A Review of REINFORCE Algorithms

Ryan Peck, Louis Renaux

May 16, 2019

Abstract

We were first introduced to REINFORCE algorithms in the context of derivative free optimization (DFO). Random Search is a DFO algorithm that falls into the classification of a REINFORCE algorithm and was the first REINFORCE algorithm presented to us. DFO, including Random Search and thus REINFORCE, was shown in lecture notes to have particularly poor convergence properties relative to methods that rely on gradient updates. However, REINFORCE was not intended to be used in a setting where true gradient updates are available. Although rigorous convergence proofs such as those presented in lecture notes are not available for many new algorithms based on REINFORCE, it is our hope that the following paper helps to lay out some intuition for convergence *properties* in both old and new algorithms based on REINFORCE. We will start with a history of the acronym and algorithm, present issues with the entire set of algorithms, discuss some older examples of REINFORCE, and finally discuss some newer improvements upon REINFORCE. We will conclude with empirical results, some of our own making and some sourced from online, to demonstrate the qualities we talk about.

1 History of REINFORCE

Ronald J. Williams wrote what is frequently cited as the first paper using the term "REINFORCE" [1]. In this paper, he is quick to point out that this class of algorithm is not new, it has simply not been recognized. He then goes on to point out manifestations of it that were already being used and discusses new uses. First, he explains the difficult REINFORCE acronym: **RE**ward **I**ncrement = **N**onnegative **F**actor times **O**ffset **R**einforcement times **C**haracteristic **E**ligibility. The acronym becomes clearer, although no less unwieldy, when he describes the general algorithm itself.

In the context of this review, a feedforward learning network operates by receiving an input from the environment (x^i) and then propogating that input through the system. At the end of the system, the output is fed to the environment, and a reinforcement signal is returned. We will denote this signal r. This r is used to update the parameters of the network and hopefully improve its performance. Williams described a REINFORCE algorithm as any algorithm that has an update function of the form:

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij})e_{ij} \tag{1}$$

Here, he let w_{ij} represent the parameters of the network, α_{ij} represent the learning rate factor, b_{ij} represent the reinforcement baseline and $e_{ij} = \partial \ln g_i / \ln w_{ij}$ represent the characteristic eligibility of w_{ij}. In this last equation, $g_i(\xi, w^i, x^i) = \Pr\{y_i = \xi | W, x^i\}$. Note that w^i is a vector of parameters, and x^i likewise a vector of inputs. According to Williams, α_{ij} can usually be taken as a constant. The interesting property of REINFORCE algorithms is summarized in the following equation:

$$E\{\Delta W|W\}^T \nabla_W E\{r|W\} \geq 0 \tag{2}$$

This means that the updated parameters move in a direction of increasing reward from the environment. In the next few sections, we will discuss the ramifications of this property and problems with REINFORCE algorithms. We will demonstrate these by discussing earlier uses followed by a discussion of more modern techniques.

2 Flaws with REINFORCE

One issue with REINFORCE algorithms is that they have a very high variance. This means that you have to draw a lot of samples from the environment before you get a good approximation of the actual gradient. To show this, we can first show how the gradient update term was derived using the log-trick.

$$
\begin{aligned}
\nabla J(\theta) &= \int R(h) \nabla_\theta p(h|\theta) dx \\
&= \int R(h) \frac{\nabla_\theta p(h|\theta)}{p(h|\theta)} p(h|\theta) dx \\
&= \int R(h) \nabla_\theta log(p(h|\theta)) p(h|\theta) dx \\
&= \int p(h|\theta) \sum_{t=1}^{T} \nabla_\theta log(p(a_t^n|s_t^n, \theta)) R(h^n) dx
\end{aligned}
$$

Since the actual value of $p(h|\theta)$ cannot be known, the gradient update is approximated as shown in equation 3 where a baseline is also added. It is very important to note that this gradient update is not a true gradient update. REINFORCE algorithms fall into the class of derivative free optimization algorithms, and come with all the expected detriments to performance.

$$
\hat{\nabla}_\theta J(\theta) = \frac{1}{N} \sum_{n=1}^{N} (R(h^n) - b) \sum_{t=1}^{T} \nabla_\theta log(p(a_t^n|s_t^n, \theta)) \tag{3}
$$

Calculating the true gradient would require knowing the probability of every action in every state, which is impossible in almost all cases. Therefore, the gradient is approximated with so-called "roll-out" samples (h) containing both state and action information. These samples are where a set of actions are simulated for T timesteps. To further explain equation 3 and put it in the context of REINFORCE, it's useful to see the update step in equation 4 where ϵ is a small constant.

$$
\theta_{k+1} = \theta_k + \epsilon \hat{\nabla} J(\theta) \tag{4}
$$

Now, the algorithm shows a reward-based increment of a non-negative factor (ϵ) times a term containing offset reinforcement (b) times characteristic eligibility ($\nabla_\theta log(p(a_t^n|s_t^n, \theta))$. Figure 1 shows this high variance in action. While training on a simple game with a max reward of 200, the algorithms natural policy gradient (which we will discuss later) and policy gradient (an umbrella term including REINFORCE) both waver tremendously. The dark lines represent the mean of 10 trials, and the shaded regions represents one standard deviation from the mean of those trials. A consequence of this high variance, even more pronounced in the original REINFORCE algorithm, is that the algorithm converges very slowly if at all [2].

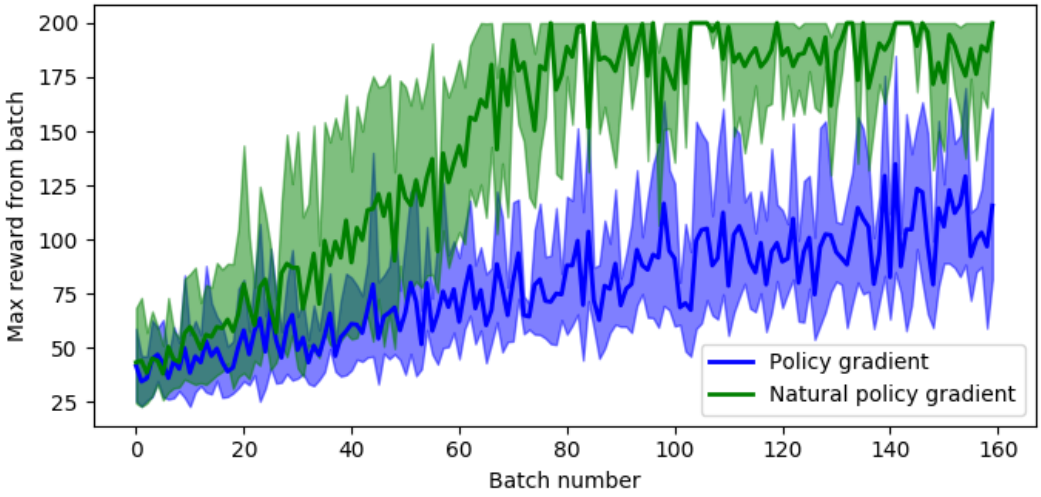


Figure 1: Two reinforcement learning algorithms trained on a simple game [3]

Now, let's find an upper bound for the variance of the REINFORCE algorithm to show that adding a baseline reduces it[2]. This should help provide context for future discussions on variance and provide a starting point for discussing adding a baseline, which we will discuss in detail in section 4.1.

$$
\begin{aligned}
\mathrm{Var}[\nabla_\mu J(\theta)] &= \frac{1}{N}\mathrm{Var}\left[\sum_{t=1}^{T}\nabla_\mu logp(a_t|s_t,\theta)\gamma^{t-1}r(s_t,a_t,s_{t+1})\right] \\
&= \frac{1}{N}\mathrm{Var}[R(h)f(h)] \\
&\leq \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[(Rf_n)^2] \\
&= \frac{1}{N}\mathbb{E}[R^2 f^T f] \\
&= \frac{1}{N}\int_h p(h)(\sum_{t=1}^{T}\gamma^{t-1}r(s_t,a_t,s_{t+1})^2(\sum_{t=1}^{T}\frac{(a_t-\mu^T s_t)}{\sigma^2}s_t)^T(\sum_{t=1}^{T}\frac{(a_t-\mu^T s_t)}{\sigma^2}s_t)dh \\
&\leq \frac{\beta^2(1-\gamma^T)^2}{N\sigma^2(1-\gamma)^2}\mathbb{E}\left[(\sum_{t,t'=1}^{T}\frac{(a_t-\mu^T s_t)(a_{t'}-\mu^T s_{t'})}{\sigma^2}s_t^T s_{t'})\right] \\
&= \frac{\beta^2(1-\gamma^T)^2}{N\sigma^2(1-\gamma)^2}\mathbb{E}\left[(\sum_{t,t'=1}^{T}\xi_t\xi_{t'}s_t^T s_{t'})\right] \\
&= \frac{\beta^2(1-\gamma^T)^2}{N\sigma^2(1-\gamma)^2}\left[\sum_{t=1}^{T}\mathbb{E}[\xi_t^2 s_t^T s_{t'})] + \sum_{t,t'=1,t\neq t'}^{T}\mathbb{E}[\xi_t s_t]\mathbb{E}[\xi_{t'}s_{t'}])\right] \\
&= \frac{\beta^2(1-\gamma^T)^2}{N\sigma^2(1-\gamma)^2}\sum_{t=1}^{T}\|s_t\|^2\mathbb{E}[\xi_t^2] \\
&= \frac{\beta^2(1-\gamma^T)^2}{N\sigma^2(1-\gamma)^2}\sum_{t=1}^{T}\|s_t\|^2 \\
&\leq \frac{D_T\beta^2(1-\gamma^T)^2}{N\sigma^2(1-\gamma)^2}
\end{aligned}
$$

Thus, we have $\mathrm{Var}[\nabla_\mu J(\theta)] \leq \frac{D_T\beta^2(1-\gamma^T)^2}{N\sigma^2(1-\gamma)^2}$ with a probability $(1-\delta)^{1/2N}$.
The same way we prove that:

$$\mathrm{Var}[\nabla_\sigma J(\theta)] \leq \frac{2T\beta^2(1-\gamma^T)^2}{N\sigma^2(1-\gamma)^2}$$

Additionally, we have that, for any baseline b and the optimal baseline b^*[4]:

$$\mathrm{Var}[\nabla_\theta \hat{J}^b(\theta)] - \mathrm{Var}[\nabla_\theta \hat{J}^{b^*}(\theta)] = \frac{(b-b^*)^2}{N}\mathbb{E}\left[\left\|\sum_{t=1}^{T}\nabla_\theta log(p(a_t|s_t,\theta))\right\|^2\right]$$

Thus,

$$\mathrm{Var}[\nabla_\theta \hat{J}(\theta)] - \mathrm{Var}[\nabla_\theta \hat{J}^{b^*}(\theta)] = \frac{b^*2}{N}\mathbb{E}\left[\left\|\sum_{t=1}^{T}\nabla_\theta logp((a_t|s_t,\theta))\right\|^2\right]$$

We can thus see that any baseline reduces the variance. Continuing to analyze the optimal baseline,

$$\begin{aligned}
\text{Var}[\nabla_\theta \hat{J}^{b^*}(\theta)] &= \text{Var}[\nabla_\theta \hat{J}(\theta)] - \frac{b^{*2}}{N}\mathbb{E}\left[\left\|\sum_{t=1}^{T}\nabla_\theta log p((a_t|s_t,\theta))\right\|^2\right] \\
&\leq \frac{D_T\beta^2(1-\gamma^T)^2}{N\sigma^2(1-\gamma)^2} - \frac{b^{*2}}{N}\mathbb{E}\left[\left\|\sum_{t=1}^{T}\nabla_\theta log(p(a_t|s_t,\theta))\right\|^2\right] \\
&\leq \frac{D_T\beta^2(1-\gamma^T)^2}{N\sigma^2(1-\gamma)^2} \\
&\leq \text{Var}[\nabla_\theta \hat{J}(\theta)]
\end{aligned}$$

Then, we have the reduction of variance:

$$\text{Var}[\nabla_\theta \hat{J}^{b^*}(\theta)] \leq \text{Var}[\nabla_\theta \hat{J}(\theta)]$$

3 Original Implementations

One algorithm that Williams cites as an example of REINFORCE is the associative reward inaction (A_{R-I}) algorithm. This is a special case of the associative reward penalty (A_{R-P}) algorithm first discussed by Andrew Barto and P. Anandan in 1985 [5]. Williams uses a Bernoulli semilinear unit for g, which makes the characteristic equation $(y_i - p_i)x_j$. In the notation of Williams, this A_{R-P} has an update rule:

$$\Delta w_{ij} = \alpha[r(y_i - p_i) + \lambda(1-r)(1-y_i-p_i)]x_j$$

Here, y_i is the output of the system and p_i is the probability that $y_i = 1$. This is a two-action system since y_i can only equal 1 or 0. The λ in this example can be chosen on the range of $0 < \lambda \leq 1$. Since λ does not equal 0, this is not a REINFORCE algorithm. A_{R-I} arises from the special case when $\lambda = 0$ because that makes the update rule:

$$\Delta w_{ij} = \alpha r(y_i - p_i)x_j$$

In the 1985 paper on the A_{R-P} algorithm, the authors prove strong convergence with several constraints, notably including that λ must be greater than 0. Additionally, the authors ran simulations showing that if $\lambda = 0$, there were some cases of divergence, therefore stating that A_{R-I} could not be proven to converge. Interestingly, the authors proved that A_{R-I} can be reduced to the linear reward inaction stochastic learning automaton (L_{R-I}) under very specific conditions. The L_{R-I} algorithm was cited in the 1992 paper by Williams as a REINFORCE algorithm as well. In a 1986 paper on L_{R-I}, the authors prove that this algorithm does converge, albeit very slowly, for the particular case of a system implemented on an ergodic (every state is aperiodic and expected to be reached in a finite number of steps) Markov chain with perfect state observation, meaning there can be no noise in the signal from the environment [6].

A 1993 paper attempts to extend A_{R-P} to problems that require continuous (as opposed to binary) outputs and have noisy signals [7]. This paper proposes an algorithm called SRV, which is not a REINFORCE algorithm but is similar to A_{R-P}. After being modified slightly and being restricted by several conditions, it was shown to converge in the presence of noise of a bounded variance. In conclusion, REINFORCE algorithms around the time of Williams converged only in very special cases, and even related algorithms converged only conditionally.

4 Modern Modifications

4.1 Changing Baseline

To reduce the variance of the REINFORCE method, we can add a baseline as discussed in section 2. We will now discuss whether adding a baseline has an effect on bias. The first possible baseline is to use a constant one, as we can simply show that it does not bias the policy:

$$\begin{aligned}
\mathbb{E}[\nabla_\theta \log p_\theta(\tau) b] &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) b d\tau \\
&= \int p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} b d\tau \\
&= \int \nabla_\theta p_\theta(\tau) b d\tau \\
&= b \nabla_\theta \int p_\theta(\tau) d\tau \\
&= b \nabla_\theta 1 \\
&= 0
\end{aligned}$$

Thus, we can add any baseline that does not depend on τ in our policy without adding any bias. However, we can find a better baseline. Indeed, we will show that a state-dependent baseline also leaves the policy unbiased. Thus, let's prove that we have the following equality:

$$\mathbb{E}[\sum_{t=1}^{T} \nabla_\theta \log p_\theta(a_t|s_t) b(s_t)] = 0$$

Therefore, because of the linearity of the expectation we have:

$$\sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log p_\theta(a_t|s_t) b(s_t)] = 0$$

Thus, $\forall t$

$$\begin{aligned}
\mathbb{E}[\nabla_\theta \log p_\theta(a_t|s_t) b(s_t)] &= \int \nabla_\theta \log p_\theta(a_t|s_t) b(s_t) p_\theta(\tau) d\tau \\
&= \int \int \int \nabla_\theta \log p_\theta(a_t|s_t) b(s_t) p_\theta(s_t, a_t) p_\theta(\tau/s_t, a_t|s_t, a_t) d(\tau/s_t, a_t) ds_t da_t \\
&= \int \int \nabla_\theta \log p_\theta(a_t|s_t) b(s_t) p_\theta(s_t, a_t) ds_t da_t \int p_\theta(\tau/s_t, a_t|s_t, a_t) d(\tau/s_t, a_t) \\
&= \int \int \nabla_\theta p_\theta(a_t|s_t) b(s_t) \frac{p_\theta(s_t, a_t)}{p_\theta(a_t|s_t)} ds_t da_t \\
&= \int b(s_t) p_\theta(s_t) ds_t \int \nabla_\theta p_\theta(a_t|s_t) da_t \\
&= \int b(s_t) p_\theta(s_t) ds_t \nabla_\theta \int p_\theta(a_t|s_t) da_t \\
&= \int b(s_t) p_\theta(s_t) ds_t \nabla_\theta 1 \\
&= 0
\end{aligned}$$

Therefore, we can use a state-dependant baseline to reduce our variance in our policy and it will not harm our approach towards an optimal policy [8].

4.2 Changing Gradient Update

While intelligently selecting a reinforcement baseline is one method of reducing the variance in a REINFORCE algorithm, recent research has focused on changing the gradient update term entirely. Three main algorithms are used to extend the original REINFORCE algorithm and improve both variance and computational complexity. For clarity, these algorithms are technically not REINFORCE algorithms, but they were indirectly built upon REINFORCE.

4.2.1 Natural Policy Gradient

In the EE227C lecture notes, the natural gradient is presented as the Hessian approximation of Bregman divergence [9]. For background, typical gradient descent minimizes the following function with respect to x:

$$f(x) + \langle \nabla f(x), x - x_0 \rangle + \frac{1}{\alpha}||x - x_0||^2$$

The distance term is $\frac{1}{\alpha}||x - x_0||^2$, and it can be replaced by another distance term. In the context of natural policy gradient, that distance term is usually the Fisher Information Matrix [10]. It is equivalent to the Hessian approximation of KL-divergence, a type of Bregman divergence [11]. The following shows that natural policy gradient has same update rule as equation 4, along with a new gradient term dependent on $F(\theta)$, the Fisher Information Matrix.

$$\begin{aligned}
\theta_{k+1} &= \theta_k + \epsilon \hat{\nabla} J(\theta) \\
\hat{\nabla} J(\theta) &= F(\theta)^{-1} \nabla J(\theta) \\
F(\theta) &= E_\theta[\nabla_\theta log(p(a|s,\theta) \nabla_\theta log(p(a|s,\theta)^T]
\end{aligned}$$

Here, $\nabla J(\theta)$ is the true gradient of average reward. That is not always available, so it can be replaced with equation 3. Using this Fisher Information Matrix solves a problem of typical policy gradient algorithms (including REINFORCE) of being noncovariant. This means that the gradient term is basis-dependent. Natural policy gradient is covariant, meaning its gradient term is independent of the basis. Additionally, natural policy gradient moves towards a greedy optimal solution. According to Sham Kakade in [10], non-covariant gradient updates move towards *better* policies, while covariant gradient updates move towards the *best* policy. We already knew that REINFORCE moved towards a better policy from equation 2, but now we see that natural policy gradient can guarantee movement towards a possibly better solution.

Although natural policy gradient is seen as a significant improvement, it still faces some major issues. For one, calculating the Fisher Information Matrix is very computationally expensive. Additionally, just because it moves *towards* the best policy does not necessarily mean that it always converges to it. Figure 2 shows the algorithm in pseudocode. Note that the "trajectories" are the roll-out samples (h) referenced earlier, the "advantages" are the cumulative rewards ($R(h)$), and the policy gradient is $\hat{\nabla} J(\theta)$.

Algorithm 1 Natural Policy Gradient

Input: initial policy parameters θ_0
for $k = 0, 1, 2, \ldots$ **do**
Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Form sample estimates for

- policy gradient $\hat{g}_k$ (using advantage estimates)
- and KL-divergence Hessian / Fisher Information Matrix $\hat{H}_k$

Compute Natural Policy Gradient update:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$$

end for

Figure 2: Natural Policy Gradient pseudocode [11]

4.2.2 Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) attempts to reduce the computational complexity of natural policy gradient by using the conjugate gradient method to approximate the inverse of the Fisher Information Matrix. This

approximation can lead to the update term violating KL-divergence or having negative reward. For that reason, after approximating the Fisher Information Matrix, the algorithm checks if the direction has positive reward and doesn't violate KL Divergence. Finally, it performs line search by scaling the step size by an exponentially decaying factor α [12].

Figure 3 shows this line search, with the two quality checks on line 4. Using line search was actually recommended in [10] as a method of making sure that natural gradient moved towards a greedy action that also improved the policy in the short term. Figure 4 shows the TRPO algorithm in full, with the line search step at the end.

Algorithm 2 Line Search for TRPO

Compute proposed policy step $\Delta_k = \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$
for $j = 0, 1, 2, ..., L$ **do**
  Compute proposed update $\theta = \theta_k + \alpha^j \Delta_k$
  if $\mathcal{L}_{\theta_k}(\theta) \geq 0$ and $\bar{D}_{KL}(\theta || \theta_k) \leq \delta$ **then**
    accept the update and set $\theta_{k+1} = \theta_k + \alpha^j \Delta_k$
    break
  end if
end for

Figure 3: Line search for TRPO [11]

Algorithm 3 Trust Region Policy Optimization

Input: initial policy parameters θ_0
for $k = 0, 1, 2, ...$ **do**
  Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Form sample estimates for

- policy gradient $\hat{g}_k$ (using advantage estimates)
- and KL-divergence Hessian-vector product function $f(v) = \hat{H}_k v$

  Use CG with n_{cg} iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$
  Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$
  Perform backtracking line search with exponential decay to obtain final update

$$\theta_{k+1} = \theta_k + \alpha^j \Delta_k$$

end for

Figure 4: TRPO pseudocode [11]

4.2.3 Proximal Policy Optimization

The original Proximal Policy Optimization (PPO) algorithm was introduced in 2017, around the same time as TRPO and 16 years after natural policy gradient. When we say PPO, we are referring to clipping PPO as it was presented in the paper, since this is the variation the authors proved had the best performance. This algorithm avoids the computational complexity inherent in natural policy gradient and in TRPO by avoiding KL-divergence entirely [13]. The purpose of adding KL-divergence was to create a basis-independent distance limitation on update steps. This would prevent algorithms from overstepping into dangerous territory. The authors of the PPO paper's primary goal was to find a simpler way to limit distance. They settled on a gradient term summarized in equation 5. The clipping function clips the probability ratio r_t, effectively putting a ceiling on the update term when r_t is more than ϵ away from 1.

$$J(\theta) = E_t[min(r_t(\theta)R_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)R_t)] \tag{5}$$

The ratio r_t is a function of the previous and current parameter. It takes advantage of a phenomenon called importance sampling, well covered in [11].

$$r_t(\theta_k) = \frac{p_{\theta_k}(a|s, \theta_k)}{p_{\theta_{k-1}}(a|s, \theta_{k-1})}$$

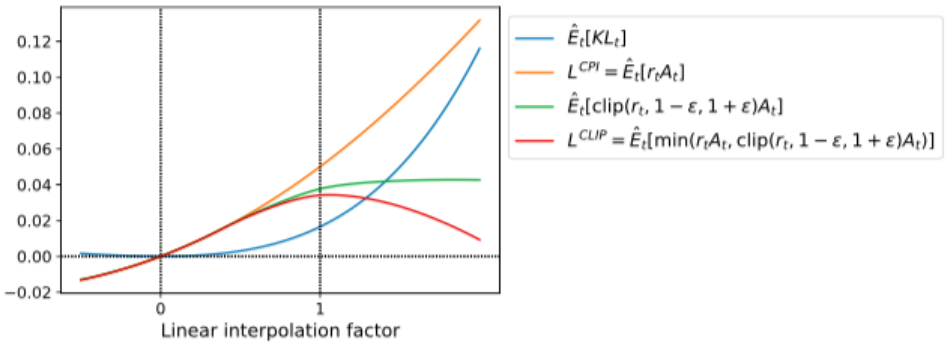


Figure 5: Clipping as a penalty for excess distance [13]

The value of this new update term is best illustrated by figure 5. This graph uses different notation, but the orange line on top represents the update distance used with TRPO while the red line that finishes lowest represents the update distance used with PPO. The x axis is a linear interpolation between the old policy (at 0) and the new policy (at 1). The y-axis is the KL-divergence. The conclusion from this graph should be that PPO results in a lower bound on the KL-divergence found by TRPO. It does this as a first-order optimization as opposed to the second-order optimization of natural policy gradient and TRPO, decreasing computational complexity. Finally, figure 6 shows the algorithm in pseudocode.

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

Figure 6: PPO pseudocode [11]

5 Empirical Results

5.1 REINFORCE With and Without Baseline

In studying the efficacy of REINFORCE, we first turned to Zhao et al for a simulation setup [2]. They created a toy environment upon which the algorithms could be tested. The toy environment could be defined by the following initial parameters, all continuous scalars:

$$\begin{aligned} s_0 &\sim \mathcal{N}(0,\, 1) \\ a_0 &\sim \mathcal{N}(0,\, 1) \\ \mu &= -1.5 \\ \sigma &= 1 \end{aligned}$$

Where s_0 and a_0 are the initial state and action respectively, and μ and σ are the respective initial mean and initial standard deviation that collectively make up our update parameter θ. The update steps are as follows. For T time steps, a roll-out sample is collected.

$$\begin{aligned} s_{t+1} &= \frac{s_t + a_t + \epsilon}{8000} \\ a_{t+1} &\sim \mathcal{N}(\mu s_t,\, \sigma^2) \end{aligned}$$

Note that the 8000 in the denominator of the state update was introduced to prevent the state from becoming too large, as occasionally happened during simulation. The final term in the numerator of that equation, ϵ is to simulate noise and is drawn from $\mathcal{N}(0,\, .5^2)$. While each roll-out sample is collected, gradient update terms are being accumulated like so:

$$\begin{aligned} \nabla_\mu log(p(a_t^n|s_t^n, \theta) &= \frac{(a_t^n - \mu s_t^n)s_t^n}{\sigma^2} \\ \nabla_\sigma log(p(a_t^n|s_t^n, \theta) &= \frac{(a_t^n - \mu s_t^n)^2 - \sigma^2}{\sigma^3} \end{aligned}$$

Reward for the entire roll-out is calculated with a decaying exponential exponent.

$$R(h) = \sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t)$$
$$r(s, a) = e^{-s^2/2 - a^2/2} + 1$$

For our simulation, γ was set to 0.9. Finally, these roll-outs were averaged over N collections per update step i, giving us the following update term from equation 3 and respective update equations for the mean and standard deviation.

$$\hat{\nabla}_\theta J(\theta) = \frac{1}{N} \sum_{n=1}^{N} (R(h^n) - b) \sum_{t=1}^{T} \nabla_\theta log(p(a_t^n | s_t^n, \theta))$$
$$\mu_{i+1} = \mu_i + \alpha \hat{\nabla}_\mu J(\theta)$$
$$\sigma_{i+1} = \sigma_i + \alpha \hat{\nabla}_\sigma J(\theta)$$

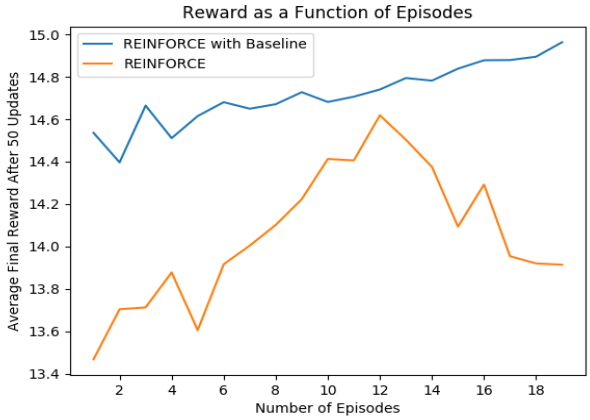
For our simulation, the learning term α was arbitrarily set to 0.01 due to the cleanliness of the result. Figure 7a shows the results of the simulation. We used the following parameters:

$$T = 20$$
$$max(N) = 20$$
$$I = 50$$
$$A = 50$$

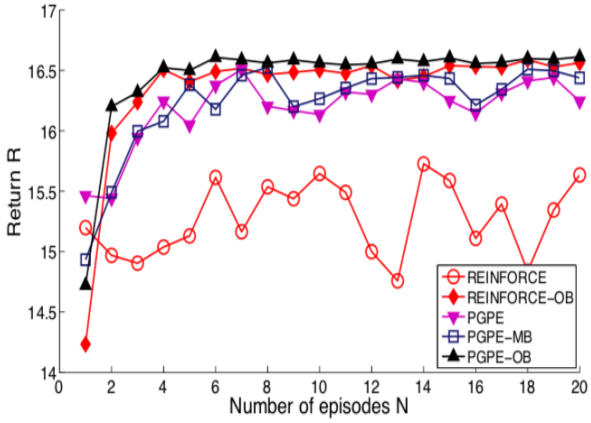
These parameters mean that we took roll-out samples of length 20, averaged them for each N in $[1, max(N)]$ collections, updating our μ and σ $I = 50$ times each. The variable A represents that we ran 50 simulations of this entire process, averaging the final reward over all simulations. The baseline was set to 0 for the plain REINFORCE algorithm, and to an adaptive baseline for the REINFORCE with Baseline algorithm:

$$b_n = \gamma R(h^n) + (1 - \gamma) b_{n-1}$$

Figure 7a demonstrates the high variance of plain REINFORCE leading to poor performance and that a baseline can greatly reduce that variance. For comparison, figure 7b is the same simulation results from the paper by Zhao et al, except with no mention of the 8000 in the denominator of the state update term, $A = 20$, and no explicit learning rate α. REINFORCE-OB is their implementation of REINFORCE with a baseline. To confirm that REINFORCE without baseline has higher variance, figures 8a and 8b show the calculated variance of the different trials (each a in A, which was only 20 for this calculation).

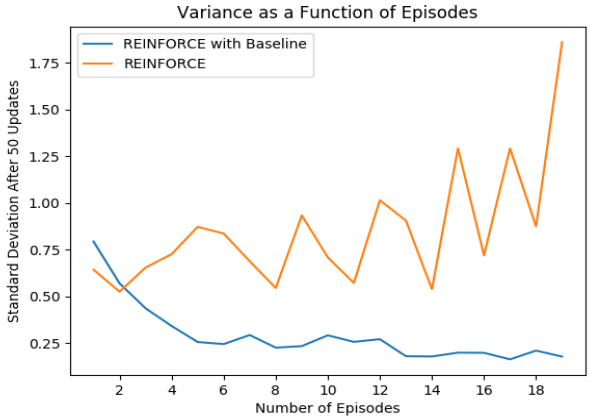


(a) Our comparison of reward from REINFORCE with and without baseline

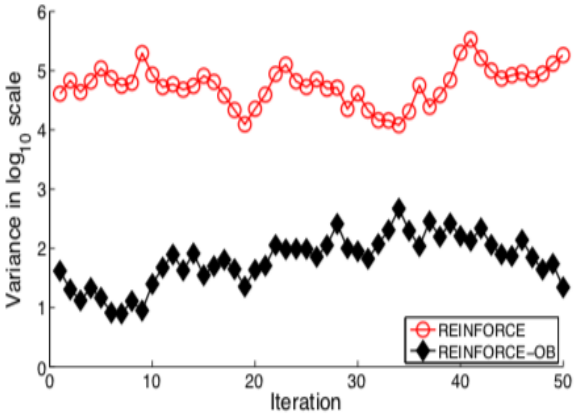


(b) A comparison of reward from REINFORCE with and without baseline along with other algorithms from [2]

Figure 7: A comparison of reward from our simulation and Zhao et al



(a) Our comparison of variance from REINFORCE with and without baseline



(b) A comparison of variance from REINFORCE with and without baseline from [2]

Figure 8: A comparison of variance from our simulation and Zhao et al

5.2 Newer Algorithms

Additionally, we wanted to compare the 4 algorithms we analyzed (policy gradient, natural policy gradient, TRPO, and PPO) on a more complex reinforcement learning environment. To do so, we used a github repository[14] that uses MuJoCo to train different humanoïds and animals how to walk. The goal of that experiment was to determine if TRPO and PPO would actually improve the policy thanks to the restriction of the trust region that would make the policy not wander in unknown areas. Figure 9 is the plot of the reward from the Ant-v2 performance. PG corresponds to Vanilla Policy Gradient and NPG to Natural Policy Gradient.

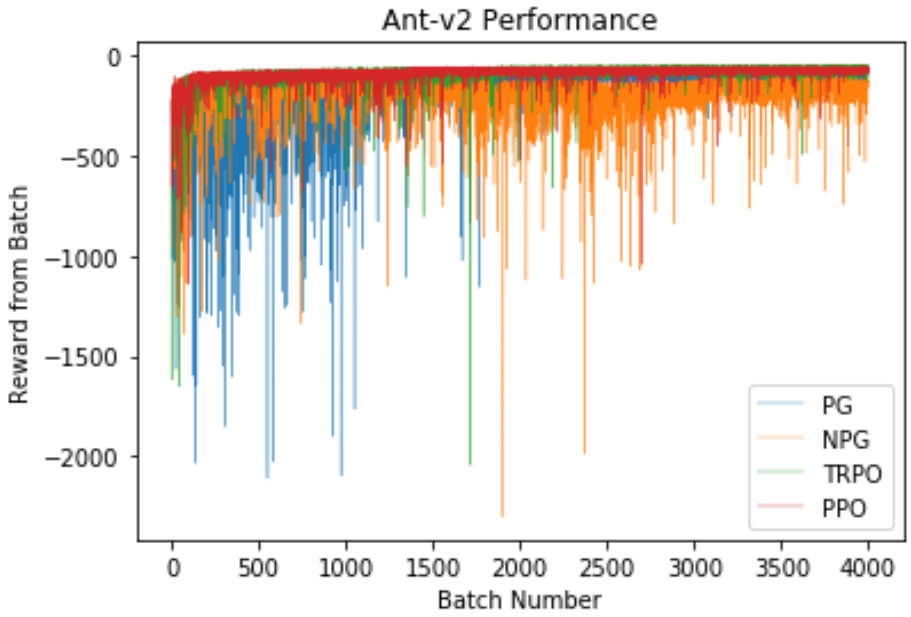


Figure 9: A comparison of Policy Gradient, Natural Policy Gradient, TRPO and PPO from [14]

Thus, we clearly see that the Vanilla Policy Gradient (in blue) suffers the most as the reward drop very frequently, which is exactly what the trust region is meant to correct. We also see that the TRPO algorithm which is using the hessian of the KL divergence and the backtracking line-search is dropping more than the PPO algorithm. PPO is more effective and has lower computational complexity.

6 Conclusion

REINFORCE algorithms are derivative free optimization algorithms that are particularly useful for tasks formulated as games. These games have close to infinite combinations of states and actions, resulting in a difficult optimization problem. Finding an optimal policy for selecting the next action is an exciting field of research. While REINFORCE was introduced to us through the context of Random Search which came with a thorough convergence proof, other REINFORCE algorithms do not. In this paper, we have presented some intuitions about the convergence of both original and modern REINFORCE algorithms in various contexts. It is our hope that this review helps to clarify the field of reinforcement learning as a whole for other readers just entering the field.

References

[1] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8 (1992), pp. 229–256.

[2] Tingting Zhao et al. "Analysis and Improvement of Policy Gradient Estimation". In: *Advances in Neural Information Processing Systems* (2011).

[3] Travis DeWolf. *Natural policy gradient in TensorFlow*. 2018. URL: https://studywolf.wordpress.com/2018/06/20/natural-policy-gradient-in-tensorflow/.

[4] E. Greensmith, P. L. Bartlett, and J. Baxter. "Variance reduction techniques for gradient estimates in reinforcement learning". In: *Journal of Machine Learning Research* 37.5:1471–1530 (2004).

[5] Andrew G. Barto and P. Anandan. "Pattern Recognizing Stochastic Learning Automata". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-15.3 (1985), pp. 360–375.

[6] Richard M. Wheeler Jr. and Kumpati S. Narendra. "Decentralized Learning in Finite Markov Chains". In: *IEEE Transactions on Automatic Control* AC-31.6 (1986), pp. 519–526.

[7] Vijaykumar Gullapalli. "Associative Reinforcement Learning of Real-Valued Functions". In: *COINS Technical Report* (1993), pp. 90–129.

[8] Sergey Levine. *CS294-112 Deep Reinforcement Learning HW2: Policy Gradients*. 2018. URL: http://rail.eecs.berkeley.edu/deeprlcourse/static/homeworks/hw2.pdf.

[9] Benjamin Recht. *Natural Gradient and Mirror Descent*. 2019.

[10] Sham Kakade. "A Natural Policy Gradient". In: *NIPS'01 Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic* (2001), pp. 1531–1538.

[11] Joshua Achiam. *Advanced Policy Gradient Methods*. 2017. URL: rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_13_advanced_pg.pdf.

[12] John Schulman et al. "Trust Region Policy Optimization". In: *International Conference on Machine Learning* 37 (2017).

[13] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *arXiv:1707.06347* (2017).

[14] Woongwon Lee et al. *Comparison of Gradient Descent, Natural Gradient Descent, PPO and TRPO*. 2018. URL: https://github.com/reinforcement-learning-kr/pg_travel.