Assignment 1: Welcome to TensorFlow

CS20: TensorFlow for Deep Learning Research (cs20.stanford.edu)
Due 1/31 at 11:59pm
Prepared by Chip Huyen (chiphuyen@cs.stanford.edu)

I'd like the assignments to more closely resemble what you'd encounter the in the real world, so the problems are more open-ended and less structured. Problems often don't have starter code or very simple starter code. You get to implement everything from scratch!

Most problems have several options for you to choose from. Some options are easier than others, and that will be taken into account when I grade. You'll be graded on both functionality and style, e.g. how elegant your code is.

I'm more interested in the process than the results, so you might still get good grade even if you don't arrive at the results you want to, as long as you explain the difficulties you encountered along the way and how you tackled them.

I hope that through this problem set, you'd familiarize yourself with TensorFlow's official documentation.

Problem 1: Op is all you need

This first problem asks you to create simple TensorFlow ops to do certain tasks. You can see the tasks at `assignments/01/q1.py` on the GitHub repository for this class.

These tasks are simple--their purpose is to get you acquainted with the TensorFlow API.

Problem 2: Logistic regression

2a. Logistic regression with MNIST

We never got around to doing this in class but don't worry, you'll have a chance to do it at home. You can find the starter code at `examples/03_logreg_starter.py` on the GitHub repository of the class.

For the instruction, please see the lecture note 03.

2b. More logistic regression

You can choose to do one of the two tasks below.

Task 1: Improve the accuracy of logistic regression on MNIST

We got the accuracy of ~91% on our MNIST dataset with our vanilla model, which is unacceptable. The state of the art is above 99%. You have full freedom to do whatever you want here as long as your model is built in TensorFlow.

You can reuse the code from part 1, but please save your code for part 2 in a separate file and name it `q2b.py`. In the comments, explain what you decide to do, instruction on how to run your code, and report your results. I'm happy with anything above 97%.

Task 2: Logistic regression on notMNIST

MNIST is so popular as a toy dataset for computer vision tasks that the machine learning community got a little bit sick of it. Yaroslav Bulatov, a research engineer at OpenAI, created a similar dataset and literally named it notMNIST[1]. This rebel is designed to look like the classic MNIST dataset, but less 'clean' and extremely cute. The images are still 28x28 and the there are also 10 labels, representing letters 'A' to 'J'. You should save your file in `q2b.py`.

[1] Bulatov, Y. "Notmnist dataset." Google (Books/OCR), Tech. Rep.[Online]. Available: http://yaroslavvb. blogspot. it/2011/09/notmnist-dataset. html (2011).

The format of notMNIST is not the same as MNIST, but David Flanagan is very kind to publish his script to convert notMNIST to MNIST format. After converting notMNIST to MNIST format, you can use the `read_mnist` module in `utils.py` to read it in.

Once you have the data ready, you can use the exact model you built for MNIST for notMNIST. Don't freak out if the accuracy is lower for notMNIST. notMNIST is supposed to be harder to train than MNIST.

Problem 3: More on word2vec

The answers for questions 3a, 3b, and 4 should be saved in a file named a1_answers in any format of your choice.

3a. Application of word embeddings

Briefly give three examples of real world applications that use word embeddings.

3b. Visualize word embeddings and find interesting word relations

We've implemented word2vec in TensorFlow and it's pretty neat. You should visualize the word embeddings on TensorBoard. Then you can either use this visualization or write a script of your own to find the interesting word relations. For example, you can find five words closest to certain country names to see if our embeddings catch any racial bias. Another idea is that you can find word spectra: you start with two opposite words, get the line connecting that two words, get 100 words closest to that line and arrange them by the order they appear on that line, you'll see how a word fades into another.

For inspiration, you can read about sexism in word embeddings here. I've also written a post about the inherent biases of Artificial Intelligence here.

3c. word2vec

We implemented word2vec with skip-gram model in class. Now, you can try to implement the CBOW model. Save the model in `cbow.py`.

Problem 4: Feedback

This is an opportunity for you to give me feedback on the class. Please answer some or all of the questions below:

- What parts of the class do you find useful?
- What parts do you find confusing?
- What parts do you find too easy?
- What do you want to see/do more in the class?
- What parts did you find less interesting?
- What do you think of the first assignment?
- How long did it take you to do the first assignment?
- Any recommendations for me?

Submission Instructions
Your submission should contain the following files:
q1.py
03_logreg_starter.py
q2b.py
cbow.py (optional)
a1_answers (contains your answers to questions 3a, 3b, and 4)

Compress your submission into a zip file and send it to cs20-win1718-staff@lists.stanford.edu with subject title: "[Your SUNetID]_assignment1"

Thank a lot, guys!