CodaLab Submission Instructions

CS 224N: Default Final Project

February 18, 2018

Contents

1 Introduction

We will be accepting and evaluating your submissions on CodaLab, an online platform for computational research built at Stanford 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.

For now, the main CodaLab terminology you'll need to understand are:

- **Bundles** are immutable files/directories that represent the code, data, and results of an experimental pipeline. You can upload bundles to CodaLab, that contain your data and/or code. You can also create **run-bundles**, which run some command, dependent on the contents of other bundles (which may contain the necessary code and data). The idea is that these run-bundles can be reproduced exactly.

- **Worksheets** organize and present an experimental pipeline in a comprehensible way, and can be used as a lab notebook, a tutorial, or an executable paper. Once you have created your CodaLab account, you can view your worksheets in-browser at https://worksheets.codalab.org/ (go to My Dashboard).

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 project development machine (the one where you train your final models; most likely your Azure VM) to make uploading your submission easier. Instructions to install the CLI are here: https://github.com/codalab/codalab-worksheets/wiki/CLI-Basics

You should now be able to use the `cl` command. Execute the following commands to create the CodaLab 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 (this can be whatever you like). Worksheets have a global namespace, so your worksheet name `cs224n-GROUPNAME` will need to be unique.

```
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-win18-staff read     # give us read access (IMPORTANT)
```

If you are working in a group, then execute the following commands to create a group on CodaLab, add each of your members to it, and 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 further with the CLI: https://github.com/codalab/codalab-worksheets/wiki/CLI-Basics.

3 Leaderboards

We are hosting three leaderboards on CodaLab, which each display the EM and F1 scores of the submitted models:

- The **sanity check** leaderboard. This evaluates your model on a small subset (810 examples) of the dev set. Evaluation on this subset is fast, so you will use it to debug your CodaLab submission process.
  - CodaLab tag: `cs224n-win18-sanity-check`
  - Leaderboard URL:
    http://cs224n-win18-leaderboard.westus.cloudapp.azure.com/sanity-check.html

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

  - Submission limit: unlimited

- The **dev** leaderboard. This evaluates your model on the official SQuAD dev set (which is also available to you locally as `dev-v1.1.json`). You will use this leaderboard to measure your progress with respect to other teams.
  - CodaLab tag: `cs224n-win18-dev`
  - Leaderboard URL:
    `http://cs224n-win18-leaderboard.westus.cloudapp.azure.com/dev.html`
  - Submission limit: 10 per day
- The **test** leaderboard. This evalutes your model on the official (secret) SQuAD test set. You will use this leaderboard for your final submission.
  - CodaLab tag: `cs224n-win18-test`
  - Leaderboard URL:
    `http://cs224n-win18-leaderboard.westus.cloudapp.azure.com/test.html`
  - Submission limit: 3 total

4 Submitting to CodaLab

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 upload your model and 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, check out the CodaLab wiki at `https://github.com/codalab/codalab-worksheets/wiki`

4.1 Run official eval locally

At this point, we assume that you have a trained model checkpoint saved on your development machine. Before uploading to CodaLab, you should first run the official evaluation pipeline on your development machine.

First, download the sanity check dataset `tiny-dev.json` to the `data` directory:

```
cd cs224n-win18-squad                           # Go to the root of the repository
cl download -o data/tiny-dev.json 0x4870af    # Download the sanity check dataset
```

Now, re-run the command you used to train the model, but replace `--mode=train` with `--mode=official_eval`, and supply new arguments `--json_in_path` and `--ckpt_load_dir`:

```
source activate squad     # Remember to activate your project environment
cd cs224n-win18-squad     # Go to the root of the repository

python code/main.py <OTHER FLAGS> --mode=official_eval \
--json_in_path=data/tiny-dev.json \
--ckpt_load_dir=experiments/<YOUR EXPERIMENT NAME>/best_checkpoint
```

Note: *this command calls the function* `QAModel.get_start_end_pos()` *which is defined in* `qa_model.py`. *If you have edited the code in such a way that* `get_start_end_pos` *function no longer works, then you will need to fix it before you can proceed.*

If successful, the above command loads a model checkpoint from file (specified by `--ckpt_load_dir`), reads a SQuAD data file in JSON format (specified by `--json_in_path`), generates answers for the (context, question) pairs inside, and writes those answers to another JSON file (specified by `--json_out_path`, which defaults to `predictions.json`).

If everything goes smoothly, you should now see a file `predictions.json` in the `cs224n-win18-squad` directory. Inside, you should see a mapping from unique ids (like `57277373dd62a815002e9d28`) to SQuAD answers (text).

Next, run the official SQuAD evaluation script (which can be found at `code/evaluate.py`) on your output:

```
python code/evaluate.py data/tiny-dev.json predictions.json
```

After a few seconds you should see a printout with your F1 and EM scores on the sanity check dataset:

```
{"f1": 39.45802615545679, "exact_match": 32.96296296296296}
```

If you wish, you can repeat this process to evaluate your model on the entire dev set `dev-v1.1.json` instead of `tiny-dev.json`. Note: these F1 and EM scores may be different to what you can see in TensorBoard; see FAQ A.2.2.

4.2 Run official eval on CodaLab

Once you've successfully run offical eval locally, it's time to upload your code and model to CodaLab.

```
cd cs224n-win18-squad           # Go to the root of the repository
cl work main::cs224n-GROUPNAME  # Ensure you're on your project worksheet

# Upload your latest code
cl upload code

# Upload your best checkpoint
cl upload experiments/<YOUR EXPERIMENT NAME>/best_checkpoint
```

Note: ***do not*** *upload the whole* `experiments` *directory, or the entire* `<YOUR EXPERIMENT NAME>` *directory. These contain large files (for example, the TensorBoard logging files) that we do not want to upload to CodaLab. Similarly, you do not need to upload the* `data` *directory unless you have created some new data files. All the data required for the baseline model is already available on CodaLab; see the explanation of the* `cl run` *command for more information.*

To see your newly uploaded bundles and inspect their contents, you can go to `https://worksheets.codalab.org`, click on My Dashboard, then `cs224n-GROUPNAME`, to see your worksheet. Alternatively you can run the following commands:

```
cl ls                    # See your bundles
cl cat code              # Look inside your uploaded code directory
cl cat best_checkpoint   # Look inside your uploaded checkpoint directory
```

Now you will run your official eval command again, but on CodaLab. Make sure to include the other flags you used where it says `<OTHER FLAGS>`:

```
cl run --name gen-answers --request-docker-image abisee/cs224n-dfp:v4 \
:code :best_checkpoint glove.txt:0x97c870/glove.6B.100d.txt data.json:0x4870af \
'python code/main.py <OTHER FLAGS> --mode=official_eval \
--glove_path=glove.txt --json_in_path=data.json --ckpt_load_dir=best_checkpoint'
```

Let's break down this command to understand it:

- `cl run`: Runs the command in your CodaLab worksheet.
- `--name gen-answers`: A tag for this run-bundle.

- `--request-docker-image abisee/cs224n-dfp:v4`: Loads a Docker image that includes all the dependencies required for the baseline code. If you edited the code to require new dependencies, then you may need to create your own Docker image to run on CodaLab – see section A.1. Once you've built your own Docker image, you should replace `abisee/cs224n-dfp:v4` with the tag of your own image.
- `:code :best_checkpoint`: Give access to the directories you uploaded.
- `glove.txt:0x97c870/glove.6B.100d.txt`: Maps the string `glove.txt` to the copy of the 100-dimensional GLoVE word embeddings file stored on CodaLab. The directory with the UUID `0x97c870` contains all the GLoVE files that you have in your `data` directory[2]. If your model uses a different GLoVE dimensionality, change this part accordingly.
- `data.json:0x4870af`: Maps the string `data.json` to the copy of the sanity check dev set stored on CodaLab, which is accessed via its UUID `0x4870af`. This means that CodaLab will run your command using the sanity check dataset as `json_in_path`. If you wanted to use the dev set (`dev-v1.1.json`) instead, use the UUID `0x8f29fe`.
- `'python code/main.py <OTHER FLAGS> --mode=official_eval --glove_path=glove.txt --json_in_path=data.json --ckpt_load_dir=best_checkpoint'`: This is the same command you ran before, but we supply mappings to tell CodaLab where to find the GloVe embeddings, the JSON input file, and the model checkpoint.

Once you have run this command, 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 gen-answers

# Blocks until the job is complete, while tailing the output
cl wait --tail gen-answers

# Inspect the resulting files
cl cat gen-answers                     # List the files
cl cat gen-answers/stderr              # Inspect stderr
cl cat gen-answers/stdout              # Inspect stdout
cl cat gen-answers/predictions.json    # Inspect specific file
```

The `cl info` command shows the `state` of your run. For a while (potentially several minutes), this will say `running`, then when finished, either `failed` or `ready`. If it says `failed`, you can see why by looking at `stderr`[3]. Once `state` shows `ready` (i.e. succesfully finished), you should be able to see and inspect the completed `predictions.json` file.

As an extra sanity check, you may wish to run the `evaluate.py` script again on CodaLab, to check that the EM and F1 scores match what you got locally (if they don't match, see FAQ A.2.3). To do this, run:

```
cl run --name run-eval --request-docker-image abisee/cs224n-dfp:v4 \
:code data.json:0x4870af preds.json:gen-answers/predictions.json \
'python code/evaluate.py data.json preds.json'

# Look at the status of the run
cl info --verbose run-eval

# Once the above command shows state 'ready', inspect stdout.
# You should see the same numbers you saw when you ran locally.
cl cat run-eval/stdout
```

[2]There are even more word vector resources available in this CodaLab worksheet
https://worksheets.codalab.org/worksheets/0xc946dfbd2215486493672a5e5b0c88d8/

[3]You may see a `FutureWarning` message in `stderr`. This is expected: see FAQ A.2.1

If you wish, you can repeat the process described in this section (4.2) to evaluate your model on the entire dev set on CodaLab. To do so, just use the UUID `0x8f29fe` for `dev-v1.1.json` instead of the UUID `0x4870af` for the sanity check dataset. However, note that you won't be able to submit that run-bundle to the leaderboards – all run-bundles submitted to leaderboards need to have run on the *sanity check dataset*. This is because the leaderboards work by replacing the UUID for the sanity check dataset with the UUID for the appropriate dataset.

4.3 Submitting your model to the leaderboards

You're almost there! The last thing you need to do is to re-tag your `gen-answers` run-bundle (that generated answers for the sanity check dataset) with one of the leaderboard tags. Our leaderboard script will then find your bundle, re-run the command, replacing the UUID for the sanity check dataset with the UUID for the appropriate dataset, and display the results on the leaderboard.

Run this command to submit your bundle to the sanity check leaderboard:

```
cl edit gen-answers -T cs224n-win18-sanity-check --description "description here"
```

Note: the description is optional, and if supplied will appear publicly on the leaderboard. However, we advise you to enter a unique description, which will help you to keep track of your various submissions.

After doing this, go to the leaderboard URL[4] to check your results. Within a minute or two, you should see an entry under your username with the status `running` (if you see nothing, see FAQ A.2.4). Some time later (this should only take a few minutes, but may take longer when there is high demand), you should see either `success` status (with your F1 and EM score) or `failed` status. If you're seeing `failed`, see section A.2.5. In the case of success, the F1 and EM scores should exactly match what you got when you ran locally and on your own CodaLab worksheet (if they don't, see FAQ A.2.6).

Once you've successfully submitted to the sanity-check leaderboard, you can submit to the dev and test leaderboards by re-tagging the same bundle with the appropriate tags (see section 3). Your score on the full dev set should be similar to your score on the sanity check dataset (plus or minus a few percent). If it's much lower, see FAQ A.2.7.

Remember that because we choose our hyperparameters based on the dev set, it's possible to overfit to the dev set. For that reason and others, your score on the test set may be lower than on the dev set. If it's a lot lower, see A.2.8. Remember that submissions to the dev and test leaderboards are limited (see section 3).

[4] `http://cs224n-win18-leaderboard.westus.cloudapp.azure.com/sanity-check.html`

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 baseline code, including TensorFlow. It is available on DockerHub as `abisee/cs224n-dfp:v4`. You can find the specification for the Docker image in the project code repository at `cs224n-win18-squad/docker/Dockerfile`. A Dockerfile is a simple text-based specification that describes how a Docker image should be built.[5]

However, you may want to include other dependencies in your code, in which case you'll need a Docker image that includes those dependencies. 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 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 to 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 on the `abisee/cs224n-dfp:v4` image we gave you. We will start a container with that image, loading your code and checkpoint into the container.

```
cd cs224n-win18-squad             # Go to the root of the repository
docker pull abisee/cs224n-dfp:v4  # Download the Docker image

# Start the container
cl edit-image --request-docker-image abisee/cs224n-dfp:v4 :code :best_checkpoint
```

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/main.py
Traceback (most recent call last):
  File "code/main.py", line 30, in <module>
    import marshmallow
ImportError: No module named marshmallow
```

[5]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

If anything fails due to a missing dependency, just install inside the container, e.g. using `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
```

Repeat this process 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 Docker image, tag it with a name of your choice, and push it to Docker Hub.

```
cl commit-image 701c0bfa <YOUR DOCKERHUB ID>/cs224n-dfp:v1
cl push-image <YOUR DOCKERHUB ID>/cs224n-dfp:v1
```

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

A.2 FAQs

A.2.1 Unexpected warning message in stderr on CodaLab

When running TensorFlow using the `abisee/cs224n-dfp:v4` Docker image, you may see the following warning message in `stderr`:

```
/usr/local/lib/python2.7/dist-packages/h5py/__init__.py:36:
FutureWarning: Conversion of the second argument of issubdtype from `float` to
`np.floating` is deprecated. In future, it will be treated as
`np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

Ignore this; it is just a warning about how the package `h5py` is using `numpy` notation that will soon be deprecated. Unfortunately this is a consequence of building our Docker image based on the official Tensorflow 1.4.1 Docker image. See here for more details: `https://github.com/h5py/h5py/issues/961`.

A.2.2 Why are the F1/EM scores from `official_eval` mode different to those shown in TensorBoard?

The F1/EM scores shown on Tensorboard are computed by the `QAModel.check_f1_em()` function defined in `qa_model.py`. Read the header of that function to see an explanation of why the scores are different. Probably the biggest difference is that the `official_eval` mode compares your predicted answer to *all three* gold answers provided in the dataset, and awards the maximum F1/EM score across the three. This will most likely give you *better* F1/EM scores in `official_eval` mode than in TensorBoard.

A.2.3 The F1/EM scores I get on my CodaLab worksheet are different to what I get locally

Have you uploaded your latest code and latest checkpoint to CodaLab? Make sure that the `ckpt_load_dir` you supplied when running `official_eval` mode locally matches the `best_checkpoint` directory you uploaded to CodaLab, and that when you ran the `official_eval` command on CodaLab, you supplied that directory as the `ckpt_load_dir`. You can run `cl cat best_checkpoint` to check that your uploaded checkpoint files have the same name (i.e. training iteration number) as the checkpoint you're using locally. Lastly, you can always visit your project worksheet in-browser at `https://worksheets.codalab.org/`. This allows you to click on a particular run-bundle and view which exact dependencies it used.

A.2.4 My submission isn't showing up on the leaderboard

Are you sure you entered the leaderboard tag correctly? Are you sure you entered the run-bundle name correctly? Remember that the run-bundle you submit needs to be one that generates answers for the *sanity check dataset*, using its CodaLab UUID. Were you logged onto your CodaLab worksheet? Were there any error messages when you ran the `cl edit` command? Try opening your worksheet in your browser, clicking on the run-bundle you tried to submit, and checking that it does have the correct tag. You may need to wait a while for your submission to appear on the leaderboard, especially if many students are submitting to the leaderboard at the same time.

Note: the leaderboards periodically scan CodaLab for all run-bundles tagged with the leaderboard's tag. For each user, the latest run-bundle with the appropriate tag is selected. (Note: here 'latest' means the most recently created run-bundle, not the most recently tagged run-bundle). If that run-bundle has not been evaluated by the leaderboard yet, and is more recent than the user's latest submission on the leaderboard, then the leaderboard evaluates the new run-bundle. If your run-bundle isn't showing up, it may be because there's a more recently-created run-bundle that you've tagged with the leaderboard's tag. If in doubt, just create a new run-bundle!

If you're still having difficulty, post on Piazza with the tag `default_project`, and include your CodaLab username and the UUID of the run-bundle you tried to submit. You can find the UUID of your run-bundle via the web interface, or via the `cl info --verbose <BUNDLE NAME>` command.

A.2.5 My leaderboard submission shows 'failed'

Did you step through all the instructions in section 4.2, including running the `evaluate.py` script in your CodaLab worksheet? Ensure that all of those commands worked correctly. If you're having trouble submitting to the dev set, step through these instructions using the dev set instead of the sanity check set (see A.2.7). Make sure that you are able to step through these instructions without any problems, then submit the exact same run-bundle to the leaderboard.

If you're still having difficulty, post on Piazza with the tag `default_project`, and include your CodaLab username and the UUID of the run-bundle you tried to submit. You can find the UUID of your run-bundle via the web interface, or via the `cl info --verbose <BUNDLE NAME>` command. On the leaderboard, click on `failed` and get the UUID there (this is the UUID of the the leaderboard re-running your run-bundle). Also include that UUID in your Piazza post.

A.2.6 The F1/EM scores I get on the leaderboard are different to what I get on my own CodaLab worksheet

On the leaderboard, look at the timestamp of your submission. Does the timestamp match what you expect? If it's an older timestamp, perhaps your latest model wasn't correctly submitted (see A.2.4). On the leaderboard, look at the description. Does it match what you expect?

Open your project worksheet at `https://worksheets.codalab.org/` and find the run-bundle with the same timestamp. This is the run-bundle that was replicated by the leaderboard. You can see other details here, such as which exact version of the code and data it used. Check these are the (latest) versions of the code and the checkpoint that you intended to use.

A.2.7 My F1/EM score on the dev set leaderboard is much lower than on the sanity check leaderboard

You have full access to the dev set. Locally, it can be found in `data/dev-v1.1.json`, and on CodaLab, it can be accessed via its UUID `0x8f29fe`. This means that you can debug what's going wrong. Step through sections 4.1 – 4.2 again, but this time running on the full dev set (this means substitute in `dev-v1.1.json` when running locally and `0x8f29fe` as the UUID when running on CodaLab). If these runs give you the same poor F1/EM scores, then you can inspect your model locally to figure out what's going wrong. If these runs give you good F1/EM scores but when you submit to the dev leaderboard you get bad scores, then see A.2.6.

A.2.8 My F1/EM score on the test set leaderboard is much lower than on the dev set leaderboard

Unfortunately, the test set is secret, so you can't debug on it. If you've made multiple test set leaderboard submissions, are you sure that the test leaderboard is showing your latest submission (look at the timestamp)? Make sure that the run-bundle you submit to the test leaderboard is the exact same run-bundle that you submitted to the dev leaderboard (and got a good score on the dev leaderboard).