Reinforcement Learning

Eilif Solberg

September 26, 2018

Contents

1 Introduction

Up until now we have assumed that the data is *given* to us, and is has been our job to *analyze* it. We have looked at how we can use e.g. attention mechanisms to focus more on some parts of the data than others, but we haven't had any way to actually influence the data distribution itself. Moving over to reinforcement learning, this will no longer be so. Indeed for some applications, taking actions to obtain the necessary information will be one of the most important tasks that must be learned.

Reinforcement learning may be viewed as an interactive play with two actors, the *agent* and the *environment*. The agent performs actions that affect the environment. The environment provides the agent with useful information about the current situation the agent finds itself in, as well as feedback through a scalar reward signal. Note that unlike supervised learning, the environment is not a teacher that tells the agent what actions he should have made. Of the two actors the environment is clearly the passive one. It does not have any goals of its own and we shall assume its behaviour to be the same throughout[1]. The agent on the other hand wants to get as much reward as possible, and changes his policy π to achieve this goal. In general we shall not assume that the agent knows anything about the workings of the environment. The agent must find this out for himself by interacting with the environment. We shall let $\mathcal{A}$ denote the set of actions the agent may perform, this *action space* may be both finite and infinite. Similarly we let $\mathcal{O}$ denote the *observation space*. The rewards are real numbers, and we shall assume them to be bounded, i.e. we can always find bounds m and M so that rewards are always greater than m and less than M[2]. We shall operate with a discrete notion of time, where the sequence of agent-environment dynamics plays out as follows: The agent is presented with an initial observation o_0 and then for $t = 0, 1, \ldots$

1. The agent performs an action a_t according to its policy π, i.e. $a_t \sim \pi(o_0, a_0, r_1, o_1, \ldots, a_{t-1}, r_t, o_t), \quad a_t \in \mathcal{A}$.
2. The environment rewards the agent with $r_{t+1} \in \mathbb{R}$ and emits the observation $o_{t+1} \in \mathcal{O}$.

We define the *history* h_t to be $o_0, a_0, r_1, o_1, \ldots, a_{t-1}, r_t, o_t$ The situation just presented is quite general, for the remainder of the chapter we shall make

[1] This is how we model the situation anyhow. In many applications the environment may contain other actors that continually change and certainly do have goals of their own.

[2] Without this assumption you would at least need to put some other assumptions on it if you would like convergence guarantees for any of the algorithms we will look at here.

a couple of quite strong assumptions. The first assumption shall basically ensure that we know what information the environment is using to make its decision. The history h_t is in practice often not manageable for us and we will have to base our decisions on only a subset of this information. The second assumption shall just assume that even with this restricted version, the agent still has sufficient information to learn the optimal behaviour. We state the assumptions as follows:

- The distribution of (o_{t+1}, r_{t+1}) is fully determined by the history h_t and the action a_t.
- The agent is given a function f, $s_t = f(h_t)$, such that the distribution of (o_{t+1}, r_{t+1}) is independent of h_t given s_t and a_t. We shall call s_t the *state* at time t and assume the states s_t to live in some state space $\mathcal{S}$.

What do we do if either of these assumptions are not satisfied? The game of *Texas hold'em poker* is a situation where the first assumption is violated. The actions of the other players (which is part of the environment for us) will depend on their *personal* cards, which are unknown to us. One will then usually try to reason on what kind of cards the opponent is likely to have, given our own cards, the community cards and the actions taken by the players so far. This kind of problem is not something we will try to handle here, but we refer the reader to the literature on *partially observable Markov decision processes*. Given that the first assumption is true it is within our power to satisfy the second one. Sometimes however we shall have to give this up in order to obtain any sort of practical solution. In a few cases it may be possible to restrict the state to be only the last few observations, i.e. $s_t = (o_{t-k}, \dots, o_{t-1}, o_t)$ for some k. For a chess game we may even take $k = 0$, using only the current board position to make our decisions. The moves of our opponent may of course depend on history going back several moves, making our predictions of his moves slightly off. He can't however gain anything from this, and restricting ourselves to only the current board position is probably a good decision. For a given f it is often hard to know if the conditional assumption is actually satisfied. For the theory we shall assume it to be true, in applications we often have to balance likely deviations against the complexity of the state space in constructing f. We may also try to learn a state function, based on the whole history, through e.g. an RNN model.

With the introduction of the state, we shall going forward ignore the observations and assume the agent is given the state directly. Though the agent could potentially base his policy on the whole history of states, there

would be no use to it as the environment dynamics is independent of the previous states, given the current one. We shall thus assume that our policy π is given as $\pi\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$. For a given state s and action a, $\pi(a|s)$ represents for a *discrete* action space the *probability* for the agent to perform action a given that we are in state s. If we are dealing with a *continuous* action space, the number represents a *density* value. We may in the future use these terms interchangeably. For a given state s we shall let $\pi(s)$ denote the probability distribution the policy π induces over the action space $\mathcal{A}$. The agents next action is then drawn from this distribution, i.e.

$$a \sim \pi(s), \quad s \in \mathcal{S}, \;\; a \in \mathcal{A} \tag{1}$$

The environment, from the point of view of the agent, is fully defined by the conditional distributions $\mathcal{P}_s^a$, which may be described in terms of its density values

$$p(r, s'|S_t = s, A_t = a) \tag{2}$$

which denote the density value for the outcome $R_{t+1} = r$ and $S_{t+1} = s$. The model presented is called a *Markov Decision Process* and we will assume this for the remainder of the chapter. The dynamics of the system then unfolds according to

1. $a_t \sim \pi(s_t)$
2. $(s_{t+1}, r_{t+1}) \sim \mathcal{P}_{s_t}^{a_t}$

A state s such that $P[S_{t+1} = s|S_t = s, A_t = a] = 1$ for all actions a, is called an *absorbing* state as we can't escape from it. We shall assume that all the rewards are zero from that point onwards, and that we *know* when we are in an absorbing state. A transition into such a state effectively marks an end of the sequence, as nothing more can be learned or gained going forward. We call environments where the optimal behaviour always leads to such an absorbing state an *episodic* environment. *Continual* environments on the other hand either don't have absorbing states, or it is not a great idea to get in one! We shall also assume that we always *know* if our environment is episodic or continual. Some of the methods we will look at are only applicable for episodic environments as the learning is based on seeing the entire outcome of the episode. We will also look at methods that works for continual environments, these will be able to learn *as they go* without the need for an end. Methods that work for continual environments automatically work for episodic environments as well.

Note: The material of these notes are largely based on a course in reinforcement learning given by David Silver. Look at `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`, which also contains a link to videos of the lectures.

2 Prediction - evaluating a policy

Before we look into how we can find the optimal policy we shall look at how we can evaluate how *good* a policy is. Why would we want that? One reason is that it can clearly be useful to know how well the policy is working, so to be able to assess if it provides an acceptable solution. Another point is that evaluating two policies let us *compare* them and we can make a more informed decision on which policy to choose. A third reason, is that we shall later see that our ability to evaluate a policy will also prove useful in *improving* it.

For supervised learning the problem of evaluation is usually trivial. We have a fixed evaluation set $\{(X_i, Y_i)\}_{i=1}^N$ where we aggreate some kind of loss or accuracy g over the samples

$$L = \frac{1}{N} \sum_{i=1}^{N} g(f(X_i), Y_i) \tag{3}$$

The case in reinforcement learning is a bit different. We don't have a fixed evaluation set we can run over, but must instead sample data through our interactions with the environment. There may be introduced randomness both through *our policy* and from the *environment dynamics*

1. In case of a *stochastic* policy we may sample different actions from the same state, leading to different future states and rewards.

2. The environment may introduce some randomness in its state transitions and reward signal.

Though this makes the problem of policy evaluation highly non-trivial, we will see that there are viable approaches to the problem. We define the *return*[3] at t, denoted by G_t, to be the *discounted* cumulative reward from t and onwards

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4}$$

[3]We may in the future take the term *reward* to mean *return* if it is clear from context.

where $\gamma \in [0, 1]$. For theoretical analysis it is very convenient to have $\gamma < 1$. For $\gamma = 1$ one often needs to make additional assumptions in order to obtain similar results, for one thing we can't even guarantee that G_t is finite if $\gamma = 1$! We may also view γ as a mechanism to allow us to care more about rewards in the near rather than distant future. Not to digress too much we have postponed a longer discussion on the justification and uses of the γ factor to Section 7.1.

We shall make use of two different representations of the expected return, which we call *value functions*. The first function $v_\pi \colon \mathcal{S} \to \mathbb{R}$, which we call the *state*-value function, is defined as

$$v_\pi(s) = E_\pi[G_0 | S_0 = s] \tag{5}$$

The function tells us the amount of reward we can expect from a given initial state s, following policy π. As our distributions are stationary with respect to time it is clear that also $v_\pi(s) = E_\pi[G_t | S_t = s]$ for any t. Thus at any time we find ourselves in state s the value $v_\pi(s)$ is the expected discounted reward from that point and onwards.

We note that the state-value function introduces a weak ordering of policies. We say that $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$ and $\pi > \pi'$ if the inequality is strict for at least one state s. We use the term *weak* ordering as in general we may have $v_\pi(s) > v_{\pi'}(s)$ for a state s and $v_{\pi'}(s') > v_\pi(s')$ for another state s'.

The *action*-value function is a representation that will often prove convenient when used in the setting of policy improvement, and is defined as

$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a] \tag{6}$$

$q_\pi(s, a)$ tells us how much future reward we can expect by taking action a from state s and *then* following policy π from that point forward. Note that by the law of iterated expectations $v_\pi(s) = E_\pi[G_t | S_t = s] = E_\pi[E_\pi[G_t | S_t = s, A_t = A]] = E_\pi[q_\pi(s, A)]$. The relation between them can also be written as

$$v_\pi(s) = \int \pi(a|s) q_\pi(s, a) da \tag{7}$$

If the policy π is deterministic equation (7) simplifies and the relationship is simply given by $v_\pi(s) = q_\pi(s, \pi(s))$.

We shall now present two methods that may be used to estimate the state-value and action-value functions. The methods find themselves at opposite positions in the infamous bias-variance trade-off.

2.1 Monte Carlo

Our first approach is quite direct. We shall illustrate it using the state-value function only, as the equations we derive will be of the exact same form for the action-value function. Let's recall the definition of the state-value function

$$v_\pi(s) = E_\pi[G_t|S_t = s] \tag{8}$$

Let us now assume that we are in an episodic enviroment, so that all sequences terminate. In that case why not just estimate the expectation directly through sampling say N episodes? Assume first that we have a discrete state space. For each episode, the first time we run into a state s we increase the zero-initialized count $N[s]$ by one, and add the observed future reward to an accumulator $A[s]$[4]. Based on these $N[s]$ estimates we then have an approximation for $v_\pi(s)$ which we store in an array V

$$V[s] \leftarrow \frac{1}{N[s]} A[s] \tag{9}$$

By the *law of large numbers*, we will have that $A[s]/N[s] \rightarrow v_\pi(s)$ as $N[s] \rightarrow \infty$ [5].

Instead of collecting these statistics only to update our estimates in V at the end, we can also create an equivalent *online* update rule. Recall that G_t is the return for state t. At the end of the episode we may update the running average for each state visited during the episode by

$$V[s_t] \leftarrow V[s_t] + \frac{1}{N[s_t]}(G_t - V[s_t]) \tag{10}$$

where t here is assumed to be the time index for the first encounter of state s_t during the episode. To see that this is equivalent to our previous update

[4] This is called *first-visit* Monte Carlo. There is a also a variant called *every-visit* Monte Carlo which only calculate the statistics based on *every* visit of a state for each episode. Every-visit Monte Carlo will clearly have more statistics to base its expectation on, and may thus seem superior. It does however have the disadvantage that it's harder to estimate its variance as the samples are no longer independent.

[5] This is trivially true for *first-visit* Monte Carlo as we are dealing with IID data. This should also hold for *every-visit* Monte Carlo though under mild assumptions

note that

$$\bar{x}_n := \frac{1}{n}\sum_{i=1}^{n} x_i \tag{11}$$

$$= \frac{1}{n}((n-1)(\frac{1}{n-1}\sum_{i=1}^{n-1} x_i) + x_n) \tag{12}$$

$$= \frac{1}{n}((n-1)\bar{x}_{n-1}) + x_n) \tag{13}$$

$$= \bar{x}_{n-1} + \frac{1}{n}(x_n - \bar{x}_{n-1}) \tag{14}$$

Although the updates of equations (9) and (10) are equivalent, the latter formulation will be useful for us later when we shall look at cases where the policy *changes* as we do the evaluation! For those cases we don't want to take an equally weighted average, but rather gradually forget the early samples as these are obtained using a potentially quite different policy then the current one. We shall then look at update equations of the form

$$V[s_t] \leftarrow V[s_t] + \alpha(G_t - V[s_t]) \tag{15}$$

where $\alpha \in (0, 1)$. What if our state space is too large to keep such an array, or the state space is not discrete? We will in those cases resort to *function approximation*. We shall often take the approximator to be a neural network parametrixed by η and will then let v_η denote this function. Function approximation may also be used in cases where we in theory could keep our estimates in an array, but would like to have the potential generalization benefits that comes with function approximation. With function approximation updates to one state can potentially benefit other, similar states. In cases where we use a function approximator, we can't "assign" the target to the function in any meaningful way, like we did in e.g. equation (9). We will instead let the right-hand side be the *target* for the value of the approximator at s, i.e. $v_\eta(s)$. We will then make an update of the parameters so to make this value closer to this target. We define the loss as

$$l(\eta) = \frac{1}{2}\big(G_t - v_\eta(s_t)\big)^2 \tag{16}$$

and we find by using the chain rule that the gradient is given by

$$\nabla_\eta l(\eta) = -(G_t - v_\eta(s_t))\nabla_\eta v_\eta(s_t) \tag{17}$$

Making an update in the direction of steepest *descent* gives an update in the opposite direction of the gradient, i.e.

$$\eta = \eta + \alpha\big(G_t - v_\eta(s_t)\big)\nabla_\eta v_\eta(s_t) \tag{18}$$

where $\alpha > 0$. Note the simliarities between this update and the one in equation (15). Unlike for that case we cannot however know exactly how the update affects our update $v_\eta(s_t)$. In addition this update will also affect other states, perhaps also pushing our value estimates at states "similar" to s_t towards G_t. We call G_t the *target* of the Monte Carlo update. The target certainly provides *unbiased* estimates of the true value function. This follows from the definition of the value function, $E[G_t|s_t] = v_\pi(s_t)$. Unfortunately the target may sometimes suffer from high *variance*. We may need to sample a large number of episodes in order to reduce the variance to an acceptable level, especially for states not visited that often.

2.2 Temporal Difference

When we looked at the Monte Carlo method above we estimated $v_\pi(s)$ *independently* for each s[6]. We treated the values $v_\pi(s)$ as if they lived in separate universes. We know that this is certainly not the case however, and will see that the value of other states will pose very strong restrictions on $v_\pi(s)$. Assume we find ourselves in state s and take and action a, according to our policy π. We get an immediate reward r and end up in a new state s', where $v_\pi(s')$ gives us the expected future reward from there. We might be unlucky and find ourselves worse off then our current expectations. By this we mean that from the perspective of our new situation, we expect to get less total reward from state s, i.e. $r + \gamma v_\pi(s') < v_\pi(s)$. The reason for this might be due to randomness in our policy, in the environment, or both:

- In case of a stochastic policy, our action $a \sim \pi(s)$ may turn out to be one of the worse actions of our policy.
- The action a produced a lower immediate reward then usual or we happened to transition to an unusually bad state.

Another time the luck is on our side and the future suddenly looks brighter, i.e. $r + v_\pi(s') > v_\pi(s)$. On *average* however we should expect

[6]With function approximation it is not *completely* independent, but the coming argument still applies.

that our estimate remains the *same*, we should neither expect to be positively nor negatively surprised. This consistency is what is captured in the *Bellman expectation equations*. We have that

$$v_\pi(s_t) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})] \tag{19}$$

$$= \int\int\int \pi(a|s_t)p(r, s'|s_t, a)(r + \gamma v_\pi(s'))ds'drda \tag{20}$$

Notice how equation (20) summarizes our intutiton from above. When we average out the randomness that comes from either our action, next state or immediate reward, the estimates from the point of view of the future should be consistent with our current estimate. This gives us a *necessary* condition that our value function must satisfy, and we should somehow be able to take advantage of this when we estimate the value function. In fact the situation is even better, as long as $\gamma < 1$ it is indeed a *sufficient* condition[7].

Let's see if we can now try to make our value function estimates satisfy the equations of (20). Let's first look at the case with a discrete state space without function approximation. Assume that we have an array V such that $V[s]$ contains our current estimate of $v_\pi(s)$. Assuming we also have a discrete action space, equation (20) transforms into

$$V[s] = \sum_a \sum_r \sum_{s'} \pi(a|s)p(r, s'|s, a)(r + \gamma V[s']) \tag{21}$$

Imagine if the equation does not hold true, and say that the prediction of future rewards consistently worsened when taking one step forward from s. This inconsistency may of course be on *either* side of equation (21), i.e. we are too positive to the prospects from state s, or we are in general too negative to the likely future states we typically find ourselves in one step ahead. In theory one might thus want to change either, or both, sides to obtain balance. We could e.g. justify our current estimate by increasing our predictions $V[s']$ for likely transition states s'. If these then indicated being too optimistic, we could justify these again by increasing future prospects. If our episodes never came to an end, it seems like we in this way could keep changing our future predictions so that they fit with our early predictions, and so obtain equality in this way. We will however see that it is not so. Assume for now that we are in an episodic environment. All episodes comes to an end, and we will finally be held accountable for our predictions. If

[7] This can be proved by application of Banach's fixed-point theorem.

we come to the end of an episode and figure out our reward estimate was way too optimistic, we should take this reality into account and update our beliefs based on it. The ending of episodes provides potential *grounding* of our estimates and may prevent us from creating consistent, but fantasy predictions. The information must necessarily be propagated *backwards* from the final state, as there are no future states. This tells us that we should thus update the *left-hand side* of equation (21) in terms of the *right-hand side*, so that we update the state in terms of grounded future states, propagating the grounding backwards in time.

In the case of general environments with $\gamma < 1$, we argue in Section 7.1 that this basically also forces us to only consider a perhaps long, but finite future. It indeed forces us to ground our reward estimates in the "*near*" future as argued in the appendix. With $\gamma = 1$ in continual environments we get into trouble because we neither get the grounding the episode end provides, nor from the myopic world view the discount-factor forces upon us.

Anyway, the arguments given were only to give some intution why it makes sense to update the left-hand side over the right-hand side. The math tells the same story and doing it the other way will in general not give convergence. In fact we can come up with a single-state MDP with $\gamma < 1$ that gives divergence in that case, perhaps you can think of such?

Let us now move on to see how we can actually make the updates. If we knew the dynamics of the environment, i.e. $p(r, s'|s, a)$ from above we could turn the equation (21) directly into an update of every state by

$$V[s] \leftarrow \sum_a \sum_r \sum_{s'} \pi(a|s) p(r, s'|s, a)(r + \gamma V[s']) \tag{22}$$

We can not assume this in general though and shall instead resort to methods based on *sampling*, as was the case for the Monte Carlo methods above. Note that again sampling-based updates affects our update scheme in *two* quite different aspects. Firstly it affects *how* we update a certain state s, as we need to sample the right-hand side instead of taking the full expectation. Secondly, it affects *which* states and in what order we update them. We may not be able to magically put ourselves in any state s and then update that state. Instead we will in general we can only update the states we happen to visit as we interact with the envirionment. We shall in general sample the states we update by following our policy π, as we ususally do for Monte Carlo. Unlike with Monte Carlo, however, we shall not assume an episodic environment, and we will look at methods that update after *each* step, i.e. online methods. By moving one step forward we obtain an unbiased sample $R + V[S']$ of the right-hand side of (22). As this is only a

single sample however, we shall not trust it completely by overwriting the old value. Instead we will do an incremental update of the form

$$V[s_t] \leftarrow V[s_t] + \alpha((R_{t+1} + \gamma V[S_{t+1}]) - V[s_t]) \tag{23}$$

We here call $R_{t+1} + \gamma V[S_{t+1}]$ the *TD-target*. Generally $R_{t+1} + \gamma \hat{v}_\pi(S_{t+1})$ is the TD-target for a value function estimator $\hat{v}_\pi$.

Let's now see how we can make updates when we use function approximation for the value function. If we implement the value function with a differentiable model parameterized by η, we may define a loss function l

$$l(\eta) = \frac{1}{2}\big((R_{t+1} + v_\eta(S_{t+1})) - v_\eta(s_t)\big)^2 \tag{24}$$

In the spirit of updating the left-hand side towards the right-hand side, we shall in taking the gradient assume the parameters η of the TD-target to be constant. Let η' be a constant copy of η, then

$$\nabla_\eta l(\eta) = -\big((R_{t+1} + v_{\eta'}(S_{t+1})) - v_\eta(s_t)\big)\nabla_\eta v_\eta(s_t) \tag{25}$$

and following the direction of steepest descent gives an update rule

$$\eta = \eta + \alpha\big((R_{t+1} + v_{\eta'}(S_{t+1}) - v_\eta(s_t))\big)\nabla_\eta V_\eta(s_t) \tag{26}$$

Notice the similarity of the updates in (23) and (26), to the corresponding Monte Carlo updates in equations (15) and (18). The only change is that the *target* we update towards is $R_{t+1} + \hat{v}_\pi(S_{t+1})$ instead of G_t, where $\hat{v}_\pi$ is either the array V or function v_η from above. The target for the TD-method often has much lower variance than the Monte Carlo target. The only issue is that the target is generally not correct in expectation, i.e.

$$E_\pi[R_{t+1} + \hat{v}_\pi(S_{t+1})] \neq E_\pi[R_{t+1} + v_\pi(S_{t+1})] = v_\pi(s_t) \tag{27}$$

where the last equality follows from the Bellman equations. This shows that the TD-target introduces some bias. This bias is not enough to throw us off convergence at least in the case *without* function approximation. With function approximation there are known theoretical divergence issues, but it may very well converge in practice.

What about the case of the action-value function? The same reasoning that led us to believe that there must be some relations between the values $v_\pi(s)$ of the state-value function, should lead us to the same conclusion for

the action-value function. Assume that we find ourselves in state s_t and choose action a_t. As a consequence we observe a reward r and state s'. Our expected future reward from state s' is $v_\pi(s')$. Again we should have that the estimate before the action was applied $q_\pi(s_t, a_t)$ should on average be consistent with the estimate we get after observing the environment response, $r + \gamma v_\pi(s')$. We have

$$q_\pi(s_t, a_t) = \int\int p(r, s'|s_t, a_t)(r + \gamma v_\pi(s'))ds'dr \tag{28}$$

To get a recursive relation, we want to write this in terms of q though. This is possible as by equation (7) we have $v_\pi(s) = \int \pi(a|s)q_\pi(s, a)da$. We thus get the relation

$$\begin{aligned} q_\pi(s_t, a_t) &= \int\int p(r, s'|s_t, a_t)(r + \gamma \int \pi(a'|s')q_\pi(s', a')da')ds'dr & (29)\\ &= \int\int\int p(r, s'|s_t, a_t)\pi(a'|s')(r + \gamma q_\pi(s', a'))ds'drda' & (30)\\ &= E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s_t, A_t = a_t] & (31) \end{aligned}$$

Although the recursive relations in (20) and (30) look remarkably similar, there is an important difference when it comes to *sampling* their right-hand side. In case of the state-value function we first sample and action $a \sim \pi(s)$ and then observerve the response of the environment through r and s'. When sampling (29) we first sample the environments response r and s', and then we sample an action $a' \sim \pi(s')$. As there is no randomness to the result of our action sample, the value is given by $q(s', a')$, we may actually integrate it out and avoid sampling the action altogether. If this is worthwhile depends on the computational cost of this integration. For each of the TD-update equations for the state-value function we then have *two* different versions for the action-value function; depending if we choose to integrate out randomness of our action or not. The corresponding update to (23) has the two versions

$$(a)\;\; Q[s, a] \leftarrow Q[s, a] + \alpha\big((R + \sum_{a'} \pi(a'|S')Q[S', a']) - Q[s, a]\big) \tag{32}$$

$$(b)\;\; Q[s, a] \leftarrow Q[s, a] + \alpha\big((R + Q[S', A']) - Q[s, a]\big) \tag{33}$$

and (26) transforms into the options

$$(a)\;\; \eta \leftarrow \eta + \alpha\big((R + \int \pi(a'|S')q_\eta(S', a')da') - q_\eta(s, a)\big)\nabla_\eta q_\eta(s, a) \tag{34}$$

$$(b)\;\; \eta \leftarrow \eta + \alpha\big((R + q_\eta(S', A')) - q_\eta(s, a)\big)\nabla_\eta q_\eta(s, a) \tag{35}$$

Again we have assumed the parameters in the target to be constant when taking derivatives.

2.3 TD(λ)

Monte Carlo evaluation had no bias, but high variance, while temporal difference learning had lower variance at the cost of some variance. In some respects this are at different ends of the bias-variance trade-off. It turns out that there is a whole spectrum of trade-offs in-between. There is also an elegant method in which we can smoothly change the trade-off by varying a single real-valued parameter. This technique is called $TD(\lambda)$. We will not cover this method here, but encourage the eager reader to look it up.

3 Control - finding the optimal policy

Given any initial state s the goal of the agent is to maximize the expected return, i.e. design a policy π such that

$$v_\pi(s) = max_{\pi'} v_{\pi'}(s) \tag{36}$$

for all states s.

How can we find a policy π with a high expected reward? As usual we shall treat problem as an *iterative* optimization problem, where we have an initial policy π_0 which we gradually try to improve. The optimization however will not be as straightforward as for the case with *supervised* learning. For supervised learning we had *targets* that told us what the desired output was. This allowed us to create a differentiable loss function and we could use stochastic gradient descent for the optimization. For reinforcement learning we don't have targets for the outputs of π. The reward for an episode gives us some *overall* feedback for the actions we performed, however. A high reward may indicate that the choices we made were pretty good. We may then decide to update the policy to increase the probability of the actions taken. An issue is however that the reward doesn't tell us *which* actions were good. We might be tempted to believe that an action a_t that gets a high *immediate* reward r_t from state s_t is a good action. We need to remember however that actions we take now also affects future states and rewards. The action may not look so good anymore if it comes at the cost of a much higher reward later. One of the major problems in reinforcement learning is to *assign credit* to the actions for their contribution to the sparse reward signal, i.e. to figure out which actions were *responsible* for the high or low reward.

Another interesting aspect with reinforcement learning is that for some environments it may not be immediately clear if a reward is good or not. Even a high positive reward may not be impressive if we could potentially get a much higher reward. On the other hand, a small negative reward may be great if all rewards are negative. In these cases we will also need to try to learn the *distribution* of rewards to figure out what we can consider a good or bad reward. For most of the problems that we will look at it is the *credit assignment problem* that will be the greatest challenge. There are, however, some problems in reinforcement learning, e.g. the *multi-armed bandit*, which soley concern themselves with the distribution problem.

As a final thought is important to keep in mind that an action can rarely be seen as good or bad on its own, we need to consider it together with the rest of the policy. A good move for an expert chess player that leads to a quite complicated position, but eventually will pay off after a series of follow-up moves, may not work out well for a novice player who does not know how to proceed. As the novice improves his play, eventually it may become a good move for the novice as well. It is thus the *policy* as a whole that may be considered good or bad, not the individual actions.[8] When we later in this chapter refer to *good* and *bad* actions, this should then not be interpreted as a statement about the action on its own, but also how it fits with the rest of the policy.

3.1 Policy gradient - policy based control

Or first approach for solving the reinforcement learning problem will be the most direct one. We will start by creating a parametrized space of policy functions π_θ, $\theta \in \Theta$. By varying θ we get different policies with different expected rewards. Our problem is then reduced from that of (36) to

$$\pi_{\theta^*} = argmax_\theta E_{\pi_\theta}(G_0) \tag{37}$$

Of course we have $E_{\pi_{\theta^*}}(G_0) \leq E_{\pi*}(G_0)$. As we in general may not able to cover all possible policies within our parametrized family, the inequality will often be strict. However, at least we have reduced the problem to something that looks more manageable. If we implement π_θ as a neural network we may usually increase the capacity of our policy family if necessary. We shall assume that π_θ is a differentiable function of θ.

[8] No rules without exception though, there can of course be cases where one move really *is* better than another. E.g. if two actions leads to the same next state, but only one of the actions gives a reward.

We shall approach the problem in equation (37) by imitating the standard optimization approach in supervised learning with differentiable parametrized models. Let's start by refreshing our memory on how that worked. During training we have our input-target pair (X, Y), a parametrized function f_θ which predicts Y from X, and a differentiable loss function l measuring how far away our prediction were from the target. The problem we want to solve is

$$\min_\theta E(l(f(X;\theta), Y)) \tag{38}$$

In order to proceed with our gradient descent optimization, we then need to find the gradient of equation (38) with respect to θ. If we had a finite number of pairs, as is often the case for supervised learning, we could in theory calculate the full gradient. In practice we usually approximate the gradient using only a small number of sampled pairs, as this is may provide a sufficiently good estimate at a much lower computational cost. We will now show how we can get an unbiased estimate of the gradient with respect to the full distribution of (X, Y). We shall use the trick of "derivation under the integral sign". You may assume this to be valid for all integrals encountered in this chapter, though we refer the the uneasy reader to Section 7.2 for some justifications. We have

$$\nabla_\theta E(l(f(X;\theta), Y)) = \nabla_\theta \int\int p(x,y) l(f(x;\theta), y) dxdy \tag{39}$$

$$= \int\int \nabla_\theta (p(x,y) l(f(x;\theta), y)) dxdy \tag{40}$$

$$= \int\int p(x,y) \nabla_\theta l(f(x;\theta), y) dxdy \tag{41}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta l(f(x_i;\theta), y_i) \tag{42}$$

We will now see if can carry this approach over to the reinforcement learning problem. We shall only derive the method in the case of *episodic* environments, but we will in Section 3.3 see a version that can also be applied for *continual* learning. Let z denote an arbitrary episode, $z = (s_0, a_0, r_1, s_1, , a_{\tau-1}, r_\tau, s_\tau)$ where s_τ is a terminal state. We define $r(z)$ to be the total reward for the episode, i.e. $r(z) = \sum_{t=0}^{\tau-1} \gamma^t r_{t+1}$. We can then

write the expected return $E_{\pi_\theta}(G_0)$ as[9]

$$E_{\pi_\theta}(G_0) = \int p(z;\theta) r(z) dz \tag{43}$$

To follow the same approach as for the supervised learning case we next want to find a way to approximate the gradient of the expected reward with respect to our parameters θ.[10] We have

$$\nabla_\theta E_{\pi_\theta}(R) = \nabla_\theta \int p(z;\theta) r(z) dz \tag{44}$$

$$= \int \nabla_\theta p(z;\theta) r(z) dz \tag{45}$$

Now we start getting into trouble. The distribution is not constant with respect to the variables θ, which we are taking the derivatives with respect to, and thus we can't factor it out as before. If we go back and look at equations (40) - (42), we see that this was what put us in position to approximate the integral by sampling. Here we see the effect of a difference we pointed out between supervised learning and reinforcement learning in the introduction; with reinforcement learning our actions affect the data distribution itself. So is this the end of the road then? Is the policy gradient method restricted to the simple cases where we are able to evaluate the integral analytically? Not so fast.

For a function f of one variable, we have by the chain rule that

$$\frac{d}{dx} \log f(x) = \frac{\frac{d}{dx} f(x)}{f(x)} \tag{46}$$

We may easily verify that this translates to a function of several variables θ as

$$\nabla_\theta \log f(\theta) = \frac{\nabla_\theta f(\theta)}{f(\theta)} \tag{47}$$

and thus we we that $\nabla_\theta f(\theta) = f(\theta) \nabla_\theta \log f(\theta)$. This is known as the *log-derivative trick*. Using this we get that

[9] We of course also have $E_{\pi_\theta}(G_0) = \int p(r;\theta) r dr$, but it is less clear how to proceed from there.

[10] For now we shall assume that this exists!

$$\int \nabla_\theta p(z;\theta) r(z) dr = \int p(z;\theta) \nabla_\theta \log p(z;\theta) r(z) dr$$

We now see that we have been able to get a factor $p(z;\theta)$ *outside* the gradient. This is good news as we know how to sample from $p(z;\theta)$, this is just episodes generated by following policy π_θ. We can thus approximate the gradient by sampling episodes using our policy π_θ

$$\nabla_\theta E_{\pi_\theta}(G_0) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p(z_i;\theta) r(z_i) \tag{48}$$

It remains to show that p(z; θ) is actually differentiable with respect to θ, and hopefully get an expression for $\nabla_\theta \log p(z;\theta)$ which we can compute efficiently. We will accomplish this by choosing an appropriate factorization[11] of $p(z;\theta)$, where all factors are either applications of π_θ or factors that are constant with respect to θ. As we have assumed that π_θ is differentiable with respect to θ, the result will follow. We shall use a factorization that is natural with respect to how the episode *unfolds* or is *generated*. For an episode $(s_0, a_0, r_1, s_1, \ldots, a_{\tau-1}, r_\tau, s_\tau)$ to take place we must first start with the initial state s_0. Then for $t = 0, \ldots, \tau - 1$ we must have

1. Given s_t we follow up with action a_t, i.e. a_t is drawn from $\pi_\theta(s_t)$. The probability for this is $\pi_\theta(a_t|s_t)$.

2. The environment rewards the agent with r_{t+1} and returns a new state s_{t+1}, which has probability $p(r_{t+1}, s_{t+1}|s_t, a_t)$.

Let us denote by $h_t = (s_0, a_0, r_1, s_1, \ldots, a_{t-1}, r_t, s_t)$. We can write this

[11] We will use that $p(x,y) = p(x)p(y|x) = p(y)p(x|y)$. We already see that for two variables, there is no unique factorization. With n variables we may factorize $p(x_1, x_2, \ldots, x_n)$ as $p(x_{\sigma(1)})p(x_{\sigma(2)}|x_{\sigma(1)})p(x_{\sigma(3)}|x_{\sigma(1)}, x_{\sigma(2)}) \cdots p(x_{\sigma(n)}|x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n-1)})$, where σ is any permutation of $\{1, 2, \ldots, n\}$.

factorization as

$$p(s_0, a_0, r_1, s_1, \ldots, a_{\tau-1}, r_\tau, s_\tau; \theta) \tag{49}$$

$$= p(s_0) \prod_{t=0}^{\tau-1} p(a_t|h_t; \theta) p(r_{t+1}, s_{t+1}|h_t, a_t) \tag{50}$$

$$= p(s_0) \prod_{t=0}^{\tau-1} \pi_\theta(a_t|h_t) p(r_{t+1}, s_{t+1}|h_t, a_t) \tag{51}$$

$$= p(s_0) \prod_{t=0}^{\tau-1} \pi_\theta(a_t|s_t) p(r_{t+1}, s_{t+1}|s_t, a_t) \tag{52}$$

where we in the next to last step used that the probability of the next action is given by our policy. In the final step, we used our assumptions that both the agent and the environment only uses s_t and not the whole history to determine the future. Note that there is no dependency on θ in $p(r_{t+1}, s_{t+1}|s_t, a_t)$. The environment only depends on our policy, and thus θ, through the *actions* it produces. As we have already conditioned on the actions, the policy does not have any more influence on these probabilities. The environment doesn't know what policy produced the actions, neither does it care, when it decides on its reward r_{t+1} and next state s_{t+1}. The probability in (52) is as argued above clearly differentiable with respect to θ. As z was arbitrary, this show that this is true for all sequences z. We will now find an expression for the derivative of the *log*-likelihood $\nabla_\theta \log p(z; \theta)$. As the product in (52) turns into a sum, we have

$$\nabla_\theta \log p(z; \theta) \tag{53}$$

$$= \nabla_\theta \log \Big(p(s_0) \prod_{t=0}^{\tau-1} \pi_\theta(a_t|s_t) p(r_{t+1}, s_{t+1}|s_t, a_t) \Big) \tag{54}$$

$$= \nabla_\theta \Big(\log p(s_0) + \sum_{t=0}^{\tau-1} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{\tau-1} \log p(r_{t+1}, s_{t+1}|s_t, a_t) \Big) \tag{55}$$

$$= \sum_{t=0}^{\tau-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{56}$$

as the gradients of all the constant terms (with respect to θ) are zero. Our approximation of the gradient then becomes

$$\nabla_\theta E_{\pi_\theta}(G_0) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{\tau^{(i)}} \Big(\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) r^{(i)} \Big) \tag{57}$$

where we have made the dependency of the actions, rewards and states on the episode explicit by a superscript, where $r^{(i)} = r(z_i)$. Following the direction of steepest ascent, we then get an update

$$\theta \leftarrow \theta + \alpha \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{\tau^{(i)}} \left(\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) r^{(i)} \right) \tag{58}$$

Before we move on, let's see if we can get some intuition on the updates for our policy function π_θ. First we note that for each action a_t, $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is a direction which *increases* the probability of choosing a_t from state s_t. We thus see that if we get a positive reward we get a contribution for this term on θ that increases the future frequencies for action a_t. Furthermore, the higher the reward, the bigger the change. If the reward is negative on the other hand, the direction is turned around, and the term gives a contribution that *decreases* the probability of the action. We also see that as we use the *sum* of the rewards $r(z_i)$, and not the immediate rewards, we will have that either *all* terms try to increase the probability of the actions taken in the episode, or *all* terms try to decrease the probability of those actions. An intuitive reasoning may go something like this: If the outcome was good, as measured by a high reward, the actions taken were probably pretty good. We update the parameters θ so to increase the probability of those actions so that they are picked more often. With a negative reward the same reasoning leads us to update the parameters so that we are less likely to see those actions in the future. Although this at first seems to make intuitive sense, the reasoning should however also point us to a possible weakness of the policy gradient method. A single mistake may ruin an until then well played chess game. On the other hand, even a clear win may contain a few suboptimal moves the opponent was not able to take advantage of. By treating all actions as either good or bad[12], we completely ignore the *credit assignment* problem discussed in the introduction. We will see that we will be able to address this problem later in Section 3.3 on *actor-critic* methods.

There is however a couple of quick fixes we can do even before then. You may have been wondering about why we multiply $\log \pi_\theta(a_t|s_t)$ with the whole cumulative reward $r(z_i)$ and not just the *future* rewards after action a_t has been taken. Clearly action a_t cannot take credit or blame for any rewards that happened *before* the action was taken. Indeed it is so. With a

[12]The norwegian expression "skjære alle over én kam" seems fitting here.

clever trick we can show that

$$\nabla_\theta E_{\pi_\theta}(G_0) = E[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t] \tag{59}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{\tau^{(i)}} \left(\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) G_t^{(i)} \right) \tag{60}$$

Another practical trick we can do is in normalizing rewards. Imagine e.g. if all rewards were positive, then no matter what we do we will increase the probability of the sampled actions! At first it might sound like this cannot be correct, how can we improve our policy if this was the case? There is no fault however as the updates for the actions with high reward will be *larger*. Intuitively it is clear however that we will make more updates in the wrong direction, even though they are correct *on average*. We would instead like that unusually low returns were *negative* so that actions that led to such an outcome are discouraged; thought where we still have that especially high returns stay positive. We could solve this problem by subtracting an appropriate scalar m. In fact we are fully in the right doing so. Recall that our definition of the gradient we have just estimated was

$$\nabla_\theta E_{\pi_\theta}(G_0) = \nabla_\theta \int p(z;\theta) r(z) dz \tag{61}$$

If we now subtract a constant m from our return[13] we have

$$\nabla_\theta \int p(z;\theta)(r(z) - m) dz = \nabla_\theta \int p(z;\theta) r(z) dz - m \nabla_\theta \int p(z;\theta) dz \tag{62}$$

$$= \nabla_\theta \int p(z;\theta) r(z) dz - m \nabla_\theta 1 \tag{63}$$

$$= \nabla_\theta \int p(z;\theta) r(z) dz \tag{64}$$

and we thus see that the gradient does not change. Our new gradient estimate then becomes

$$\nabla_\theta E_{\pi_\theta}(G_0) = E[\sum_{t=1}^{\tau} \nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - m)] \tag{65}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{\tau^{(i)}} \left(\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})(G_t^{(i)} - m) \right) \tag{66}$$

[13] Note that shifting the *immediate* rewards r_t however would usually get us into trouble, see problem in Section 5.1

This is a poor man's version of what is to come in Section 3.3. We may also *scale* the rewards. This is equivalent to a change in the update parameter α, but scaling the rewards to fall within some fixed range may let us get away with fewer changes to this parameter across experiments.

3.2 Policy iteration and value iteration - value based control

3.2.1 Policy iteration

Unlike *policy gradient* our next approach will not be that direct. Instead of directly optimizing a policy, we will start with the perhaps less ambitious goal of *prediction*. Assume that we have used one of the techniques described in Section 2 to estimate the action-value function $q_\pi(s, a)$. We saw earlier that the value-function is a weighted average of the state-action function, where we average actions with weights given by our policy, i.e.

$$v_\pi(s) = \int \pi(a|s) q_\pi(s, a) da$$

Lets fix a state $s' \in \mathcal{S}$. Just among the actions where $\pi(a|s') > 0$ we must have at least one action a such that $q_\pi(s', a) \leq v_\pi(s')$ and an action a' such that $q_\pi(s', a') \geq v_\pi(s')$ (why?). If we look at all actions we may typically have some actions a such that $q_\pi(s', a) < v_\pi(s')$ and some actions a' such that $q_\pi(s', a') > v_\pi(s')$. It thus seems that from a given a state s' there are some actions that promises a better future than others. If you were given the option to deviate from the policy π this one time, what would you do? It is tempting to choose $a' = argmax_a q_\pi(s', a)$. Our expected future rewards are certainly no worse than what we could have gotten following our policy, i.e. $q_\pi(s', a') \geq v_\pi(s')$ and quite possibly better. It thus seems like we can increase our rewards by sometimes deviating from our policy. Intuitively it seems like if it makes sense to choose a' from state s' once, it should make sense to this *every* time, as we each time find ourselves in the same situation. This would correspond to a *policy change*, defining a new policy π' such that $\pi'(a|s) = \pi(a|s)$ for all $s \neq s'$ and $\pi'(s') = a'$ (i.e. we deterministially choose a' from s' under π'). Before you implement this policy change you may suddenly have seconds thought however. We know for sure that it was a good idea to pick a' from s' if we were to follow policy π from then on. When we change the future actions as well, we might imagine that it wasn't such a great idea after all. We shall soon let all your worries come to rest, and see that your initial intuition were entirely correct. What we shall prove though is the case where we take the tactic above to the extreme, so that we

define our new policy

$$\pi'(s) := argmax_a q_\pi(s,a) \tag{67}$$

for all $s \in \mathcal{S}$, then

$$\pi' \geq \pi \tag{68}$$

Furthermore if $\pi' = \pi$, in the sense that $v_{\pi'}(s) = v_\pi(s)$ for all $s \in \mathcal{S}$ then $\pi = \pi^*$. This is called the *policy improvement* theorem. A proof of the first part can be found in Section 7.3.

The policy improvement theorem then tells us one way we could improve our policy. After we have our improved policy π' we may estimate its value function $q_{\pi'}$. As there is no reason to believe that this new policy is optimal, we may apply the same improvement step outlined above to $q_{\pi'}$. This alternation between *prediction* and *control* leads to an algorithm which we call *policy iteration*. Assume that we have an initial policy π_0. For $i = 0, 1, 2, \ldots$ repeat the following two steps

1. **Policy evaluation** Estimate the value function for policy π_i.
2. **Policy improvement** Define a policy π_{i+1} by acting *greedily* with respect to the value function estimated in the previous step.

For the first step we may use any of the sampling based approaches introduced in Section 2. The second step is easily accomplished if we have a finite action space with a manageable number of states, as the optimization problem in (67) is simply picking the largest number in a small set. In other cases it may be prove more challenging however. Note that we need not actually store the policies π_{i+1} as it is implicitly defined through q_i. There is only one problem with the discussions so far. We have assumed that in the first step we are able to fully figure out the state-value function $q_\pi(s,a)$. It turns out that when we use sampling-based approaches, which not necessarily update all state-action pairs, the approach may actually break down and we may get stuck with a suboptimal policy. We need to keep *exploring* to make sure that we don't prematurely rule out an action a even though our initial estimate of $\hat{q}(s,a)$ is low. In the case of a finite action space a simple way to make sure that we keep visiting all state-action pairs is to pick a random action ϵ of the time, choosing the greedy policy the rest of the time. The policy iteration method for this case becomes

1. **Policy evaluation** Estimate the value function $\hat{q}_{\pi_i}$

2. **Policy improvement** Define a policy π_{i+1} by acting ϵ-*greedily* with respect to $\hat{q}_{\pi_i}$

$$\pi_{i+1}(a|s) = \begin{cases} 1 - \epsilon + \epsilon/K & \text{for a } = argmax_{a'} \hat{q}_{\pi_i}(s, a') \\ \epsilon/K & else \end{cases}$$

In practice we often do not even attempt to estimate q_{π_i} very accurately at each iteration. The policy evaluation may be a very expensive operation to have in an inner loop, and we might not want to to spend too much time evaluating a policy we are about to change anyway. We may also get policy improvement as long as the *relative* estimates are somewhat correct, even though the value estimates themselves are not. How much time to spend on policy evaluation between policy improvements may be treated as a hyperparameter of the algorithm.

3.2.2 Value iteration

We have so far learned about the Bellman *expectation* equations, which led to an algorithm for policy *evaluation* through temporal-difference learning. They do however also exist in another form, the Bellman *optimality* equations[14], which will lead to an algorithm for policy *optimizing*. The Bellman optimaliy equations will not try to estimate the value function for a *given* policy, but try to directly estimate the *optimal* value function! If we let v_* denote the optimal value function, the Bellman optimality equations for the optimal state-value function v_* are

$$v_*(s) = \max_a E[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \tag{69}$$

$$= \max_a \int \int p(r, s'|s, a)(r + \gamma v_*(s'))ds'dr \tag{70}$$

As usual we shall often find the action-value function more convenient to work with in settings with an unknown environment. The Bellman optimality equations for q_* are

$$q_*(s, a) = E[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] \tag{71}$$

$$= \int \int p(r, s'|s, a)(r + \gamma \max_{a'} q_*(s', a'))ds'dr \tag{72}$$

[14] This is indeed usually the form implied when only the term *Bellman equations* is used

Note that given q_* the optimal policy is given by $\pi_*(s) = argmax_a q_*(s,a)$. We may think of value iteration as combining the two steps of policy iteration into a singe one! If we knew the dynamics of the environment we could turn either of the equations above into an update equation. In Section 2.2 we argued in the context of the Bellman *expectation* equations we should update the left-hand side in the term of the right-hand side. It turns out that this is also the right thing to to for the Bellman *optimality* equations. As we usually do not know the environment dynamics, we will generally need to *sample* the right hand side. To be concrete we shall first look at the case with discrete state and action spaces without function approximation (where we store estimates in an array or table). The equation (71) transforms into the update

$$Q[s,a] \leftarrow Q[s,a] + \alpha((R + \max_{a'} Q[S',a']) - Q[s,a]) \tag{73}$$

With function approximation we have the update

$$\lambda \leftarrow \lambda + \alpha\big((R + \max_{a'} q_\eta(S',a')) - q_\eta(s,a)\big)\nabla_\eta q_\eta(s,a) \tag{74}$$

What we have not yet discussed is which policy we should use to sample our actions. It is interesting to compare the updates in (73) and (74) to the TD-updates for policy evaulation in equations (32) and (34). It is like we are doing policy evaluation with respect to the policy $\pi(s) = argmax_a \hat{q}_*(s,a)$[15]. This should perhaps suggest that we should use this policy to sample actions. As we are actually not doing policy *evalution*, but policy *optimization*, this unfortunately suffers the same issues that the greedy update of policy iteration. Due to our incomplete knowledge of the environment we need to *explore*. We may apply the same fix as for policy iteration, though, take the greedy action with probability $(1-\epsilon)$ and sample a random action the rest of the time.

3.3 Actor-critic - policy *and* value based control

Recall the policy gradient estimate from Section 3.1

$$\nabla_\theta E_{\pi_\theta}(G_0) = E[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t] \tag{75}$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{\tau^{(i)}} \Big(\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) G_t^{(i)}\Big) \tag{76}$$

[15]Note that we can write $\max_{a'} \hat{q}_*(S',a') = \hat{q}_*(S', argmax_{a'} \hat{q}_*(S',a'))$

If the reward G_t following action a_t is positive we make an update to increase the probability of the action. The more positive it is, the more we try to increase it. If the reward is negative, we make an update to *decrease* the frequencey of action a_t from state s_t. Thus G_t in somes sense act as a *critic* judging the goodness of the actions of the *actor*. The actor listens to the critic when adjusting his policy, incresing or decreasing its probabilities depending on the evaluation of the critic. The critic G_t on *average* brings us in the right direction, but has potentially very high variance. E.g. if we have only positive rewards this critic will always judge *any* action to be good, which will necessarily make many of our updates go in the wrong direction. We introduced at the end of Section 3.1 of a simple fix to this overly positive critic. This only touched the surface of the problem, however. A series of actions that leads to a miraculous save from a seemingly hopeless situation should probably get a good evaluation, even if we don't end up with a high reward. We should thus take into account our *expectations* from a particular situation. Intuitively we would like to increase that probability of actions that give better outcomes than we get on average following our policy. E.g. if we take action a from state s and observe the reward G_t to be quite a bit higher than what we typically get, $v_\pi(s)$, it seems reasonable that we should increase the frequency of action a from state s. This thus gives rise to the critic $G_t - v_\pi(s_t)$. As the value function v_π is not given to us, we must estimate it, which will introduce some bias in our critic. Our gradient estimate then becomes

$$\nabla_\theta E_{\pi_\theta}(G_0) = E[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - v_\pi(s_t))] \tag{77}$$

$$\approx E[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - \hat{v}_\pi(s_t))] \tag{78}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{\tau^{(i)}} \left(\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})(G_t^{(i)} - \hat{v}_\pi(s_t)) \right) \tag{79}$$

Although this critic captures most of the properties that we would like a critic to have, it still suffers from the high variance of G_t. A critic, similar in spirit but with lower variance, is $(R_{t+1} + \gamma v_\pi(S_{t+1})) - v_\pi(s)$. We only look one step ahead, and see if our situation has improved or not. Our gradient estimate is then

$$\nabla_\theta E_{\pi_\theta}(G_0) = E[\sum_{t=1}^{\tau} \nabla_\theta \log \pi_\theta(a_t|s_t)((R_{t+1} + \gamma v_\pi(S_{t+1})) - v_\pi(s_t))] \tag{80}$$

$$\approx E[\sum_{t=1}^{\tau} \nabla_\theta \log \pi_\theta(a_t|s_t)((R_{t+1} + \gamma \hat{v}_\pi(S_{t+1})) - \hat{v}_\pi(s_t))] \tag{81}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{\tau^{(i)}} \Big(\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})((R_{t+1}^{(i)} + \gamma \hat{v}_\pi(S_{t+1}^{(i)})) - \hat{v}_\pi(s_t^{(i)})) \Big) \tag{82}$$

Actor-critic methods naturally alternates between policy updates, updating π, and policy evaluation, updating $\hat{v}_\pi$, and may be viewed as *generalized* policy iteration methods. For policy evaluation we might use any of the methods presented in Section 2.

4 Conclusion

Reinforcement learning is a very elegant framework and many learning problems can naturally be stated as a reinforcement learning problem. One of the really appealing properties about reinforcement learning is that it can potentially learn without direct supervision. For supervised learning the problem we are really concerned with is *imitation*. We try to copy and also generalize based on examples. It is a problem that at least someone already knows how to solve. If we had a huge database of chess games of expert players, we could use supervised learning to try to copy their behaviour. The supervised learning problem is not trivial as it would have to make intelligent generalizations to make good moves for board positions never seen during training. If we succeed we would probably get a pretty decent chess player. However as the goal was to imitate, it will also inherit the same imperfections as the human chess players had. With reinforcement learning, however, we try to learn an *optimal* policy. Not only does it require no supervision at all, it can go *beyond* the capabilities of any human.

Unfortunately reinforcement learning has some obstacles that may complicate or prohibit its use. First of all the trial-and-error based learning approach taken in reinforcement learning may be both challenging and computationally very expensive. Furthermore for many learning problems it may be hard to create a suitable environment for the agent to learn in. We can't just put a self-driving car out in the street, and give it a negative reward

every time it runs over a human. Providing *virtual environments* through simulation could in many cases be an acceptable solution. Creating *realistic enough* virtual environments may however be too difficult or too expensive for some applications. Think about a robot that needs to interact with humans. To make the simulation realistic we would have to be able to fully simulate human behaviour! You should hope that realistic *enough* for your application may be satisfied by much cruder models of human beings. Last, but not least, designing the appropriate reward function may be challenging in many real-world scenarios. It can be very difficult, if not impossible, to capture every expectation we have of the agent into a reward function. We may end up putting too much emphasis on the things that are easy to measure. One of the stated advantages of reinforcement learning is that we don't have to specify *how* the agent solves to problem. This may however make the model unpredictable and given an imperfect reward function we may find that we don't really *like* the *how*! An intersting blog post both illustrating the problem and suggesting ways to counteract it can be found at `https://blog.openai.com/faulty-reward-functions/`.

5 Problems

5.1 Shifting rewards

Normalizing rewards in episodic and non-episodic environments... The questions here shall be

5.1.1 Problem

Assume that you are in an episodic environment. What happens if you shift all rewards r_t by subracting a positive constant m, i.e. $r'_t = r_t - m$ are our new rewards? What happens if you do the same in a continual environment?

5.2 Discount factor

5.2.1 Problem

Imagine you try to learn the agent to play a game, e.g. chess, with very high discounting, e.g. $\gamma = 0.5$. What difference in learned behaviour would you expect to see as a result of the high discounting factor?

6 Bibliography

References

7 Appendix

7.1 The purpose of the discount factor

Why do we sometimes discount future rewards with a factor $\gamma \in (0,1)$? The reward is then

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{83}$$

The question really contains two questions.

1. Why do we introduce a function $g(k)$ where $g(k) \in (0,1]$ for $k \in \mathbb{N}$ and $g(k) \to 0$ as $k \to \infty$ such that

$$G_t = \sum_{k=0}^{\infty} g(k) r_{t+k+1} \tag{84}$$

   ?

2. Why do we let $g(k) = \gamma^k$, $\gamma \in (0,1)$?

The first is that we for some reason value rewards now more than later. It might be argued that the future is uncertain, we might not even live tomorrow, so it's better to take a sure thing now.

The answer to the second question is twofold, where the first answer also complements the answer to the first question! Firstly, having a function $g(k)$ such that $\lim_{l\to\infty} \sum_{k=l}^{\infty}(k) \to 0$[16] is very convenient for dealing with continual learning, i.e. non-episodic environments. Assuming we have a bounded rewards, which we usually do, we effectively create finite episodes! Let M be the bounds for our rewards. Now for any $\epsilon > 0$ we can choose l such that $\sum_{k=l}^{\infty} g(k) < \epsilon$. We now split the future rewards into two, the rewards we get before time $t+l$ and the rewards from then

$$G_t = \sum_{k=0}^{l-1} g(k) r_{t+k+1} + \sum_{k=l}^{\infty} g(k) r_{t+k+1} \tag{85}$$

[16] To see that $g(k) = \gamma^k$ satisfies this, we note that $\sum_{k=l}^{\infty} \gamma^k = \gamma^l \sum_{k=0}^{\infty} \gamma^k = \gamma^l \frac{1}{1-\gamma} \to 0$ as $\gamma^l \to 0$ as $l \to \infty$.

If we look at the second term we see that

$$|\sum_{k=l}^{\infty} g(k) r_{t+k+1}| \leq \sum_{k=l}^{\infty} g(k)|r_{t+k+1}| \leq \sum_{k=l}^{\infty} g(k)M = M\sum_{k=l}^{\infty} g(k) \leq M\epsilon \quad (86)$$

As ϵ was arbitrary we see that we may make the second term as small as we would like. We can thus reduce the problem of maximizing $\sum_{k=0}^{\infty} g(k) r_{t+k+1}$ to that of maximizing $\sum_{k=0}^{l-1} g(k) r_{t+k+1}$ with only the possibility of a very small error.

This doesn't answer the question fully though, why not use a different function $g(k)$ that satisfies $\lim_{l\to\infty} \sum_{k=l}^{\infty} g(k) \to 0$, e.g. $g(k) = \frac{1}{(k+1)^2}$? It turns out however that other functions are not as convenient to work with. $g(k) = \gamma^k$ has the nice property that $g(k+1) = \gamma g(k)$ for all k. This is very convenient and allows us to get nice recursive relationships independent of time. It turns out that only functions that has the property $g(k+1) = \gamma g(k)$ for all k are functions of the form $g(k) = k\gamma^k$, $k \in \mathbb{R}$.

7.2 Derivate under the integral sign

In reinforcement learning we often find ourselves in situations where our data-distribution depends on some parameters θ. When our objective function depends on this distribution, we often find that we would like to find the gradient of the objective function with respect to our parameters. Often the objective function is defined as an expectation, and the gradient function is of the form

$$\nabla_\theta E_\theta[f(Z)] = \nabla_\theta \int p(z;\theta) f(z) dz \quad (87)$$

It is usually not possible to perform the integration, and even harder to get an analytical solution so to allow us to derive the gradient with respect to θ. It is then very tempting if we could just

$$\nabla_\theta \int p(z;\theta) f(z) dz = \int \nabla_\theta p(z;\theta) f(z) dz \quad (88)$$

because of the log-derivative trick we know that

$$\int \nabla_\theta p(z;\theta) f(z) dz = \int p(z;\theta) \nabla_\theta \log p(z;\theta) f(z) dz \quad (89)$$

which is something we can sample from, given that we know how to generate samples z from $p(z,\theta)$. Under what conditions is this operation actually valid though, if at all?

We might at first try an argument as follows: We know that the derivative of a sum is the sum of derivatives, i.e.

$$\frac{d}{dt}\sum_{x=1}^{n} g(x,t) = \sum_{x=1}^{n} \frac{d}{dt} g(x,t) \tag{90}$$

As integration and summation is basically the same, this should also work with integration. Not so fast! The result above was only for *finite* sums, and it will be the potential unboudedness of the infinite that may cause issues for us.

Let $g\colon \mathbb{R}\times\mathbb{R}\to\mathbb{R}$ and assume g is differentiable in the second variable. We want to find the derivative of the function $h\colon \mathbb{R}\to\mathbb{R}$, where $h(t) = \int g(x,t)dx$. This is thus

$$\frac{d}{dt}h(t) = \frac{d}{dt}\int g(x,t)dx \tag{91}$$

By using the the definition of the derivative we have

$$\begin{aligned}
\frac{d}{dt}h(t) &= \lim_{\Delta\to 0}\frac{h(t+\Delta)-h(t)}{\Delta} && (92)\\
&= \lim_{\Delta\to 0}\frac{1}{\Delta}\Big(\int g(x,t+\Delta)dx - \int g(x,t)dx\Big) && (93)\\
&= \lim_{\Delta\to 0}\int \frac{g(x,t+\Delta)-g(x,t)}{\Delta}dx && (94)
\end{aligned}$$

If we could only bring the limit *inside* the integral we would have that

$$\int \lim_{\Delta\to 0}\frac{g(x,t+\Delta)-g(x,t)}{\Delta}dx = \int \frac{\partial}{\partial t}g(x,t)dx \tag{95}$$

which would give the desired result

$$\frac{d}{dt}\int g(x,t)dx = \int \frac{\partial}{\partial t}g(x,t)dx \tag{96}$$

It turns out that this is sometimes possible, other times it is not. We shall now look at a *sufficient* conditions for this to be true, and see that indeed there is hope for many of the cases we care about.

- Add proof here...

We shall now look at the case

$$\nabla_\theta E_\theta[f(Z)] = \nabla_\theta \int p(z;\theta) f(z) dz \tag{97}$$

where we assume f is bounded, i.e. $f(z) \leq M$ for all z. We have that

$$\nabla_\theta p(z;\theta) f(z) = p(z;\theta) \nabla_\theta \log p(z;\theta) \tag{98}$$

It is now sufficient to show that $|\frac{\partial}{\partial \theta_i} \log p(z;\theta)| \leq L$ for some $L \in \mathbb{R}$. To show this it is sufficient to show that the function is Lipschitz-continuous, i.e.

If you want to try to go about and prove this rigoursly, it might be useful to keep in mind that if $g(x) = g^(n)(g^{(n-1)}(\dots g^{(1)}(x)\dots))$ it is sufficient to show that each function $g^{(i)}$ is Lipschitz to prove that g is Lipschitz with respect to x[17]. The converse is not true, g can be Lipschitz even if not all the functions $g^{(i)}$ are Lipschitz. If this was not the case we would never have $\log p(z;\theta)$ to be Lipschitz! The simple example $g^(1)(x) = 1x$, $g^(2)(x) = x^2$, shows that indeed none of them need to be!

7.3 Proof of policy improvement theorem

We only prove the first part, i.e. that defining a new policy π' by acting greedily with respect to a value function q_π always improves the policy, in the sense that $v_{\pi'}(s) \geq v_\pi(s)$ for all $s \in \mathcal{S}$. Assume bounded rewards and discount factor $\gamma < 1$. Let $s \in \mathcal{S}$. We will prove the result by first showing that from state s we can improve our expected return by picking our inital step from π' instead of π. Then we will show that actually it is even better to take *two* steps of policy π' before we switch back to policy π. By an inductive argument it may then be shown that following π' for *one more step* always improves our expected return. In the limit, we are actually following policy π', and as we have just kept improving our prospects it will follow that $v_{\pi'}(s) \geq v_\pi(s)$. As s was arbitrary we obtain the desired inequality $\pi' \geq \pi$. Let's now prove each of the steps outlined. The first step is easy, clearly

$$v_\pi(s) = E_\pi[q_\pi(s, A_t)] \leq max_a q_\pi(s,a) = E_{\pi'}[q_\pi(s, A_t)] \tag{99}$$

which shows that taking the *first* action according to π' is a good idea. The second step will require some more work. We shall first show derive a result

[17] This can be proved by induction.

which shall come in handy at least a couple of times. For every $s \in \mathcal{S}$ and $a \in \mathcal{A}$

$$\begin{aligned}
q_\pi(s,a) =& E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] && (100)\\
\leq& E_\pi[R_{t+1} + \gamma \max_{a'} q_\pi(S_{t+1}, a')|S_t = s, A_t = a] && (101)\\
=& E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, argmax_{a'} q_\pi(S_{t+1}, a'))|S_t = s, A_t = a] && (102)\\
=& E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})|S_t = s, A_t = a] && (103)\\
=& E_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] && (104)
\end{aligned}$$

This also implies the inequality for the random variables below

$$q_\pi(s, A_t) \leq E_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t] \quad (105)$$

Taking the expectation with respect to π' and using the law of iterated expectation

$$\begin{aligned}
E_{\pi'}[q_\pi(s, A_t)] \leq& E_{\pi'}[E_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t]] && (106)\\
=& E_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s] && (107)
\end{aligned}$$

We have now proved that picking our initial *two* steps according to π' is a good idea. We can now proceed by induction. Assume that the inequality below holds for all n such that $1 \leq n \leq m-1$

$$E_\pi[q_\pi(s, A_t)] \leq E_{\pi'}[\sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n q_\pi(S_{t+n+1}, A_{t+n+1})|S_t = s] \quad (108)$$

We now want to show that it also for $n = m$. Setting $n = m-1$ the following holds true

$$E_\pi[q_\pi(s, A_t)] \leq E_{\pi'}[\sum_{k=0}^{m-2} \gamma^k R_{t+k+1} + \gamma^{m-1} q_\pi(S_{t+m}, A_{t+m})|S_t = s] \quad (109)$$

By application of the result obtained through equations (100) - (104), we have that

$$q_\pi(S_{t+m}, A_{t+m}) \leq E_{\pi'}[R_{t+m+1} + \gamma q_\pi(S_{t+m+1}, A_{t+m+1})|S_{t+m}, A_{t+m}] \quad (110)$$

Note that both the left-hand and right-hand side are random variables, but for any samples of S_{t+m} and A_{t+m} the inequality holds true. Inserting this into (111) we get

$$E_{\pi'}[\sum_{k=0}^{m-2}\gamma^k R_{t+k+1} + \gamma^{m-1} q_\pi(S_{t+m}, A_{t+m})|S_t = s] \tag{111}$$

$$\leq E_{\pi'}[\sum_{k=0}^{m-2}\gamma^k R_{t+k+1} + \gamma^{m-1} E_{\pi'}[R_{t+m+1} + \gamma q_\pi(S_{t+m+1}, A_{t+m+1})|S_{t+m}, A_{t+m}]|S_t = s] \tag{112}$$

$$= E_{\pi'}[\sum_{k=0}^{m-2}\gamma^k R_{t+k+1} + \gamma^{m-1}\big(R_{t+m+1} + \gamma q_\pi(S_{t+m+1}, A_{t+m+1})\big)|S_t = s] \tag{113}$$

$$= E_{\pi'}[\sum_{k=0}^{m-1}\gamma^k R_{t+k+1} + \gamma^{m} q_\pi(S_{t+m+1}, A_{t+m+1})|S_t = s] \tag{114}$$

where the second to last equality follows from the law of iterated expectations, and the last is simply rearranging. We have just showed the induction step, which proves that equation (108) holds true for any n. Spitting the right-hands side into two, and factoring out the γ^n factor, we have

$$E_\pi[q_\pi(s, A_t)] \leq E_{\pi'}[\sum_{k=0}^{n-1}\gamma^k R_{t+k+1}|S_t = s] + \gamma^n E_{\pi'}[q_\pi(S_{t+n+1}, A_{t+n+1})|S_t = s] \tag{115}$$

Taking the limit as $n \to \infty$ the inequality

$$E_\pi[q_\pi(s, A_t)] \leq \lim_{n\to\infty}(E_{\pi'}[\sum_{k=0}^{n-1}\gamma^k R_{t+k+1}|S_t = s] + \gamma^n E_{\pi'}[q_\pi(S_{t+n+1}, A_{t+n+1})|S_t = s]) \tag{116}$$

$$= \lim_{n\to\infty} E_{\pi'}[\sum_{k=0}^{n-1}\gamma^k R_{t+k+1}|S_t = s] + \lim_{n\to\infty}\gamma^n E_{\pi'}[q_\pi(S_{t+n+1}, A_{t+n+1})|S_t = s] \tag{117}$$

where the split can be justified as both sequences are bounded, we are assuming bounded rewards and $\gamma < 1$. Clearly the right-term converges to

zero. Furthermore we may move the limit inside the integral[18] to obtain

$$\lim_{n\to\infty} E_{\pi'}[\sum_{k=0}^{n-1} \gamma^k R_{t+k+1}|S_t = s] = E_{\pi'}[\lim_{n\to\infty} \sum_{k=0}^{n-1} \gamma^k R_{t+k+1}|S_t = s] \tag{118}$$

$$= E_{\pi'}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \tag{119}$$

$$= v_{\pi'}(s) \tag{120}$$

Recalling our work we have now shown that

$$v_\pi(s) = E_\pi[q_\pi(s, A_t)] \leq v_{\pi'}(s) \tag{121}$$

which finishes the proof.

[18] This is an application of the Dominated Convergence Therem, which is also gives sufficient conditions on "differentiation under the integral sign" as can be read about in Section 7.2