CMPUT 652, Fall 2019 - Assignment #2

October 15, 2019

Version: 1.1. October 16, 2019.

Due: October 29, 23:59 MST

This assignment consists of two parts. The first are written questions which are related to the course lectures. The second part is a programming assignment. You will need to submit both code and a write up. Please provide these as a single archive. Submit to both armahmood@ualberta.ca and sherstan@ualberta.ca.

1 Written Questions

Total 30 points. Complete within the given space using LaTeX or handwritten notes. Show the derivations or the steps for obtaining partial points. On the other hand, mistakes, missteps or bad reasoning behind a correct answer will result in points deducted.

1.1 Consider a finite MDP, where a state is represented by a d-dimensional feature vector $\mathbf{x}(s)$, and the feature vectors for all N states form a $N \times d$ feature matrix $\mathbf{X}$ with rank d. The Mean Squared Value Error (MSVE) objective is $\|\mathbf{v}_\pi - \mathbf{X}\mathbf{w}\|^2_{\mathbf{D}_\pi}$, where $\mathbf{v}_\pi$ is an N-dimensional vector, $[\mathbf{v}_\pi]_s = v_\pi(s)$ is the value of state s induced by policy π, and $\mathbf{D}_\pi$ is a $N \times N$ diagonal matrix with the steady-state probabilities $d_\pi(s)$ induced by π along the diagonal. Show that the solution to this objective is $\mathbf{w}^*_{\text{VE}} = (\mathbf{A}_{\text{VE}})^{-1}\mathbf{b}_{\text{VE}}$, where $\mathbf{A}_{\text{VE}} = \mathbf{X}^\top \mathbf{D}_\pi \mathbf{X}$ and $\mathbf{b}_{\text{VE}} = \mathbf{X}^\top \mathbf{D}_\pi \mathbf{v}_\pi$ (15 points).

$$\begin{aligned}
\|\mathbf{v}_\pi - \mathbf{X}\mathbf{w}\|^2_{\mathbf{D}_\pi} &= (\mathbf{v}_\pi - \mathbf{X}\mathbf{w})^\top \mathbf{D}_\pi (\mathbf{v}_\pi - \mathbf{X}\mathbf{w}) \\
&= \mathbf{v}_\pi^\top \mathbf{D}_\pi \mathbf{v}_\pi - \mathbf{w}^\top \mathbf{X}^\top \mathbf{D}_\pi \mathbf{v}_\pi - \mathbf{v}_\pi^\top \mathbf{D}_\pi \mathbf{X}\mathbf{w} + \mathbf{w}^\top \mathbf{X}^\top \mathbf{D}_\pi \mathbf{X}\mathbf{w}
\end{aligned}$$

$$\begin{aligned}
\nabla_{\mathbf{w}} \|\mathbf{v}_\pi - \mathbf{X}\mathbf{w}\|^2_{\mathbf{D}_\pi} &= -2\mathbf{X}^\top \mathbf{D}_\pi \mathbf{v}_\pi + 2\mathbf{X}^\top \mathbf{D}_\pi \mathbf{X}\mathbf{w} = 0 \\
&\rightarrow \mathbf{X}^\top \mathbf{D}_\pi \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{D}_\pi \mathbf{v}_\pi \\
&\rightarrow \mathbf{w}^*_{\text{VE}} = (\mathbf{X}^\top \mathbf{D}_\pi \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{D}_\pi \mathbf{v}_\pi \\
&\text{The inverse is valid here because } \mathbf{X}, \mathbf{D}_\pi \text{ are both full-rank matrices} \\
&\rightarrow \mathbf{w}^*_{\text{VE}} = (\mathbf{A}_{\text{VE}})^{-1} \mathbf{b}_{\text{VE}}
\end{aligned}$$

1.2 Consider a vector y that may not be in the linear span of the features, that is, $\mathrm{y} \neq \mathbf{X}\mathrm{w}$ for any $\mathrm{w} \in \mathbb{R}^d$. The projection matrix $\mathbf{\Pi}$ is such that $\mathbf{\Pi}\mathbf{y} = \mathbf{X}\mathrm{w}_y$, where $\mathrm{w}_y = \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathrm{w}\|_{\mathbf{D}}^2$, that is, it is the matrix that linearly transforms or "projects" y to the span of features according to norm $\|\cdot\|_{\mathbf{D}}^2$. Here $\mathbf{D}$ is a diagonal matrix with positive diagonal elements. Show that $\mathbf{\Pi} = \mathbf{X}(\mathbf{X}^\top\mathbf{D}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{D}$, $\mathbf{\Pi}\mathbf{X} = \mathbf{X}$ and $\mathbf{X}^\top\mathbf{D}\mathbf{\Pi} = \mathbf{X}^\top\mathbf{D}$ (15 points).

Based on the answer for Q1.1, we know:

$$\mathbf{w}_y = (\mathbf{X}^\top\mathbf{D}_\pi\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{D}_\pi y$$

$$\begin{aligned}\mathbf{\Pi}\mathbf{y} &= \mathbf{X}\mathbf{w}_y\\ &= \mathbf{X}(\mathbf{X}^\top\mathbf{D}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{D}y\\ &\rightarrow \mathbf{\Pi} = \mathbf{X}(\mathbf{X}^\top\mathbf{D}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{D}\end{aligned}$$

$$\begin{aligned}\mathbf{\Pi}\mathbf{X} &= \mathbf{X}(\mathbf{X}^\top\mathbf{D}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{D}\mathbf{X}\\ &= \mathbf{X}(\mathbf{X}^\top\mathbf{D}\mathbf{X})^{-1}(\mathbf{X}^\top\mathbf{D}\mathbf{X})\\ &= \mathbf{X}\end{aligned}$$

$$\begin{aligned}\mathbf{X}^\top\mathbf{D}\mathbf{\Pi} &= \mathbf{X}^\top\mathbf{D}\mathbf{X}(\mathbf{X}^\top\mathbf{D}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{D}\\ &= (\mathbf{X}^\top\mathbf{D}\mathbf{X})(\mathbf{X}^\top\mathbf{D}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{D}\\ &= \mathbf{X}^\top\mathbf{D}\end{aligned}$$

2 Programming

100 points. You will implement a basic policy-gradient method: REINFORCE with a Baseline. The purpose is to gain experience implementing PG methods in PyTorch. Additionally, the questions are meant to help you develop the tools you will need for performing empirical analysis.

See Sections 13.3 and 13.4 of **Sutton2018** for information on the REINFORCE algorithm.

```
REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$                                   $(G_t)$
        $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
```

Figure 1: (**Sutton2018**)

The episodic REINFORCE algorithm collects entire episodes of trajectories before updating policy parameters. It is a Monte Carlo method based on Stochastic Gradient Descent. In the its returns can have high variance, making learning difficult. By subtracting off a baseline, the value estimate, the variance of the policy parameter updates can be reduced and learning can be improved.

Your task is to implement REINFORCE with a baseline to solve the classic Cart Pole task: https://gym.openai.com/envs/CartPole-v1/.

Note that the algorithm specified in the above collects an episode of transitions and then loops over each transition of the episode computing gradients and updating weights at each timestep. You could also implement the update as a batch where the errors are taken across the entire episode trajectory before gradients are computed. Feel free to implement either approach. However, for reporting losses please average losses over the whole episode.

A note on γ. First, γ is sometimes approached in two different ways. One way claims that γ is part of the problem definition and the other treats γ as part of the solution. We will take the second view here. We will evaluate your algorithm based on the total undiscounted reward received over an entire episode. You are free to adjust γ as you need, but a good place to start would be to simply set $\gamma = 1$ and then try other values. Further, the last line of the equation in Figure 1 has the additional term γ^t. This term was not always included in the algorithm and you will find implementations online which leave out this term. I expect it will impact the values of γ for which your algorithm performs well. You are free to include it or leave it out, but you should be clear which version you implemented in your pseudocode.

2.1 Setup

Create a python virtualenv (feel free to use another dependency manager if you like) for running your code. We will use python3.6+. A requirements.txt file is included specifying all the packages which you will require for your virtualenv. Do not make your code dependent on any additional packages as they will not be part of the environment on which I test.

Create your virtualenv as follows (adjust paths as required):

```
# create a virtualenv called assign2
virtualenv --python=python3.6 assign2
# you will need to activate this environment before running your script.
source assign2/bin/activate
```

```
pip install -r requirements.txt
```

You have been provided with two skeleton files: main.py and network.py. Edit these files to implement your code.

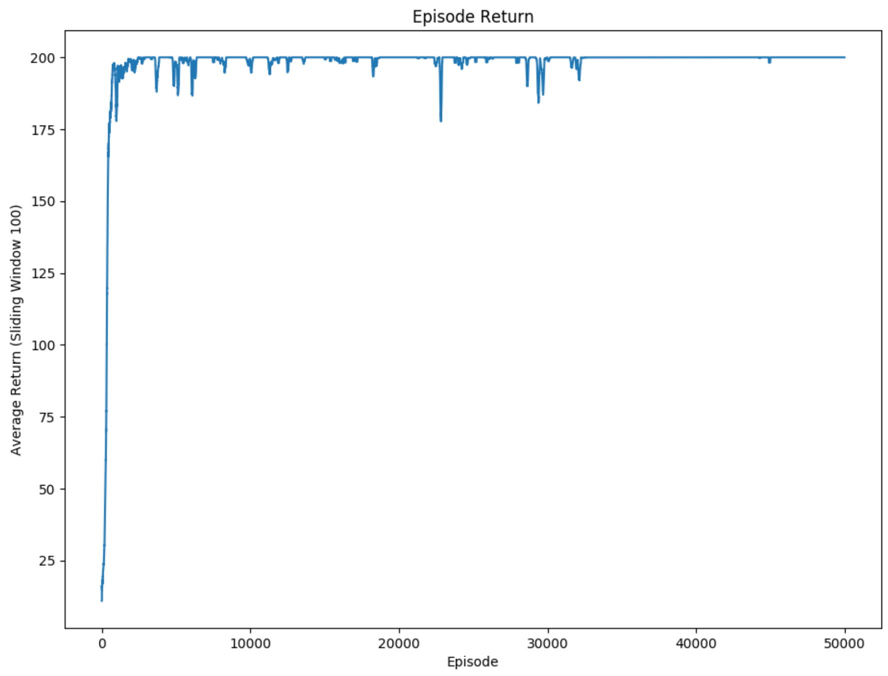
2.2 Deliverables

Note, that I am not evaluating your code based on whether or not it can beat some baseline or the performance of your classmate's algorithm. I am looking for code that works. Your agent should be able to fairly frequently achieve a perfect score of 200 by at least 10k episodes. I will be lenient in grading, but I do expect that everyone put in the effort and does their own work - of course you are free to consult with one another. The key to full marks is to simply implement the algorithm, achieve reasonable performance, and put in sufficient effort in answering the questions and formatting your plots where possible. If everything looks good from a high level, I won't go looking for problems.

The plots provided below should only be taken as examples, not optimal.

In the following code we have chosen to plot returns as a function of the episode. We do this for simplicity. A fairer comparison would be to compare based on the number of observations made. In the next assignment we will compare in this way.

2.2.1 Running Code (50 points)



Algorithm 1 REINFORCE with Baseline

```
Require: Initialize policy network $\pi(A|S,\theta)$ and value network $v(S|w)$
  for $episode = 1, M$ do
      Generate an episode $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T$ with policy $\pi(\cdot|\cdot,\theta)$
      for each step of the episode $t = 0, 1, \cdots, T-1$ do
          $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
          $\delta \leftarrow G_t - v(S_t, w)$
          $w \leftarrow \alpha\delta\nabla v(S_t, w)$
          $\theta \leftarrow \alpha\delta\nabla \ln \pi(A_t|S_t,\theta)$
      end for
  end for
```

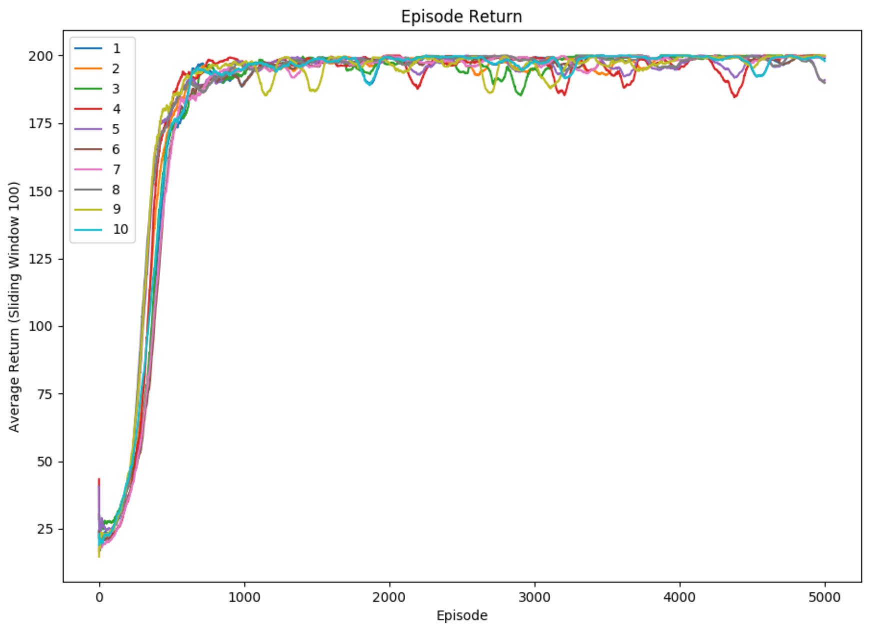
2.2.2 Saved Policy (5 points)

See ”saved.pkl”, load it by ”python3 main.py -l”

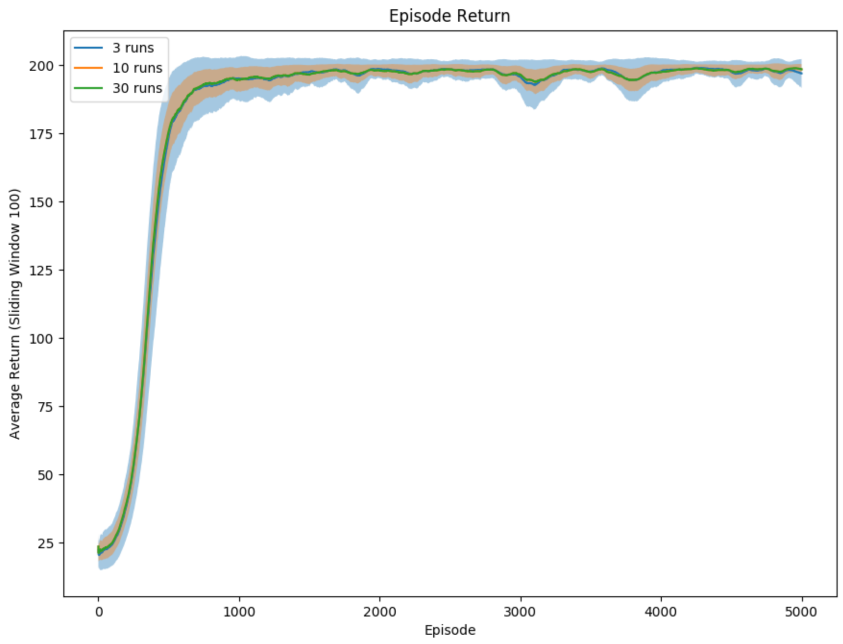
2.2.3 Plot Return vs. Episode (15 points)

Perform 30 different runs for your algorithm (each run should be a different random initialization). Each run should be 5000 episodes in length. For each run you will need to save the list of episode returns. Using this data you will produce two different plots looking at the mean performance across different runs. Plots should be labeled and legible - they should look nice and be something you would include in a paper submission.

1. From your data set sample 3 runs and take the mean across the runs. Do this 10 times, each time resampling from your set of 30 runs. Plot each of these means together on a single plot. To be clear the x-axis will be the episode number.



2. From your dataset plot means for the following: 3 runs, 10 runs, 30 runs. Plot these all on the same plot. Label the lines in some way.

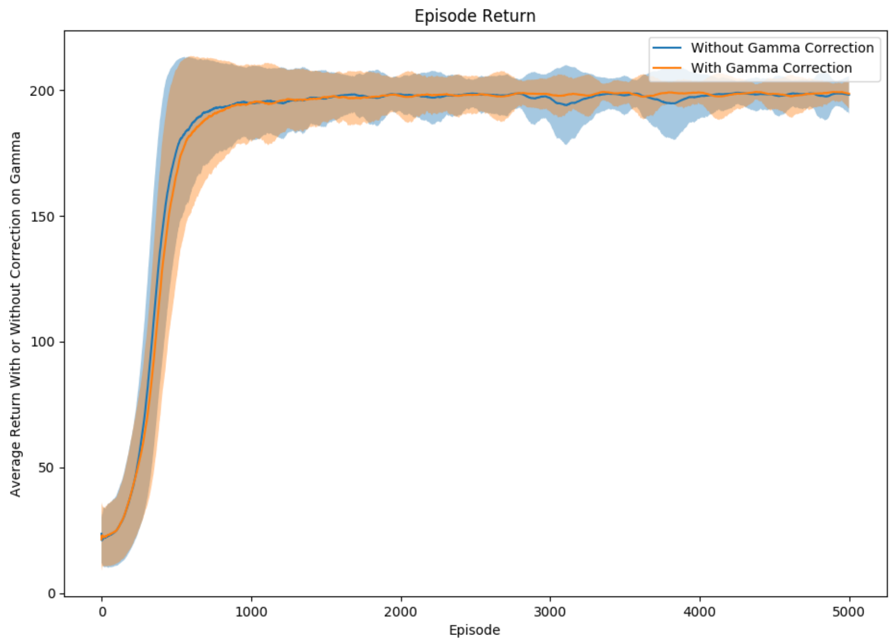


Question: There are different statistics which might be reported for each mean: standard deviation, max-min, standard error. Which of these would you use? When? Why? In the previous plot which would you use to give a comparison between the series? Include your answer to these questions.

I would use standard deviation to compare between the previous results once the learning curves converge. Standard deviation is used here because it's an appropriate statistic for possible deviation from the expected value within certain confidence interval. Max-min is not used here because it's extremely sensitive to outliers and contains no information about confidence interval. Standard error is not used as in reinforcement learning, we usually have limited number of returns whose standard deviation can be explicitly computed instead of estimating with standard error.

2.2.4 Algorithm Comparison: (15 points)

In this question, I compared the performance of REINFORCE with or without the term γ^t in the policy update $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\gamma^t\delta\nabla\ln\pi(A_t|S_t,\mathbf{w})$. In my previous implementation, I didn't include this term as current algorithms usually ignore this term. The figure given below is the mean and standard deviation of 30 run with random initialization.



In first roughly 800 episode, where the episode return is rapidly increasing, update without the γ^t term learns slightly faster than the other one, but they have similar standard deviation in performance. My explanation here is that adding the γ^t term has the effect of making smaller steps in gradient update in general which resulted in slower learning.

However, when both methods converge, method with the γ^t term has noticeable lower variance and the performance curve is also more flatten than the other one. I think this is also because of the smaller steps in gradient update by γ^t. Making gradient update with a large step size is likely to degrade current policy, especially when current policy is near-optimal. Such a problem is addressed in methods like TRPO or PPO, but using smaller learning rate could yield the similar effect to some extent.

2.2.5 Tensorboard file: (15 points)

See ”./runs”

2.3 Submission Checklist

☑ **2.2.1** Source Code and output plot from one run. Written algorithm.

☑ **2.2.2** One saved policy.

☑ **2.2.3** Two plots and 1 answered question.

☑ **2.2.4** A short writeup of your comparison. One plot showing the comparison.

☑ **2.2.5** One TensorBoard events file.

2.4 Common Problems to Watch For

- Forgetting to link your optimizer with your policy parameters.

```
        torch.optim.Adam(network.parameters())
```

- Forgetting to `zero_grad` before calling `backward`.
- Broadcasting issues. If two vectors of different sizes are attempted to combine, e.g. [26] and [26,1], torch and numpy may happily combine these, but the results may not be what you expect. This can give incorrect results.
- Wrapping data in `torch.Tensor()` breaks the gradient flow. For example, if you want to convert a list of tensors to a tensor use `torch.stack()` instead.
- Forgetting to call `detach()` on a bootstrapped target. You don't, usually, want the gradient taken through your target. When we use a bootstrapped target, like in TD learning, we need to make sure to detach the gradient through the target.