On the policy gradient method

Kamila Zdybał

kamilazdybal.github.io, kamila.zdybal@gmail.com

Please feel free to contact me with any suggestions, corrections or comments.

Preface

Reinforcement learning (RL) is all about finding optimal *policies*–rules of acting on or behaving in an environment. The policy, π, is simply a function that takes observations of the environment as inputs and it outputs actions to be executed in that environment. There's a couple of different methodological approaches to how one can obtain the optimal policy (or *train* the RL). In particular, the policy gradient method (PGM) is one of the RL algorithms where we directly learn the policy function, $\pi_{\boldsymbol{\theta}}$, with its trainable parameters $\boldsymbol{\theta} \in \mathbb{R}^d$. The functional form for the policy can be imposed by us in any way that we want, as long as it is differentiable[1] with respect to $\boldsymbol{\theta}$. Since the advent of deep learning, we commonly use artificial neural networks as the function approximators for $\pi_{\boldsymbol{\theta}}$. In PGM, π is a very specific function and one that outputs **the probability of each action being selected**[2]. To actually select an action, we sample from those probabilities. In other words, we write that $\pi = \pi(a_t|s_t)$. With sufficient training, the optimal PGM policy returns the maximum probability for the *best* action to be taken in each state in the environment.

This document is an educational tutorial deriving and explaining the PGM in more depth. The pre-requisites are that you understand what RL is in general (what is state, action, reward) and that you've been exposed to training of artificial neural networks (ANNs). I specifically wanted to root the intuition behind PGM in the intuition that you may already have from training general-purpose ANNs. This document is a bit mathy, but stick with it and you will find that it's actually not that much more of a conceptual advance from the math of gradient descent!

Keywords

reinforcement learning, policy gradient method, deep neural networks

[1] The differentiability requirement is necessary because the core mechanism of the PGM involves computing the gradient of $\pi_{\boldsymbol{\theta}}$, as we shall see later. Hence the algorithm is called the *policy gradient* method.

[2] This is an important assumption that we make use of in the derivation of the PGM. If the policy ever outputs anything else, the present derivation of the PGM may need a modification.

The core concept

The PGM, at its core, is trained in the same way as we train artificial neural networks in general. We compute the gradient of some performance measure with respect to the trainable parameters, $\boldsymbol{\theta}$, and we update $\boldsymbol{\theta}$ in the direction dictated by that gradient by taking a small step (measured by the learning rate, α) in that direction.

In a classic regression problem, the performance measure could be the mean-squared-error between the true and the predicted targets, call it $\text{MSE}(\boldsymbol{\theta})$. We want to nudge $\boldsymbol{\theta}$ in the direction that *minimizes* $\text{MSE}(\boldsymbol{\theta})$, so we would be taking a step in the direction *opposite* to the gradient of $\text{MSE}(\boldsymbol{\theta})$. The parameter update rule in that case is

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta}) \,. \tag{1}$$

Doing this update iteratively gives us the classic gradient *descent* algorithm.

To be completely precise, in the deep learning world, we use a *stochastic* gradient descent, which means that $\text{MSE}(\boldsymbol{\theta})$ is estimated from samples of true *vs*. predicted targets. We cannot know the raw $\text{MSE}(\boldsymbol{\theta})$ for *all* possible values of targets–there's astronomically many of them, if not infinitely many. We can only estimate $\text{MSE}(\boldsymbol{\theta})$ from those values of targets that we have actually sampled as our "training data". But that's okay. As long as our data sampling was statistically representative, the MSE that we compute on those samples is a good enough approximation of the MSE that would have been computed on all possible samples. We denote this stochastic estimate of $\text{MSE}(\boldsymbol{\theta})$ as $\widehat{\text{MSE}(\boldsymbol{\theta})}$ and so, instead of Eq. (1), we actually compute

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} \widehat{\text{MSE}(\boldsymbol{\theta})} \,. \tag{2}$$

Conversely, in the PGM, our performance measure, call it $J(\boldsymbol{\theta})$, is constructed (in some smart way) from the rewards given to the RL agent in the environment. We want to maximize the rewards in the long run, right? So we should be taking steps in the direction that *maximizes* $J(\boldsymbol{\theta})$, or in the direction *of* the gradient of $J(\boldsymbol{\theta})$. Hence, the parameter update rule in PGM is

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \,. \tag{3}$$

As opposed to training the regressor, this is the gradient *ascent* algorithm.

So far so good. If we can compute, or even reasonably approximate $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, then we are all set. But the name of the game in PGMs is how do we get $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ in practice? In the next sections, I would like to first show you why getting the raw $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is impossible in practice (for much the same reasons why we couldn't get the true $\text{MSE}(\boldsymbol{\theta})$ in practice) and then show you a bunch of neat math rearrangements that make estimating $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ possible!

How do we construct the performance measure, $J(\boldsymbol{\theta})$?

Before we construct a specific $J(\boldsymbol{\theta})$, let's take a step back and see what options and restrictions we are dealing with.

1. We've already said that it makes sense that $J(\boldsymbol{\theta})$ is *somehow* constructed from the rewards given to the RL agent as he navigates the environment. In the end, the essence of RL is learning from experience by receiving feedback signals (rewards), which are the only mechanism through which we correct and improve the behavior of the RL agent in the future. We expect that the optimal policy, $\pi^*_{\boldsymbol{\theta}}$, leads to solving the task in the environment perfectly, which we can recognize by observing that the agent receives the highest rewards possible. Usually in RL we use the discounted future rewards, but there is some leeway to how exactly we can make the rewards enter the performance measure.
2. We have to be able to estimate $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ from actual rollouts in the environment. There's no other way. We have to set the agent running and interacting with the environment and see what he accomplishes. We do not have any other mechanism for obtaining the "training data". In that sense, similarly as in all neural network training, we will be computing stochastic estimates of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, which we denote as $\widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$. So, instead of Eq. (3), in PGM we actually compute

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}\,, \tag{4}$$

   for reasons analogous to those that made us arrive at the stochastic estimate of the MSE($\boldsymbol{\theta}$)[3].

Okay, so let's use the reasoning from #1 and write out what $J(\boldsymbol{\theta})$ could be. The most mathematically general statement would be to say that

$$J(\boldsymbol{\theta}) \coloneqq \mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}\left[R(\tau)\right]\,. \tag{5}$$

Here, τ is the particular trajectory that the RL agent takes in the environment, sampled using the current version of the policy, $\pi_{\boldsymbol{\theta}}$, when its parameters are $\boldsymbol{\theta}$. $R(\tau)$ is the "total return" from following the trajectory τ and it is *somehow* constructed from the rewards in the environment, perhaps like so:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t\,, \tag{6}$$

where γ is the discount factor (a number between 0 and 1) and r_t is the reward value given to the agent at the time step t. The sum then takes account of all incremental rewards that were obtained throughout the duration of the trajectory τ, *i.e.*, from time step $t = 0$ to some final time step $t = T$.

[3]For the moment, it is not clear why we are after the stochastic estimate of the **gradient** of $J(\boldsymbol{\theta})$ and not the stochastic estimate of $J(\boldsymbol{\theta})$ directly. This is where Eq. (4) differs from Eq. (2). Please be patient, as this will become clear later!

Let's ponder about that expected value, $\mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}$, for a bit. First, we could also expand the definition of the expected value (in the discrete setting) and write Eq. (5) differently as

$$J(\boldsymbol{\theta}) = \sum_{i=0}^{N} \left[R(\tau^{(i)}) \cdot P(\tau^{(i)}|\boldsymbol{\theta})\right]\,, \tag{7}$$

In other words, the expected value across all sampled trajectories is the sum of total returns weighted by the probability, P, of the specific trajectory τ occurring under the current policy parameters $\boldsymbol{\theta}$. N can be a *really* huge number. In fact, it can be infinite and we would then write the expected value in the continuous setting as

$$J(\boldsymbol{\theta}) = \int_{\tau} \left[R(\tau) \cdot P(\tau|\boldsymbol{\theta})\right]\,. \tag{8}$$

But for the moment, the expected value itself is not scary. We could, in the end, estimate it from samples of a couple of rollouts actually taken, thereby satisfying our restriction #2. We could let the agent take a few, or even many trajectories, τ, and we could average the total returns across those to give us some estimate of $J(\boldsymbol{\theta})$. In other words, getting $\widehat{J(\boldsymbol{\theta})}$ wouldn't be difficult at all if we ever needed that. But recall that in Eq. (3) we need the gradient of $J(\boldsymbol{\theta})$ in order to know how to improve the policy. And things complicate when we take that gradient...

The complication that arises when taking the gradient of $J(\boldsymbol{\theta})$

Let's take the gradient of $J(\boldsymbol{\theta})$ as defined in Eq. (4) to see what happens! First, since the gradient is with respect to the parameters $\boldsymbol{\theta}$, we ask if the expected value is a function of $\boldsymbol{\theta}$? Well, yes! It is, because the probability of a given trajectory occurring depends on the current policy (which guides the dynamics of the agent's movement across the environment). And the policy explicitly depends on $\boldsymbol{\theta}$. Note that $R(\tau)$ is not a function of the policy nor its parameters–the way that rewards are placed in the environment is simply a function of the environment itself. Eq. (7) shows explicitly that there is a dependence on $\boldsymbol{\theta}$, namely, the probability P is a function of $\boldsymbol{\theta}$. So, at most we can write that

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{i=0}^{N} \left[R(\tau^{(i)}) \cdot \nabla_{\boldsymbol{\theta}} P(\tau^{(i)}|\boldsymbol{\theta})\right]\,. \tag{9}$$

Computing (or estimating) the right-hand-side of the above equation is impossible in practice because we have no way of knowing how the probability P changes as we vary the parameters $\boldsymbol{\theta}$ by just sampling a few trajectories. We'd have to sample *all of them* to know $\nabla_{\boldsymbol{\theta}} P(\tau|\boldsymbol{\theta})$ because those probabilities are coupled. The current form of the gradient of $J(\boldsymbol{\theta})$ violates our restriction #2. But a beautiful simplification arises when we use one math trick and also expand a bit what that probability P is equal to!

Where maths comes to the rescue

First, we are going to use the "log-derivative trick". It states that

$$\nabla_{\boldsymbol{\theta}} P(\tau|\boldsymbol{\theta}) = P(\tau|\boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} \ln\left(P(\tau|\boldsymbol{\theta})\right) . \tag{10}$$

This simply comes from rearrangement[4] of what the derivative of a natural logarithm of a multivariate function is equal to. We could plug in the result from Eq. (10) into Eq. (9) but that doesn't improve our situation, we still ultimately have to deal with the gradient of that probability P (or the natural logarithm of it). Let's see if we can do something about that P.

What is $P(\tau|\boldsymbol{\theta})$, really?

How would we compute $P(\tau|\boldsymbol{\theta})$ for one specific sampled trajectory, τ, given that τ is the following sequence of states, s_t, and actions, a_t,

$$\tau = \{s_0, a_0, s_1, a_1, s_2, a_2, \ldots s_{T-1}, a_{T-1}, s_T\} . \tag{11}$$

Well, three questions contribute to *this particular* trajectory (sequence of states and actions) arising:

1. What was the chance that τ started in *this particular* initial state, s_0?
2. What was the chance that the action selected in s_0 was *this particular* a_0?
3. What was the chance that by executing this particular a_0 in this particular s_0 we then entered *this particular* next state, s_1?

and then, you can continue asking yourself the questions 2. and 3. in a loop as you traverse all the states in τ, all the way till you get to the terminal state, s_T!

So let's write these three factors mathematically in terms of probabilities:

1. The probability that τ started in *this particular* initial state, s_0, is equal to $P_0(s_0)$, where P_0 is some distribution over the possible starting states in the environment. For instance, for entirely fair environments, like a 2D grid world, all grid cells may be equally likely, so $P_0(s_0) \in \mathcal{U}$. But some environments, like games, may have more likely starting positions, and some starting positions are impossible. You may even implement a completely deterministic starting position, so all τ originate from the same starting position. Sky's the limit! But the important take-away is that $P_0(s_0)$ is a property of the environment, and it is not a function of the policy!
2. The probability that the action selected in s_0 was *this particular* a_0 is simply equal to $\pi_{\boldsymbol{\theta}}(a_0|s_0)$–our policy explicitly outputs that probability! (Recall what I mentioned in the preface about π needing to output probabilities explicitly. This is where we are using this assumption!)
3. The probability that by executing this particular a_0 in this particular s_0 we then entered *this particular* next state, s_1, is equal to $P_t(s_1|s_0, a_0)$, where P_t is also known as the state transition probability. This probability is the property of the environment, too. More specifically, a property of any *stochastic* environment. A deterministic environment would have each $P_t = 1$. It defines the dynamics of the environment. It is also not a function of the policy!

With all of this, we are ready to write what is $P(\tau|\boldsymbol{\theta})$ equal to:

$$P(\tau|\boldsymbol{\theta}) = P_0(s_0) \cdot \prod_{t=0}^{T-1} \pi_{\boldsymbol{\theta}}(a_t|s_t) \cdot P_t(s_{t+1}|s_t, a_t) . \tag{12}$$

Note that we simply multiply all of those probabilities because all of those events are independent from one another.

Putting it all together

The mathematical beauty comes from substituting Eq. (12) into the gradient-of-the-logarithm term in Eq. (10)!

First, if we take the natural logarithm of Eq. (12), all products become sums of logarithms:

$$\ln\left(P(\tau|\boldsymbol{\theta})\right) = \ln\left(P_0(s_0)\right) + \sum_{t=0}^{T-1} \ln\left(\pi_{\boldsymbol{\theta}}(a_t|s_t)\right) + \sum_{t=0}^{T-1} \ln\left(P_t(s_{t+1}|s_t, a_t)\right) . \tag{13}$$

Second, if we now take the gradient of Eq. (14), the first and the third term on the right-hand-side vanish, because as we've reasoned earlier, they are not a function of the policy parameters, $\boldsymbol{\theta}$! So we are left with

$$\nabla_{\boldsymbol{\theta}} \ln\left(P(\tau|\boldsymbol{\theta})\right) = \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(a_t|s_t)\right) . \tag{14}$$

This is **a really key result**! It allows us to shift from the need of computing gradients of those probabilities of trajectories, P, which we've mentioned are intractable, to only needing to compute gradients of the policy function itself! (The natural logarithm of it, that is.) And that we can do because we know the functional form of the policy. If it is a neural network, we can always compute how action probabilities change with a specific change in $\boldsymbol{\theta}$. In fact, this is what automatic differentiation keeps track of anyway when we use a package like PyTorch or TensorFlow.

Alright, but let's write out the complete version of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ to see what we have accomplished. We have obtained

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{i=0}^{N} \left[R(\tau^{(i)}) \cdot P(\tau^{(i)}|\boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} \ln\left(P(\tau^{(i)}|\boldsymbol{\theta})\right) \right] \tag{15}$$

from just using the "log-derivative trick". And now we also have

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{i=0}^{N} \left[R(\tau^{(i)}) \cdot P(\tau^{(i)}|\boldsymbol{\theta}) \cdot \sum_{\substack{t=0 \\ \tau=\tau^{(i)}}}^{T-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(a_t|s_t)\right) \right] . \tag{16}$$

[4] For an independent variable, x, we have that $\frac{d}{dx} \ln(x) = \frac{1}{x}$, whereas for a multivariate function f, the chain rule gives $\nabla \ln(f) = \frac{1}{f} \cdot \nabla f$. Now just multiply the latter by f and you get the "log-derivative trick".

Note that we still have the probability P appearing in this equation, but not its gradient. In fact, due to this probability appearing explicitly, we can again recognize the definition of an expected value in Eq. (16)! Hence, we can write that

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[R(\tau^{(i)}) \cdot \sum_{\substack{t=0 \\ \tau=\tau^{(i)}}}^{T-1} \nabla_{\boldsymbol{\theta}} \ln \left(\pi_{\boldsymbol{\theta}}(a_t|s_t) \right) \right] . \quad (17)$$

The result in Eq. (17) allows $\widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$ to be entirely computable from just samples of selected trajectories. In other words, we can estimate this present expected value by running a couple of rollouts in the environment and averaging them in some way. For each rollout, at each time step, we will also be computing the gradient of the policy term.