Lab 5: RL in the Cloud

Deep Reinforcement Learning Bootcamp

August 26-27, Berkeley CA

1 Introduction

In this lab, you will get some experience with running RL experiments on the Elastic Compute Cloud (EC2).

2 Environment Setup

You should have your environment set up as specified in the pre-lab setup instructions.

3 EC2 Setup

3.1 Signing In

First, visit this URL: https://deeprlbootcamp.signin.aws.amazon.com/console. You should see a page like the following:

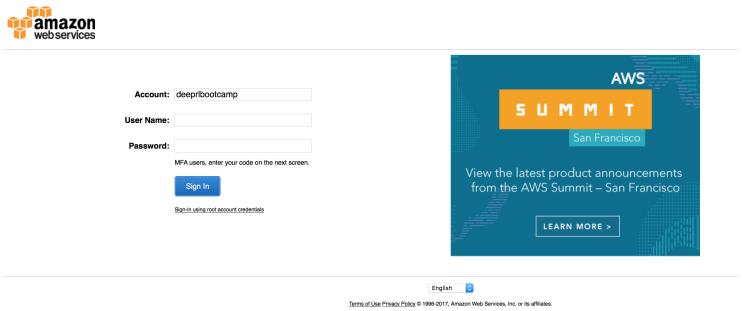

Figure 1: Log In

Your user name is `attendee_ORDERID`, where `ORDERID` should be replaced by the Eventbrite order ID (only include the digits). For example, if your order

ID is `612345678`, the user name would be `attendee_612345678`. The initial password is the email address you used to register the event.

Upon signing in, you must update your password to continue. Fill in the required information and then click "Confirm password change."

3.2 Generate Access Key

After signing in, visit this URL in the same browser: https://console.aws.amazon.com/iam/home#/users/attendee_ORDERID?section=security_credentials. Again, replace `ORDERID` with your actual order ID. Scroll down and click on a button named "Create access key" as in Figure 2.

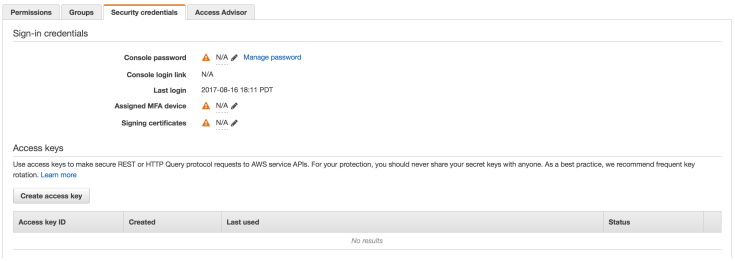

Figure 2: Create Access Key

You should see a small screen popping up as in Figure 3. Click on "Show" right below the column "Secret access key." Note down both the access key ID and the secret access key.

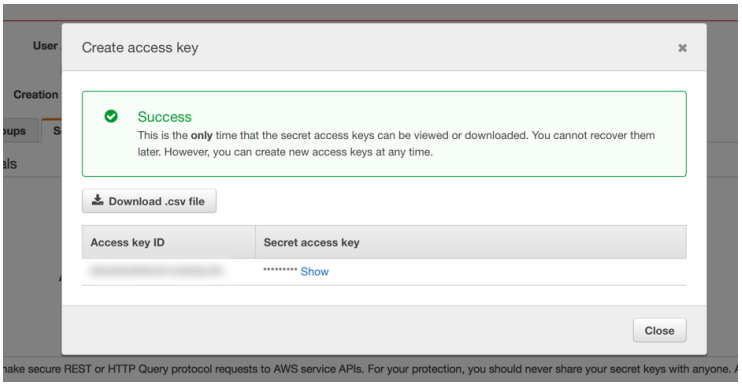

Figure 3: Show Secret Access Key

Now, in the project folder you should notice a file named `cloudexec.yml.template`. Make a copy of the file and name it `cloudexec.yml`. We need to make a few edits to this new file. First, replace `YOUR_ID_HERE` with your user name used to sign in to EC2, i.e. of the form `attendee_ORDERID`. Then, replace

YOUR_ACCESS_KEY_ID_HERE with the access key ID, and YOUR_SECRET_ACCESS_KEY_HERE with the secret access key (do not include quotes around them).

3.3 Generate Key Pairs

Finally, we need to generate key pairs used to access the machines remotely. This will be done from the terminal. From the project folder, run the following (this will only work after you set up `cloudexec.yml` properly):

```
./docker_run.sh scripts/generate_key_pairs.py
```

This should generate key pairs as needed, and display something like the following:

```
aws_key_pairs:
    - us-east-1: attendee_XXXXXXXXX_us-east-1
```

If you see an authentication error like

```
AWS was not able to validate the provided access credentials
```

, try restarting Docker (from menubar) and running it again. This could be due to the time in Docker not synchronized, and we observed that restarting Docker could fix it.

In `cloudexec.yml`, find `YOUR_KEY_PAIR_NAME` and replace it with the displayed key pair name.

Finally, run the following command to check if you have everything set up:

```
./docker_run.sh scripts/test_ec2_setup.py
```

Ideally, you should see a line saying "Your EC2 configuration has passed all checks!"

4 Your First Cloud Experiment

4.1 Testing Locally

You are now ready to launch your very first cloud experiment. Open the file `experiments/run_cloud_trpo_cartpole.py`. You will notice that the content is very similar to `experiments/run_trpo_cartpole.py` in Lab 4, except that the body is wrapped in a method named `run`, and instead of running it directly, we pass it to `cloudexec.remote_call`.

Typically, we want to make sure that the program runs locally before running it in the cloud. Notice that currently the mode is set to `cloudexec.local_mode`, so we are good to go. Run the following command:

```
./docker_run.sh experiments/run_cloud_trpo_cartpole.py
```

This should behave exactly the same as when we ran the TRPO-CartPole experiment in the previous lab.

4.2 Submitting to EC2

After confirming that the program works, change the mode to `cloudexec.ec2_mode`, and re-run the same bash command. A prompt will show up:

```
Running in EC2 mode. Confirm? [Y/n]
```

This is to ensure that you did not accidentally launch remote experiments. Type `y` (it's case insensitive) and press `Enter`. Then, internally, the program packages all the code in the current directory (except some excluded files, which can be configured in `cloudexec.yml` through `s3_code_sync_ignores`) and uploads it to S3, Amazon's Simple Storage Service.

Ideally, you should see a line similar to the following printed before the program exits:

```
Submitted job trpo-cartpole_XXX to EC2 in region us-east-1
```

Notice that the job name is prefixed by the `exp_group` parameter in the `cloudexec.Config` object, which is passed to `cloudexec.remote_call`. It is concatenated with a time stamp and an experiment ID to identify different experiments.

4.3 Monitor Launched Jobs

After launching the job, you can run the following command repeatedly to monitor the status of all current jobs:

```
./docker_run.sh scripts/ec2ctl.py jobs
```

Initially, running it may print nothing, since it takes a while for the instance to be launched. However, in a minute or two you should see the following:

```
(None) us-east-1a
```

This means an instance has been launched, but it has not been tagged with what job it runs. In a few more minutes, running the command should output the actual job name followed by its region:

```
trpo-cartpole_XXX us-east-1a
```

4.4 Remote SSH

Now let's try to SSH into the remote machine. In the terminal, run

```
./docker_run.sh ./scripts/ec2ctl.py ssh JOB_NAME
```

where you should replace `JOB_NAME` with the actual job name (without its region). This command will keep trying to ssh into the remote server until it succeeds. Multiple retries may be needed since the SSH server on the remote machine may take a while to start.

After signing into the machine, you will either land in the data folder of the experiment (in `~/code/data/local/...`) or the home folder, depending on whether the data folder has been created yet. In either case, run the following command:

```
cd ~
tail -f user_data.log
```

This will show you the standard output of the script that's currently running on this machine. You can press `Ctrl+C` to exit.

If the data folder is created, you can also `cd` into the data folder (the most convenient way is to first type `cd ~/code/data`, and then press `Tab` repeatedly to auto-complete the rest). The full command should look like the following:

```
cd ~/code/data/local/trpo-cartpole/trpo-cartpole_XXX
tail -f log.txt
```

This will show you similar content to `user_data.log`, but it will only show the logs explicitly stored rather than all standard output content.

When you are finished, first press `Ctrl+C` to exit `tail`, and then press `Ctrl+D` to exit SSH.

4.5 Pulling Experiment Logs

Once the experiment is running, the experiment logs will be periodically uploaded to S3, which can then be downloaded to your local machine.

In the terminal (locally, not on the remote machine), and from the project folder, run the following

```
./docker_run.sh scripts/sync_s3.py trpo-cartpole
```

This will download the periodically-synced log files to `data/s3`. You can then visualize these data:

```
./docker_run.sh viskit/frontend.py data/s3/trpo-cartpole
```

You can then navigate to the URL displayed in the terminal to visualize plots of various logged quantities.

4.6 Shutting Down Experiments

By default, the instance will be automatically terminated after the experiment finishes. If you need to terminate the instance earlier than that, you can use the following command:

```
./docker_run.sh scripts/ec2ctl.py kill_f JOB_NAME
```

where you should replace `JOB_NAME` with the actual job name. It will search for matching jobs and confirm before terminating the instances. This command also supports specifying a job name prefix, which can be convenient if you want to terminate multiple instances.

After shutting down experiments, try running `ec2ctl.py jobs` again, and you should see that these experiments are gone. However, the experiment logs will still live on S3.

5 Hyperparameter Tuning

One benefit of running experiments on the cloud is that it's very convenient to launch multiple experiments with different hyperparameter configurations. In this section, we will utilize EC2 to compare the effect of different baselines on the Pendulum task, when using TRPO as the RL algorithm.

Let's take a look at `experiments/run_cloud_trpo_pendulum_baseline.py`. Inside the method `run`, we check which baseline has been chosen and construct the corresponding baseline object. Then, below this method, we construct a `VariantGenerator`, which is a convenient class to configure different hyperparameters.

Again, let's start by verifying that the program works locally. Double check that the mode has been set to `cloudexec.local_mode`. In your terminal, run

```
./docker_run.sh experiments/run_cloud_trpo_pendulum_baseline.py
```

and make sure it works locally. Notice that the first line prints the number of variants generated this way (you should check that there are 9 variants).

After verifying it locally, change the mode to `cloudexec.ec2_mode`. Run the same command again and confirm to launch experiments on EC2. This time, it will launch 9 jobs instead of just 1.

After all jobs are launched, use the following command again to monitor the status of all jobs:

```
./docker_run.sh scripts/ec2ctl.py jobs
```

You should first see jobs appearing without names until there are 9 of them. Then, gradually each of them will be tagged with the names of their job. As soon as the name shows up, you can choose to ssh into one of the remote machines to see if the experiment has started running. Then, run the following command in your local terminal:

```
./docker_run.sh scripts/sync_s3.py trpo-pendulum-baseline
```

After data starts pouring in, launch VisKit:

```
./docker_run.sh viskit/frontend.py data/s3/trpo-pendulum-baseline
```

Then, navigate to the URL indicated in the terminal. You should see 3 curves, each representing a single type of baseline that's used. The solid line is the average of the logged quantity, and the shaded color indicates the standard deviation. Which one appears to be performing better?

6 Case Studies

Now that you've learned how to run experiments in the cloud, you can now design and run more experiments to investigate the behavior of algorithms under different hyperparameter settings. In this section, we present some case studies that you can try out, and you are encouraged to design and run your own experiments too.

Note that since we have a finite compute budget, **please do not have more than 20 instances running simultaneously at any given time**. In addition, since EC2 bills to each whole hour (which means that if you launch an instance and immediately terminate it, it still costs an hour), please try to avoid frequently launching / terminating instances if possible. Intentional abuse of compute resources may result in termination of your account before the end of the bootcamp.

6.1 Tuning λ

The λ parameter used in Generalized Advantage Estimation [1] interpolates between 1-step TD prediction ($\lambda = 0$) and full Monte Carlo returns ($\lambda = 1$). In this case study we will investigate the effect of different λ parameters.

Start by making a copy of `experiments/run_cloud_trpo_pendulum_baseline.py` in the same folder, and name it `run_cloud_trpo_pendulum_gae_lambda.py`. You need to make the following edits to this file:

- Near the end of the file, change the value of `exp_group` from `trpo-pendulum-baseline` to `trpo-pendulum-gae-lambda`. This will ensure that the logs are saved to a new folder, different from the previous one.
- Change the mode to `cloudexec.local_mode`, since we want to test locally first.
- Edit the list of baseline options to only include the best-performing baseline found in the previous section. For example, if the linear feature baseline performed the best, you should have

  ```
  vg.add("baseline", ['linear_feature'])
  ```

  This ensures that we will not be searching through different baselines for this case study (otherwise, we would be launching too many instances).
- After the line above, add the following:

  ```
  vg.add("gae_lambda", ...)
  ```

where you should replace "..." with a list of numbers between 0 and 1 that you want to experiment with. You should have at most 6 numbers so that the total number of experiments is at most 20.

- In the method call to `trpo`, after `n_iters=100` (or elsewhere within the method call), add `gae_lambda=v['gae_lambda']`.

After making these edits, double check that we are using the local mode. Then, let's run this file:

```
./docker_run.sh experiments/run_cloud_trpo_pendulum_gae_lambda.py
```

You should verify that it runs locally, and that the total number of variants is at most 20. Then, change the mode back to `cloudexec.ec2_mode`, and run the same file again to submit the jobs to EC2.

Recall that you can monitor job status using:

```
./docker_run.sh scripts/ec2ctl.py jobs
```

To pull the log files, run

```
./docker_run.sh scripts/sync_s3.py trpo-pendulum-gae-lambda
```

To visualize the results, run

```
./docker_run.sh viskit/frontend.py data/s3/trpo-pendulum-gae-lambda
```

6.2 Tuning batch size

In this case study, we will investigate the effect of different batch sizes on training, and see which batch size would lead to the fastest learning for the `Pendulum-v0` task.

Start by making a copy of `experiments/run_cloud_trpo_pendulum_baseline.py` in the same folder, and name it `run_cloud_trpo_pendulum_batch_size.py`. You need to make the following edits to this file:

- Near the end of the file, change the value of `exp_group` from `trpo-pendulum-baseline` to `trpo-pendulum-batch-size`.
- Change the mode to `cloudexec.local_mode`.
- Edit the list of baseline options to only include the best-performing baseline found in the previous section. For example, if the linear feature baseline performed the best, you should have

  ```
  vg.add("baseline", ['linear_feature'])
  ```

- After the line above, add the following two lines:

  ```
  vg.add("batch_size", [1000, 2500, 5000, 10000])
  vg.add("n_iters", lambda batch_size: [1000000 // batch_size])
  ```

The first line adds a hyperparameter for the batch size. The second line is more interesting. Rather than specifying a list of possible values for `n_iters`, it is specified as a function of the batch size. Indeed, in this case we want the total number of samples to be consistent under different batch sizes. In the previous section we used 1M samples. Hence, each batch size will correspond to only one value of `n_iters`, computed by the given expression. Note that the lambda function should return a list rather than a single number. This allows for the possibility of multiple options.

The ability to specify a function that returns a list, rather than a fixed list can be very useful. Note that it's not limited to a function of `batch_size`, but any other hyperparameters (as long as they do not create a circular dependency). Internally, the code parses the input argument name to the lambda function, and check whether a hyperparameter with the corresponding name exists. You can also have multiple input arguments to the lambda function, in which case a separate list of `n_iters` will be generated for each combination of the hyperparameters corresponding to the input arguments.

- In the method call to `trpo`, replace `batch_size=10000` with `batch_size=v['batch_size']`, and `n_iters=100` with `n_iters=v['n_iters']`.

After making these edits, double check that we are using the local mode. Then, let's run this file:

```
./docker_run.sh experiments/run_cloud_trpo_pendulum_batch_size.py
```

You should verify that it runs locally. Also, check that there should be 12 variants. Then, change the mode back to `cloudexec.ec2_mode`, and run the same file again to submit the jobs to EC2.

Recall that you can monitor job status using:

```
./docker_run.sh scripts/ec2ctl.py jobs
```

To pull the log files, run

```
./docker_run.sh scripts/sync_s3.py trpo-pendulum-batch-size
```

To visualize the results, run

```
./docker_run.sh viskit/frontend.py data/s3/trpo-pendulum-batch-size
```

For this particular case study, the default x-axis (the iteration number) is not very convenient for determining which batch size is more sample-efficient, since each option runs for a different number of iterations. You can toggle the `X-Axis Attribute` to something more sensible such as `TotalNSamples` to compare them more easily.

For learning curves with very large variance, you may want to switch the `Display Mode` to `Individual` which displays each individual curves without

averaging them. This can give you a better idea of how well different runs are performing.

You should observe that typically, a larger batch size leads to more stable learning. On the other hand, smaller batch sizes can sometimes achieve better sample complexity but tend to be less stable.

References

[1] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR2016)*, 2016.