Homework 6: Inference in Graphical Models, MDPs

Introduction

In this assignment, you will practice inference in graphical models as well as MDPs/RL.

Resources and Submission Instructions

For readings, we recommend Sutton and Barto 2018, Reinforcement Learning: An Introduction, CS181 Lecture Notes, and Section 10 and 11 Notes.

Please type your solutions after the corresponding problems using this LaTeX template, and start each problem on a new page.

Please submit the **writeup PDF to the Gradescope assignment 'HW6'**. Remember to assign pages for each question.

Please submit your **LaTeX file and code files to the Gradescope assignment 'HW6 - Supplemental'**.

You can use a **maximum of 2 late days** on this assignment. Late days will be counted based on the latest of your submissions.

Problem 1 (Hidden Markov Models, 15 pts)

In this problem, you will be working with one-dimensional Kalman filters, which are *continuous-state* Hidden Markov Models. Let $z_1, \cdots, z_t$ be the hidden states of the system and $x_1, \cdots, x_t$ be the observations produced. Then, state transitions and emissions of observations work as follows:

$$\begin{aligned} z_{t+1} &= z_t + \epsilon_t \\ x_t &= z_t + \gamma_t \end{aligned}$$

where $\epsilon_t \sim N(0, \sigma_\epsilon^2)$ and $\gamma_t \sim N(0, \sigma_\gamma^2)$. The value of the first hidden state follows the distribution $p(z_o) \sim N(\mu_p, \sigma_p^2)$.

1. Draw the graphical model corresponding to the one-dimensional Kalman filter.
2. In this part we will walk through the derivation of the conditional distribution of $z_t|(x_0, \cdots, x_t)$.
   (a) How does the quantity $z_t|(x_0, \cdots, x_t)$ relate to $\alpha_t(z_t)$ and $\beta_t(z_t)$ from the forward-backward algorithm for HMMs? What is the operation we are performing called?
   (b) The distribution of $p(z_t|x_0, \cdots, x_t)$ will be Normal with a mean μ_t and a variance σ_t^2. We start our derivation of μ_t and σ_t^2 by writing:
   $$p(z_t|x_0, \cdots, x_t) \propto p(x_t|z_t)p(z_t|x_0, \cdots x_{t-1})$$
   What is the distribution of $p(x_t|z_t)$?
   (c) Suppose we are given the mean and variance of the distribution $z_{t-1}|(x_0, \cdots, x_{t-1})$ to be μ_{t-1}, σ_{t-1}^2. Find the distribution of $z_t|(x_0, \cdots x_{t-1})$
   Hint 1: You may use the fact that
   $$\int N(y - x; \mu_a, \sigma_a^2)N(x; \mu_b, \sigma_b^2)dx = N(y; (\mu_a + \mu_b), (\sigma_a^2 + \sigma_b^2))$$
   Hint 2: Start by marginalizing out over z_{t-1}.
   (d) Combine your answers from parts (b) and (c) to get a final expression for $p(z_t|x_0, \cdots, x_t)$. Feel free to use the result that the product of two Normal PDFs yields a Normal PDF. Report the mean μ_t and variance σ_t^2 of this Normal.
   Hint: Rewrite $N(x_t; z_t, \sigma_\gamma^2)$ as $N(z_t; x_t, \sigma_\gamma^2)$.
3. Interpret μ_t in terms of how it combines observations from the past with the current observation.

Solution

Your solution here.

Problem 2 (Policy and Value Iteration, 15 pts)

You have a robot that you wish to collect two parts in an environment and bring them to a goal location. There are also parts of the environment that you wish the robot avoid to reduce wear on the floor.

Eventually, you settle on the following way to model the environment as a Gridworld. The "states" in Gridworld are represented by locations in a two-dimensional space. Here we show each state and its reward:

R=4	R=0	R=-10	R=0	R=20
R=0	R=0	R=-50	R=0	R=0
START R=0	R=0	R=-50	R=0	R=50
R=0	R=0	R=-20	R=0	R=0

The set of actions is {N, S, E, W}, which corresponds to moving north (up), south (down), east (right), and west (left) on the grid. Taking an action in Gridworld does not always succeed with probability 1; instead the agent has probability 0.1 of "slipping" into a state on either side, but not backwards. For example, if the agent tries to move right from START, it succeeds with probability 0.8, but the agent may end up moving up or down with probability 0.1 each. Also, the agent cannot move off the edge of the grid, so moving left from START will keep the agent in the same state with probability 0.8, but also may slip up or down with probability 0.1 each. Lastly, the agent has no chance of slipping off the grid - so moving up from START results in a 0.9 chance of success with a 0.1 chance of moving right.

Also, the agent does not receive the reward of a state immediately upon entry, but instead only after it takes an action at that state. For example, if the agent moves right four times (deterministically, with no chance of slipping) the rewards would be +0, +0, -50, +0, and the agent would reside in the +50 state. Regardless of what action the agent takes here, the next reward would be +50.

In this problem, you will first implement policy and value iteration in this setting and discuss the policies that you find. Next, you will interrogate whether this approach to modeling the original problem was appropriate.

Problem 2 (cont.)

Your job is to implement the following three methods in file `homework6.ipynb`. Please use the provided helper functions `get_reward` and `get_transition_prob` to implement your solution. *Do not use any outside code. (You may still collaborate with others according to the standard collaboration policy in the syllabus.)*

Important: The state space is represented using integers, which range from 0 (the top left) to 19 (the bottom right). Therefore both the policy `pi` and the value function `V` are 1-dimensional arrays of length `num_states = 20`. Your policy and value iteration methods should only implement one update step of the iteration - they will be repeatedly called by the provided `learn_strategy` method to learn and display the optimal policy. You can change the number of iterations that your code is run and displayed by changing the `max_iter` and `print_every` parameters of the `learn_strategy` function calls at the end of the code.

Note that we are doing infinite-horizon planning to maximize the expected reward of the traveling agent. For parts 1-3, set discount factor $\gamma = 0.7$.

1a. Implement function `policy_evaluation`. Your solution should learn value function V, either using a closed-form expression or iteratively using convergence tolerance `theta = 0.0001` (i.e., if $V^{(t)}$ represents V on the t-th iteration of your policy evaluation procedure, then if $|V^{(t+1)}[s] - V^{(t)}[s]| \leq \theta$ for all s, then terminate and return $V^{(t+1)}$.)

1b. Implement function `update_policy_iteration` to update the policy `pi` given a value function `V` using **one step** of policy iteration.

1c. Set `max_iter = 4`, `print_every = 1` to show the learned value function and the associated policy for the first 4 policy iterations. Do not modify the plotting code. Please fit all 4 plots onto one page of your writeup.

1d. Set `ct = 0.01` and increase `max_iter` such that the algorithm converges. Include a plot of the final learned value function and policy. How many iterations does it take to converge? Now try `ct = 0.001` and `ct = 0.0001`. How does this affect the number of iterations until convergence?

2a. Implement function `update_value_iteration`, which performs **one step** of value iteration to update `V`, `pi`.

2b. Set `max_iter = 4`, `print_every = 1` to show the learned value function and the associated policy for the first 4 value iterations. Do not modify the plotting code. Please fit all 4 plots onto one page of your writeup.

2c. Set `ct = 0.01` and increase `max_iter` such that the algorithm converges. Include a plot of the final learned value function and policy. How many iterations does it take to converge? Now try `ct = 0.001` and `ct = 0.0001`. How does this affect the number of iterations until convergence?

3. Compare and contrast the number of iterations, time per iteration, and overall runtime between policy iteration and value iteration. What do you notice?

4. Plot the learned policy with each of $\gamma \in (0.6, 0.7, 0.8, 0.9)$. Include all 4 plots in your writeup. Describe what you see and provide explanations for the differences in the observed policies. Also discuss the effect of gamma on the runtime for both policy and value iteration.

5. Now suppose that the game ends at any state with a positive reward, i.e. it immediately transitions you to a new state with zero reward that you cannot transition away from. What do you expect the optimal policy to look like, as a function of gamma? Numerical answers are not required, intuition is sufficient.

Problem 2 (cont.)

Now you will interrogate your solution in terms of its applicability for the intended task of picking up two objects and bringing them to a goal location.

6. In this problem, we came up with a model for the problem, solved it, and then we had a policy to use on the real robot. An alternative could have been to use RL on the robot to identify a policy that achieved your objective. What is the value of the approach we took? What are some limitations (in general)?
7. Do any of the policies learned actually accomplish the task that you desired? What modeling shortcuts were made that result in some policies matching your true objective and some not?
8. Describe at least three modeling choices that were made in turning your original goal into this abstract problem, and potential implications of those choices.

Solution

Your solution here.

Problem 3 (Reinforcement Learning, 20 pts)

In 2013, the mobile game *Flappy Bird* took the world by storm. You'll be developing a Q-learning agent to play a similar game, *Swingy Monkey* (See Figure 1a). In this game, you control a monkey that is trying to swing on vines and avoid tree trunks. You can either make him jump to a new vine, or have him swing down on the vine he's currently holding. You get points for successfully passing tree trunks without hitting them, falling off the bottom of the screen, or jumping off the top. There are some sources of randomness: the monkey's jumps are sometimes higher than others, the gaps in the trees vary vertically, the gravity varies from game to game, and the distances between the trees are different. You can play the game directly by pushing a key on the keyboard to make the monkey jump. However, your objective is to build an agent that *learns* to play on its own.

You will need to install the `pygame` module (`http://www.pygame.org/wiki/GettingStarted`).

Task: Your task is to use Q-learning to find a policy for the monkey that can navigate the trees. The `homework6_soln.ipynb` file contains starter code for setting up your learner that interacts with the game. This is the **only code file** you need to modify. At the beginning of the code, you will import the `SwingyMonkey` class, which is the implementation of the game that has already been completed for you. Note that by default we have you import this class from the file `SwingyMonkeyNoAnimation.py`, which allows you to speed up testing. To actually see the game animation, you can instead import from `SwingyMonkey.py`. Additionally, we provide a video of the staff Q-Learner playing the game at `https://youtu.be/xRD6xBQbauw`. It figures out a reasonable policy in a few iterations. You'll be responsible for implementing the Python function `action_callback`. The action callback will take in a dictionary that describes the current state of the game and return an action for the next time step. This will be a binary action, where 0 means to swing downward and 1 means to jump up. The dictionary you get for the state looks like this:

```
{ 'score': <current score>,
  'tree': { 'dist': <pixels to next tree trunk>,
            'top': <height of top of tree trunk gap>,
            'bot': <height of bottom of tree trunk gap> },
  'monkey': { 'vel': <current monkey y-axis speed>,
              'top': <height of top of monkey>,
              'bot': <height of bottom of monkey> }}
```

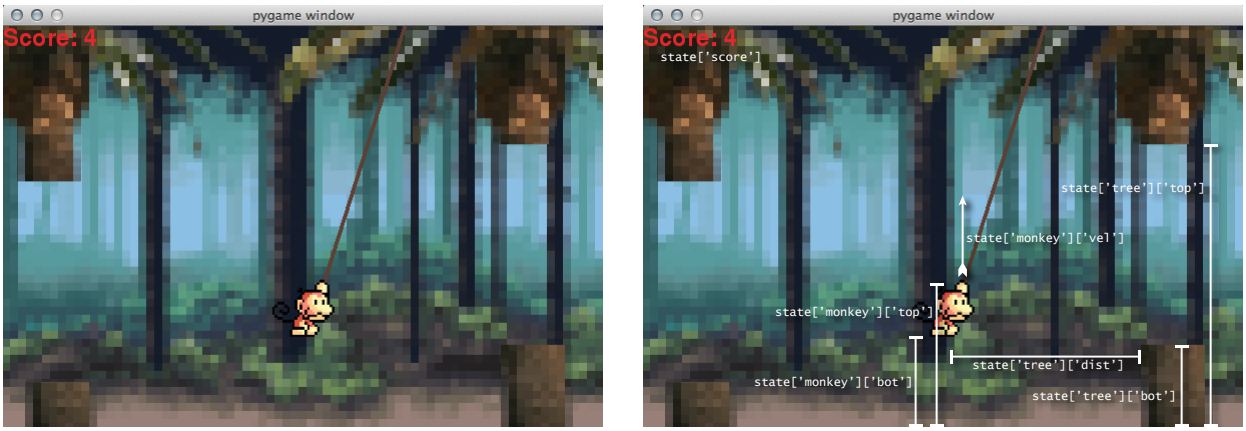
All of the units here (except score) will be in screen pixels. Figure 1b shows these graphically. Note that since the state space is very large (effectively continuous), the monkey's relative position needs to be discretized into bins. The pre-defined function `discretize_state` does this for you.

Requirements

Code: First, you should implement Q-learning with an ϵ-greedy policy yourself. You can increase the performance by trying out different parameters for the learning rate α, discount rate γ, and exploration rate ϵ. *Do not use outside RL code for this assignment.* Second, you should use a method of your choice to further improve the performance. This could be inferring gravity at each epoch (the gravity varies from game to game), updating the reward function, trying decaying epsilon greedy functions, changing the features in the state space, and more. One of our staff solutions got scores over 800 before the 100th epoch, but you are only expected to reach scores over 50 before the 100th epoch. **Make sure to turn in your code!**

Evaluation: In 1-2 paragraphs, explain how your agent performed and what decisions you made and why. Make sure to provide evidence where necessary to explain your decisions. You must include in your write up at least one plot or table that details the performances of parameters tried (i.e. plots of score vs. epoch number for different parameters).

Note: Note that you can simply discretize the state and action spaces and run the Q-learning algorithm. There is no need to use complex models such as neural networks to solve this problem, but you may do so as a fun exercise.



(a) SwingyMonkey Screenshot

(b) SwingyMonkey State

Figure 1: (a) Screenshot of the Swingy Monkey game. (b) Interpretations of various pieces of the state dictionary.

Solution

Your solution here.

Name

Collaborators and Resources

Whom did you work with, and did you use any resources beyond cs181-textbook and your notes?