CS294-112 Deep Reinforcement Learning Plotting and Visualization Handout

1 General Best Practices

Plotting and visualization are an important component of designing, debugging, prototyping, and evaluating your deep reinforcement learning algorithms. The following tips might help you structure your code in a way that makes it easy to produce good plots:

- Your learning code should log results to an external file, such as a csv or pkl file, rather than producing the final plot directly. This way, you can run the learning process once, and then experiment with different ways to plot the results. It might be a good idea to log more than you think is strictly necessary, since you never know what information will be most useful for understanding what happened. Keep an eye on file size, but generally it might be good to log some of the following: average reward or loss at each iteration, some of the sampled trajectories (for subsequent visualization), useful secondary metrics such as Bellman error or gradient magnitudes.

- You should have a separate script that loads up one or more logs and plots the results. If you run the algorithm multiple times with different hyperparameters or random seeds, run different algorithms to compare, or run variants of your method, it's a good idea to load up all of the data together (perhaps from different files) and plot it on the same plot, with an automatically generated legend and a color scheme that makes it easy to distinguish different methods.

- Deep RL methods, especially model-free methods that you'll learn about in the course, tend to experience considerable variability between runs. It's therefore a good idea to run multiple times with multiple different random seeds. When plotting the results for multiple runs, it may be a good idea at least initially to plot all of the runs on the same plot, with the average performance also plotted with a thicker line or in a different color. When plotting many different methods, you may find it convenient to summarize this into mean and standard deviation plots. However, the distribution doesn't always follow a normal curve, so plotting all the runs,

at least initially, might give you a better sense for the variability between random seeds.

2 Example Code

In python, matplotlib and seaborn are useful tools for plotting data. Here is some example code for plotting with shaded regions to indicate standard deviation:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

  # This is just a dummy function to generate some arbitrary data
  def get_data():
    base_cond = [[18,20,19,18,13,4,1],
         [20,17,12,9,3,0,0],
         [20,20,20,12,5,3,0]]

    cond1 = [[18,19,18,19,20,15,14],
         [19,20,18,16,20,15,9],
         [19,20,20,20,17,10,0],
         [20,20,20,20,7,9,1]]
    cond2= [[20,20,20,20,19,17,4],
          [20,20,20,20,20,19,7],
          [19,20,20,19,19,15,2]]
    cond3 = [[20,20,20,20,19,17,12],
          [18,20,19,18,13,4,1],
          [20,19,18,17,13,2,0],
          [19,18,20,20,15,6,0]]
    return base_cond, cond1, cond2, cond3

# Load the data.
results = get_data()
fig = plt.figure()

# We will plot iterations 0 ... 6
xdata = np.array([0,1,2,3,4,5,6])/5.

# Plot each line
# (may want to automate this part e.g. with a loop).
sns.tsplot(time=xdata, data=results[0], color='r', linestyle='-')
sns.tsplot(time=xdata, data=results[1], color='g', linestyle='--')
sns.tsplot(time=xdata, data=results[2], color='b', linestyle=':')
sns.tsplot(time=xdata, data=results[3], color='k', linestyle='-.')
```

```
# Our y-axis is "success rate" here.
plt.ylabel("Success Rate", fontsize=25)

# Our x-axis is iteration number.
plt.xlabel("Iteration Number", fontsize=25, labelpad=-4)

# Our task is called "Awesome Robot Performance"
plt.title("Awesome Robot Performance", fontsize=30)

# Legend.
plt.legend(loc='bottom left')

# Show the plot on the screen.
plt.show()
```

Note that in practice, you may want to automate your code to load a set of files, automatically draw a reasonable legend, and generate automatic colors.