Literature review: Reinforcement Learning

Daniel Hernandez

0.1 Markov Decision Processes

subfiles

Bellman (1957) introduced the concept of a Markov Decision Process (MDP) as an extension of the famous idea of Markov chains. Markov decision processes are a standard model for sequential decision making and control problems. An MDP is fully defined by the 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}(\cdot|\cdot,\cdot), \mathcal{R}(\cdot,\cdot), \gamma)$. Whereby:

- $\mathcal{S}$ is the set of states $s \in \mathcal{S}$ and $s_t \in \mathcal{S}$ is the state at time t.
- $\mathcal{A}$ is the set of actions $a \in \mathcal{A}$ and $A_t \subset \mathcal{A}$ is the subset of actions available in state s_t. If an state s_t has no available actions, it is said to be a *terminal* state.
- $\mathcal{P}(s'|s,a) = PrSt_{t+1} = s|s_t = sa_t = a$ where $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$ is a transition kernel which states the probability of transitioning to state s' from state s after performig action a. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. If the environment is stochastic, as opposed to deterministic, the function $\mathcal{P}$ maps a state-action pair to a distribution over states in $\mathcal{S}$.
- $\mathcal{R}(s,a)$ where $s \in \mathcal{S}$, $a \in \mathcal{A}$; is the reward function, which returns the immediate reward (typically in the range $[-1, 1]$) of performing action a in state s. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$. The reward at time step t can be interchangably written as r_t or $r(s_t, a_t)$.
- $\gamma \in [0, 1]$ is the discount factor, which represent the rate of importance between the current reward and future rewards. If $\gamma = 0$ the agent cares only about the immediate reward, if $\gamma = 1$ all rewards r_t are taken into account, this is only allowed in episodic tasks, as otherwise $t \to \infty$ (Sutton et al., 1999). γ is often used as a variance reduction method, and aids proofs in infinitely running environments.

From here we can introduce the notion of an agent. (link to control theory?), An agent is an entity that on every state $s_t \in \mathcal{S}$ it can take an action $a_t \in \mathcal{A}$ in an environment transforming the environment from s_t to s_{t+1}. The beheviour of an agent is fully defined by a policy π. A policy π is a mapping from states to actions, $\pi : \mathcal{S} \to \mathcal{A}$. The agent chooses which action a_t to take in every state s_t by querying its policy such that $a_t = \pi(s_t)$. If the policy is stochastic, π will map an action to a distribution over action $a_t \sim \pi(s_t)$. The objective for an agent is to find an *optimal* policy, which tries to maximize the cumulative sum of possibly discounted rewards.

There are two functions of special relevance in reinforcement learning, the *state value* function $V^\pi(s)$ and the *action value* function $Q^\pi(s,a)$:

- The state value function $V^\pi(s)$ under a policy π, where $s \in \mathcal{S}$, represents the expected sum of rewards obtained by starting in state s and following the policy π until termination. Formally defined as $V^\pi(s) = \mathbb{E}^\pi[\sum_{t=0}^{\infty} r(s_t, a_t)|s_0 = s]$
- The state-action value function $Q^\pi(s,a)$ under a policy π, where $s \in \mathcal{S}$, $a \in \mathcal{A}$, represents the expected sum of rewards obtained by performing action a in state s and then following policy π. Formally defined as: $Q^\pi(s,a) = \mathbb{E}^\pi[r(s_0, a_0) + \sum_{t=1}^{\infty} r(s_t, a_t)|s_0 = s, a_0 = a]$

The Bellman equations are the most straight forward, dynamic programming approach at solving MDPs (Bertsekas, 2007; Bellman, 1957).

0.2 Bellman equations and optimality principle

Note that in general it is not the case that all actions $a \in \mathcal{A}$ can be taken on every state $s_t \in \mathcal{S}$.

The optimality principle, found in Bellman (1957), states the following: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute

an optimal policy with regard to the state resulting from the first decision. The optimality principle, coupled with the proof of the existance of a deterministic optimal policy for any MDP as outlined in (Borkar, 1988) give rise to the optimal state value function $V^*(s) = \text{argmax}_\pi V^\pi(s) = V^{\pi^*}(s)$ and the optimal action value function $Q^*(s,a) = \text{argmax}_\pi Q^\pi(s,a) = Q^{\pi^*}(s,a)$. The optimal value functions determine the best possible performance in a MDP. An MDP is considered *solved* once the optimal value functions are found.

Most of the field of reinforcement learning research focuses on approximating these two equations (Tamar et al., 2017) (Watkins and Dayan, 1992) (Mnih et al., 2013). (cite many more)

Bellman (1957) outlined two analytical equations for the state value and action value function:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) * (r(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) * V^\pi(s')) \tag{1}$$

$$Q^\pi(s,a) = r(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) * (\sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s',a')) \tag{2}$$

Most RL algorithms can be devided into the following categories: Policy based Value based actor critic

A further categorization of algorithms is the notion of *model free* and *model based* algorithms. Consider a *model* of an environment to be the transition function $\mathcal{P}$ and reward function $\mathcal{R}$. Model free algorithms aim to approximate an optimal policy without them. Model based algorithms are either given a prior model that they can use for planning (Browne et al., 2012; Soemers, 2014), or they learn a representation via their own interaction with the environment (Sutton, 1991; Guzdial et al., 2017). Note that an advantage of learning your own model is that you can choose a representation of the environment that is relevant to the agent's actions, which can have the advantage of modelling uninteresting (but perhaps complicated) environment behaviour (Pathak et al., 2017).

0.3 Q-learning

subfiles

The Q-learning algorithm was first introduced by Watkins (1989), and is arguably one of the most famous and widely implemented methods in the entire field. Given an MDP, Q-learning aims to calculate the corresponding optimal action value function Q^*, following the principle of optimality and proof of the existence of an optimal policy as described in Section 0.1. It is model free, learning via interaction with the environment, and it is an offline algorithm. The latter is because, even though we are learning the optimal action value function Q^*, we can choose any to gather experience from the environment with any policy of our choosing. This policy is often named an *exploratory* policy. Researchers like Tijsma et al. (2017) benchmarked the efficiency of using various exploratory policies in grid world stochastic maze environments.

Q-learning has been proven to converge to the optimal solution for an MDP under some assumptions:

1. Each state-action pair is visited an infinite number of times. (Watkins, 1989)
2. The sequence of updates of Q-values has to be monotonically increasing $Q(s_i, a_i) \leq Q(s_{i+1}, a_{i+1})$. (Thrun and Schwartz, 1993).
3. The learning rate α must decay over time, and such decay must be slow enough so that the agent can learn the optimal Q values. Expressed formally: $\sum_t \alpha_t = \infty$ and $\sum_t (\alpha_t)^2 < \infty$. (Watkins, 1989)

Algorithm 1: Q-learning

repeat
  Select action `a` from policy ;
  Observe successor state `s'` and reward `r` after taking action `a` ;
  Update $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \text{argmax}_{a'} Q(s',a') - Q(s,a)]$;
until *done*;

Q-learning features its own share of imperfections, Watkins and Dayan (1992) tell us that the algorithm converges to optimality with probability 1 if each state-action pair is represented discretely, that is,

if it is implemented in tabular form. If there is a function approximator[1] in place, Thrun and Schwartz (1993) shows that if the approximation error is greater than a threshold which depends on the discount factor γ and episode length, then a systematic overestimation effect, which happens mainly due to the joint effort of function approximation methods and the max operator. On top of this, Kaisers and Tuyls (2010) introduces the concept of *Policy bias*, which states that state-action pairs that are favoured by the policy are chosen more often, biasing the updates. Ideally all state-action pairs are updated on every step. However, because agent's actions modify the environment, this is generally not possible in absence of an environment model. Frequency Adjusted Q-learning (FAQL) proposes scaling the update rule of Q-learning inversely proportional to the likelihood of choosing the action taken at that step (Kaisers and Tuyls, 2010). Abdallah et al. (2016) introduces Repeated Update Q-learning (RUQL), a more promising Q-learning spin off that proposes running the update equation *multiple times*, where the number of times is inversely proportional to the probability of the action selected given the policy being followed.

1 Policy gradient methods

subfiles

In order to comply with notation used in the field of direct optimization, we shall use u for actions.

Consider a stochastic control policy $\pi_\theta(s)$ parameterized by a parameter vector θ, that is, a distribution over the action set $\mathcal{A}$ given a state $s \in \mathcal{S}$. θ is a D-dimensional real valued vector, The parameter vector $\theta \in \mathbb{R}^D$, where D is the number of parameters / dimensions and $D << |\mathcal{S}|$. This prameterized policy function will be denoted by π_θ.[2].

There are strong motivations for using policy gradient approaches versus the already discussed RL methods:

1. A more direct way of approaching the problem. Instead of computing the value functions V or Q and from those computing a policy function, we are calculating the policy function directly.
2. Using a stochastic policy is that it smoothes the optimization problem. With a deterministic policy, changing which action to do in a given state can have a dramatic effect on potential future rewards[3]. If we assume a stochastic policy, shifting a distribution over actions slightly will only slightly modify the potential future rewards. This is what is commonly reffered as smooth.
3. Often π can be simpler than V or Q.
4. If we learn Q in a large or continuous actions space, it can be tricky to compute $\underset{u}{argmax}\ Q_\theta(s,u)$

Policy gradient methods are on-policy. In them, the agent acts with using a policy which is improved gradually over time. This contrasts with off-policy algorithms, such as the Q-learning algorithm introduced in Section 0.3, which allows the agent to interact with the environment with a policy while it is simultaneously learning another policy. There is ongoing research looking at off-policy variants of policy gradient methods (Mnih et al., 2013, 2016).

Starting from the basics: the goal of an reinforcement learning agent is to maximize the (possibly discounted)[4] sum of rewards, generally over a time horizon H. This poses the following optimization problem:

$$\max_\theta = \mathbb{E}[\sum_{t=0}^{H} r(s_t)|\pi_\theta] \tag{3}$$

For an episode of length H let τ be the trajectory followed by an agent in an episode. This trajectory τ is a sequence of state-action tuples $\tau = (s_0, a_0, \ldots, s_H, a_H)$. We overload the notation of the reward function $\mathcal{R}$ thus: $\mathcal{R}(\tau) = \sum_{t=0}^{H} r(s_t, u_t)$, indicating the total accumulated reward in the trajectory. We will also use $r(s_t) \in \mathbb{R}$ to refer to the scalar reward obtained at timestep t in the trajectory. From here, the utility of a policy parameterized by θ is defined as:

$$U(\theta) = \mathbb{E}[\sum_{t=0}^{H} r(s_t, u_t)|\pi_\theta] = \sum_\tau P(\tau;\theta)\mathcal{R}(\tau) \tag{4}$$

[1] With neural networks being the most famous function approximators in reinforcement learning at the time of writing.

[2] Some researchers prefer the notation $\pi(\cdot, \theta)$ to make the parameters of the function approximator explicit parameters of the policy. These notations are equivalent.

[3] An example of this concept are *greedy* or ϵ-*greedy* policies derived thus: $\pi(s) = \text{argmax}_{a \in \mathcal{A}} Q(s,a)$.

[4] Williams (1992); Sutton et al. (1999) present proofs of this same derivation using a discount factor.

Where $P(\tau;\theta)$ denotes the probability of trajectory τ happening when taking actions sampled from a parameterized policy π_θ. More informally, how likely is this sequence of state-action pairs to happen as a result of an agent following a policy π_θ. Going back to equation 4, our optimization problem becomes:

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau;\theta)\mathcal{R}(\tau) \tag{5}$$

The policy gradient theorem (Williams, 1992; Sutton et al., 1999) introduces the notion of updating the policy parameter vector θ using well-studied gradient based methods such as gradient descent (Schulman et al., 2017; Mnih et al., 2013), natural gradiant (Wu et al., 2017) and other approacheswhich are discussed later on. The Appendix section shows the derivation from equation 4 to equation 6

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(\tau) R(\tau)] \tag{6}$$

A key advantage of the policy gradient theorem, as inspected by Sutton et al. (1999) (and formalized in the Appendix Section), is that equation 6 does not contain any term of the form $\nabla_\theta \mathcal{P}(\tau;\theta)$. This means that we don't need to model the effect of policy changes on the distribution of states. (Sutton et al., 1999) goes farther and shows proof of local optima convergence for policy iteration methods with function approximators for both the policy and advantage functions.

2 Appendix

subfiles

Take equation 4 from Section 1, representing the utility of a policy π_θ parameterized by a D-dimensional real valued parameter vector $\theta \in \mathbb{R}^D$

$$U(\theta) = \sum_\tau P(\tau;\theta)R(\tau) \tag{7}$$

The goal is to find the expression $\nabla_\theta U(\theta)$ that will allow us to update our policy parameter vector θ in a direction that improves the estimated value of the utility of the policy π_θ. Taking the gradient w.r.t θ gives:

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta)R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau;\theta)R(\tau) && \text{(Move gradient operator inside sum)} \\
&= \sum_\tau \nabla_\theta \frac{P(\tau;\theta)}{P(\tau;\theta)} P(\tau;\theta)R(\tau) && \text{(Multiply by} \frac{P(\tau;\theta)}{P(\tau;\theta)}\text{)} \\
&= \sum_\tau P(\tau;\theta)\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)}R(\tau) && \text{(Rearrange)} \\
&= \sum_\tau P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)R(\tau) && \text{(Note:} \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} = \nabla_\theta \log P(\tau;\theta)\text{)} \\
\nabla_\theta U(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau;\theta)R(\tau)] && (\mathbb{E}[f(x)] = \sum_x x f(x))
\end{aligned} \tag{8}$$

(9)

This leaves us with an expectation for the term $\nabla_\theta \log P(\tau;\theta)R(\tau)$. Note that as of now we have not discussed how to calculate $P(\tau;\theta)$. Let's define the probability of a trajectory under a policy π_θ as:

$$P(\tau;\theta) = \Pi_{t=0}^H \underbrace{P(s_{t+1}|s_t,u_t)}_{\text{dynamics models}} \underbrace{\pi_\theta(u_t|s_t)}_{\text{policy}} \tag{10}$$

From here we can calculate the term $\nabla_\theta \log P(\tau;\theta)$ present in equation 8

$$
\begin{aligned}
\nabla_\theta \log P(\tau;\theta) &= \nabla_\theta \log[\Pi_{t=0}^{H} P(s_{t+1}|s_t,u_t)\pi_\theta(u_t|s_t)] \\
&= \nabla_\theta[(\sum_{t=0}^{H} \log P(s_{t+1}|s_t,u_t)) + (\sum_{t=0}^{H} \log \pi_\theta(u_t|s_t))] \\
&= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t|s_t)}_{\text{no dynamics required!}}
\end{aligned}
\tag{11}
$$

Plugging the result of equation 11 into equation 8 we obtain the following equation for the gradient of the utility function w.r.t to parameter vector θ:

$$
\nabla_\theta U(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(\tau) R(\tau)] \tag{12}
$$

We can compute an empirical approximation of that expresion by taking m sample trajectories (or paths) under the policy π_θ. This works even if the reward function R is unkown and/or discontinuous. This works in discrete state spaces. The likelihood ratio changes the probability of experienced paths. That is, the probability of sampling trajectories. Thus we use a Monte Carlo approach to approximate the gradient of the utility of π_θ: (REPHRASE) Which, if we plug into the original equation, we get:

$$
\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=0}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)}) \tag{13}
$$

$$
\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=0}^{m} \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})(\sum_{k=0}^{H} R(s_t^{(i)},u_t^{(i)})) \tag{14}
$$

Sutton et al. (1999) offers a different approach to this derivation by calculating the gradient for the state value function on an initial state s_0, calculating $\nabla_\theta V_{\pi_\theta}(s_0)$.

This is an unbiased estimate and it works in theory. However it requries an impractical amount of samples, otherwise the approximation is very noisy. In order to overcome this limitation we can do the following tricks:

- Add a baseline
- Add temporal structure
- Use trust region and natural gradient.

2.1 Add a baseline

Substract a baseline from the equation 15 to reduce variance without introducing varience (Williams, 1992):

$$
\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=0}^{m} \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})(\sum_{k=0}^{H} R(s_t^{(i)},u_t^{(i)}) - b) \tag{15}
$$

The problem with this equation is that each action u_i is being scaled by the whole sum of rewards $R(\tau)$. This means that, for instance, the last action in a trajectory is taking into account rewards that happened much earlier in the episode. A way to compensate for this is to scale the value of an action u_i (clarify what is meant by this) only by rewards that depend on u_i. So the equation becomes:

$$
\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=0}^{m} \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})(\sum_{k=t}^{H} R(s_t^{(i)},u_t^{(i)})) \tag{16}
$$

There are other proposed variance reduction techniques via tweaking the baseline which some researchers have studied (Greensmith et al., 2004)

References

Abdallah, S., Org, S., Kaisers, M., and Nl, K. (2016). Addressing Environment Non-Stationarity by Repeating Q-learning Updates. *Journal of Machine Learning Research*, 17:1–31.

Bellman, R. (1957). *Dynamic Programming*, volume 70.

Bertsekas, D. P. (2007). *Dynamic Programming and Optimal Control, Vol. II*, volume 2.

Borkar, V. S. (1988). A convex analytic approach to Markov decision processes. *Probability Theory and Related Fields*, 78(4):583–602.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of {Monte Carlo Tree Search} methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Greensmith, E., Bartlett, P., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *The Journal of Machine Learning ...*, 5:1471–1530.

Guzdial, M., Li, B., and Riedl, M. O. (2017). Game Engine Learning from Video. *International Conference on Artificial Intelligence (IJCAI)*.

Kaisers, M. and Tuyls, K. (2010). Frequency Adjusted Multi-agent Q-learning. *Learning*, pages 309–316.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. 48.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. pages 1–9.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-Driven Exploration by Self-Supervised Prediction. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2017-July:488–489.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. pages 1–12.

Soemers, D. (2014). Tactical Planning Using MCTS in the Game of StarCraft. page 12.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163.

Sutton, R. S., Mcallester, D., Singh, S., and Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *In Advances in Neural Information Processing Systems 12*, pages 1057–1063.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. (2017). Value iteration networks. *IJCAI International Joint Conference on Artificial Intelligence*, (Nips):4949–4953.

Thrun, S. and Schwartz, A. (1993). Issues in Using Function Approximation for Reinforcement Learning. *Proceedings of the 4th Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, pages 1–9.

Tijsma, A. D., Drugan, M. M., and Wiering, M. A. (2017). Comparing exploration strategies for Q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*.

Watkins, C. J. (1989). *Learning from delayed rewards*. PhD thesis, King's College.

Watkins, C. J. and Dayan, P. (1992). Technical Note: Q-Learning. *Machine Learning*, 8(3):279–292.

Williams, R. J. (1992). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3):229–256.

Wu, Y., Mansimov, E., Liao, S., Grosse, R., and Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. pages 1–14.