RL — Policy Gradient Explained

Jonathan Hui · Follow
9 min read · Sep 11, 2018

Photo by Alex Read

Policy Gradient Methods (**PG**) are frequently used algorithms in reinforcement learning (RL). The principle is very simple.

> *We observe and act.*

A human takes actions based on observations. As a quote from Stephen Curry:

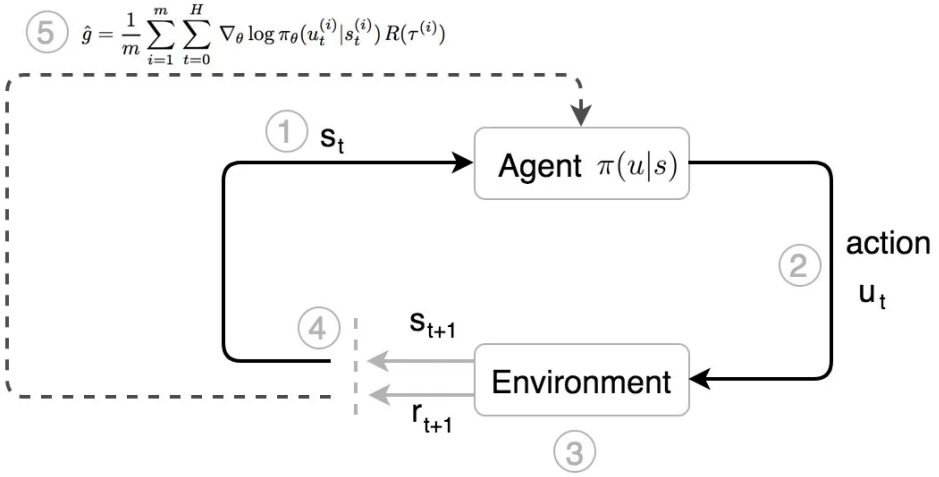
You have to rely on the fact that you put the work in to create the muscle memory and then trust that it will kick in. The reason you practice and work on it so much is so that during the game your instincts take over to a point where it feels weird if you don't do it the right way.

Source

Constant practice is the key to build muscle memory for athletes. For PG, we train a policy to act based on observations. The training in PG makes actions with high rewards more likely, or vice versa.

> *We keep what is working and throw away what is not.*

In policy gradients, Curry is our agent.

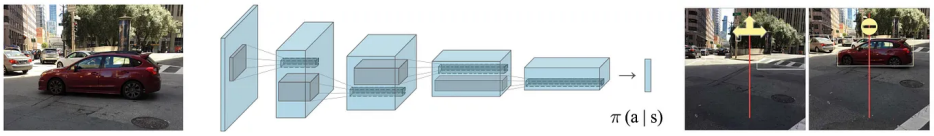


1. He observes the state of the environment (s).
2. He takes action (u) based on his instinct (a policy π) on the state s.
3. He moves and the opponents react. A new state is formed.
4. He takes further actions based on the observed state.
5. After a trajectory τ of motions, he adjusts his instinct based on the total rewards $R(\tau)$ received.

Curry visualizes the situation and instantly knows what to do. Years of training perfects the instinct to maximize the rewards. In RL, the instinct may be mathematically described as:

$$\pi\left(u|s\right)$$

the probability of taking the action u given a state s. π is the **policy** in RL. For example, what is the chance of turning or stopping when you see a car in front:



Objective

How can we formulate our objective mathematically? The expected rewards equal the sum of the probability of a trajectory × corresponding rewards:

$$J(\theta) = \mathrm{E}\Big[\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta\Big] = \sum_{\tau} P(\tau;\theta) R(\tau)$$

And our objective is to find a policy θ that create a trajectory τ

$$(s_1, u_1, s_2, u_2, \ldots, s_H, u_H)$$

that maximizes the expected rewards.

$$\max_\theta J(\theta) = \max_\theta \sum_{\tau} P(\tau;\theta) R(\tau)$$

Input features & rewards

$$\pi_\theta(u|s)$$

s can be handcrafted features for the state (like the joint angles/velocity of a robotic arm) but in some problem domains, RL is mature enough to handle raw images directly. π can be a deterministic policy which output the exact action to be taken (move the joystick left or right). π can be **a stochastic policy** also which outputs the possibility of an action that it may take.

We record the reward r given at each time step. In a basketball game, all are 0 except the terminate state which equals 0, 1, 2 or 3.

$$(s_1, u_1, r_1, s_2, u_2, r_2, \ldots, s_H, u_H, r_H)$$

Let's introduce one more term H called the horizon. We can run the course of simulation indefinitely ($h \to \infty$) until it reaches the terminate state, or we set a limit to H steps.

Optimization

First, let's identify a common and important trick in Deep Learning and RL. The partial derivative of a function *f(x)* (R.H.S.) is equal to *f(x)* times the partial derivative of the *log(f(x))*.

$$\mathrm{f(x)}\ \nabla_\theta \log \mathrm{f(x)} = \mathrm{f(x)}\ \frac{\nabla_\theta \mathrm{f(x)}}{\mathrm{f(x)}} = \nabla_\theta \mathrm{f(x)}$$

Replace f(x) with π.

$$\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau)$$

Also, for a continuous space, expectation can be expressed as:

$$E_{\mathrm{x}\sim \mathrm{p(x)}}[\mathrm{f(x)}] = \int \mathrm{p(x)}\ \mathrm{f(x)}\ dx$$

Now, let's formalize our optimization problem mathematically. We want to model a policy that creates trajectories that maximize the total rewards.

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)}$$

However, to use gradient descent to optimize our problem, do we need to take the derivative of the reward function ***r*** which may not be differentiable or formalized?

Let's rewrite our objective function ***J*** as:

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau) r(\tau) d\tau$$

The gradient (**policy gradient**) becomes:

$$\begin{aligned}\nabla_\theta J(\theta) &= \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau \\ &= E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]\end{aligned}$$

Great news! The policy gradient can be represented as an expectation. It means we can use sampling to approximate it. Also, we sample the value of ***r*** but not differentiate it. It makes sense because the rewards do not directly depend on how we parameterize the model. But the trajectories τ are. So what is the partial derivative of the log $\pi(\tau)$.

$\pi(\tau)$ is defined as:

$$\underbrace{\pi_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

Take the log:

$$\log \pi_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

The first and the last term does not depend on θ and can be removed.

$$\nabla_\theta \left[\log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \right]$$

So the policy gradient

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

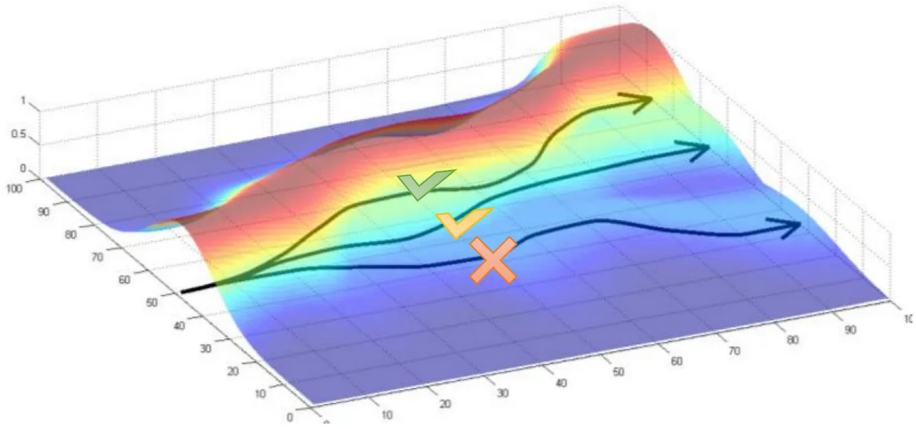
And we use this policy gradient to update the policy θ.

Intuition

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

How can we make sense of these equations? The underlined term is the maximum log likelihood. In deep learning, it measures the likelihood of the observed data. In our context, it measures how likely the trajectory is under the current policy. By multiplying it with the rewards, we want to increase the likelihood of a policy if the trajectory results in a high positive reward. On the contrary, we want to decrease the likelihood of a policy if it results in a high negative reward. In short, keep what is working and throw out what is not.

If going up the hill below means higher rewards, we will change the model parameters (policy) to increase the likelihood of trajectories that move higher.



Source

There is one thing significant about the policy gradient. The probability of a trajectory is defined as:

$$\pi_\theta(\tau) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

States in a trajectory are strongly related. In Deep Learning, a long sequence of multiplication with factors that are strongly correlated can trigger vanishing or exploding gradient easily. However, the policy gradient only sums up the gradient which breaks the curse of multiplying a long sequence of numbers.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

The trick

$$\pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau)$$

creates a maximum log likelihood and the log breaks the curse of multiplying a long chain of policy.

Policy Gradient with Monte Carlo rollouts

Here is the REINFORCE algorithm which uses **Monte Carlo** rollout to compute the rewards. i.e. play out the whole episode to compute the total rewards.

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Source

Policy gradient with automatic differentiation

The policy gradient can be computed easily with many Deep Learning software packages. For example, this is the partial code for TensorFlow:

Policy gradient:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# q_values – (N*T) x 1 tensor of estimated state-action values
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

Source

Yes, as often, coding looks simpler than the explanations.

Continuous control with Gaussian policies

How can we model a continuous control?

$$\text{sample } \{\tau^i\} \text{ from } \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \text{ (run the policy)}$$

Let's assume the values for actions are Gaussian distributed

$$f(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right\}$$

and the policy is defined using a Gaussian distribution with means computed from a deep network:

$$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t); \Sigma)$$

With

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

We can compute the partial derivative of the log π as:

$$\log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2} \| f(\mathbf{s}_t) - \mathbf{a}_t \|_\Sigma^2 + \text{const}$$
$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2} \Sigma^{-1} (f(\mathbf{s}_t) - \mathbf{a}_t) \frac{df}{d\theta}$$

So we can backpropagate

$$-\tfrac{1}{2} \Sigma^{-1} (f(\mathbf{s}_t) - \mathbf{a}_t) \left(\textstyle\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right)$$

through the policy network π to update the policy θ. The algorithm will look exactly the same as before. Just start with a slightly different way in

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i) \right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Source

Policy Gradients improvements

> *Policy Gradients suffer from high variance and low convergence.*

Monte Carlo plays out the whole trajectory and records the exact rewards of a trajectory. However, the stochastic policy may take different actions in different episodes. One small turn can completely alter the result. So Monte Carlo has no bias but high variance. Variance hurts deep learning optimization. The variance provides conflicting descent direction for the model to learn. One sampled rewards may want to increase the log likelihood and another may want to decrease it. This hurts the convergence. To reduce the variance caused by actions, we want to reduce the variance for the sampled rewards.

$$\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

> *Increasing the batch size in PG reduces variance.*

However, increasing the batch size significantly reduces sample efficiency. So we cannot increase it too far, we need additional mechanisms to reduce the variance.

Baseline

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=1}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)}_{Q(s,\, a)}$$

We can always subtract a term to the optimization problem as long as the term is not related to θ. So instead of using the total reward, we subtract it with $V(s)$.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - V(\mathbf{s}_{i,t}) \right)$$

We define the advantage function A and rewrite the policy gradient in terms of A.

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

In deep learning, we want input features to be zero-centered. Intuitively, RL is interested in knowing whether an action is performed better than the average. If rewards are always positive (R>0), PG always try to increase a trajectory probability even if it receives much smaller rewards than others. Consider two different situations:

- Situation 1: Trajectory A receives+10 rewards and Trajectory B receives -10 rewards.
- Situation 2: Trajectory A receives +10 rewards and Trajectory B receives +1 rewards.

In the first situation, PG will increase the probability of Trajectory A while decreasing B. In the second situation, it will increase both. As a human, we

will likely decrease the likelihood of trajectory B in both situations.

By introducing a baseline, like V, we can recalibrate the rewards relative to the average action.

Vanilla Policy Gradient Algorithm

Here is the generic algorithm for the Policy Gradient Algorithm using a baseline b.

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

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b
for iteration=1, 2, ... **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$
end for

Source

Causality

Future actions should not change past decisions. Present actions only impact the future. Therefore, we can change our objective function to reflect this also.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

Reward discount

Reward discount reduces variance which reduces the impact of distant actions. Here, a different formula is used to compute the total rewards.

$$Q^{\pi,\gamma}(s,u) \longleftarrow r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = s, a_0 = a$$

And the corresponding objective function becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

Part 2

This ends part 1 of the policy gradient methods. In the second part, we continue on the Temporal Difference, Hyperparameter tuning, and importance sampling. Temporal Difference will further reduce the variance and the importance sampling will lay down the theoretical foundation for more advanced policy gradient methods like TRPO and PPO.

RL — Policy Gradients Explained (Advanced topic)

In the first part of the Policy Gradients article, we cover the basic. In the second part, we continue on the Temporal...

medium.com

Credit and references

UCL RL course

UC Berkeley RL course

UC Berkeley RL Bootcamp

A3C paper

GAE paper

Machine Learning | Reinforcement Learning | Artificial Intelligence | Data Science | Deep Learning