Summary - Chapter 1

Ameet Deshpande

November 24, 2017

1 Summaries

1.1 Section 1.1

- Reinforcement Learning is more goal directed than other Machine Learning approaches
- The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning.
- The exploration exploitation tradeoff is a common trade off in Reinforcement Learning.

1.2 Section 1.2

Mainly examples

1.3 Section 1.3

- The four main things that characterize problems are *policy, reward function, value function* and optionally a *model* for the environment. Policies can also be stochastic just like the reward function.
- Reward function indicates what is good in the immediate sense and the value function indicates what is good in the longer run as it takes into account reward functions from multiple states. Thus action choices are usually made based on value functions rather than rewards.
- The model of the environment is used to estimate things like that the next reward and the state will be, given the current state and action that is being taken.
- Reinforcement Learning methods are closely related to dynamic programming methods and state space planning methods.

1.4 Section 1.4

- Minimax solution makes assumptions about the play of the opponent. It would never reach a state from where it can lose, even if it has won all the games from there on.
- Most of the times we move greedily and occasionally we choose an exploratory move.
- Temporal difference can be used with an appropriate step-size parameter α which controls how the averaging is done. If the parameter is reduced appropriately over time, the value functions converge. If the step-size parameter is not reduced all the way to 0, the agent can also play well against an opponent who changes with time. For example, the case of keeping α a constant.
- In games which have a large number of states, how well a model performs depends on how well it can generalize on from past experiences. It should be able to use past experience from different states to be able to take actions in states it has probably not seen before. Symmetrical positions in Tic-Tac-Toe board could be one example.

1.5 Section 1.5

Summary section

2 Exercises

2.1 Exercise 1.1

Self Play. **Suppose, instead of playing against a random opponent, the reinforcement learning algorithm described above played against itself. What do you think would happen in this case? Would it learn a different way of playing?**

The first thing to note is the fact that the agent is not playing against a fixed opponent. Given that the same algorithm is being used to train both the agents, it is highly likely that the agents learn very similar policies. The agents will be playing greedy most of the times.
In usual cases when the agent is playing with an opponent, it involves exploiting patterns of the opponent (which may or not change over time). This is the reason the RL algorithm is different from the classical *minimax* algorithm. But since it is playing against itself now, the second agent takes a move which maximizes its probability of winning. It does not follow some inbuilt pattern or anything like that. Thus both agents will end up learning policies very similar to the *minimax* algorithm.
Exploration does not help in this case either. This is because, taken a non-optimal move will most likely end up in the other agent winning all the time. To show this, consider an ϵ greedy method. If an agent takes a non-optimal move, the probability that the other agent also takes a non-optimal move is lesser. Thus, exploration will end up in a loss.

2.2 Exercise 1.2

Symmetries. **Many Tic-Tac-Toe positions appear different but are really the same because of symmetries. How might we amend the reinforcement learning algorithm described above to take advantage of this? In what ways would this improve it? Now think again, suppose the opponent did not take advantage of symmetries. In that case, should we? It is true then that symmetrically equivalent positions should necessarily have the same value?**

It is an easy problem to recognize what states are symmetrical. In the original problem, we were working with a set of states say, s. We can establish symmetries between states and keep that information in disjoint set data structure or any other data structure like a *set* in python. To take advantage of the symmetries, we can change the algorithm to make updates to all the symmetrical states instead of just the single state which had been encountered. We could also make another modification. We could change the $Q(s, a)$ value for the encountered state and for all its symmetrical states, we could change that value by a decayed factor of γ. This can be beneficial if the opponent is not taking advantage of symmetries.
The advantage of this algorithm is that the agent will learn much quicker than the other algorithm. This is because the generalization power is higher.
If the opponent does not take into account the symmetries, it may not be the best thing for us to take it into account. But we could introduce γ as a tuneable parameter. Setting γ to 0 means that no symmetry information is being used. Setting it to 1 would mean all the information is being used. But if the agent we are playing with foes not exploit symmetries, it might be beneficial for us to not exploit symmetries. For example, say that there are two symmetrical states s_1 and s_2. If the opponent always wins when they are in state s_1 and loses when in s_2, we might not want the Q values for our agent to be the same for our agent. This allows the agent to learn opponent specific policies.
Thus, symmetrically equivalent positions need not necessarily have the same value.

2.3 Exercise 1.3

Greedy Play. **Suppose the reinforcement learning player was greedy, that is, it always played the move that brought it to the position that it rated the best. Would it learn to play better, or worse, than a non-greedy player? What problems might occur?**

In general the player will play worse than a non-greedy player. If any action gives a positive reward, the agent will keep choosing that action to play the game even if there is a more optimal move. For example, say there are two moves m_1 and m_2. The first one results in a win in 10 moves and the second one wins in 1 moves. If the agent chooses the first move first and finds out that the reward is positive, it will keep choosing that same move as there will always be positive rewards and it will never explore to find out if there is a better move like m_2.

The agent will also play bad against opponents which change their playing strategy over time. This is because the agent does not explore and is thus stuck with the same moves it think is optimal for it. Since the current move is still giving positive (yet probably much lower than optimal) rewards

it will keep choosing that move.

2.4 Exercise 1.4

Learning from Exploration. **Suppose learning updates occurred after all moves, including exploratory moves. If the step-size parameter is reduced over time appropriately, then the state values would converge to a set of probabilities. What are the two sets of probabilities computed when we do, and when we do not, learn from exploratory moves? Assuming that we do continue to make exploratory moves, which set of probabilities might be better to learn? Which would result in more wins?**

The update is usually of the form,

$$V(s) \leftarrow V(s) + \alpha \times (V(s^f) - V(s))$$

where α is the step size parameter and s^f is the next state that the agent reached. In section 1.4, the book mentions that the values are backed up when the move taken after reaching that state is a greedy one. Basically, if the learning updates happened only after greedy moves were taken, the probability values that the states converge to will be the optimal values for that state. This does not however mean that learning does not happen through exploration. Consider a state s_1 and an optimal move e^* and another move e. Taking the move e from the state s_1 leads the agent to state s_2. If an optimal move is taken from the state s_2, then the values are backed up for state s_1. If however a non optimal value is taken from s_2, then s_2 will not be considered for learning updates. This makes sense because we want to learn in steps. When we are learning through exploration for a state s_1, we want everything else to be optimal.

Thus, when we are making updates only after greedy moves, the probability values for all the states will be the optimal values. However, assuming that the rate of exploration remains the same throughout the game, using exploratory moves also for learning updates will give the expected value after considering the exploratory moves also as the probability value. This may not be desirable. Consider the following figure which shows when and when not the updates are made.

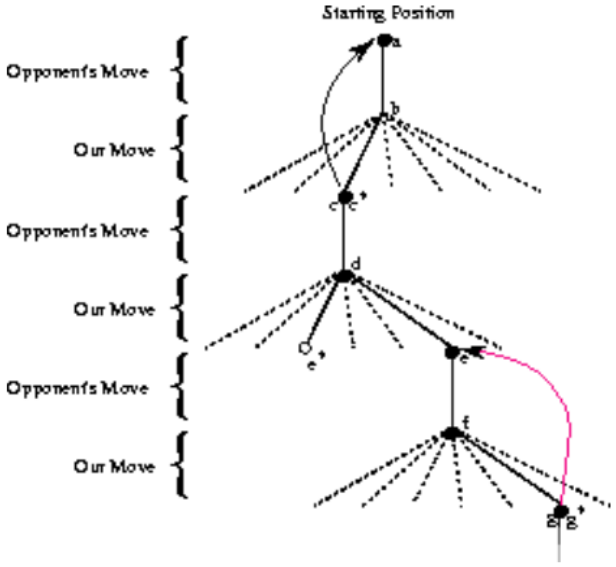

Figure 1: Updating Strategy

As explained before, we would expect the strategy which uses only the greedy moves for update to do better, all other things equal.

2.5 Exercise 1.5

Other Improvements. **Can you think of other ways to improve the reinforcement learning player? Can you think of any better way to solve the Tic-Tac-Toe problem?**

- There is no reward for finishing the game in lesser number of moves. Instead of rewarding only based on if the agent wins the game or not, the reward could also be based on if the number of moves taken by the agent is lesser.
- Using symmetries along with another hyperparameter as discussed in the second question.
- We could alter the exploration rate of the agent based on the variance in the opponent's moves. If the opponent is playing the same move again and again, exploration will actually end up losing the game.
- For an opponent who is changing with respect to time, it could be beneficial to decay the learning update rule. This would result in faster improvement.