Homework 4: TensorFlow
Due December 19, 11:59 pm
Worth 20 points

December 4, 2017

Warning: Owing to the grade submission deadline, you may not use late days to extend the deadline for this homework or any other homework beyond December 19th. Any work turned in after 11:59pm on December 19 will be considered late and will receive a grade of 0.

Instructions on writing and submitting your homework.
See the instructions posted on Canvas or on the course webpage at http://www-personal.umich.edu/~klevin/teaching/Fall2017/STATS700-002/hw_instructions.html. Please note that overly complicated solutions; solutions that suggest a failure to read the slides and/or documentation; or solutions that suggest an incomplete understanding of the subject matter will not receive full credit.

1 Warmup: Constructing a 3-tensor (5 points)

You may have noticed that the TensorFlow logo, seen in Figure 1 below, is a 2-dimensional depiction of a 3-dimensional orange structure, which casts shadows shaped like a “T” and an “F”, depending on the direction of the light. The structure is five “cells” tall, four wide and three deep.

![TensorFlow logo](image)

Figure 1: The TensorFlow logo.
Create a TensorFlow constant tensor `tflogo` with shape 5-by-4-by-3. This tensor will represent the 5-by-4-by-3 volume that contains the orange structure depicted in the logo (said another way, the orange structure is inscribed in this 5-by-4-by-3 volume). Each cell of your tensor should correspond to one cell in this volume. Each entry of your tensor should be 1 if and only if the corresponding cell is part of the orange structure, and should be 0 otherwise. Looking at the logo, we see that the orange structure can be broken into 11 cubic cells, so your tensor `tflogo` should have precisely 11 non-zero entries. For the sake of consistency, the (0, 3, 2)-entry of your tensor (using 0-indexing) should correspond to the top rear corner of the structure where the cross of the “T” meets the top of the “F”. Note: if you look carefully, the shadows in the logo do not correctly reflect the orange structure—the shadow of the “T” is incorrectly drawn. Do not let this fool you!

Hint: you may find it easier to create a Numpy array representing the structure first, then turn that Numpy array into a TensorFlow constant. Second hint: as a sanity check, try printing your tensor. You should see a series of 4-by-3 matrices, as though you were looking at one horizontal slice of the tensor at a time, working your way from top to bottom.

# Building and training simple models (10 points)

In this problem, you’ll use TensorFlow to build the loss functions for a pair of commonly-used statistical models. In all cases, your answer should include placeholder variables `x` and `ytrue`, which will serve as the predictor (independent variable) and response (dependent variable), respectively. Please use `W` to denote a parameter that multiplies the predictor, and `b` to denote a bias parameter (i.e., a parameter that is added).

1. **Logistic regression with a negative log-likelihood loss.** In this model, which we discussed briefly in class, the binary variable `Y` is distributed as a Bernoulli random variable with success parameter \( \sigma(W^T X + b) \), where \( \sigma(z) = \frac{1}{1 + \exp(-z)} \) is the logistic function, and \( X \in \mathbb{R}^6 \) is the predictor random variable, and \( W \in \mathbb{R}^6, b \in \mathbb{R} \) are the model parameters. Derive the log-likelihood of `Y`, and write the TensorFlow code that represents the negative log-likelihood loss function. **Hint:** the loss should be a sum over all observations of a negative log-likelihood term.

2. **Estimating parameters in logistic regression.** The zip file at [http://www-personal.umich.edu/~klevin/teaching/Fall2017/STATS700-002/logistic.zip](http://www-personal.umich.edu/~klevin/teaching/Fall2017/STATS700-002/logistic.zip) contains four Numpy `.npy` files that contain train and test data generated from a logistic model:

   - `logistic_xtest.npy`: contains a 500-by-6 matrix whose rows are the independent variables (predictors) from the test set.
   - `logistic_xtrain.npy`: contains a 2000-by-6 matrix whose rows are the independent variables (predictors) from the train set.
   - `logistic_ytest.npy`: contains a binary 500-dimensional vector of dependent variables (responses) from the test set.
   - `logistic_ytrain.npy`: contains a binary 2000-dimensional vector of dependent variables (responses) from the train set.

The \( i \)-th row of the matrix in `logistic_xtrain.npy` is the predictor for the response in the \( i \)-th entry of the vector in `logistic_ytrain.npy`, and analogously for the two test set files. **Note:** we didn’t discuss reading Numpy data from files. To load the
files, you can simply call `xtrain = np.load('xtrain.npy')` to read the data into the variable `xtrain`. `xtrain` will be a Numpy array.

Load the training data and use it to obtain estimates of $W$ and $b$ by minimizing the negative log-likelihood via gradient descent. **Another note:** you’ll have to play around with the learning rate and the number of steps. Two good ways to check if optimization is finding a good minimizer:

- Try printing the training data loss before and after optimization.
- Use the test data to validate your estimated parameters.

3. **Evaluating logistic regression on test data.** Load the test data. What is the negative log-likelihood of your model on this test data? That is, what is the negative log-likelihood when you use your estimated parameters with the previously unseen test data?

4. **Evaluating the estimated logistic parameters.** The data was, in reality, generated with

$$W = (1, 1, 2, 3, 5, 8), \quad b = -1.$$  

Write TensorFlow expressions to compute the squared error between your estimated parameters and their true values. What is the squared error? **Note:** you need only evaluate the error of your final estimates, not at every step.

5. **Probit regression with negative log-likelihood loss.** The probit model is similar to logistic regression, but instead of modeling the response as a Bernoulli with success parameter $\sigma(W^T X + b)$, we model it as a Bernoulli with success parameter $\Phi(W^T X)$, where $\Phi$ is the cumulative distribution function of a standard normal random variable, and now $W \in \mathbb{R}^5$. Derive the log-likelihood of $Y$ and write TensorFlow code that represents the negative log-likelihood loss function of this model. **Note:** your function should now only involve $x, y$ and $W$, since there is no bias term. **Hint:** you’ll want to read the documentation surrounding TensorFlow distribution objects, in particular their `cdf()` method: [https://www.tensorflow.org/api_docs/python/tf/distributions/Normal](https://www.tensorflow.org/api_docs/python/tf/distributions/Normal)

6. **Estimating parameters in probit regression.** The zip file located at [http://www-personal.umich.edu/~klevin/teaching/Fall2017/STATS700-002/probit.zip](http://www-personal.umich.edu/~klevin/teaching/Fall2017/STATS700-002/probit.zip) contains four files named and organized analogously to the files in `logistic.zip`. Load the training data and use it to obtain an estimate of $W$. This time, instead of using gradient descent, use Adagrad, supplied by TensorFlow as the function `tf.train.AdagradOptimizer`. Adagrad is a **stochastic gradient descent algorithm**, popular in machine learning. You can call this just like the gradient descent optimizer we used in class—just supply a learning rate. Documentation for the TF implementation of Adagrad can be found here: [https://www.tensorflow.org/api_docs/python/tf/train/AdagradOptimizer](https://www.tensorflow.org/api_docs/python/tf/train/AdagradOptimizer) See [https://en.wikipedia.org/wiki/Stochastic_gradient_descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent) for more information about stochastic gradient descent and the Adagrad algorithm.

7. **Evaluating probit regression on test data.** Load the test data from `probit.zip`. What is the negative log-likelihood of your probit model on this test data? That is, what is the negative log-likelihood when you use your estimated parameters with the previously unseen test data?
8. **Evaluating the estimated probit parameter.** The probit data was, in reality, generated with true parameter

\[ W = \frac{1}{2}(2, -3, 5, -7, 11). \]

Write a TensorFlow expression to compute the squared error between your estimated parameters and their true values. What is the squared error? **Note:** again, you need only evaluate the error of your final estimates, not at every step.

### 3 Building a Complicated Model (5 points)

The TensorFlow documentation includes tutorials on building a number of more complicated neural models in TensorFlow: [https://www.tensorflow.org/tutorials/](https://www.tensorflow.org/tutorials/). Choose one of these tutorials (except for the GPU tutorial and the two numerical computing tutorials on the Mandelbrot set and PDEs) and follow it. Some of the tutorials include instructions along the lines of “We didn’t discuss this trick, try adding it!” You do not need to do any of these additional steps (though you will certainly learn something if you do!). **Warning:** some of the tutorials require large amounts of training data. If this is the case, please do not include the training data in your submission! Instead, include a line of code to download the data from wherever it is stored. Also, some of the tutorials require especially long training time, so budget your time accordingly!

Your submission for this problem should include code that loads the training and test data, builds and trains a model, and evaluates that model on test data. That is, your code should perform all the training and testing steps performed in the tutorial, but without having to be run from the command line. Depending on which model you choose, training may take a long time if you use the preset number of training steps, so be sure to include a variable called `nsteps` that controls the number of training steps.

**Note:** it will not be enough to simply copy the tutorial’s python code into your jupyter notebook, since the demo code supplied in the tutorials is meant to be run from the command line.

**Another note:** If it was not clear, you are, for this problem and this problem only, permitted to copy-paste code from the TensorFlow tutorials as much as you like without penalty.