CS/CE 352/368: Introduction to Reinforcement Learning: Assignment #2

Dr. M. Shahid Shaikh

Due on April 4, 2025, 11.59pm

Name & ID:

Overview

In this assignment, you will:

- Implement various reinforcement learning algorithms (value iteration, policy extraction, Monte Carlo methods) in a FrozenLake environment.
- Compare and contrast the performance of these algorithms in terms of convergence speed, accuracy, and applicability.
- Provide theoretical justifications and empirical evaluations of your implementations.

The part of the assignment is divided into four tasks which constitute 50% of the assignment grade. The remaining 50% of the assignment grade comes from the completion of Recitation Module 2, refer to this link.

Submission Guidelines:
Submit the entire HW as a zipped file containing the following:

1. Submit your solutions, including content, proofs, etc., as a latex pdf.
2. Submit the programming part as a copy of Rec_mod_2.ipynb file (this link).

Problem 1

(10 points) **Value-Iteration Vs. Policy-Iteration**

- (5 points) How do the convergence speeds (i.e., number of iterations) differ between the two algorithms in practice? Support your analysis with visual plots.
- (3 points) How does the overall computational complexity (in terms of state and action space sizes) differ between the two methods?
- (2 points) How do the intermediate policies generated during the iterative process compare between the two algorithms?

Solution (a)

From empirical results, we observe that Policy Iteration usually needs fewer iterations to converge but each iteration is computationally heavier due to full policy evaluation. Value Iteration has lighter iterations, but needs more steps to converge due to gradual updates.
(b)
If we look at the algorithm, we can see that policy iteration requires two computationally heavy steps. i.e Poliy Evaluation and the policy improvement while value iteration only requires a single step followed by determining deterministic policy. In terms of state and action spaces,
Policy Iteration
Let n be the number of convergence steps for policy evaluation, m be the steps it takes to obtain a stable policy, s, a be number of states and action space size respectively, then

$$T(n, s, a) = m * (T(policyeval) + T(PolicyImprov))$$

$$T(n, s, a) = m * O(ns^2) + O(as^2)$$

$$T(n,s,a) = max(O(mns^2), O(mas^2))$$

Value Iteration

$$T(n,s,a) = T(ValueIter) + T(OutputDetermeinsticPolicy)$$

$$T(n,s,a) = O(nas^2) + O(as^2)$$

$$T(n,s,a) = O(nas^2)$$

In terms of theoretical complexity, both algorithms are of the same order. Computationally, the policy iteration would require twice as much time as value iteration since it comprise of two steps instead of one. This however is mitigated by the fact that policy iteration typically takes fewer iterations than value iteration to converge making policy iteration generally faster than value iteration. This validates our empirical analysis above.
(c)
Policy Iteration
In policy iteration, each policy is strictly an improvement over (is better than) the previous policy. Hence, each steps leads towards a better policy. As a results, the policy coverges quicker and fewer iterations are needed. Because the policy is updated greedily, changes are sometimes abrupt. An entire row of actions might change at once.
Value Iteration In Value Iteration, since the value function is only partially improved in each step, the induced policies improve more slowly and may oscillate before settling. As a result, it typically requires more steps to converge fully to the optimal policy. Since Value Iteration makes incremental updates, intermediate policies might look almost random or suboptimal.

Problem 2

(10 points) **First-Visit Vs. Every-Visit**

- (4 points) Under what conditions do the two methods produce unbiased estimates of the value function, and are there any scenarios where one method might introduce bias over the other?
- (4 points) What role do correlations between returns (from multiple visits to the same state) play in the stability of the learning process for every-visit MC?
- (2 points) In empirical studies, under what circumstances does one method outperform the other in terms of convergence speed and final accuracy of the value function?

Solution

(a)
Both methods are unbiased under the following conditions:

- Episodes are generated following the same fixed policy. This ensures that the observed returns reflect the expected return under the target policy.
- Returns are averaged correctly i.e, each observed return contributes equally to the estimated value of a state.
- Episodes are independent and identically distributed. This ensures that each sample is a fair and independent representation of the policy's behavior.

Problem 2 continued on next page. . .

- All states are visited sufficiently often. This Ensures the law of large numbers can be applied for convergence.

The following conditions introduce bias in the estimation of value function of the two approaches.

- If each visit to a state in the same episode is treated as an independent sample without normalizing for multiple visits, the resulting value estimate may be skewed toward states that appear more frequently within episodes. This violates the assumption of equal contribution per episode.
- In environments where certain states appear multiple times in episodes due to loops or structure, Every-Visit MC can overweight those states unless proper normalization is applied.

(b)
Since in every visit algorithm, returns of repeating states in a single episode are highly corelated, they have an effect on the stability of the learning process.

- When a state is visited multiple times in a single episode, the corresponding returns used in Every-Visit MC are not independent. They are all derived from the same trajectory, and thus share common transitions, rewards, and outcomes. These returns are highly correlated, especially in deterministic or low-variance environments.
- Using multiple, highly correlated returns as if they were independent samples can lead to underestimation of variance in value estimates. In practice, this increases the variance of the learning process and can result in unstable or noisy value updates, especially in early stages with limited data.
- Correlated returns can lead to overrepresentation of frequently revisited states within episodes. This skews the learning process by giving disproportionate weight to these states, which may not accurately reflect their long-term value under the policy.
- Although Every-Visit MC updates more frequently, the redundancy of correlated returns means that not all updates contribute equally to learning. This can reduce the effective sample efficiency, making the method less stable compared to First-Visit MC in environments with frequent loops or revisits.

(c)
Theoretically, both methods First visit as well as every visit asymptotically converge to the optimal policy π despite their differences. Practically however, running algorithms for infinity is generally not possible which is why factors like convergence speed and accuracy matter.
Converge Speed
Every visit MC updates a state on every occurrence compared to first visit which only updates on the last occurrence of a state in an episode. As a result, every visit MC typically converges faster than first visit MC. It makes more frequent updates per episode, leading to quicker adjustments in the value estimates.

Accuracy
First-Visit MC often yields more stable and accurate estimates especially when the environment has repeated visits to the same state in an episode, such as in gridworlds with loops. They have an advantage over every visit MC where episodes are long and correlations between returns from the same episode can affect estimate quality.

From empirical results, Every-Visit MC tends to outperform in terms of convergence speed, particularly in stochastic environments with frequent state revisits. However, First-Visit MC often provides superior final accuracy in environments where intra-episode correlations are strong or episodes are long.

Problem 3

(25 points) **Dynamic Programming Vs. Monte-Carlo Methods**

- (10 points) Discuss and compare the two methods in terms of:
  - Convergence speed.
  - Computational complexity.
  - Practical applicability to different environments.

  Include visualizations (e.g., plots, heat maps, etc.) to support your analysis.
- (5 points) Monte Carlo methods are often subject to high variance because of sampling over complete episodes. How does this variance impact the stability and accuracy of the estimated value functions compared to the deterministic updates in DP methods?
- (5 points) Discuss possible ways to improve the performance of each method.
- (5 points) What trade-offs emerge in practice when choosing between DP and MC approaches for a given reinforcement learning problem?

Solution

(a)

Convergence Speed

- DP:
  1. DP methods like value iteration and policy iteration are known for converging quickly in terms of iterations, provided the environment's transition model is known.
  2. These methods update values or policies based on the entire state space at each iteration, which leads to rapid progress, especially in small to moderate state spaces.
  3. However, convergence speed can become slow for large state spaces as the number of states grows, requiring more iterations to update all states and becoming computationally expensive.
- Monte Carlo:
  1. MC methods, by contrast, rely on complete episodes of interaction with the environment before updating the value function. Therefore, MC convergence is typically slower than DP's, as it requires multiple episodes for each update.
  2. However, MC methods can be very efficient when the state space is large and the agent can explore extensively, as they don't require the entire model to be known upfront.

Computational Complexity

- DP:
  1. The computational complexity of DP methods is generally $O(s^2)$ for value iteration (where s is the number of states) and $O(s^3)$ for policy iteration (due to the need for solving a system of equations in each iteration).
  2. DP methods require the full model (transition probabilities and reward function) to compute updates, which makes them computationally intensive, especially for large state spaces or environments with a complex model.

Problem 3 continued on next page. . .

- Monte Carlo:
    1. MC methods do not need a model and only depend on experience gained through interactions with the environment. The complexity of MC methods is therefore mainly dependent on the number of episodes required for convergence.
    2. The per-step complexity of MC methods can be lower than that of DP, but because each episode must be completed before an update can occur, the overall learning can still be slow. Moreover, if a model is simulation based, then MC methods can incur high simulation computation costs

Practical Applicability to Different Environments

- DP:
    1. DP methods are ideal when the environment's model is fully known, including all state transitions and rewards. They are most effective in deterministic environments or when you have enough computational power to simulate every possible state transition.
    2. DP is also beneficial when the state space is relatively small, making it computationally feasible to compute the value for all states.
- Monte Carlo:
    1. MC methods are more suited for environments where the transition model is unknown or too complex to model explicitly. They can learn entirely from interaction, making them model-free and more flexible for real-world applications.
    2. MC methods are more effective in stochastic or partially observable environments, as they only require experience from episodes, not a full model.

(b)
Monte Carlo (MC) methods and Dynamic Programming (DP) methods differ fundamentally in how they estimate value functions. MC methods rely on sample-based returns from complete episodes, while DP methods rely on expectations computed using a known transition model. As a result, Monte Carlo methods typically exhibit higher variance, which affects both the stability and accuracy of value function estimates when compared to the deterministic updates used in DP.
Monte Carlo methods estimate the value function V(s) using the empirical average of returns observed from sampled episodes. These returns can vary significantly due to stochastic transitions and rewards in the environment, different trajectories resulting from exploration, sparse or delayed rewards, especially in long episodes. This introduces high variance in the estimates, especially when the number of episodes is small.
Impact on Stability
Monte Carlo methods are inherently less stable than DP methods during learning because each update is based on a potentially noisy sample. This instability can lead to fluctuating value estimates, particularly early in training or when episodes vary widely in length and outcome. In contrast, DP methods perform deterministic updates by computing expected values using the known model of the environment, leading to smooth and stable convergence.
Impact on Accuracy
High variance in MC methods can lead to slower convergence and inaccurate value estimates in the short term. However, as the number of episodes increases and more samples are collected, the law of large numbers ensures that MC estimates converge to the true value function $V_{(s)}$, assuming sufficient exploration. DP methods on the other hand given accurate model information, can compute exact value functions with no variance, making them more accurate and data-efficient in settings where the model is known.

The high variance in Monte Carlo methods can significantly reduce the stability and short-term accuracy of value estimates compared to Dynamic Programming. However, with sufficient sampling, MC methods can

Problem 3 continued on next page. . .

still provide unbiased and consistent estimates. In contrast, DP methods achieve more stable and accurate estimates by leveraging complete knowledge of the environment. Therefore, while DP is preferable in model-known settings, Monte Carlo remains a practical alternative in model-free scenarios, albeit at the cost of increased variance.

(c)

Improving Monte Carlo Methods

- **Variance Reduction Techniques**:
  When using off-policy MC methods, importance sampling can be used to correct for the difference between behavior and target policies.
- **Bootstrapping Hybrid Approaches**:
  Methods like TD() or Eligibility Traces combine the long-term information of MC with the lower variance of TD methods. These bootstrapped MC methods can provide faster convergence and improved stability.
- **Using Exploration Strategies**:
  Using -greedy, softmax, or Upper Confidence Bound strategies to ensure adequate exploration, especially early in training can mitigate the bias.
- **Long runs and sufficient sampling**: With sufficient sampling, MC methods can still provide unbiased and consistent estimates. Hence running MC for higher number of episodes, often in parallel can improve accuracy.

Improving DP Methods

- **Value Function Approximation**:
  Use of function approximators to generalize across large or continuous state spaces. This leads to Approximate Dynamic Programming methods that are more scalable.
- **Prioritized Sweeping**:
  Rather than updating all states uniformly, prioritize updates for states whose values are likely to change significantly. This reduces unnecessary computation and improves convergence speed.
- **Asynchronous and Real-Time DP**:
  Update only a subset of states in each iteration in an asynchronous manner (like real-time dynamic programming), which makes DP more efficient in large-scale problems.
- **Efficient Data Structures**:
  Use of sparse matrices, hash tables, or tree-based representations for state and transition spaces to reduce memory usage and speed up lookup and update operations.

(d)

DP is ideal when you have a known model of the environment, and computational resources allow processing the state space thoroughly. It's more efficient when dealing with deterministic environments and when high-quality updates per iteration are required. Monte Carlo based RL is better when you don't have a model of the environment, need to learn directly from experience, and can afford slower learning with potentially less computational overhead.

1. Model Requirements

- **DP:** DP methods, such as value iteration and policy iteration, require a known model of the environment, specifically the transition probabilities and reward function. This makes them impractical when the model of the environment is unknown or hard to obtain.

Problem 3 continued on next page. . .

- **MC:** MC methods do not require knowledge of the transition model. They learn purely from experience, making them more suitable for environments where the model is unknown or too complex to represent explicitly.

2. Sample Efficiency

- **DP:** DP methods are sample-efficient in terms of the number of updates required because they use the full model and can compute updates deterministically. However, they require full knowledge of the environment, which is often not feasible in practice.
- **MC:** MC methods tend to be less sample-efficient since they require generating complete episodes (trajectories) to update values. The learning process can take longer, especially when episodes are long, and might not generalize well from sparse data.

3. Computation Complexity

- **DP:** The computational complexity of DP methods is generally high, especially for large state spaces, since they involve iterating over all states and applying updates. For problems with large or continuous state spaces, DP can be computationally prohibitive.
- **MC:** MC methods can be computationally less intensive in terms of individual updates, but because they require generating and processing entire episodes, they can still be slow, especially in large state spaces.

4. Convergence

- **DP:** DP methods converge more quickly (in terms of iterations) and are guaranteed to converge to the optimal solution, provided the model is accurate and all states are fully explored.
- **MC:** MC methods require complete episodes for updates, and thus convergence can be slower, especially if the exploration of the state space is inefficient or not thorough. However, they are also guaranteed to converge to the true value function under certain conditions.

5. Exploration and Generalization

- **DP:** DP assumes the agent knows the model and can compute all state transitions, so exploration in the environment is not inherently a concern. It often works better when the environment is fully known and can be simulated.
- **MC:** MC methods require exploration and are better at handling problems where exploration is part of the learning process. However, the exploration strategy (e.g., epsilon-greedy) can significantly impact performance, and poor exploration can lead to suboptimal learning.

6. Applicability to Online Learning

- **DP:** DP is typically more suited to offline learning, where you have a model of the environment to process all transitions and rewards before updating. Online learning with DP can be more difficult and requires full model access.
- **MC:** MC methods are inherently better suited for online learning because they can update their value estimates incrementally as episodes are completed, without needing the full model of the environment.

7. Suitability for Partially Observable or Stochastic Environments

- **DP:** DP methods require full observability and are most effective in deterministic or fully observable environments. They struggle with partially observable or highly stochastic environments unless modified (e.g., with Partially Observable Markov Decision Processes or stochastic DP).

Problem 3 continued on next page. . .

- **MC:** MC methods are generally more robust to partially observable environments, as they only need experience in the form of episodes to learn from. They can handle stochasticity as long as sufficient exploration is done.

8. Real-Time Performance

- **DP:** In environments where decisions need to be made in real-time, DP may be too slow or computationally expensive, especially with large state spaces. This is due to the need to calculate updates for all states iteratively.
- **MC:** MC methods can sometimes provide faster, real-time updates after each episode, making them more suitable for situations where quick feedback is needed, even if the trade-off is in learning efficiency.

Problem 4

(5 points) **Assignment Feedback** Provide one or two line response

- How clear were the objectives and topics covered in the recitation module (e.g., dynamic programming and Monte Carlo methods)? Answer: The topic were all covered, however there were errors in boiler plate code given. For instance, the balck jack code doesnt follow discrete spaces, but the given codes were written considering it as a discrete space env.
- Were the concepts explained in a way that helped you understand the material? If not, what could be improved?
  Answer: Yes. they were.
- How well did recitation module 2 prepare you for this part of the assignment tasks?
  Answer: It provided empirical insights thorugh teh results which helped validate thoertical concepts.
- Do you prefer this style of assignment, i.e., assignment being paired with Lab style Recitation Module 2 in 50% and 50% division?
  Answer: Yes, I think some coding part is good.
- How many hours did it take to complete the assignment?
  Answer: 8 hours
- How difficult did you find the complete assignment?
  Answer: Debugging the boiler plate code was difficult.