scores $\mathbf{s} = \langle \mathbf{s}_1, \cdots, \mathbf{s}_n \rangle$, the class probabilities over n classes, denoted $\mathbf{y}'$, are the vector $\mathbf{y}' \stackrel{\text{def}}{=} \text{softmax}(\mathbf{s}) \stackrel{\text{def}}{=} \frac{1}{\sum_k \exp(\mathbf{s}_k)} \exp(\mathbf{s})$

Putting all this together, the probability of class j given input $\mathbf{x}$ is:

$$P(y' = j | \mathbf{x}, \mathbf{W}, \mathbf{b}) = \frac{\exp\left(\mathbf{W}_j^T \mathbf{x} + \mathbf{b}_j\right)}{\sum_k \exp\left(\mathbf{W}_k^T \mathbf{x} + \mathbf{b}_k\right)}$$

In the probabilistic notation, we use y' to refer to the class instead of one-hot vector encoding of class or class probabilities. Prediction is done by taking the class of highest probability:

$$y_{pred} \stackrel{\text{def}}{=} \text{argmax}_i P(y' = i|\mathbf{x}, \mathbf{W}, \mathbf{b})$$

For each input vector $\mathbf{x}$ and one-hot-encoded ground truth vector $\mathbf{y}$ in dataset $\mathbf{X}$, $\mathbf{Y}$, the cross-entropy loss is

$$\begin{aligned} L(\mathbf{x}, \mathbf{y}; \mathbf{W}, \mathbf{b}) &\stackrel{\text{def}}{=} -\sum_j \mathbf{y}_j \log \mathbf{y}'_j \\ &= -\mathbf{y} \odot \log \mathbf{y}' \end{aligned}$$

where $\odot$ refers to dot product.

In this homework, you will not be processing each instance at a time but instead handle a subset ("batch") of the training dataset. Let $\mathbf{X}$ be the input matrix consisting of all the images in one training batch. $\mathbf{X}_i$ is the i-th image represented by the i-th column. Let $\mathbf{Y}$ be the ground truth matrix one-hot-encoded and $\mathbf{Y}'$ be the estimated probability matrix. Let N be the batch size. For one batch of 100 images of size 28*28 in 10 classes, the dimensions of $\mathbf{X}$ and $\mathbf{Y}$ are $(100, 784)$ and $(100, 10)$, respectively. The loss function for a batch is thus:

$$\begin{aligned} \mathbf{L}(\mathbf{X}, \mathbf{Y}; \mathbf{W}, \mathbf{b}) &\stackrel{\text{def}}{=} -\frac{1}{N} \sum_i \mathbf{Y}_i \odot \log \mathbf{Y}'_i \\ &= -\frac{1}{N} \sum_i \sum_j \mathbf{Y}_{i,j} \log \mathbf{Y}'_{i,j} \end{aligned}$$

Now we have the relationship between $\mathbf{L}$ and $\mathbf{W}$,$\mathbf{b}$, by using the chain rule, we can derive the gradient $\frac{\partial \mathbf{L}}{\partial \mathbf{W}}$ and $\frac{\partial \mathbf{L}}{\partial \mathbf{b}}$. Most deep learning software calculates the gradients automatically for you. In Level 3 of this homework, you will need to derive it by yourself. After deriving the gradient values, we can perform stochastic gradient descent to find the optimal $\mathbf{W}$ and $\mathbf{b}$.

3 Before You Start

3.1 Software Requirements

This assignment requires Python 3.7, `numpy`, `gzip`, `keras`, `tensorflow`, `matplotlib`, and `jupyter`. We require that you install specific versions of these libraries inside an anaconda environment. Please follow the provided instructions below.

3.2 Installation Instructions

1) First install `Miniconda` with Python 3.7 (instructions at https://docs.conda.io/en/latest/miniconda.html).

2) Once `Miniconda` is installed, create the virtual environment from the provided `18739_hw12020.yml` by running

`$ conda env create -f 18739_hw.yml`.

This will create an environment, `fsdl`, that can be used by running

`$ conda activate fsdl`.

Optional

In order to use Jupyter Notebooks, you will first need to install it under the fsdl environment by running:

`$ pip install jupyter`.

Jupyter is then started using the following command:

`$ jupyter notebook`.

This will locally host all the notebooks in the current directory for interaction through a browser. Your browser should automatically open, but the notebook can be accessed by navigating to `localhost:8888/` in your browser.

Manually Installation Guide

If the instruction above does not work on your computer, it is probably due to the conda version issues. If updating conda version does not solve your problem, try to install packages by the following command:

Create a new environment by:

```
conda create -n fsdl python=3.7
```

Install keras, numpy and tensorflow:

```
conda install numpy keras tensorflow
```

3.3 Dataset and Loading

The MNIST dataset `mnist.pkl.gz` is included in the release folder. The starter code for loading the dataset is included in `hw1_utils.py`. It loads the dataset into three separate sets: training set, validation set, and test set.

4 Level 1 - Keras Implementation

Keras is a high-level neural network API which runs on top of a Tensorflow, CNTK, or Theano backend. Typically one can choose the backend if they have more than one installed, however, we will be using Tensorflow in the next part of the assignment, and Tensorflow is included in the provided Anaconda environment.

4.1 Coding Exercise: Implementation *[6 points]*

Complete `build_model()` and `run_keras()` in `hw1_keras.py`. You may want to to consult the documentation for Keras on https://keras.io/. Your program should contain the following parts with each no more than a line or two:

- Load the dataset with `load_mnist()` and encode the groundtruth into the one-hot vectors. The loading part is already provided in the code template.
- Create the logistic regression model using canonical Keras via the Sequential or Functional approach.
- Compile your model with the desired loss function, optimizer, and metrics.
- Fit your training data (you can also specify validation data using the validation set here).
- Predict on the test data and report the test accuracy (the percentage of images correctly classified).

You may find the `to_categorical` function (in `keras.utils.np_utils`) helpful for converting a vector of integers into its one-hot representation.

4.2 Grading

If you pass the test cases on Gradescope, you will be given 3 points for test cases and 3 points for implementation. If you fail to pass the test cases, you will be given partial points (up to 3 points) on your implementation.

5 Level 2 - Tensorflow Implementation

While Keras is typically easiest to use, its level of abstraction is occasionally above the level required for specialized applications. Furthermore, in an industry setting, you may not always have autonomy over the framework you use. TensorFlow is a popular deep learning framework which lets you build a computational graph to represent a deep neural network and other components including the loss function. In the late 2019, Google has published Tensorflow 2.0 to replace the original Tensorflow 1.x versions. **In this homework, we will only be using Tensorflow 1.x instead of Tensorlfow 2.0.** It is required that you use the Tensorflow API in provided conda environment. More detailed discussion about using Tensorflow 1.x instead of Tensorflow 2.0 is discussed here https://www.tensorflow.org/guide/migrate

5.1 Coding Exercise: Implementation *[12 points]*

Complete the code in `build_model()` and `run_tf()` in `hw1_tf.py`; you may want to to consult the documentation for TensorFlow at https://www.tensorflow.org/api_docs/python/tf. Your training code should be batched, processing instances in batches of size specified by `run_tf()` arguments. You may not use `GradientDescentOptimizer`, or any other predefined optimizer. In this part of the homework you need to implement gradient descent using the lower level API yourself.

Your program should contain the following parts:

- Setup the input placeholders: create the input tensor `X` and groudtruth tensor `Y` (tensorflow placeholders). This is already done in the code template.
- Model construction. Define various tensors including the mutable weight and bias tensors (tensorflow variables) to store the model parameters; the derived `loss_op` and `accuracy_op` tensors for computing loss and accuracy, respectively based on model parameters and the input placeholders; and finally the parameter update operations `update_W_op` and `update_b_op`. You will find it easier to define intermediate tensors for model outputs, predictions, and gradients.

- Training. Create and run a TensorFlow session and train the model on batches of the training data for the requisite number of epochs as specified by `run_tf` method arguments.
- Evaluation. After training, compute `text_accuracy`, the accuracy of the final model on the test instances.

Hints:

- You might find it convenient to a batchable dataset using `tf.data.Dataset` and `tf.data.Iterator` .
- You will want to look at the documentation for `tf.get_variable` (or `tf.Variable`), `tf.gradients`, `tf.assign`, and `tf.Session`. Additionally there are many tensor operations available which you can use for defining the model, loss function, and accuracy.

5.2 Grading

If you pass the test cases on Gradescope, you will be given 7 points for test cases and 5 points for implementation. If you fail to pass some of the test cases, you will be given partial points (up to 5 points) on your implementation.

6 Level 3 - Python Implementation

In some highly-specialized settings, automated tools may be unable to compute the gradient of your loss function. In these cases you would need to further lower your level of abstraction to implement backpropagation.

In this section, you are going to solve the same problem using only `numpy`. You will actually derive the gradient expressions and hard-code them into the program. Also, you are going to use the validation set to select the best learning rate. You can choose your own epoch count and batch size.

6.1 Written Exercise: Derivation *[2 points]*

Derive the gradient of loss in terms of $\mathbf{W}$ and $\mathbf{b}$: $\frac{\partial \mathbf{L}}{\partial \mathbf{W}}$ and $\frac{\partial \mathbf{L}}{\partial \mathbf{b}}$. Show your work and make sure the dimensions of your vectors are consistent with the ones in the problem description.

6.2 Coding Exercise: Implementation *[7 points]*

You program in this section should be self contained and independent from other sections; do not make use of any functions other than `numpy` functions. **No Keras or Tensorflow methods are allowed in this section**. Furthermore, other than looping over epochs or batches, **do not use loops in your code**. Processing instances in a batch or values in an instance should be done using numpy vector/matrix/tensor operations. Loops include `for` and `while` statements, comprehensions, generators, and recursion.

Complete methods `softmax`, `sgd`, `evaluate` and `run_numpy` based on your derivations in `hw1_numpy.py`.

1. `softmax()` – the softmax activation function. Be aware that your input is a batch of data of size (N, 10) where N is the batch size.

2. `sgd()` – min-batch Stochastic Gradient Descent. Computes the optimal weight and bias after the given number of epochs. You should use the training parameters specified in the method arguments.

3. `evaluate()` – accuracy measurement. Given the model parameters and test data, computes the predictions and returns the accuracy of the predictions relative to the given test data.

4. `run_numpy()` – pipeline invoker. Gets the data, initialized model parameters, optimizes them, computers and returns test accuracy.

6.3 Grading

If you pass the test cases on Gradescope, you will be given 4 points for test cases and 3 points for implementation. Using looping constructs other than for epochs or batches will incur penalties in implementation points. If you fail to pass some of the test cases, you will be given partial points (up to 3 points) on your implementation.

Submission

Typeset your answer to Section 6.1 into `hw1_written.pdf`. Submit `hw1_written.pdf`, `hw1_keras.py`, `hw1_tf.py`, `hw1_numpy.py` and `mnist.pkl.gz` as a single zip file as specified by the submission procedures at the beginning of this document.

7 Common Problems

- Common issues when employing arrays or tensors is dimensionality mismatch. Inspecting the shape of the inputs involved is helpful. Both tensorflow `Tensor` and numpy `ndarray` have a `shape` attribute. Other issues might arise due to typing where the `dtype` attribute might be helpful.

- On the other hand, if numpy or tensorflow performs a tensor operation when you would expect a mismatch in dimensions, it is possible that *broadcasting* was involved. Consult the broadcasting documentation in Tensorflow (https://www.tensorflow.org/xla/broadcasting) and numpy (https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html).

- Many numpy and tensorflow operations take in an *axis* argument to specify along which dimension the operation should be carried out. The default axis or axes may not be what you need.

- If you are using jupyter to develop your solutions and get strange errors, there might be unintended interaction between the stateful tensorflow and keras APIs. Restarting the kernel usually solves these issues.

- If you are using the tensorflow data iterator approach for reading and batching data, note that `get_next` method on the iterator is connected to the computation graph and will be called once for every call to `session.run`. You should be careful to call `session.run` only once when updating the weights, biases, and calculating the accuracy.

- Within a TensorFlow *session*, you will get very bad performance if you use any calls to `tf._`. You should keep the construction of the computation graph as tensors outside the session.