Homework 10: Google TensorFlow

Due December December 20, 10:00 am

Worth 30 points

Warning: Three things to bring to your attention:

- 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 20th. Any work turned in after 10:00:am on December 20th will be considered late and will receive a grade of 0. **Note the non-standard deadline time!**
- The last problem of this homework will ask you to do a few simple things in TensorFlow on Google's computing service, Google Cloud Platform (GCP). This part of the assignment is meant to challenge you to learn a new tool from scratch by reading documentation and following its tutorials. This is tough, by design. Start early so that you have plenty of time to grapple with the material.
- As discussed in lecture, Google TensorFlow is now on version 2.0, while Google Cloud Platform (GCP) is still largely built around TensorFlow version 1.X. Thus, we are in the unfortunate situation of having to choose between learning the latest version of TensorFlow but being unable to run it on GCP or running a slightly older version of TensorFlow on GCP. It is my opinion that the latter is preferable, since getting more experience with distributed and cloud-based computing is much more useful for your careers than learning a particular version of TensorFlow (which is bound to change yet again in a couple of years, anyway). The upshot for you is that you should take care when reading the TensorFlow documentation that you are reading the docs for TF version 1, not version 2.

Instructions on writing and submitting your homework can be found at `http://www-personal.umich.edu/~klevin/teaching/Fall2019/STATS507/hw_instructions.html`. Failure to follow these instructions will result in lost points. Please direct any questions to either the instructor or your GSI.

1 Warmup: Constructing a 3-tensor (2 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.

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

Figure 1: The TensorFlow logo.

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.

2 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 $\sigma(z) = (1 + \exp(-z))^{-1}$ 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 negative log-likelihood term, summed over all the observations.

2. **Estimating parameters in logistic regression.** The zip file at `http://www-personal.umich.edu/~klevin/teaching/Fall2019/STATS507/HW10_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. Please include these files in your submission so that we can run your code without downloading them again. **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
$$W = (1, 1, 2, 3, 5, 8), \qquad b = -1.$$
Write TensorFlow expressions to compute the squared error between your estimated parameters and their true values. Evaluate the error in recovering W and b separately. What are the squared errors of these estimates? **Note:** you need only evaluate the error of your final estimates, not at every step.

5. For ease of grading, please make the variables from the above problems available in a dictionary called `results_logistic`. The dictionary should have keys `'W'`, `'Wsqerr'`, `'b'`, `'bsqerr'`, `'log_lik_test'` , with respective values `sess.run(x)` where `x` ranges over the corresponding quantities. For example, if my squared error for W is stored in a TF variable called `W_squared_error`, then the key `'Wsqerr'` should have value `sess.run(W_squared_error)`.

6. **Classification of normally distributed data.** The .zip file at `http://www-personal.umich.edu/~klevin/teaching/Fall2019/STATS507/HW10_normal.zip` contains four Numpy `.npy` files that contain train and test data generated from $K = 3$

different classes. Each class $k \in \{1, 2, 3\}$ has an associated mean $\mu_k \in R$ and variance $\sigma_k^2 \in \mathbb{R}$, and all observations from a given class are i.i.d. $\mathcal{N}(\mu_k, \sigma_k^2)$. The four files are:

- `normal_xtest.npy` : contains a 500-vector whose entries are the independent variables (predictors) from the test set.
- `normal_xtrain.npy` : contains a 2000-vector whose entries are the independent variables (predictors) from the train set.
- `normal_ytest.npy` : contains a 500-by-3 dimensional matrix whose rows are one-hot encodings of the class labels for the test set.
- `normal_ytrain.npy` : contains a 2000-by-3 dimensional matrix whose rows are one-hot encodings of the class labels for the train set.

The i-th entry of the vector in `normal_xtrain.npy` is the observed random variable from class with label given by the i-th row of the matrix in `normal_ytrain.npy`, and analogously for the two test set files. Please include these files in your submission so that we can run your code without downloading them again.

Load the training data and use it to obtain estimates of the vector of class means $\mu = (\mu_0, \mu_1, \mu_2)$ and variances $\sigma^2 = (\sigma_0^2, \sigma_1^2, \sigma_2^2)$ by minimizing the cross-entropy between the estimated normals and the one-hot encodings of the class labels (as we did in our softmax regression example in class). Please name the corresponding variables `mu` and `sigma2`. 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/versions/r1.15/api_docs/python/tf/train/AdagradOptimizer`. See `https://en.wikipedia.org/wiki/Stochastic_gradient_descent` for more information about stochastic gradient descent and the Adagrad algorithm.

Note: you'll no longer be able to use the built-in logit cross-entropy that we used for training our models in lecture. Your cross-entropy for one observation should now look something like $-\sum_k y'_k \log p_k$, where y' is the one-hot encoded vector and p is the vector whose k-th entry is the (estimated) probability of the k-th observation given its class. **Another note:** do not include any estimation of the mixing coefficients (i.e., the class priors) in your model. You only need to estimate three means and three variances, because we are building a *discriminative* model in this problem.

7. **Evaluating loss on test data.** Load the test data. What is the cross-entropy of your model on this test data? That is, what is the cross-entropy when you use your estimated parameters with the previously unseen test data?

8. **Evaluating parameter estimation on test data.** The true parameter values for the three classes were

$$\mu_0 = -1, \sigma_0^2 = 0.5$$
$$\mu_1 = 0, \sigma_1^2 = 1$$
$$\mu_2 = 3, \sigma_2^2 = 1.5.$$

Write a TensorFlow expression to compute the total squared error (i.e., summed over the six parameters) between your estimates and their true values. What is the squared error? **Note:** you need only evaluate the error of your final estimates, not at every step.

9. **Evaluating classification error on test data.** Write and evaluate a TensorFlow expression that computes the classification error of your estimated model averaged over the test data.

10. Again, for ease of grading, define a dictionary called `results_class`, with keys `'mu', 'sigma2', 'crossent_test', 'class_error'` with keys corresponding to the evaluation (again using `sess.run`) of your estimate of μ, σ^2, the cross-entropy on the test set, and the classification error from the previous problem.

3 Running Models on Google Cloud Platform (18 points)

In this problem, you'll get a bit of experience running TensorFlow jobs on Google Cloud Platform (GCP), Google's cloud computing service. Google has provided us with a grant, which will provide each of you with free compute time on GCP.

Important: this problem is **very hard**. It involves a sequence of fairly complicated operations in GCP. As such, I do not expect every student to complete it. Don't worry about that. Unless you've done a lot of programming in the past, this problem is likely your first foray into learning a new tool largely from scratch instead of having my lectures to guide you. The ability to do this is a crucial one for any data scientist, so consider this a learning opportunity (and a sort of miniature final exam). Start early, read the documentation carefully, and come to office hours if you're having trouble.

Good luck, and have fun!

The first thing you should do is claim your share of the grant money by visiting this link: `https://google.secure.force.com/GCPEDU?cid=HVe78B8tfw3bHEOFuVu%2BonEq8XNxlOpwDwEKGhM` `2Ff%2FXp6oxXcpqo6Jf15frWW` You will need to supply your name and your UMich email. Please use the email address associated to your unique name (i.e., `uniqname@umich.edu`), so that we can easily determine which account belongs to which student. Once you have submitted this form, you will receive a confirmation email through which you can claim your compute credits. These credits are valid on GCP until they expire in September 2020. Any credits left over after completing this homework are yours to use as you wish. Make sure that you claim your credits while signed in under your University of Michigan email, rather than a personal gmail account so that your project is correctly associated with your UMich email. If you accidentally claim the credits under a different address, add your unique name email as an owner.

Once you have claimed your credits, you should create a project, which will serve as a repository for your work on this problem. You should name your project `uniqname-stats507f19`, where `uniqname` is your unique name in all lower-case letters. Your project's billing should be automatically linked to your credits, but you can verify this fact in the billing section dashboard in the GCP browser console. Please add me (UMID `klevin`) as well as your GSIs Roger Fan (UMID `rogerfan`) and Su I Iao (UMID `iaosui`) as owners. You can do this in the IAM tab of the IAM & admin dashboard by clicking "Add" near the top of the page, and listing our UMich emails and specifying our Roles as Project → Owner.

Note: this problem is comparatively complicated, and involves a lot of moving parts. At the end of this problem (several pages below), I have included a list of all the files that should be included in your submission for this problem, as well as a list of what should be on your GCP project upon submission.

Important: after the deadline (December 20th at 10:00am) you **should not** edit your GCP project in any way until you receive a grade for the assignment in canvas. If your project indicates that any files or running processes have been altered after the deadline by a user other than `klevin`, `rogerfan` or `iaosui` we will assume this to be an instance of editing your assignment after the deadline, and you will receive a penalty.

1. Follow the tutorial at `https://cloud.google.com/ml-engine/docs/tensorflow/getting-started-training-prediction` which will walk you through the process of training a deep neural net similar to the one we saw in class, but this time using resources on GCP instead of your own machine.

   A few notes:

   - You may have already completed some parts of the first few steps of the tutorial in the process of claiming your credits. Some steps of the tutorial have instructions for either MacOS and Cloud Shell. You should follow the cloud shell instructions, *even if* you are running Mac OSX. Similarly, you may ignore any discussion of the *Cloud SDK*. SDK stands for "Software Development Kit". The Cloud SDK is a collection of tools for building programs that interact with Google Cloud. There's lots of cool stuff you can do with this, but it is outside the scope of this assignment (of course, you are highly encouraged to check it out, if you wish to learn more). Note, however, that there is a section titled "Verify the Google Cloud SDK components" that you should still run, despite the fact that Google Cloud SDK is in the name.
   - The section titled "Develop and validate your training application locally" refers to running a GCP training job in such a way as to avoid any charges for storage, compute time, etc. This is a good way to make sure that everything is running properly before you start paying for things. Note that the reference to "locally" may be a bit confusing. Usually, we mean "local" to refer to the machine that we are sitting at. In this case, "local" means the computer that is running your Google Cloud console (that you access through a browser), as opposed to the computers that GCP offers for rent.
   - When it comes time to train a model on a non-local GCP machine, the first thing we need to do is set up a *bucket*, which is the Google Cloud term for what is essentially a storage drive. This is done in the section titled "Set up your Cloud Storage bucket". **Important:** Please follow that section's suggestion to set your bucket name as `PROJECT_ID=$(gcloud config list project --format "value(core.project)") BUCKET_NAME=${PROJECT_ID}-mlengine` When it comes time to grade this problem of the assignment, we will look for a bucket by this name, and look for a trained model in that bucket, so please make sure that your bucket is named correctly. Provided that you have set your project name as suggested above, then the above two commands should suffice. The remainder of the section titled "Set up your Cloud Storage bucket" will walk you through moving the requisite files from your local (i.e., Cloud Shell) machine to the bucket. This is roughly analogous to moving a file from your laptop to an external harddrive.
   - Pay special attention to the section titled "Deploy a model to support prediction". That section includes ideas that you will need later in this assignment. One possible stumbling block in that section is in step 3, where you should be

careful that your `OUTPUT_PATH` variable points to a directory in your bucket where you actually wrote job output.

- If trying to launch a new job gets an error "A job with this id already exists.", you'll need to launch the job with a different name by supplying a different job name to the `gcloud ai-platform jobs submit training <job name>` command.

Important: the tutorial will tell you to tear your storage down at the end. Do not do that. Leave that up so that we can verify that you set things up correctly. It should cost at most a dollar or two to leave your storage buckets running, but if you wish to conserve your credits, you can tear everything down and go through the tutorial again on the evening of December 19th or the (early!) morning of December 20th.

2. Let us return to the classifier that you trained above on the normally-distributed data. In this and the next several subproblems, we will take an adaptation of that model and upload it to GCP where it will serve as a prediction node similar to the one you built in the tutorial above. Train the same classifier on the same training data, but this time, save the resulting trained model in a directory called `normal_trained`. You'll want to use the `tf.saved_model.simple_save` function. Refer to the GCP documentation at

   `https://cloud.google.com/ml-engine/docs/deploying-models`,

   and the documentation on the `tf.saved_model.simple_save` function, here: `https://github.com/tensorflow/docs/blob/master/site/en/r1/guide/saved_model.md` Please include a copy of this model directory in your submission. **Hint:** a stumbling block in this problem is figuring out what to supply as the inputs and outputs arguments to the `simple_save` function. Your arguments should look something like `inputs = {'x':x}, outputs = {'prediction':prediction}`.

3. Let's upload that model to GCP. First, we need somewhere to put your model. You already set up a bucket in the tutorial, but let's build a separate one. Create a new bucket called `uniqname-stats507f19-hw10-normal`, where `uniqname` is your uniqname. You should be able to do this by making minor changes to the commands you ran in the tutorial, or by following the instructions at `https://cloud.google.com/solutions/running-distributed-tensorflow-on-compute-engine#creating_a_cloud_storage_bucket`. Now, we need to upload your saved model to this bucket. There are several ways to do this, but the easiest is to follow the instructions at `https://cloud.google.com/storage/docs/uploading-objects` and upload your model through the GUI. **Optional challenge (worth no extra points, just bragging rights):** Instead of using the GUI, download and install the Google Cloud SDK, available at `https://cloud.google.com/sdk/` and use the `gsutil` command line tool to upload your model to a storage bucket.

4. Now we need to create a *version* of your model. Versions are how the GCP machine learning tools organize different instances of the same model (e.g., the same model trained on two different data sets). To do this, follow the instructions located at `https://cloud.google.com/ml-engine/docs/deploying-models#creating_a_model_version`, which will ask you to

- Upload a SavedModel directory (which you just did)
- Create a Cloud ML Engine model resource
- Create a Cloud ML Engine version resource (this specifies where your model is stored, among other information)
- Enable the appropriate permissions on your account.

Please name your model `stats507f19_hw10_normal` (note the underscores here as opposed to the hyphens in the bucket name and note that this model name should not include your uniqname; see the documentation for the `gcloud ml-engine versions` command for how to delete versions, if need be). **Important:** there are a number of pitfalls that you may encounter here, which I want to warn you about: A good way to check that your model resource and version are set up correctly is to run the command `gcloud ml-engine versions describe "your_version_name" --model "your_model_name"`. The resulting output should include a line reading `state: READY`. You may notice that the Python version for the model appears as, say, `pythonVersion: '2.7'`, even though you used, say, Python 3.6. This should not be a problem, but you **should** make sure that the `runtimeVersion` is set correctly. If the line `runtimeVersion: '1.0'` is appearing when you describe your version, you are likely headed for a bug. You can prevent this bug by adding the flag `--runtime-version 1.14` to your `gcloud ml-engine versions create` command, and making sure that you are running TensorFlow version 1.14 on your local machine (i.e., the machine where you're running Jupyter). Running other 1.X versions locally while running 1.14 on GCP also seems to work fine.

5. Create a .json file corresponding to a single prediction instance on the input observation $x = 4$. Name this .json file `instance.hw10.json`, and please include a copy of it in your submission. **Hint:** you will likely find it easiest to use nano/vim/emacs to edit edit the .json file from the tutorial (GCP Cloud Shell has versions of all three of these editors). Doing this will allow you to edit a copy of the .json file directly in the GCP shell instead of going through the trouble of repeatedly downloading and uploading files. Being proficient with a shell-based text editor is also, generally speaking, a good skill for a data scientist to have.

6. Okay, it's time to make a prediction. Follow the instructions at `https://cloud.google.com/ml-engine/docs/online-predict#requesting_predictions` to submit the observation in your .json file to your running model. Your model will make a prediction, and print the output of the model to the screen. Please include a copy-paste of the command you ran to request this prediction as well as the resulting output in your jupyter notebook. Which cluster does your model think $x = 4$ came from? **Hint:** if you are getting errors about dimensions being wrong, make sure that your instance has the correct dimension expected by your model. **Second hint:** if you are encountering an error along the lines of `Error during model execution: AbortionError(code=StatusCode.INVALID_ARGUMENT, details=\"NodeDef mentions attr 'output_type'`, this is an indication that either (a) there is a mismatch between the version of TensorFlow that you used to create your model and the one that you are running on GCP or (b) your .json file is formatted incorrectly. See the discussion of `gcloud ml-engine versions create` above.

That's all of it! Great work! Here is a list of all files that should be included for this problem in your submission, as well as a list of what processes or resources should be left running in your GCP project:

- You should leave the model and storage bucket from your GCP census classicier (i.e., the storage bucket with saved models in it) from the GCP ML tutorial running in your GCP project.
- Include in your submission a copy of your saved model directory constructed from your classifier. You should also have a copy of this directory in a storage bucket on GCP.
- Leave a storage bucket running on GCP containing your uploaded model directory. This storage bucket should contain a model with a single version.
- Include in your submission a `.json` file representing a single observation. You need not include a copy of this file in a storage bucket on GCP; it will be stored by default in your GCP home directory if you created it in a text editor in the GCP shell.
- Include in your jupyter notebook a copy-paste of the command you ran to request your model's prediction on the .json file, and please include the output that was printed to the screen in response to that prediction request. **Note:** Please make sure that the cell(s) that you copy-paste into is/are set to be Raw NBconvert cell(s), so that your commands display as code but are not run as code by Jupyter.