Assignment 1: Welcome to TensorFlow
CS20 SI: TensorFlow for Deep Learning Research (cs20si.stanford.edu)
Due 1/31 at 11:59pm
Prepared by Chip Huyen (huyenn@stanford.edu)
Reviewed by Danijar Hafner

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. Most problems don't have 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 a check plus even if you don't arrive at the results you want to, as long as you explain the problem you encountered along the way and how you tackled them.

I want you to get more familiar with the TensorFlow documentation that you can find at https://www.tensorflow.org/api_docs/python/.

Problem 1: Ops exercises
This first problem asks you to create simple TensorFlow ops to do certain tasks. You can see the tasks in the file `a1_q1_exercises.py` in the folder "assignments" of 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
You can choose to do one of the three following tasks:

Task 1: Improving the accuracy of our logistic regression on MNIST

We got the accuracy of ~90% on our MNIST dataset with our vanilla model, which is unacceptable. The dataset is basically solved and state of the art models reach accuracies above 99%. You can use whatever loss functions, optimizers, even models that you want, as long as your model is built in TensorFlow. You can save your code in the file q2.py. In the comments, explain what you decided to do, instruction on how to run your code, and report your results. I'm happy if you can reach 97%.

Task 2: Logistic regression on the notMNIST dataset

I guess the machine learning community is a bit sick of seeing MNIST pop up everywhere so they created a similar dataset and literally named it notMNIST. Created by Yaroslav Bulatov, a research engineer previously at Google and now at OpenAI, notMNIST 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'.

Since the format of the notMNIST dataset is not the same as the MNIST dataset, you can't just put the notMNIST dataset in place of the MNIST dataset for your model. David Flanagan is very kind to publish his script to convert notMNIST to MNIST's format. Even then, you still can't just call the function input_data from tensorflow.examples.tutorials.mnist to read in data for you. You'll have to read in data yourself, and you'll also have to take care of how to separate your train set from your test set.

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.

Task 3: Build a logistic regression model to predict whether someone has coronary heart disease

In the data folder on the class's GitHub repo, you'll find the file heart.txt. The first row is the name of the observed variables. There are 10 variables:

- sbp: Systolic blood pressure
- tobacco: Cumulative tobacco consumption, in kg
- ldl: Low-density lipoprotein cholesterol

- adiposity: Adipose tissue concentration
- famhist: Family history of heart disease (1=Present, 0=Absent)
- typea: Score on test designed to measure type-A behavior
- obesity: Obesity
- alcohol: Current consumption of alcohol
- age: Age of subject
- chd: Coronary heart disease at baseline; 1=Yes 0=No

Each following row contains the information of one patient. There are 462 samples in total.

We will be using the first 9 variables to predict the last variable. That is, your input will be 1-d tensor of 9 elements, and your label is binary. You should write the function to read in data yourself, and you should take care of dividing your data into train set and test set.

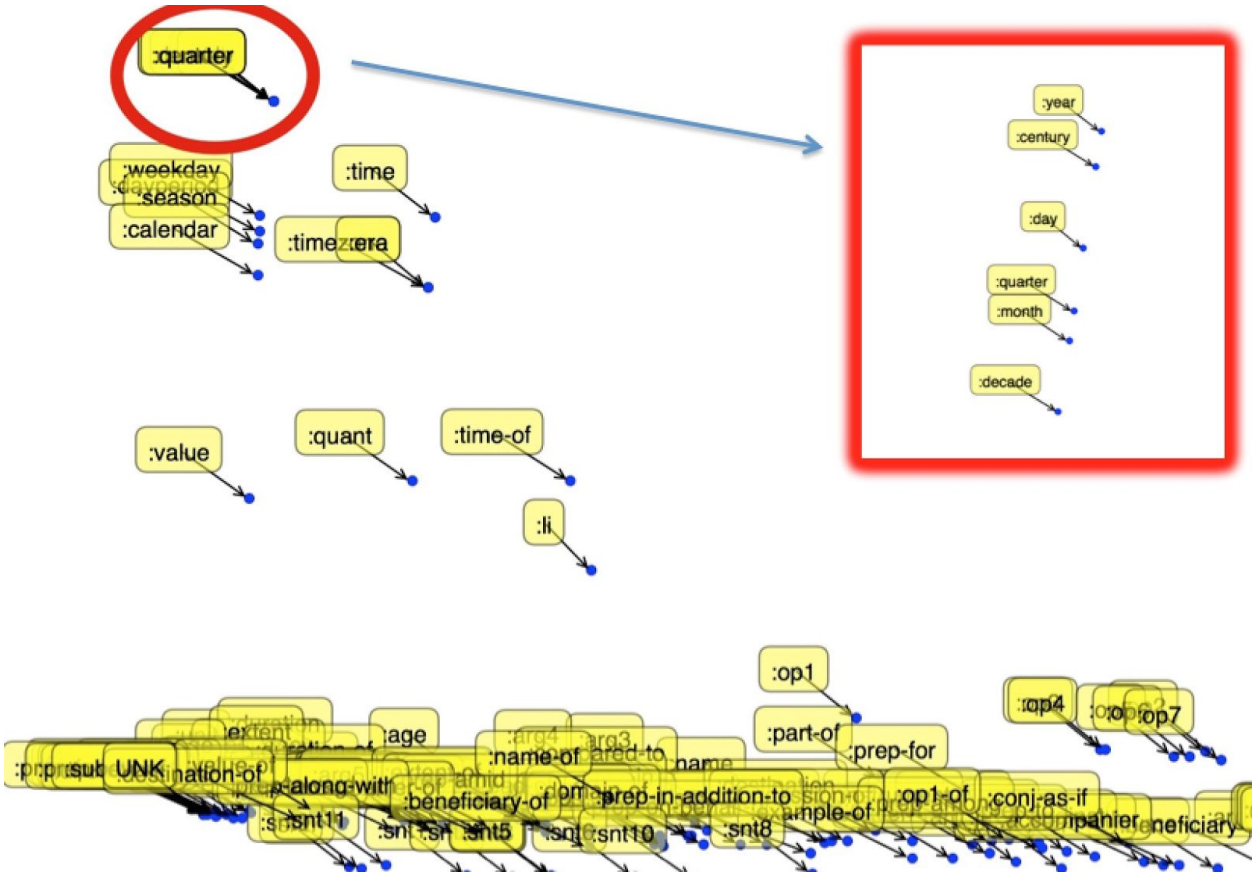
Report your results and your hyperparameters.

Source of the data:
http://statweb.stanford.edu/~tibs/ElemStatLearn/

Problem 3: Don't predict, count!
(Should attempt only after classes about word2vec on Wednesday, 1/25)

A couple of years ago, researchers at University of Trento, Italy published a great paper advocating for the switch to context-predicting models for word embeddings. The paper is named "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors". It compares the performance of the new context-predicting models such as word2vec (where you predict the context word from the center word, or vice versa), with the classical context-counting models.

The classical context-counting models still have a lot of merit. They are easier to build and faster to train (for small-enough vocabulary). They return surprisingly good word relations. For example, a context-counting model run for 10 minutes on AMR 1.0 dataset returned the following graph (this is a project I did for AMR group at University of Edinburgh last summer). You can see that time-related terms such as year, century, day, quarter, etc. are grouped so close together that you have to zoom in to see them.



In this problem, you will build a context-counting model on the dataset of your choice. I provided the text8 dataset, which is the first 100 MB of cleaned text of the English Wikipedia dump on Mar. 3, 2006. You can download the dataset from Matt Mahoney's website. The text is already processed, you can get the tokens just by splitting the text at spaces.

Steps to build the context-counting model:
Step 1: Read in your data, build the vocabulary.
Step 2: Build co-occurrence matrix.
Step 3: Use singular value decomposition (SVD) to reduce the dimensionality of your co-occurrence matrix to your embedding size. You should use **tf.svd** for this.

This resulting co-occurrence matrix is your embedding matrix, each row corresponds to the vector representation of one vector.

I advise you to keep your vocabulary under 10,000 words. You can choose the 9,999 most frequent tokens, and replace any other tokens with <UNK>. The embedding size can be anywhere from 50 to 300.

Problem 4 (extra credit): More on word2vec
(Should attempt only after classes about word2vec on Friday, 1/27)

4a. Interesting word relations in word2vec

We've implemented word2vec in TensorFlow and it's pretty neat. The problem is that it trained on only a small dataset (100MB of text is fairly small for an NLP model), and therefore, it doesn't capture all the semantics very well. In this problem, you'll train our word2vec model on the dataset fil9 -- the first 10^9 bytes of the Wikipedia dump, as described on Matt Mahoney's website.

After that, you should try to find the interesting word relations. For example, you can find the words closest to words of your choice. One of my hobbies is to find words closest to country names to see how racist our embeddings can be :-) 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.

4b. Application of word embeddings.

Briefly give examples of some of the real world applications of word embeddings.

Problem 5: 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 more in the class? What parts did you find less interesting? What do you think of the first assignment? Any recommendations for me?

Submission Instructions

I'm still trying to figure out the best way for students to submit the assignments as I've been warned that the myth submit script is pretty janky. I'll let you know soon!