Reinforcement Learning
TP3

Antonin Berthon

November 2018

On-Policy RL with Parametric Policy

REINFORCE algorithm on Gaussian Policy model

We implement the REINFORCE algorithm in the setting of the Gaussian policy model, with $\sigma_w = 0.4$, $T = 100$, $\gamma = 0.9$, such that $\theta^* \approx 0.59$.

We first consider a constant update scheme. Using 100 iterations of the algorithm, Figure 1 shows the evolution of the returns for learning rates α in $\{10^{-3}, 10^{-4}, 10^{-5}\}$. We see that for a learning rate that is too high (e.g. 0.001), the algorithm diverges. We then show the evolution of the parameters for $\alpha \in \{10^{-4}, 10^{-5}\}$ (see Figure 2). Without surprise we see that the algorithm converges more quickly with $\alpha = 10^{-4}$ than for $\alpha = 10^{-5}$, however for the latter it approaches the optimal parameter with much less variance.

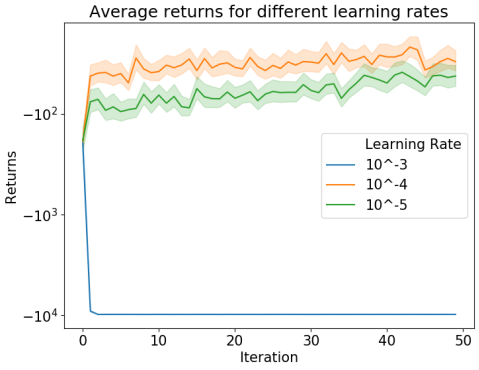

Figure 1 - Evolution of the average return for different learning rates. $\sigma_w = 0.4$, $T = 100$, $\gamma = 0.9$, $N = 50$.

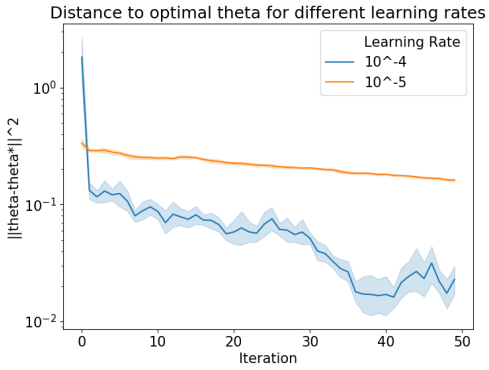

Figure 2 - Evolution instance to the optimal parameter θ^* for different learning rates. $\sigma_w = 0.4$, $T = 100$, $\gamma = 0.9$, $N = 50$.

Keeping a learning rate 10^{-4}, we now compare the constant update scheme to a annealing one in $\alpha_t = \frac{\alpha}{\sqrt{t}}$. Figure 3 and Figure 4 show the differences in average return and distance to θ^* of a the two step update schemes. We see that the annealing scheme allows for a much faster convergence.

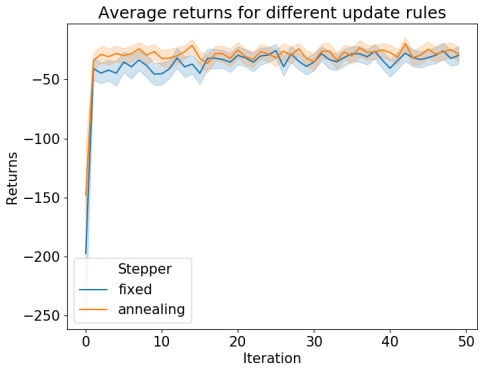

Figure 3 - Evolution of the average return for different step update methods. $\sigma_w = 0.4$, $T = 100$, $\gamma = 0.9$, $\alpha = 10^{-4}$, $N = 50$.

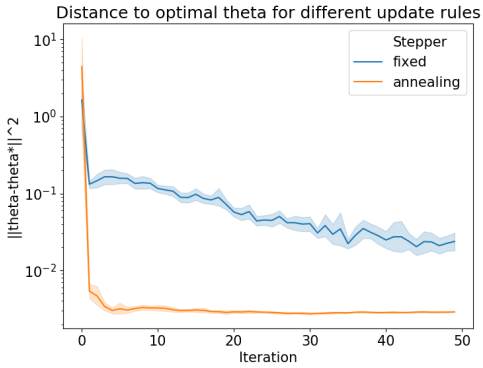

Figure 4 - Evolution instance to the optimal parameter θ^* for different step update methods. $\sigma_w = 0.4$, $T = 100$, $\gamma = 0.9$, $\alpha = 10^{-4}$, $N = 50$.

Effect of parameter N

Figure 5 and Figure 6 show the average returns and the distance to θ^* for different numbers of collected trajectories per simulation N. An increase in N allows for a more precise calculation of $\nabla_\theta J$ and therefore reduces the variance of the algorithm. However, reaching the optimal solution might take longer due to more exploration at each iteration.

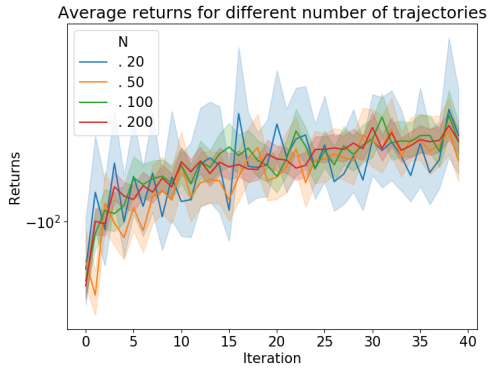

Figure 5 - Evolution of the average return for different N. $\sigma_w = 0.4$, $T = 100$, $\gamma = 0.9$, $\alpha = 10^{-5}$.

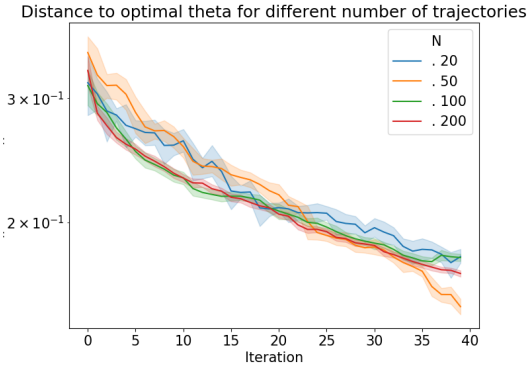

Figure 6 - Evolution of the distance to the optimal parameter θ^* for different N. $\sigma_w = 0.4$, $T = 100$, $\gamma = 0.9$, $\alpha = 10^{-5}$.

Exploration in policy gradient

Finally we implement an exploration bonus in the rewards by hashing the state and action space and counting the number of visits to each (state, action) pairs. The exploration bonus takes the form $\beta\sqrt{\frac{1}{N_t(\Phi(s,a))}}$, with β an hyper-parameter that we can tune. Below

we plot the results for average return (Figure 7) and parameter approximation (Figure 8) for $\alpha \in \{10^{-4}, 10^{-5}\}$, with $\beta = 1$ and the same parameters as in the previous section. We see that compared to the version of the algorithm with no bonus, the algorithm takes more time to converge, due to more exploration.

Figure 7 - Evolution of the average return for different learning rates with an exploration bonus. $\sigma_w = 0.4$, $T = 100$, $\gamma = 0.9$, $N = 50$, $\beta = 1$.

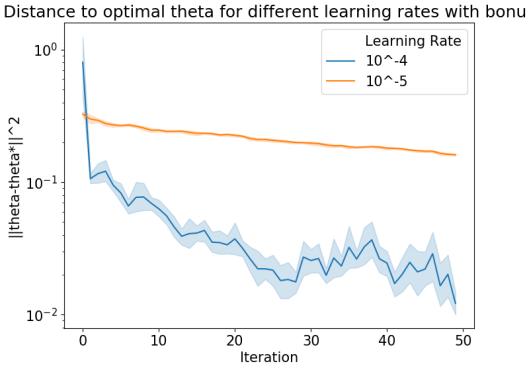

Figure 8 - Evolution of the distance to the optimal parameter θ^* for different learning rates with an exploration bonus. $\sigma_w = 0.4$, $T = 100$, $\gamma = 0.9$, $\alpha = 10^{-5}$, $\beta = 1$.