Topics in Machine Learning (CS975) **IIIT Hyderabad**
Instructor: Dr. Naresh Manwani Due: **16 October 2019 11:59pm, on Moodle**

Please use the moodle discussion thread for this assignment to communicate any queries or doubts.

2 Monte Carlo, TD Methods and Function Approximation

In this assignment, you will use Monte Carlo (MC) Methods and Temporal Difference (TD) Learning here and there on couple of games and toy problems. You will also take a shot at one of the simpler function approximation methods. If you took an extra mile last assignment, you should be able to reuse most of your plotting, environment and similar helper functions.

2.0.1 Marking

Section	Points
Monte Carlo Methods	25
Temporal Difference Methods	25
Function Approximation	50
Bonus Questions	50

Table 1: Marks Distribution, Assignment 2

The following is the overall marking rubric. In order to ensure that each section gets enough attention, bonus marks will not be carried forward across sections. For each component, working code will get you 50% of the marks (mostly binary marking here), remaining 50% is for writing up your findings. Therefore, ensure your presentation is well structured and readable[3].

2.0.2 Implementation constraints

Keep in mind that starting this assignment you do not have direct access to the MDP tuple elements S, A, R, P unlike the dynamic programming agent. All of these have to be incrementally estimated through several roll-outs and implementation of environment and agents should not share any information unless they interact. At a point in time you only have access to what you have obtained so far from calling the following, considering OpenAI gym as an example:

```
state, reward, is_terminated, [prob] = env.step(action)
```

`[prob]` is an extension which enables doing algorithms like Expected SARSA, do not use it for algorithms where it is not allowed.

[3] Move larger code segments like class or function definitions into .py files and leave only calls etc in the notebook. Notebook should be more text and bare minimum required code.

2.1 Tic-Tac-Toe via Monte Carlo Methods

haje01/gym-tictactoe contains a Tic-Tac-Toe environment designed alongside the specs of an environment in OpenAI gym. You can also find a `BaseAgent` and a `TDAgent` which is trained using Temporal Difference learning (TD-Learning), along with a trained model. For points in this question, you will pit your agents against the methods provided in the repository and train it via the following Monte Carlo Methods discussed in class.

1. On policy Monte Carlo. **[10 points]**
2. Off policy Monte Carlo through importance sampling. **[10 points]**

Few implementation details (Which player do you want to play as? Do you go for ordinary or weighted importance sampling, how do you incorporate an epsilon-soft policy, etc.) are left to you – make assumptions, discuss on moodle if required and go ahead with your implementation. The details are required to be included in your write-up for clarity and completeness.

Training curves [5 points] To indicate your training progress, present plots for the following for both on-policy and off-policy methods.[4].

1. Number of unique states covered in the rollouts so-far versus return confidence intervals among N=10 games against the tic-tac-toe agents.
2. Number of MC rollouts versus return confidence intervals among N=10 games against the tic-tac-toe agents.

Write up the inferences you can make from the above graphs. Which of the above two plots do you think is more effective, why?

Estimator bias versus variance [Bonus +15 points] For a few random states or state-action pairs, track the values of the quantity being estimated – $V(s)$ or $Q(s, a)$ over visits and consequent updates. Based on an illustration through plotting these data over time, comment on the variance of these quantities for a given state over updates, for both on-policy and off-policy methods comparing the two. From your practical experience doing the above, explain how the differences affect convergence and stability.

2.2 FrozenLake through TD Methods

Grab `FrozenLake` 8x8 from OpenAI gym and configure it to have suitable number of slippery blocks. Implement the following algorithms from the class of TD Learning algorithms. Re-use the plotting content from previous assignment to produce visualizations of the final values being estimated. Plot the rewards (confidence intervals over N runs) vs episodes for all three methods and compare.

(1) SARSA **[10 points]** (2) Q-Learning **[10 points]** (3) Expected SARSA **[5 points]**

Monte Carlo v/s TD Methods [Bonus +10 points] Tweak haje01/gym-tictactoe provided to compare your Monte Carlo methods versus TD methods. You can use the number of episodes or updates so far as the x-axis and the confidence intervals of the return obtained for test-runs on the y-axis. Argue the merits of your comparisons and comment which problems are better suited for MC or otherwise, TD.

[4] Given the environment, reward structure and transition matrix are same, you may use the same graph to compare all, provided you use a good colour scheme and legend to distinguish between the agents involved with colours. In the first assignment, many of you plotted comparison graphs which was supposed to compare the same axes for different agents/runs seperately. Do not make the same mistake this time.

2.3 Function Approximation: DQN [50 points]

Time to play some games. Atari Breakout is an arcade game where you have to break bricks guiding a ball using a board. In simple words, for points in this question - **build a Deep Q-Learning Network (DQN) which can play Atari Breakout and get the best scores**.

Due to the nature of the problem and to enable flexibility in implementation, evaluation will be done for the whole component (no parts and marks for those, unlike previous questions). An ideal submission will be written along the spirit of a distill publication (example), except in the jupyter-notebook ecosystem.

Turn in a comprehensive report[5] on your implementation. Within the report, argue with supporting evidence in that your solution:

1. Trains, converges and performs well on average across several test-runs.
2. The estimator gives good scores when the game is going good and bad scores when the game is going bad.

Also, compare how the agent behaves in different restrictions of the action space. (For example, only allow left-right and no stationary position).

You may need to use experience replay and a target network to stabilize training (Readup more on this).

Bonus [+25 points] Do either of the following:

- Use components from your DQN implementation to bring about a solution which works for Pong, repeat the deliverables above for Pong. Additionally, explain in your report the changes you had to make.
- Use double Q-Learning (deep) in place of vanilla DQN. What problems which comes with vanilla DQN does double Q-Learning solve? Use graphs and inferences from test-runs of both to support your arguments.

Supporting Material: Neural Networks, Automatic Differentiation and DQN

The content below includes pointers to help you out with implementation and connect the implementation to the lectures. We will first introduce notations where we parametrize the Q-value estimator with neural network weights. Following this approximation, we will try to see how Q-Learning becomes a regression problem. Next, we will try to shed some more light on how a forward pass on the loss function followed by Automatic Differentiation (AD) is equivalent to the update step discussed in class.

Parametrization via Deep Neural Networks Thanks to recent developments in deep learning, most Atari games can be solved quite easily provided you have a some compute time to burn. Here, you will use Q-learning through function approximation. Use a neural network parameters (weights) θ to approximate the Q-function, called the Deep Q-Learning Network (DQN) hereafter.

$$Q(s,a) \approx Q(s,a;\theta)$$

[5] embedded in a notebook after a run - primarily human readable english. Move larger segments of codes to .py files.

Predicting state values – connection to regression Using terms from a supervised learning course you should have taken before as a prerequisite, you are attempting to use regression and model the following:

$$Q(s, a; \theta) = \mathbb{E}\left[G|s, a\right]$$

But here you do not have fixed labels or predictions but is updating dynamically as your agent navigates and better estimates the environment.

The inputs to the neural network can be a concatenation the state and action vectors which is passed to the neural network. The network has to predict the Q-value given a state and action. While training we try to minimize the error (MSE, Huber Loss etc for example) between the predicted value and expected value based on feedback from environment. To minimize this error, we can use stochastic gradient descent. Below, we review the updates covered in class following which we will connect them to most solutions you will find online for reference:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[\left(r + \gamma \max_{a'} Q(s',a')\right) - Q(s,a)\right] \qquad \text{TD Update}$$

$$\theta \leftarrow \theta + \alpha\left[\underbrace{\left(r + \gamma \max_{a'} Q(s',a';\theta)\right)}_{\text{expected value or ground truth}} - \underbrace{Q(s,a;\theta)}_{\text{DQN prediction}}\right]\nabla_\theta Q(s,a;\theta) \qquad \text{SGD update of } \theta$$

Keep in mind that in essence, at each update we intend to improve upon our existing beliefs using the feedback from the environment, which is a combination of reward (r) and existing beliefs ($Q(s, a; \theta)$). There is no true-label or ground truth like normal supervised setting. This can lead to chances of instability, arrival at degenerate solutions (How would you fix this?).

SGD using automatic differentiation In automatic differentiation (AD) enabled frameworks like PyTorch or Tensorflow, you can convert the above update equation involving ∇_θ to an equivalent loss function after declaring the forward pass and let AD take care of the gradient descent over any complicated neural network, so long as the operations in the forward pass are differentiable.

$$\begin{aligned} \hat{q} &\leftarrow Q(s,a;\theta) && \text{predicted Q-value} \\ q &\leftarrow r + \gamma Q(s',a';\theta) && \text{target Q-value} \\ L &\leftarrow L(\hat{q}, q) && \text{loss} \end{aligned}$$

If you substitute L to be MSE and take into account α, you can see that the derivative turns out to be as per the update equation above. In implementation you thus end up writing the operations to get L and calling a `.backward` (PyTorch), or an optimizer (TensorFlow) or similar patterns.

2.4 Submission Instructions

Your submissions are expected to be a `.tar.gz` [6] file containing the following:

[6] (**not** .zip, .gz, .tgz, etc.)

- `main.ipynb` - A Jupyter notebook[7] with code and explanations written as a walkthrough for the code. Any proofs and derivations **should be** embedded in the notebook using MathJax + Markdown.
- `main.html` generated from the above notebook.
- Auxilliary python scripts (plotting, helpers for loading etc, abstract classes which need not appear in the notebook which you import and reuse in the notebook.
- Do not include `.ipynb_checkpoints`, `__pycache__` or other files which are not required.

Assignments will have a manual evaluation. Keep the saved models somewhere, as you will be asked to run your test-code again then. Time and venue will be announced later.

Collaboration Policy You can indicate your collaborations and avail marks $2x/3$ if you score x as a team. Groups of maximum 2 members are allowed. Extra content in your report (notebook) on how you divided work **AND** `.git` history supporting the claims are required here.

Plagiarism Policy Plagiarism detection software is guaranteed to be run before any evaluation. Trying to beat any such software will make your code significantly unreadable and easy to prove malicious intent. In case of heavy plagiarism - all parties involved (giver, taker) will get a 0.

[7] Make sure this is after a restart and run-all option, so a lot of size doesn't go up in notebook state.