PARAMETER INSPECTION FOR DIVERSITY-DRIVEN EXPLORATION IN REINFORCEMENT LEARNING

A PREPRINT

Tobias Braun
UNI: tgb2117
MS candidate in Operations Research
Columbia University
tgb2117@columbia.edu

Gerardo Antonio Lopez Ruiz
UNI: gal2140
MS candidate in Data Science
Columbia University
gal2140@columbia.edu

Carlos Omar Pardo Gomez
UNI: cop2108
MS candidate in Data Science
Columbia University
cop2108@columbia.edu

May 10, 2019

ABSTRACT

The problem of exploration versus exploitation has been haunting researchers since self-learning algorithms were first discovered. Recent research on improving ES algorithms focuses on improving the exploration strategies.

The basic idea is to favor policies that are "different" from the previously seen. In order to apply this concept, it is possible to introduce a distance factor D that measures the dissimilarity of the new policy from the previous ones and subtract it from the standard loss function $L(\pi)$. Henceforth, we propose different distance metrics D as well as various distributions over the batch used for optimization and verify them empirically using A2C and DQN with and without exploration. Furthermore, we try to achieve some theoretical guarantees regarding our best metrics. Our research concluded that there are distances and distributions that can be used to improve the algorithms previously mentioned.

Moreover, it shows that there are distances that can make the algorithm learn slower. Hence, it is important to corroborate which distance should be used since there is no "silver bullet" distance parameter that is capable of solving every environment.

1 Introduction

During the Big Data & Machine Learning course (IEOR E8100, Sect. 007, Spring 2019, Columbia University) we reviewed several strategies to improve the exploration strategies in algorithms for Reinforcement Learning (RL). One of those strategies consisted in modifying the loss function $L(\pi)$ to induce further incentive for exploration.

One of the motivations behind this is to counter the common setting of sparse rewards. Here, the environment provides a lot of small or even 0 rewards and only very few large ones. Therefore, it is likely that standard (unguided) exploration strategies such as ϵ-greedy exploration either get stuck in local maxima consisting of small rewards or take a long time to discover good policies.

One way to address this situation is to give incentives to the algorithm to explore new policies, thus making it more likely to find those sparse rewards. In mathematical terms, this can be motivated by modifying the loss function so that

$$L_{DDE}(\pi) = L(\pi) - E_{\pi' \sim \Pi}[\alpha D(\pi, \pi')],$$

where D is the distance between policies, α is a parameter which controls the relative importance of said distance, Π is the batch of the previous policies, and π' is randomly chosen from it following a given distribution Z_Π. L is a previously defined arbitrary loss.

L_{DDE} (DDE: Diversity Driven Exploration) is the objective function we now want to minimize. The *novelty incentive factor* has a negative sign, which can be interpreted as trying to maximize the expected distance with respect to the

previous policies. This translates into a behavior which looks for novelty in the new policy when the difference in the losses has been small.

The interest of this paper is to propose several ways to choose the parameters that we previously explained: the distance measure D, the probability distribution Z_Π over the batch Π of previous policies, and the relative importance of the distance α. Then, implement this logic into popular (basic) RL environments and analyze the performance differences versus baseline algorithms, and amongst themselves. To achieve this ambition, we are going to link our ideas about the parameters to the methodology and RL algorithms presented in Hong et al. [1].

2 Distance Measures

For this paper we will express policies as probability distributions over the possible actions that can be taken, given the agent is located in a fixed state s. Also, we will assume that the number of possible actions is finite or discretized. Therefore, for each state s the policy can be expressed as a vector where all the entries are ≥ 0 and the sum of them equals to 1.

2.1 KL-divergence

One of the most widely used statistical distance metrics within the RL context is the Kullback–Leibler divergence. Be P and Q two discrete probability distributions with the same support $\mathcal{X}$, the KL-divergence can be expressed as

$$D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) log\left(\frac{P(x)}{Q(x)}\right).$$

The KL-divergence can be defined similarly for continuous distributions as

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) log\left(\frac{p(x)}{q(x)}\right) dx.$$

Some advantageous properties of this metric are that

- $D_{KL}(P \parallel Q) \geq 0$
- $D_{KL}(P \parallel Q) = 0 \iff P = Q$

However, it cannot be called a distance metric, since it neither satisfies the triangle inequality nor (in general) the symmetry axiom that states $D_{KL}(P \parallel Q) = D_{KL}(Q \parallel P)$. Despite this, it has regularly shown good performance within the RL and Deep Learning (DL) contexts, and for that reason we are going to include it in our proposal. Also, this is the distance metric that Hong et al. [1] used to run their experiments.

2.2 JSD-divergence

The Jensen-Shannon divergence tackles the asymmetry of the KL-divergence in the following way. Again, Let P and Q denote two discrete probability distributions with the same support $\mathcal{X}$, the JSD-divergence is defined as

$$D_{JSD}(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel M) + \frac{1}{2} D_{KL}(P \parallel M),$$
$$M = \frac{1}{2}(P + Q).$$

The idea behind this definition is to find an average distribution M between P and Q, and then get the mean of the KL-divergences from P and Q with respect to such distribution. The reader can notice that this makes the JSD-divergence a symmetric distance measure.

Moreover, one useful feature of the JSD divergence is that it has an upper bound, in addition to the lower bound inherited from the KL-divergence:

$$0 \leq D_{JSD}(P \parallel Q) \leq \log(2)$$

This in general improves the stability of the algorithm, because the fact that the *novelty incentive factor* is not upper-bounded can lead to unwanted behaviors. The algorithm is prone to overvaluing exploration and completely disregarding the magnitude of the regular loss L.

2.3 Total Variation Distance and L_p-norms

One statistical distance metric that is widely famous within the Probability context is the Total Variation Distance (TVD). Be P and Q two distributions with the same support $\mathcal{X}$, and $\mathcal{F}$ the set of all possible subsets of $\mathcal{X}$, such distance is defined as

$$D_{TVD}(P,Q) = \sup_{A\in\mathcal{F}} |P(A) - Q(A)| .$$

This distance can be interpreted as the maximum difference that can be found between the probabilities assigned by both distributions to a single event. It is important to note that for our specific problem, even though we are working with discrete distributions, an event A can be of the form $A = \{a_1, a_3, a_7\}$, so this distance does not measure the maximum discrepancy between the probabilities assigned to a single action, but includes also all the possible sets that can be created by choosing an arbitrary number of actions from the action space. [1]

Until now, it seems like this distance metric has a significant disadvantage, since we need to calculate the difference in probability respect to $(2^{\#actions}/2) - 2$ possible subsets[2], a number that can grows exponentially large when the number of actions increases.

However, one nice consequence of the definition of the TVD is the following equality.

$$D_{TVD}(P,Q) = \sup_{A\in\mathcal{F}} |P(A) - Q(A)| = \frac{1}{2}\sum_{x\in\mathcal{X}} |P(x) - Q(x)| = \frac{1}{2}\|P - Q\|_1 = \frac{1}{2}D_{L_1}(P,Q)$$

That means that the $L1$-norm between the difference of the probability vectors is equivalent to the TVD, except for the constant factor $1/2$. Therefore, we will propose the $L1$-norm (D_{L_1}) as another possible distance between the distributions P and Q.

Along the same line, we can try what happens when we only consider action-wise differences. That means, restricting $A = \{a_i\}$ such that $a_i \in \mathcal{X}$. Recalling that the L_∞-norm of a vector x is defined as $\|x\|_\infty = \sup_i |x_i|$, and that for vectors with finite dimensionality it is $\|x\|_\infty = \max_i |x_i|$. Then, we propose the L_∞ distance between both distributions as

$$D_{L_\infty}(P,Q) = \max_{x\in\mathcal{X}} |P(x) - Q(x)| .$$

Finally, for the purpose of comparison with the two other L_p-norms, we try the widely spread Euclidean distance. We will define the L_2 distance as

$$D_{L_2}(P,Q) = \sqrt{\sum_{x\in\mathcal{X}} (P(x) - Q(x))^2}.$$

To get further understanding of the different measurements the following inequalities link all the previously defined distances.

$$D_{TVD}(P,Q) \leq \sqrt{\frac{1}{2}D_{KL}(P \parallel Q)} \implies D_{L_1}(P,Q) \leq \sqrt{2D_{KL}(P \parallel Q)},$$
$$D_{L_\infty}(P,Q) \leq D_{L_2}(P,Q) \leq D_{L_1}(P,Q) \leq \sqrt{n}D_{L_2}(P,Q) \leq nD_{L_\infty}(P,Q),$$

where $n = |\mathcal{X}|$ (the number of possible actions).

3 Distributions over previous policies

Let's recall that we are working with a batch of previous policies Π sampled from a replay buffer that we construct during the agent's training. We further assume that we can pick any of those policies π' randomly following a given

[1] For the next lines, and because it is the case of our particular problem, we will assume P and Q are discrete distributions, but almost everything can be generalized to the continuous case.

[2] The division over two is because $|P(A) - Q(A)| = |P(\mathcal{X} - A) - Q(\mathcal{X} - A)|$, and the subtraction is because $|P(\mathcal{X}) - Q(\mathcal{X})| = |P(\emptyset) - Q(\emptyset)| = 0$, always.

distribution Z_Π with support Π ($\pi' \sim Z_\Pi$). Then, it is noticeable that the way we set this distribution will have a direct effect in the *novelty incentive factor* $E_{\pi' \sim \Pi}[\alpha D(\pi, \pi')]$ because, in practice, that number is going to be estimated as a weighted average between distances, where the distribution Z_Π determines the weights.

Since we are trying to maximize the *novelty incentive factor*, and the distances are always non-negative, we should give a bigger weight to those previous policies π' from which we want to distance the most. Thus, determining which are those policies will result in the shape of Z_Π.

3.1 Uniform Distribution

The standard approach due to its simplicity is the uniform distribution. This is the first choice for any algorithm, and was the one used in Hong et al. [1]. In absence of more information about the desired behavior, it makes no assumptions about which policies should have the largest influence, and only assigns the same weight to all the previous policies. That means that the *novelty incentive factor* becomes just the regular average of all the distances, and the algorithm will avoid in the same magnitude any of the previous policies.

3.2 Exponential Discretized Truncated Distribution

This proposal takes inspiration from the Exponential Decay concept, which postulates that something decreases at a rate proportional to its current value. That means that when the value is large, it decreases quickly, and when the value is small, it decreases slowly. In fact, be Z the amount object of our study, its behavior can be modeled with the differential equation

$$\frac{dZ}{dt} = -\lambda Z,$$

where λ is a constant value. Then, after solving the equation, the value of Z at any point can be expressed as a continuous function

$$Z(t) = Z_0 e^{-\lambda t},$$

where Z_0 is the value of Z at the beginning.

This behavior can be used to model the influence of the previous policies in the actual one, where the latest is the most influential, and once we move further in time, such influence decays. This situation makes the behavior smooth because the oldest policies in the batch have just a small weight before being dropped. Choosing this kind of conduct will provoke the algorithm to constantly explore novel policies that are far from the latest, but once the time passes, the algorithm quickly *forgets* that it already visited those polices until they are eventually dropped from the replay buffer.

It is worth noticing that there is one popular distribution that follows almost exactly the same pattern and is called the Exponential Distribution. We say the random variable Y follows a Exponential Distribution with parameter λ ($Y \sim Exp(\lambda)$) if its density function is defined as

$$f(y) = \lambda e^{-\lambda y} \mathbb{1}_{y \geq 0}.$$

This density is almost equal to the Exponential Decay equation, being the only difference that Z_0 has already a fixed value equal to λ. The reason for the distribution to set this value is just that the integral over any density function must be equal to 1, and

$$\int_0^\infty Z_0 e^{-\lambda t} dt = \frac{Z_0}{\lambda}.$$

Therefore, choosing $Z_0 = \lambda = 1$ makes the Exponential Decay a well-defined distribution.

However, there are two assumptions here that don't coincide with our particular problem. Firstyl, $t \in \mathbb{R}^+$ is continuous, while for us $t \in \mathbb{N}$ is discrete, because it is the number of iterations that have happened since that policy was chosen. Secondly, t in the Exponential Distribution can be arbitrarily large, while in our problem it is bounded by the size of the the replay buffer of previous policies. (Assuming that policies get deleted according to the First in First Out concept from the replay buffer once the maximum size has been reached. Otherwise it is bounded by the number of episodes.) To counter the first issue, we will discretize the distribution, and to counter the second, we are going to truncate it.

Thus, be π_t the policy chosen t iterations before, and T the total number of policies in the batch Π, our proposal is that the probability corresponding to π_t is equal to

$$P(\pi_t) = \frac{e^{-\lambda t}}{\sum_{k=1}^{T} e^{-\lambda k}} \mathbb{1}_{t \in \{1,\dots,T\}},$$

where the denominator is just the factor that makes all the probabilities sum to 1. We are going to call this probability distribution the Exponential Discretized Truncated (EDT) one.

3.3 Reverse-Exponential Discretized Truncated Distribution

For our next proposed distribution, we will just slightly modify the math of the EDT, but causing a completely different behavior. Now, we want the oldest policies in Π to be the most influential for the next one. The intuition behind this is that we let the algorithm explore given area for a while (without forcing radical exploration into new areas), but once it has been there for a certain time, we want it to start moving to a different region.

Concerning the distribution, the trick is going to be to make a mirror of the EDT, where the oldest policy is the one with the largest weight, and then it decays as we move to the recent ones. We will call this distribution the Reverse-Exponential Discretized Truncated Distribution (R-EDT), and it will be defined as

$$P(\pi_t) = \frac{e^{-\lambda(T-t+1)}}{\sum_{k=1}^{T} e^{-\lambda(T-k+1)}} \mathbb{1}_{t \in \{1,\dots,T\}}.$$

Let's recall that one advantage of the EDT was that the oldest policies had the smaller weights, so dropping them from the batch didn't represent a significant change. It seems like now we are provoking a violent effect by dropping the one that had just the largest weight. However, it is important to recognize that we expect policies close in time not to be radically distant from each other. Thus we don't expect the oldest policy which holds the largest weight to be completely different from the one second oldest which is going to obtain the largest weight in the next iteration.

3.4 Reward-based Distribution

The *exploitation-exploration* dilemma is one commonly found in the RL context and that is particularly relevant for the task we are addressing. Once we get what seems like a good solution, should we stick to that good policy and solely use our (now limited) resources to try to find something close that only slightly improves it (*exploitation*), or should we try a radically different approach, opening the possibility to learn potentially even better strategies with the risk of getting lower rewards for the current episode (*exploration*)?

Let's remember that usually we define a RL problem with one loss function L that should be minimized. The baseline for many algorithms which tackle this problems focuses on exploitation with an ϵ- greedy exploration strategy. There is a tendency of get stuck on a local optimum once a certain time has passed as a simple time dependent ϵ will become small after a limited amount of episodes. Some algorithms such as the UCB algorithm include a term that depends on the number of times a certain strategy has already been tried into the evaluation of an action to do exploration. In this paper we are directly modifying the loss function to conduct exploration. Recalling that the loss we are trying to optimize in this paper is

$$L_{DDE}(\pi) = L(\pi) - E_{\pi' \sim \Pi}[\alpha D(\pi, \pi')],$$

we can see that optimizing just the first term would lead us to the already mentioned exploitation strategy, but by optimizing the *novelty incentive factor*, we are motivating exploration. Therefore, both strategies are in the game, and sometimes the algorithm will prefer to follow one or the other.

One way to radicalize this trade-off is by giving some penalization in the second term to those which get the best rewards, that means, best minimize the left part of the L_{DDE}. On the one hand, we will repeat a previous good policy just in case it would significantly outperform all the others. On the other hand, even when the *novelty incentive factor* doesn't give reasons to distance our new policy from the previous bad ones, the original L usually makes the algorithm avoid them.

In mathematical terms, the general idea is the following. For the off-policy DQN algorithm the approach is straight forward. We simply assigned the reward to each (state, action) pair that was observed when the action was played. For

the on-policy algorithm we focus on the rewards that were obtained by following a certain policy by taking the sum of rewards of all trajectories that were sampled from the same policy and assigning that value $r_{\pi'}$ to the policy π' . In both cases we then use a trick similar to the one we used for EDT to transform the rewards into a probability distribution. It is called the *softmax function*, to get the Reward-based (R-b) distribution as follows

$$P(\pi_t) = \frac{r_{\pi_t}}{\sum_{k=1}^{T} r_{\pi_k}} \mathbb{1}_{t\in\{1,\dots,T\}}.$$

4 Relative Importance of the Distance

This section is going to be dedicated to the way the parameter α, the one that controls the relative importance of the distance, evolves during the episodes of our algorithm. Within the RL and DL contexts, it is common that the magnitude of this kind of parameters (for instance, the learning rates or step-sizes) decreases as the number of iterations increases.

Oftentimes, the decrease is chosen in a deterministic way, the classical examples being the linear and the exponential decay. However, Hong et al. [1] demonstrated for several experiments that a linear annealing not necessarily translates into good results. Therefore, they propose two adaptive scaling methods for the value of α, which are the ones we are going to consider on this paper as well. The first are based on the distance of the current policy relative to the batch Π of previous policies, and the others are related to the actual performance of the algorithm.

4.1 Distance-based methods

In distance-based methods the idea is to adapt the value of α depending on the expected distances of the current episode from the sampled past policies. If that magnitude is small, that indicates that we have been exploring policies that are similar to the past policies. Therefore, we want the α to grow so that it gives more incentive to explore new strategies. On the other hand, if this value is large it is justified to decrease α as it implies that we are currently exploring uncharted territory.

Along this line, Hong et al. [1] proposes the following update for α, with relation to the current policy π:

$$\alpha = \begin{cases} 1.01\alpha, \text{ if } \mathbb{E}_{Z_\Pi}[D(\pi,\pi')] \leq \delta \\ 0.99\alpha, \text{ otherwise,} \end{cases}$$

where the value of δ is a previously defined hyper-parameter.

4.2 Performance-based methods

The distance-based methods tend to work fine if the replay buffer is relatively large. In on-policy algorithms the batch Π of previous policies is oftentimes smaller than in off-policy algorithms which is translated into a large variance on the expected value of the distance, causing the α to be growing and decreasing very unstably (wave with high amplitude). To counter that, Hong et al. [1] proposed to calculate the α as a function of the performance of the previous policies. To tackle this, they postulate two ideas:

$$\alpha_P(\pi) := 1 - 2\frac{P(\pi) - P_{min}}{P_{max} - P_{min}} \quad \text{(Proactive)},$$
$$\alpha_R(\pi) := 1 - \frac{P(\pi) - P_{min}}{P_{max} - P_{min}} \quad \text{(Reactive)},$$

where $P(\pi)$ is the average performance of the policy π during the last n episodes, where n is a hyper-parameter usually set equal to 5. P_{min} and P_{max} are the worst and best performances gotten by the previous policies included in the batch Π.

The reader might notice that $\alpha_P(\pi) \in [-1, 1]$ and $\alpha_R(\pi) \in [0, 1]$. The intuition behind the Proactive case is that if $P(\pi)$ is above average ($\alpha_P > 0$), the algorithm tries to find more similar policies. If it performs under-average ($\alpha_P > 0$), the algorithm will look for different policies. On the other hand, since $\alpha_R \geq 0$, the algorithm will always try to find policies far from the previous ones, even more so if they current policy is bad.

5 Advantage Actor Critic and Deep Q-Networks.

As previously mentioned, we wish to implement the previous logic into popular RL environments and analyze the performance differences. For this, we will use two popular algorithms: Advantage Actor Critic (A2C) and Deep Q-Networks (DQN); both of them with and without exploration strategies. Our baseline will be said algorithms with no exploration strategies.

Regarding the exploration strategies, we will iterate among all the different ideas previously mentioned, and choose the best ones. Since the number of parameters is relatively small, we will simply create a grid and run all the permutations among them. This will allow us to have a good understanding of what parameters are better for these environments.

Furthermore, we will also compare with OpenAi's leader board OpenAi [2], which tracks the performance of user algorithms for various tasks in gym.

5.1 Advantage Actor Critic

The advantage actor critic algorithm consists in two parts: the actor critic method and the advantage value.

The actor critic method is an improval over the policy gradient algorithm developped by Sutton et al. [3], where it solves the policy gradient's main issue: waiting for the episode to finish in order to compute the reward. Therefore, even if there is a bad action, all actions will be denominated as good. In order to avoid this, an update is made at each step.

Thus, our original policy gradient:

$$\Delta_\theta J(\theta) = \mathbf{E}_\tau[\sum_{t=0}^{T-1} \Delta_\theta log_{\pi_\theta}(a_t|s_t) G_t]$$

Where G_t is the discounted return |

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$$

Would change into:

$$\Delta_\theta J(\theta) = \mathbf{E}_\tau[\sum_{t=0}^{T-1} \Delta_\theta log_{\pi_\theta}(a_t|s_t) Q(s_t, a_t)]$$

where

$$Q(s_t, a_t) = \mathbf{E}[r_{t+1} + \gamma V(s_{t+1})]$$

The algorithm is denominated actor critic because you could see this formula as two functions:

- The critic, which estimates the action value function Q.
- The actor, which updates the policy in the direction given by Q (the critic).

Finally, the second part of A2C is the advantage value. Which intuitively is knowing if it is better to take a specific action versus the average action in a given state.

This value is given by:

$$A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t) = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$

Where V is the state-value.

Therefore, the update equation can be written as:

$$\Delta_\theta J(\theta) = \sum_{t=0}^{T-1} \Delta_\theta log_{\pi_\theta}(a_t|s_t) A(s_t, a_t)$$

Also, it is important to notice that since we only need the V function, we just need one neural network.

5.2 Deep Q-Network

Our second algorithm is not based on policy gradient, but a value function that will give us a value for each state action pair. DQN was first introduced by Mnih et al. in 2015 [4]. This algorithm is a deep neural network and reinforcement learning combination that was the "first demonstration of a general-purpose agent that is able to continually adapt its behavior without any human intervention".

DQN consist in employing a convolutional neural network to approximate the action-value function Q, given by:

$$Q^*(s,a) = \max_{\pi} \mathbf{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a.\pi]$$

this function is the maximum sum of rewards r_t discounted by γ at each step t taking a behaviour policy $\pi = P(a|s)$ where s is an observation and a is an action.

Nevertheless, doing this tends to be either unstable or diverges when a nonlinear function is used to represent the action value (see [4]).

Therefore, Mnih et al. parametrize an approximate value function $Q(s,a;\theta)$ using convolutional neural networks, where θ_i are the weights of said network at an iteration i. Furthermore, they store the experiences $e_t = (s_t, a_t, r_t, s+t+1) \mid D_t = e_1, ..., e_t$ and the Q-learning updates are applied on minibatches of experience $(s,a,r,s')\ U(D)$.

Finally, the loss function is given by:

$$L_i(\theta_i = \mathbf{E}_{(s,a,r,s')}\ U(D)[(r + \gamma \max_{a'}\ Q(s',a';\theta_i^-) - Q(s,a;\theta_i))^2]$$

where γ is the discount factor determining the agent's horizon, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i, which are only updated every C steps and are fixed between individual updates [4].

6 Results

For this section, we will compare both A2C and DQN with no exploration policies versus those exploration policies that did best among our tests. We have chosen two environments from OpenAI to do the testing: CartPole-v0 and Acrobot-v1.

6.1 CartPole-v0

For this environment, we decided to run 400 episodes, this is because more than that would make many of the paths converge and therefore, we could not clearly decide between all of them. Instead, 400 seems to be the number that made the comparison clearer.

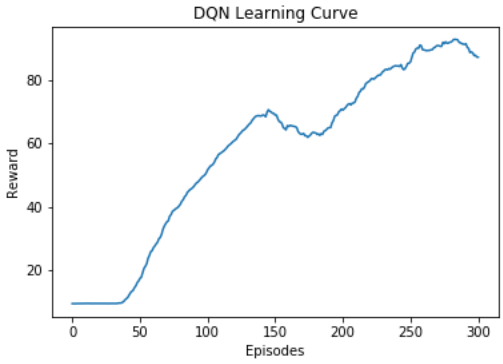
After running all the different combinations available, we chose the best ones and train them five separate times, this is because we wanted to avoid those algorithms with high variance. Below, we have chosen the three best results given the parameters versus the baseline. It is important to note that the lowest leader-board entry ([2]) in OpenAI solves the problem with 355 episodes.

DQN - Cart pole average reward over 100 consecutive trials.				
Highest Score	Lowest Score	Distance measure	Adaptive scaling strategy	Distribution
95	72	-	-	-
200	162	L1	Distance	Reward high
183	45	Linf	Distance	Reward high
121	105	L2	Reactive	Exp high recent

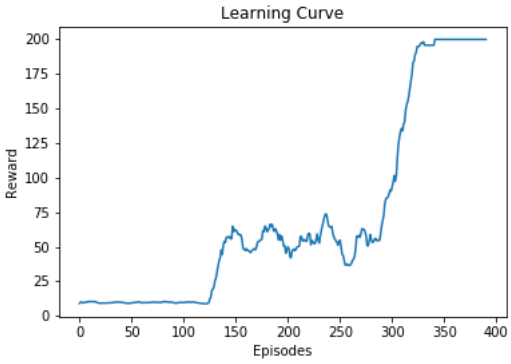
The results are quite promising. It is evident to notice a clear improval over the baseline. Also, our best algorithm was close to secure a spot in the leaderboard of OpenAI.

A2C - Cart pole average reward over 100 consecutive trials.				
Highest Score	Lowest Score	Distance measure	Adaptive scaling strategy	Distribution
87	32	-	-	-
104	83	JS	Proactive	Uniform
100	92	L2	Distance	Uniform
80	74	Linf	Distance	Uniform

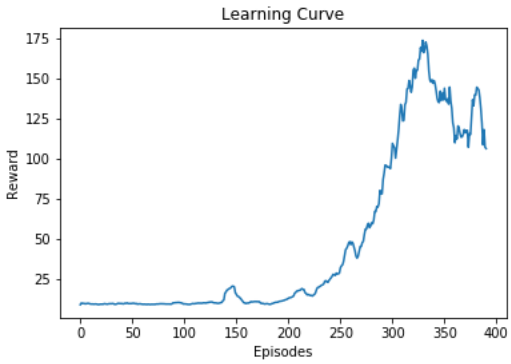
We can see that A2C did not work as well. This is evident due to the fact that the algorithm's baseline did not learn as well as DQN for this specific task. However, we were capable of improving A2C's baseline, which was our main goal.



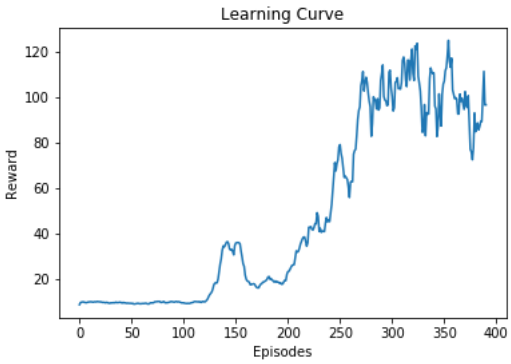
Baseline



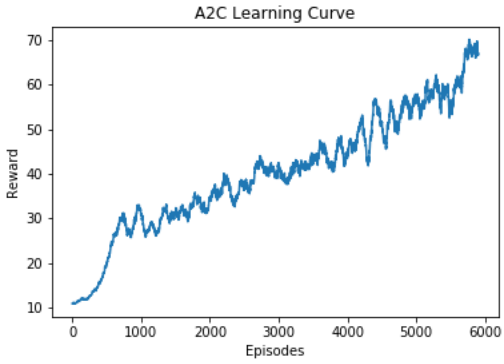
α:= distance | metric:= $L1$ | distribution:= reward high



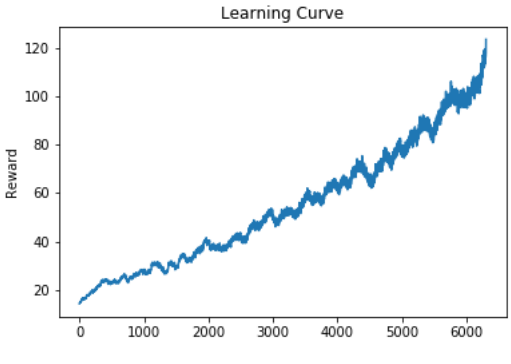
α:= distance | metric:= $Linf$ | distribution:= reward high



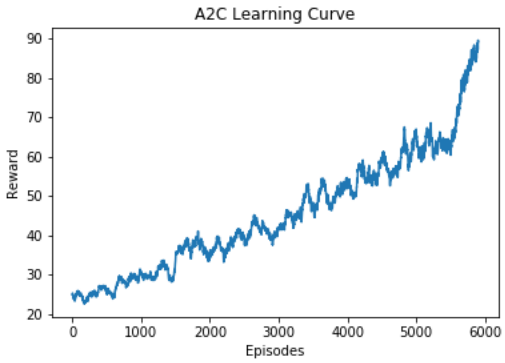
α:= reactive | metric:= $L2$ | distribution:= exp high recent



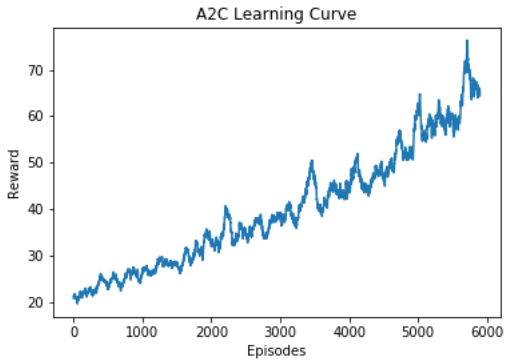
Baseline



α:= proactive | metric:= JS | distribution:= Uniform



α:= distance | metric:= $L2$ | distribution:= Uniform



α:= distance | metric:= $Linf$ | distribution:= uniform

6.2 Acrobot-v1

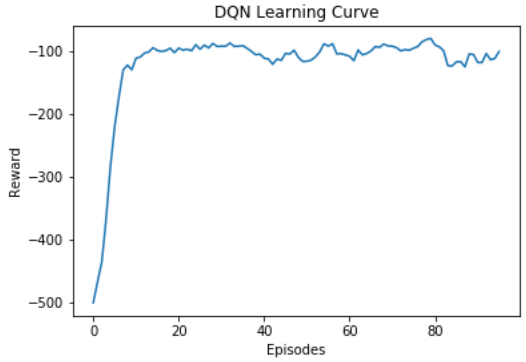
For this environment, we decided to run 100 episodes, this is because we can then compare with the leader board previously mentioned. The results in this paper would place us between the 9th and 10th place in the leader board from OpenAi.

DQN - Acrobot average reward over 100 consecutive trials.				
Highest Score	Lowest Score	Distance measure	Adaptive scaling strategy	Distribution
-102	-114	-	-	-
-82	-106	Linf	Proactive	Uniform
-88	-89	Linf	Reactive	Uniform
-88	-116	KL	Proactive	Reward high

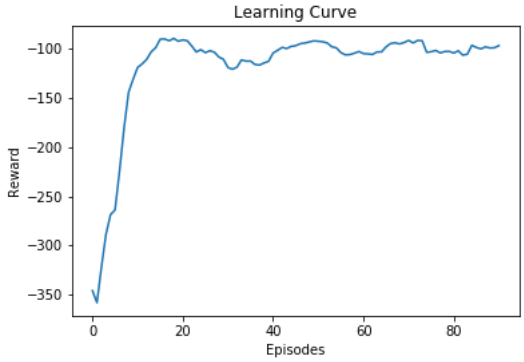
It is clear that our algorithm improves the current baseline rewards, however, most of the choices minimum overlap with the baseline maxima.

A2C - Acrobot average reward over 100 consecutive trials.				
Highest Score	Lowest Score	Distance measure	Adaptive scaling strategy	Distribution
-237	-418	-	-	-
-140	-210	JS	distance	Exp high older
-195	-318	L2	Distance	Exp high recent
-200	-406	KL	Distance	Exp high recent

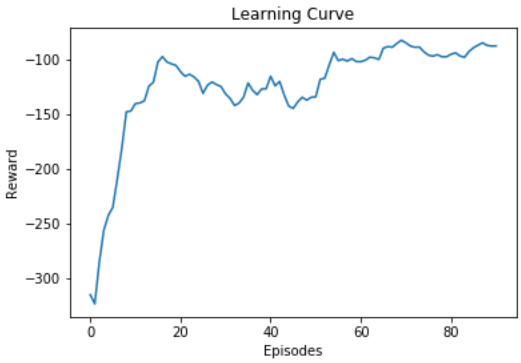
The improval is not as significant compared to the previous tries. This is because the variance is very high while running the algorithm with said parameter. However, it is a better implementation for Acrobot than the Baseline, which variance is very large.



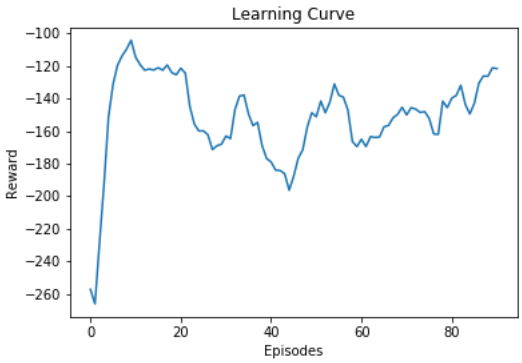
Baseline



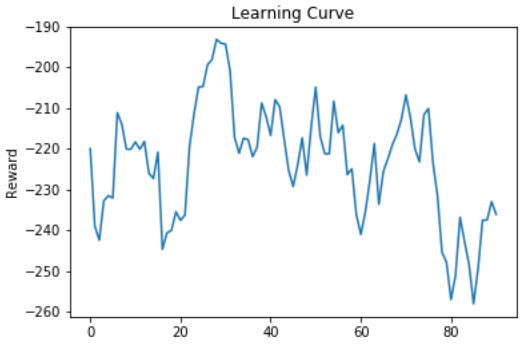
α:= proactive | metric:= $Linf$ | distribution:= Uniform



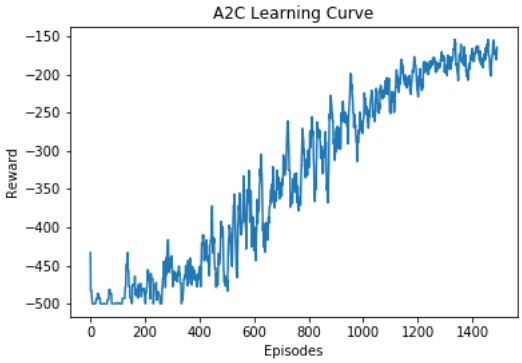
α:= reactive | metric:= $Linf$ | distribution:= Uniform



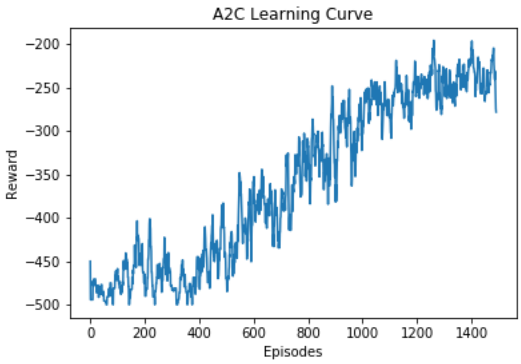
α:= proactive | metric:= KL | distribution:= reward high



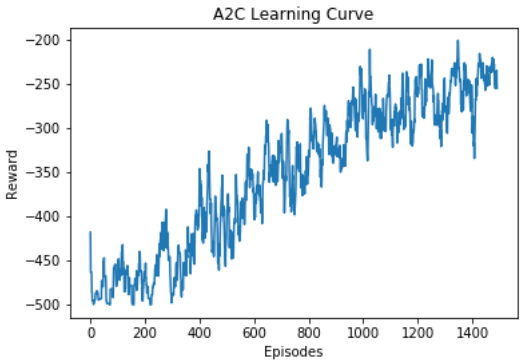
Baseline



α:= distance | metric:= JS | distribution:= exp high recent



α:= distance | metric:= $L2$ | distribution:= exp high recent



α:= distance | metric:= KL | distribution:= exp high recent

7 Conclusion and Further ideas

We believe that the results have showed us that exploration strategies work. However, it has also empirically demonstrated that not every distance and distribution work, which is very important.

References

[1] Zhang-Wei Hong, Tzu-Yun Shann, Shih-Yang Su, Yi-Hsiang Chang, and Chun-Yi Lee. Diversity-driven exploration strategy for deep reinforcement learning. 2018.

[2] OpenAi. Leaderboard. URL `https://github.com/openai/gym/wiki/Leaderboard`.

[3] Satinder P. Singh Yishay Mansour Richard S. Sutton, David A. McAllester. Policy gradient methods for reinforcement learning with function approximation. 1999. URL `https://www.semanticscholar.org/paper/Policy-Gradient-Methods-for-Reinforcement-Learning-Sutton-McAllester/160315c07d18fe785aff07f50c9e44319a0af0cb`.

[4] David Silver1 Andrei A Rusu1 Joel Veness1 Marc G. Bellemare Alex Graves1 Martin Riedmiller Andreas K. Fidjeland Georg Ostrovski Stig Petersen Charles Beattie Amir Sadik Ioannis Antonoglou Helen King Dharshan Kumaran Daan Wierstra Shane Legg Demis Hassabis Volodymyr Mnih, Koray Kavukcuoglu.

[5] Antoni Chan, Nuno Vasconcelos, and Pedro Moreno. A family of probabilistic kernels based on information divergence. 2004.

[6] Sergio Verdu. Total variation distance and the distribution of relative information. pages 1–3, 2014. ISBN 978-1-4799-3589-5. doi: 10.1109/ITA.2014.6804281.

[7] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017. URL `http://arxiv.org/abs/1707.06887`.

[8] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proc. AAAI*, pages 1433–1438, 2008.

[9] Mehdi Mirza Alex Graves Timothy P. Lillicrap Tim Harley David Silver Koray Kavukcuoglu Volodymyr Mnih, Adrià Puigdomènech Badia. Asynchronous methods for deep reinforcement learning. 2016. URL `https://arxiv.org/pdf/1602.01783`.