Reinforcement Learning

1 Markov Decision Processes

Reinforcement learning is similar to supervised learning however the generated outputs and their correctness are not the direct feedback given to the algorithm so it can improve.

1.1 Markov Decision Processes

We begin with our states S which represent the possible configurations of the agent and the world.

Actions are things that can be done at a particular state (edges connecting states). Given as A or as $A(s)$ where $A(s)$ represents the actions of the particular state s.

The model also known as the transition function or transition model, can be considered the physics of the world. $T(s, a, s')$ gives us the probability $\Pr(s' \mid s, a)$. That is the probability of transitioning to state s' given we began at state s and took action a.

Markovian property is that only the present matters, we only have to condition on the current state. We can turn any process into a Markovian process by forcing each state to remember the required context. This is similar to the method of turning a FSM into a DFSM.

The transition model is stationary and does not change.

Reward encompasses the domain knowledge and there are three main definitions $R(s)$, $R(s, a)$ and $R(s, a, s')$. All of the variations are mathematically equivalent, just indicate the specificity and granularity.

All of these factors together create a Markov Decision Problem.

A policy is the solution, that is given a state s it generates the action to take a. The policy is written $\pi(s) \to a$.

The policy that maximizes the reward function is written $\pi*$ and optimizes the reward.

MDP does not require an explicit termination point.

MDP is plan-blind and only works from the current state of the world. It uses policies which can be used to infer plans but reflects all possible states in the world. Furthermore, it is resilient to stochasticity.

Delayed rewards are based on the idea that taking an immediate action does not result in receiving a distinction whether that action was correct or not immediately. That is there is a sequence of steps to be taken before a reward is offered or denied.

Choosing which path in an MDP is the best is known as the temporal credit assignment problem.

Setting the reward to a negative value except for the terminal states, results in an optimization for the shortest path.

Minor changes to the reward function are important due to the multiplicative impact of stochasticity.

Rewards can be considered the teaching signal and act as domain knowledge.

Infinite horizons are represented by MDP that allow for infinite sequences of moves with no direct termination as a result of the number of moves taken.

With limited scope it is possible that risk/reward analysis results in different policies for the same given state. This idea can be represented by $\pi(s,t) \rightarrow a$. However this information can be encoded into the world directly by mapping it into the physics. That is, the 'separate' aspect method does not offer is not capable of a wider degree of representations, but that it may allow for those representations to be encoded with greater accessibility and more minimal scope/computation. (Consider FSM and DFSM).

The differential of the utility of sequences is independent of their common prefix states. That is given two sequences A, B where $A = (s_0, s_1, s_2, \ldots)$ and $B = (s_0, s'_1, s'_2, \ldots)$ if $U(A) > U(B)$ then $U(A/s_0) > U(B/s_0)$. This is referred to the stationary preferences. This follows from adding rewards.

We can define the utility of a sequence of states is defined $U(s_0, s_1, s_2, \ldots) = \sum_{t=0}^{\infty} R(s_t)$. Infinite sequences however, degrade the quality of this definition by making no difference between two functions that both asymptotically trend towards infinity. We can change this definition to reflect those circumstances, that is we can define $U(s_0, s_1, s_2) = \sum_{t=0}^{\infty} \gamma^t R(s_t)$ where $0 \leq \gamma < 1$. This forces the reward function to converge over infinite steps. This is called discounted sums.

It is a geometric series that is bounded $U(S) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1-\gamma}$. This is alternatively known as a discounted series that is infinite in length but returns a finite result.

The optimal policy is defined $\Pi^* = \text{argmax}_{\pi} E[\sum_t \gamma^t R(s_t) \mid \Pi]$. The utility of a particular state, based on the optimal policy is designated U^* and the utility of a particular state is defined as $U^*(s) = E[\sum_{t=0}^{\inf} \gamma^t R(s_t) \mid \Pi, s_0 = s]$ which says that the expected state of states that will be seen given that policy and state.

The reward for entering a state s_i defined $R(s_i) \neq U^*(S_i)$. Utility is defined as long term feedback,

whereas rewards are the short-term version. The utility measures the future reward to be obtained given a specific policy, its a measure of delayed reward.

The optimal policy at a given state is defined as $\Pi^*(s) = \text{argmax}_a \sum_{s'} T(s,a,s)U(s')$ where $U(s) = U^*(s)$. Then $U(s) = R(s) + \gamma \text{max}_a \sum_{s'} T(s,a,s')U(s')$, also known as the Bellman equation.

Solving for the Bellman equation is difficult, due to non-linearity as a result solving based on the nearby utilities progressively is the solution. This update equation is defined $\hat{U}_{t+1}(s) = R(S) + \gamma \text{max}_a \sum_{s'} T(s,as')\hat{U}_t(s')$. Using the Bellman update equation spreads out the reward $R(s)$ via a contraction proof via value iteration.

When considering the correct policy it is important to remember that the function of Π or the optimal policy is not a function of the utility U but of the rewards and state transitions. As a result, a policy may not get the utility information of each state correct, but so long as the orer is correct (in terms of states and transitions then this is sufficient).

We can then find the best policy by an iterative improvement in the policy. This begins with an arbitrary policy Π_0, which is then improved via the bellman equation. However, using a fixed policy prevents the introduction of the argmax function which caused nonlinearity earlier. Thus the utility equation is as follows $U_t(s) = R(s) + \Gamma \sum_{s'} T(s, \Pi_t(s), s')U_t(s')$. This function $U_t = U^{\Pi_t}$. As the series of n equations is linear it can be solved via linear algebra. The improvement step follows $\Pi_{t+1} = \arg\max_a \sum T(s,a,s')U_t(s')$. This process is known as policy iteration. The time of the algorithm is typically $O(n^3)$. This process is guaranteed to converge via the fact that there is a finite number of policies.

MDPs act as a foundation for reinforcement learning but are not reinforcmenet learning in it of themselves.

2 Reinforcement Learning

There are two major versions of determining policies:

1. Planning is the process of taking in a model and then returning a policy.
2. Reinforcement learning is the process of being given a number of state transitions and then learning from the optimal policy given those transitions.

Reinforcement learning is really reward maximization. RL is responsible for strengthening the response to the stimulus.

We can further deconstruct the reinforcement learning into two smaller components;

1. Given a set of transitions a modeler builds a model from it.
2. Giben a model a simulator generates a set of transitions from it.

Depending on the algorithm chosen and the policies, simulating can actually be highly expensive due to combinatorial explosion.

Value iteration and policy iteration are types of planning algorithms.

Model based reinforcement learning is defined as

$$\text{transitions} \rightarrow \text{modeler} \rightarrow \text{model} \rightarrow\rightarrow \text{Planner} \rightarrow \text{policy}$$

.

Reinforcement based planners are defined as

$$\text{model} \rightarrow \text{Simulator} \rightarrow \text{transitions} \rightarrow \text{Learner} \rightarrow \text{policy}$$

.

Reinforcement learning on policies directly is known as policy search. However, it suffers from issues with the temporal credit assignment problem. That is, because feedback is only recieved at the end of a series of actions and not during that then it is extraordinarily difficult to determine the next best action to take. This is overcome by changing the formulation of the problem to MDPs.

RL on the utility function is known as value-based approaches. It is difficult to transfer utility into policies direct using the bellman equations so itermediary methods are used.

RL using the transition functions are the rewards function is known as model based RL. It is possible to use value iteration to solve the bellman equations.

Encoding in desired actions taken into the Bellman equations results in the following function $Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$. That is the equation maximizes the utility of the next step given a specific state and action. It allows for the comparison of different actions.

Q learning is done by computing the Q of given values, we can define $U(s) = \max_a Q(s,a)$ simply by always taking the best action and further we can define $\Pi^*(s) = \text{argmax}_a Q(s,a)$. This technique where we solve for the Q is known as Q-learning which the evaluation of the Bellman equations from data.

Q-learning is difficult as there is no immediate access to $T(s,a,s')$ or the reward function $R(s)$. A transition is encoded as $< s,a,r,s' >$ where s is the state, a is the action taken, r is the reward given for taking an action and s' is the state after the action is taken. We use an estimate of the Q function which is defined $\hat{Q}(S,a) \xrightarrow{\alpha} r + \gamma \max_{a'} \hat{Q}(s',a')$ where α is the learning rate. $\hat{Q}(s,a)$ can be considered the utility of the current state.

The learning rate shown as $v \xleftarrow{\alpha} x$ is actually represented as $v \leftarrow (1-\alpha)v + \alpha x$.

Using a decaying learning rate, the supposition is that the Q function computes $E[r + \gamma \max_{a'} Q(s',a')]$ which can then be written as $R(s) + \gamma E_{s'}[\max_{a'} Q(s',a')]$ which further reduces to $R(s) + \gamma \sum_{s'} T(s,a,s') \hat{Q}(s',a')$. However due to the $\hat{Q}$ changing over time, this only works as a generalized explanation and does not work as a theorem of the convergence.

Given an arbitrary $\hat{Q}$, and the update function $\hat{Q}(s,a) \overset{\alpha_t}{\leftarrow} r + \gamma \max_{a'} \hat{Q}(s',a')$. Then using the update function we have $\hat{Q}(s,a) \rightarrow Q(s,a)$ if s,a are visited infinitely often and $\sum_t \alpha_t = \inf$, $\sum_t \alpha_t^2 < \inf$, s' $T(s,a,s')$ and r $R(s)$. That is a the function uses a decaying learning rate, next state s' is drawn from the state transition function and the next reward is drawn from the reward function. Then we have a solution to the Bellman equations.

Q-learning is actually a family of different algorithms wherein the initial $\hat{Q}$, decay rate of α and how actions s' are chosen differently. The policy of choosing actions is incredibly important for how the algorithm will behave. It is important to choose an action choice policy that is resistant to the arbitrary $\hat{Q}$. The greedy policy (using $\hat{Q}$) finds the local minima.

There are several ways of overcoming the greedy minima. Simulated annealing (that is taking a random action sometimes) can help to overcome these difficulties. The resultant action choosing policy $\hat{\Pi}(s) = \text{argmax}_a \hat{Q}(s,a)$ with probability $1-\epsilon$ and $\hat{\Pi}(s) =$ random action otherwise (that is probability ϵ). MDP state-action pairs that don't link to any other aren't considered in the progress and the simulated annealing as they are unreachable. Therefore it is important to start in the subgroup of states that leads to a final exit solution. However, this problem of finding a path is a complicated problem in it of itself and is a graph search problem with an exponential search factor.

If the exploration is greedy in the limit with infinite exploration it refers to Q-learning with a decaying ϵ much like a decaying learning rate. Using GLE we find that $\hat{Q} \rightarrow Q$ and $\hat{\Pi} \rightarrow \Pi^*$.

The explanation-exploitation dilemma wherein there is a trade-off between learning and use of that learning results from there being a singular agent doing the exploitation and explanation.

Encoding previous learning is possible for the learning algorithm and improve its behavior. The EED is a fundamental tradeoff in RL. There is an inherent link between model learning and planning.

By setting the initial $\hat{Q}$ high, then the algorithm will search more readily and is known as optimism in the face of uncertainty.

Generalizing MDPs allows for function approximation.

3 Game Theory

Game theory can be defined as the mathematics of conflict. That is moving from single agents to multiple agents.

Other agents can be considered in the transition model, however it is more effective to consider them explicitly.

A zero sum game is a game in which the sum of each reward state is always constant.

A strategy is a mapping of all possible states to actions.

Pure strategies are strategies that provide a complete definition of how a player will play in a game.

The matrix form of a game encodes the games outcomes as a result of any and all strategies played by all players and contains the complete outcome and understanding of the entire game space.

In a two player game, a player who goes first needs to consider the worst case counter strategy. If the two players have opposite winning conditions where one's gain is another's loss. Minimax is the way in which the game is played out by both players, the minimax strategy results in the game's value which is the output cell of the minimax strategy. Formally in a 2-player, zero-sum deterministic game of perfect information Minimax $\equiv$ Maximin and there always exists an optimal pure strategy for each player.

In games, agents try to find the strategy that maximizes reward with the understanding that all other agents are also trying to maximize their reward at that agents expense.

Von Neumann's Theorem: For 2-player zero-sum non-deterministic game of perfect information Minimax $\equiv$ Maximin and there exists an optimal pure strategy for each player.

A game of hidden information is one in which all of the states are not known to each player.

In games of hidden information Von Neumann's theorem breaks down and Maxamin $\neq$ Minimax.

Mixed strategies are used instead of pure strategies which can overcome the definition of consistency defined in perfect information games. A mixed strategy is one where the optimal strategy is not a single option from the strategical combinations but of a probability distribution over the strategies.

Mixed strategies result in a distribution over the probability of p wherein a player picks strategies with probability p. The center of the game or the 'value' of the game is the location wherein the strategical functions overlap. There are cases where the functions of the probability distribution system do not overlap. In these cases the max of the lowest line is the center of the game.

2-player non-zero sum, non-deterministic games of hidden information have several different types but one classical example is the Prisoner's Dilemma.

The game generalizes into Nash Equilibrium, whereby n players, with strategies $S_1, S_2, \ldots S_n$ are the strategy sets from which players draw. The situation wherein $S_1^* \in S_1, S_2^* \in S_2, \ldots S_n^* \in S_n$ are a nash equilibrium if and only if $\forall_i S_I^* = \text{argmax}_{s_i} \text{utility}(S_1^*, ldots S_i^* \ldots S_n^*)$. The nash equilibrium refers to the idea that no player has any reason to change their strategy. Nash equilibriums can be applied to pure and mixed strategies, the mixed version deals with probabilities over the strategy distributions.

In the n-player pure strategy game, if elmination of strictly dominated strategies eliminates all but one combination, that combination is the unique Nash Equilibrium. Furthermore, in multi-round games elimination is done until no more is possible, then on the next round the process is continued making NE an iterable process.

Any NE strategies will survive iterated elinimation of strictly dominated strategies.

If n is finite and $\forall_i S_i$ is finite then $\exists$ (mixed) NE. That is there is always a NE of some variety.

Iterated games create different dominant strategies than singular rounds wherein the dominant strategy may fall to a different NE than the initial setup of the NE.

Symmetric games suffer from the tit-for-tat strategy more strongly than other types of games.

A way of evaluating the NE of a many step iterated game is to start at the last game and determine the value of that game, then to work your way back over each game prior and then via proof by induction the games' values are built back to the original game. Thus via an iterated game is the NE of a single game. We can do this via the 'sunk cost fallacy'.

Given a game where the NE is x, n repeated iterations of the game will result in n repeated same NEs. This theorem holds for a single NE.

The matrices of the games includes all representative information encoded with it of the future, past and present. Re-encoding of the game can be done to cause reevaluation of the game conditions which causes the values of certain game cells.

Mechanism design is the process of evaluating and designing incentives to get particular behavior, ie re-encoding the game.

4 More Game Theory

A generalization of MDPs and iterative games is stochastic games (Markov) games and provide a model for multiagent RL.

The NE of a SG (stochastic game) is a pair of policies where neither wants to deviate.

Stochastic games are invented by Shapley and encode the following items:

- States S
- Actions for player i, A_i
- Transitions $T(s,(a,b),s')$
- Rewards for player i, R_i where $R_1(s,(a,b)), R_2(s,(a,b))$
- Discount γ

Differing definitions of MDPs and SG include the discount others however do not include them.

SG is a generalization of MDPs.

Zero-sum SG is based can be extrapolated from MDPs and we can use a generalized version of the Bellman equation to accomplish this.

We begin with the Bellman equation $Q_i^*(s,(a,b)) = R_i(s_i(a,b)) + \gamma \sum_{s'} T(s,(a,b),s')\max_{a',b'} Q_i^*(s',(a',b'))$. The problem with the initial formulation for SG is that the $Q_i^*(s',(a',b'))$ assumes that every joint action is designed to benefit the maximizer, which does not hold for SG (holds for single player games).

By a modification to the Bellman equation so that the next step assumes a zero sum game we wind up with the modification $Q_i^*(s,(a,b)) = R_i(s_i(a,b)) + \gamma \sum_{s'} T(s,(a,b),s')\text{minmax}_{a'b'} Q_i^*(s',(a',b'))$. The traditional zero-sum game is two player. A three player zero sum game is actually just a general sum-game. Thus Q-learning can be done iteratively whereby for the state $< s,(a,b),(r_1,r_2),s' >$: $Q_i(s,(a,b)) \overset{\alpha}{\leftarrow} r_i + \gamma \text{minimax}_{a'b'} Q(s',(a',b'))$. This is alternatively known as the minimax-Q equation.

There are a number of benefits to this modification.

- Value iteration works.
- Minimax-Q converges under the same conditions of Q-learning.
- The solution to Q^* is unique.
- The policies pursued by two agents running the minimax-Q will converge, seeing as there is one unique solution to Q^* and can be computed independently.
- The update is polynomially efficient.
- Q^* is sufficient to specify a policy.

Essentially by using minimax-q we gain all the benefits of Q-learning except for the primary disadvantadge that due to the complication of another agent, it is not possible to solve the MDP itself in polynomial time. Value iteration will get us close but is close to simulated annealing rather than an actual solution to the game created by the MDP.

General sum games are games where any number of players can be involved and any reward distribution schema. We generalize Q-learning to these circumstances by introducing the Nash Equilibrium into the iteration equation. $Q_i^*(s,(a,b)) = R_i(s,(a,b)) + \gamma \sum_{s'} T(s,(a,b),s')\text{NE}_{a'b'} Q_i^*(s',(a',b'))$ And then generating the Q-learning function we have for the state $< s,(a,b),(r_1,r_2),s' >$ that $Q_i(s,(a,b)) \overset{\alpha}{\leftarrow} r_i + \gamma \text{NE}_{a'b'} Q_i(s',(a',b'))$, which is known as Nash-Q.

However with Nash-Q, value iteration doesn't work, and Nash-Qs don't converge, there aren't unique solutions to Q^*, policies can't be determined independently. The update is not efficient and is in the class $P = PPAD$. Sadly and finally Q functions are not sufficient to specify policy.

There are a number of search/rank techniques:

- Repeated stochastic games resulting in 'folk knowledge'.

- Cheap-talk (non-binding communication), resulting in a correlated equilibria.
- Cognitive hierarchy, assume that other players don't have full computational resources then can achieve the best response.
- Side payments (coco values) which allows for balancing for the zero-sum with mutual