CodaLab and Submission Instructions

CS 224N: Programming Assignment #4

February 28, 2017

CodaLab Office Hours: Monday 1–4PM Gates AI Lounge (2nd floor, outside room 251)

Contents

1 Introduction

We will be accepting and evaluating your submissions on CodaLab, an online platform for computational research built by Percy Liang and his team. With CodaLab, you can run your jobs on a cluster, document and share your experiments, all while keeping track of full provenance, so you can be a more efficient researcher. For the purposes of this assignment, using CodaLab to manage your experiments is optional, but you will need to use CodaLab to submit your models for evaluation.

To learn more about what CodaLab is and how it works, check out the CodaLab wiki at `https://github.com/codalab/codalab-worksheets/wiki`.

2 Set up

Visit `https://worksheets.codalab.org/` to sign up for an account on CodaLab.[1] It is possible to use CodaLab entirely from the browser, and in fact the web interface provides a great view of your data and experiments. However, we also recommend installing the command-line interface (CLI) on your machine to make uploading your submission easier:

```
pip install codalab-cli
```

You should now be able to use the `cl` command. Go ahead and create the "worksheet" where you will place all of your code and data, and ensure it has the correct permissions in preparation for submission. Make sure to replace `GROUPNAME` with your group name in the following commands. Worksheets have a global namespace, so this will help avoid naming collisions.

[1] Note that your name and username on this account will be public to the world; you are responsible for your own privacy here.

```
cl work main::                          # connect and log in with your account
cl new cs224n-GROUPNAME                 # create a new worksheet
cl work cs224n-GROUPNAME                # switch to your new worksheet
cl wperm . public none                  # make your worksheet private (IMPORTANT)
cl wperm . cs224n-win17-staff read      # give us read access (IMPORTANT)
```

Since you are working in groups, you can create a group on CodaLab, add each of your members to it, then give them all full access to the worksheet.

```
cl gnew cs224n-GROUPNAME            # create the group
cl uadd janedoe cs224n-GROUPNAME    # add janedoe as a member
cl uadd marymajor cs224-GROUPNAME   # add marymajor as a member
# Give the group full access (i.e. "all") to the worksheet
cl wperm cs224n-GROUPNAME cs224n-GROUPNAME all
```

You can check out the tutorial on the CodaLab Wiki to familiarize yourself with the CLI: https://github.com/codalab/codalab-worksheets/wiki/CLI-Basics.

3 Run your model

Note: We assume here that you have been developing and training your model on your local machine or VM. These instructions go over how to now upload your model, and show us how to run your code for the leaderboards. If you'd like to use more of CodaLab's facilities to managing your experiments from end to end, skip over to Section A.2 first.

Since you don't have access to the test set, you will have to submit your code so that we can run it for you. Of course, the tricky part is that we have to know how to run your code, to which you may have made all sorts of modifications. Thankfully, you just need to upload your code (along with the trained model and any other dependencies) to CodaLab and run `qa_answer.py` on an example dataset, which we will call the "tiny dev set". Our leaderboard script will re-execute that run, substituting the actual test set in for the tiny dev set.

The tiny dev set is available globally on CodaLab, and can be loaded into any of your runs by its UUID 0x4870af2556994b0687a1927fcec66392.

```
cd path/to/assignment4
# Train your model
python code/train.py
# Make sure you're on your private pa4 worksheet
cl work main::cs224n-GROUPNAME
# Upload your latest code, data, and model parameters
cl upload code
cl upload data
cl upload train
# To see your newly uploaded bundles and inspect their contents (you can also
# go to https://worksheets.codalab.org and click on My Dashboard and then
# cs224n-GROUPNAME to see your worksheet).
cl ls
cl cat data
# Run your prediction code: This loads your code, model parameters, data,
# and the tiny dev set into a sandbox directory, inside a container based
# on the sckoo/cs224n-squad:v4 Docker image.
cl run --name run-predict --request-docker-image sckoo/cs224n-squad:v4 \
    :code :data :train dev.json:0x4870af2556994b0687a1927fcec66392 \
    'python code/qa_answer.py --dev_path dev.json'
```

You can check the status and results of the run with one or more of these commands:

```
# Look at the status of the run
cl info --verbose run-predict
# Blocks until the job is complete, while tailing the output
cl wait --tail run-predict
# Inspect the resulting files
cl cat run-predict  # list the files
cl cat run-predict/stderr  # inspect stderr
cl cat run-predict/dev-prediction.json  # read specific file
```

You may need to modify some of the commands above, in particular the run command, depending on how you built your model. If you built your own Docker image[2] for example, just replace `sckoo/cs224n-squad:v4` with the tag of your own image. The most important part is that you create a run of `qa_answer.py` on the tiny dev set, then tag the resulting run so that our leaderboard script knows what to look for.

4 Submit your model

Submitting your project for the leaderboards will simply involve tagging your prediction run-bundles with the appropriate tag. Our leaderboard script will then be able to find your bundle, re-run it with the corresponding dataset.

To make sure everything works, you can test your submission on the sanity-check leaderboard, which just re-runs your prediction run-bundle on the tiny dev set again. Just tag the bundle with `cs224n-win17-submit-sanity-check`.

```
cl edit run-predict -T cs224n-win17-submit-sanity-check
```

After you submit, go to the leaderboard to check out your results. (We will release the leaderboard URL on Piazza when it's ready.)

The tags for the other leaderboards will be made available on Piazza when they are ready. A leaderboard may be throttled (e.g. the final leaderboard for the test set will only allow one submission total), so make sure to that everything worked as expected with the sanity-check before you submit to other leaderboards.

If you have any problems, submit a post on Piazza and tag it with "hw4". Make sure to include the UUID of the bundle you're having problems with (e.g., `0x4870af2556994b0687a1927fcec66392`), or else we won't be able to help you. An even better option is to come to CodaLab office hours, which are normally held Mondays from 1:00PM to 4:00PM in the AI Lounge (the area outside Professor Chris Manning's office on the second floor of Gates).

A Appendix

A.1 Build a Docker image for your code

Docker images provide a convenient way to package all of the dependencies that you need for running your code. When you submit a job on CodaLab, you can specify a Docker image to use. CodaLab will download that Docker image, then start a new Docker container based on that image, and execute your code inside the new container. We've already built a basic Docker image for you that contains all the dependencies required by the starter code, including TensorFlow. It is available on DockerHub as `sckoo/cs224n-squad:v4`. You can find the specification for the image in your starter code at `assignment4/code/docker/Dockerfile`. A Dockerfile is a simple text-based specification that describes how a Docker image should be built.[3]

However, as your model increases in complexity, you may want to include other dependencies in the image, so that we can still run your code. If you want to learn more about Docker and have more control over how your images are built, we recommend modifying the Dockerfile we gave you and building your own images from scratch. Otherwise, CodaLab has some basic facilities to let you easily build and test your own images.

First, you will need to install Docker. Docker runs natively on Linux (installation instructions: https://docs.docker.com/engine/installation/linux/), but they have recently released an

[2] See Section A.1

[3] You can learn more about how Docker images and Dockerfiles work at https://www.digitalocean.com/community/tutorials/docker-explained-using-dockerfiles-to-automate-building-of-images

excellent set of tools called Docker for Mac that runs a Linux kernel in the native macOS hypervisor (installation instructions: https://docs.docker.com/docker-for-mac/install/), if you're developing on a Mac for some reason.

Second, you will need head over to https://hub.docker.com/ and create a DockerHub account. DockerHub, as its name suggests, is a public repository for Docker images. When you "push" an image that you've created onto DockerHub, it can now be shared and used by other people, such as the CodaLab servers.

Now let's build our image, based off of the `sckoo/cs224n-squad:v4` image we gave you. We will start a container with that image, loading in your code, data, and model directories into the container.

```
cd path/to/assignment4
docker pull sckoo/cs224n-squad:v4  # download the image
cl edit-image --request-docker-image sckoo/cs224n-squad:v4 :code :data :train
```

You will now be given a Bash shell inside the new container. You can poke around it to see that your files have been loaded into the working directory. Now let's try running your code inside the container.

```
====
Entering container 701c0bfa
Once you are happy with the changes, please exit the container (ctrl-D)
and commit your changes to a new image by running:

        cl commit-image 701c0bfa [image-tag]

====
root@701c0bfa8a9d:~# python code/qa_answer.py
Traceback (most recent call last):
  File "dev/qa_answer.py", line 30, in <module>
    import marshmallow
ImportError: No module named marshmallow
```

If anything fails due to a missing dependency, go ahead and just install inside the container, whether it's with `apt-get` or `pip`.

```
root@701c0bfa8a9d:~# pip install marshmallow
Collecting marshmallow
  Downloading marshmallow-2.13.0-py2.py3-none-any.whl (45kB)
    100% |################################| 51kB 499kB/s
Installing collected packages: marshmallow
Successfully installed marshmallow-2.13.0
```

Just repeat this until your code finally works, then exit out of the container.

```
root@701c0bfa8a9d:~# exit
exit
====
Exited from container 701c0bfa
If you are happy with the changes, please commit your changes to a new
image by running:

        cl commit-image 701c0bfa [image-tag]

====
```

Now you can commit your image, tag it with a name of your choice, then push it to Docker Hub.

```
cl commit-image 701c0bfa YOUR_DOCKERHUB_ID/squad:v1
cl push-image YOUR_DOCKERHUB_ID/squad:v1
```

You can now refer to this Docker image in CodaLab by its tag.

A.2 Train your model and manage your experiments on CodaLab

Note: The public CodaLab worker nodes that execute your jobs by default do not have GPUs. As you know, training a deep model involves a huge amount of matrix computation that could take a prohibitive amount of time to complete on CPUs alone. Luckily, you can actually run a worker from your own VMs! If you choose to manage your training on CodaLab, you must set up a worker daemon on your GPU instance by following the instructions at https://github.com/codalab/codalab-worksheets/wiki/Execution#running-your-own-worker.

When you run your programs on CodaLab, it keeps track of full provenance so that you always know how you got any set of results. For example, you can train your model with many different hyperparameter settings all at the same time, then easily compare the results without ever losing track of the hyperparameters that gave you the best score.

First, set things up and upload your code.

```
cd path/to/assignment4
cl work main::cs224n-GROUPNAME  # make sure you're on your private pa4 worksheet
cl upload code
```

Then prepare the preprocessed data.

```
# A) Run data preprocessing on CodaLab, using a globally-available
#    download directory (with UUID 0x2afcbeeda5074afa9606d5139e656d20)
#    that we've already prepared for you on the server.
cl run --name run-preprocess --request-docker-image sckoo/cs224n-squad:v4 \
    :code download:0x2afcbeeda5074afa9606d5139e656d20 \
    'code/get_started.sh'
# B) OR just run the data preprocessing on your own machine, and
#    upload the resulting data directory.
code/get_started.sh
cl upload data
```

Now run your model training!

```
# A) If you ran preprocessing on CodaLab
#    Note that we reference a directory inside the previous run-bundle
cl run --name run-train --request-docker-image sckoo/cs224n-squad:v4 \
    :code data:run-preprocess/data \
    'python code/train.py --epochs 5 --awesomeness 2.7'
# B) Or if you just uploaded your data directory
cl run --name run-train --request-docker-image sckoo/cs224n-squad:v4 \
    :code :data \
    'python code/train.py --epochs 5 --awesomeness 2.7'
```

You can run this last run command multiples times, each with a different set of command line arguments, and they will all be scheduled to run. Then if you visit your worksheet on the website[4] and track the progress of the runs. You can modify the tables to display your hyperparameters as custom columns. By modifying your code to output a TSV or JSON file containing the training loss and accuracy, you can even make your worksheet display plots of your training progress. Learn more about the worksheet markdown syntax at https://github.com/codalab/codalab-worksheets/wiki/Worksheet-Markdown.

There are many other features in CodaLab that are not covered here. You can even build up a pipeline of chained dependent jobs, then easily re-run that pipeline while just substituting one of the components. If you want to learn more, you can find more detailed documentation at https://github.com/codalab/codalab-worksheets/wiki/CLI-Reference.

[4] You can find your worksheet by looking under your worksheets list on your dashboard: https://worksheets.codalab.org/rest/worksheets/?name=dashboard