7-2020

Noisy Importance Sampling Actor-Critic: An Off-Policy Actor-Critic With Experience Replay

Miriam A M Capretz
Western University, mcapretz@uwo.ca

Norman Tasfi
Western University, ntasfi@uwo.ca

Noisy Importance Sampling Actor-Critic: An Off-Policy Actor-Critic With Experience Replay

Norman Tasfi & Miriam Capretz
Department of Electrical And Computer Engineering
Western University
London, Ontario, Canada
{ntasfi, mcapretz}@uwo.ca

***Abstract*—This paper presents Noisy Importance Sampling Actor-Critic (NISAC), a set of empirically validated modifications to the advantage actor-critic algorithm (A2C), allowing off-policy reinforcement learning and increased performance. NISAC uses additive action space noise, aggressive truncation of importance sample weights, and large batchsizes. We see that additive noise drastically changes how off-sample experience is weighted for policy updates. The modified algorithm achieves an increase in convergence speed and sample efficiency compared to both the on-policy actor-critic A2C and the importance weighted off-policy actor-critic algorithm. In comparison to state-of-the-art (SOTA) methods, such as actor-critic with experience replay (ACER), NISAC nears the performance on several of the tested environments while training 40% faster and being significantly easier to implement. The effectiveness of NISAC is demonstrated against existing on-policy and off-policy actor-critic algorithms on a subset of the Atari domain.**

***Index Terms*—deep learning, reinforcement learning, off-policy learning**

I. INTRODUCTION

Recent advances in reinforcement learning (RL) have enabled the extension of long-standing methods to complex and large-scale tasks such as Atari [1], Go [2], and DOTA [3]. The key driver has been the use of deep neural networks, a non-linear function approximator, with the combination usually referred to as *Deep Reinforcement Learning* (DRL) [1], [4]. However, deep learning-based methods are usually data-hungry, requiring millions of samples before the network converges to a stable solution. As such, DRL methods are usually trained in a simulated environment where an arbitrary amount of data can be generated.

RL algorithms can be classified as either learning in an *off-policy* or *on-policy* setting. In the *on-policy* setting, an agent learns directly from experience generated by its current policy. In contrast, the *off-policy* setting enables the agent to learn from experience generated by its current policy or/and other separate policies. An algorithm that learns in the *off-policy* setting has much greater sample efficiency as old experience from the current policy can be reused; it also enables *off-policy* algorithms to learn an optimal policy while executing an exploration-focused policy [5].

A famous off-policy method is Q-Learning [6] which learns an action-value function, $Q(s, a)$, that maps the value to a state s and action a pair. Deep Q-Learning (DQN), the marriage of Q-Learning with deep neural networks, was popularised by Mnih *et al.* [1] and used various modifications, such as experience replay, for stable convergence. Within DQN, experience replay [7] is often motivated as a technique for reducing sample correlation. Unfortunately, action-value methods, including Q-Learning, have two significant disadvantages. First, they learn deterministic policies, which cannot handle problems that require stochastic policies. Second, finding the greedy action with respect to the Q function can be costly for large action spaces. To overcome these limitations, one could use policy gradient algorithms [8], such as A2C an on-policy actor-critic method [9], which learn in an *on-policy* setting at the cost of sample efficiency.

The ideal solution would be to combine the sample efficiency of *off-policy* algorithms with the desirable attributes of *on-policy* algorithms. Work along this line has been done by using importance sampling [10] which adjusts off-policy samples according to a weight ρ, the ratio between the current policy $\pi(a|s)$ and experience policy $\mathcal{B}(a|s)$. Where the weight is defined as $\rho = \frac{\pi(a|s)}{\mathcal{B}(a|s)}$. NISAC modifies A2C by using additive action space noise, aggressive truncation of importance sample weights, and large batchsizes enabling off-policy learning from stored trajectories. The contributions of this work are as follows:

- We introduce Noisy Importance Sampling Actor-Critic (NISAC), a fully *off-policy* actor-critic algorithm, that learns from stored off-policy trajectories.
- Experimentally prove in the Atari domain, that NISAC outperforms, in performance and sample efficiency, both A2C [9] and the off-policy truncated importance sampling method [11].
- Show the addition of additive action space noise, to the numerator of ρ, changes the distribution of weights, improves learning, and that Gumbel noise, rather than Normal or Uniform, is most performant.
- NISAC training time is 40% faster than ACER, a SOTA off-policy actor-critic methods, nears its performance on several environments, and is easier to implement.

The paper is organized as follows: Sec. II covers background information, Sec. III describes the NISAC algorithm, Sec. IV details the experimental results, analysis, and ablations of our methodology. Sec. VI discusses future work, and finally Sec. V provides concluding remarks.

N. Tasfi, M. A. M. Capretz, Noise Importance Sampling Actor-Critic: An Off-Policy Actor-Critic With Experience Replay, Proc. of the IEEE 2020 International Joint Conference on Neural Networks, 2020.

II. Background and Notation

Problem Setup: Consider an agent interacting with a Markov Decision Process (MDP) consisting of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition function $P : \mathcal{S} \times \mathcal{A} \rightarrow (\mathcal{S} \rightarrow [0, 1])$, and a reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Within this work, discrete actions and time steps are assumed. A *policy* is a probability distribution over actions conditioned on states, $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. At each time step t, the agent observes the environment state $s_t \in \mathcal{S}$, chooses an action $a_t \in \mathcal{A}$ from its policy $\pi(a_t|s_t)$, and receives a reward r_t from the environment. The goal of the agent is to maximize the discounted future return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. The discount factor $\gamma \in [0, 1)$ trades off the importance of immediate and future rewards. Following from this, the value function of a policy π, in a discounted problem, is defined as the expected return $V^\pi(s_t) = \mathbb{E}_{a\sim\pi}[G_t|S_t = s]$ and its action-value counterpart as $Q^\pi(s_t, a_t) = \mathbb{E}_{a\sim\pi}[G_t|S_t = s, A_t = a]$.

Policy Gradient Methods: In reinforcement learning (RL) the direct optimization of a stochastic policy, with parameters θ, requires the use of the policy gradient theorem [8]. The policy gradient theorem provides an expression for the gradient of the discounted reward objectives with respect to the parameter $theta$. Therefore, the parameters θ of the differentiable stochastic policy $\pi_\theta(a_t|s_t)$ are updated using the following gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\Psi^\pi(s_t, a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{1}$$

where $\Psi^\pi(s_t, a_t)$, as shown by Schulman *et al.* [12], can be replaced with quantities such as: the total reward of the trajectory, the temporal difference residual, or the state-action value function $Q^\pi(s_t, a_t)$. The choice of Ψ^π affects the variance of the estimated gradient. This work uses the advantage function $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, which provides a relative measure of value for each action. The advantage function helps to reduce the variance of the gradient estimator while keeping the bias unchanged. The A3C algorithm, Asynchronous Advantage Actor-Critic [9], uses policy-gradients with an advantage function with many parallel critics. A3C uses multiple actors, trained in parallel, each interacting with its own environment. The parameters of each actor are synced periodically with the global parameters. A2C, the synchronous and deterministic version syncs all actors with the same set of parameters after a fixed number of steps have been completed.

Gumbel Distribution: The Gumbel distribution is a probability distribution that is used to model the maximum of value from a set of independent samples [13]. The density of this distribution is defined as:

$$p(x) = \frac{1}{\beta} \exp(-z - \exp(-z)) \tag{2}$$

where $z = \frac{x-\mu}{\beta}$ and parameterized by scale β and location μ. We evaluate noise sampled from the Gumbel distribution as a potential candidate for additive noise in NISAC. The Gumbel distribution has several applications within machine learning, such as the Gumbel-Max [14] and Gumbel-Softmax tricks [15]. However, our usage of the Gumbel distribution differs from the Gumbel-Max and Gumbel-Softmax in that we see Gumbel noise as perturbing the current policy – as noise would from any other distribution.

Importance Sampling: In practice, the policy gradient is estimated from a trajectory of samples generated by the *on-policy* stationary distribution $\pi(a|s)$. This limits the efficiency of typical policy gradient methods, such as actor-critic, compared to methods like Deep Q-Learning which can learn *off-policy*. A common approach to using *off-policy* samples is a technique known as *importance sampling* [10], [16]–[18].

Given a trajectory of samples generated by some behaviour policy $\mathcal{B}(a|s)$, the policy gradient from Equation 1 is modified to be:

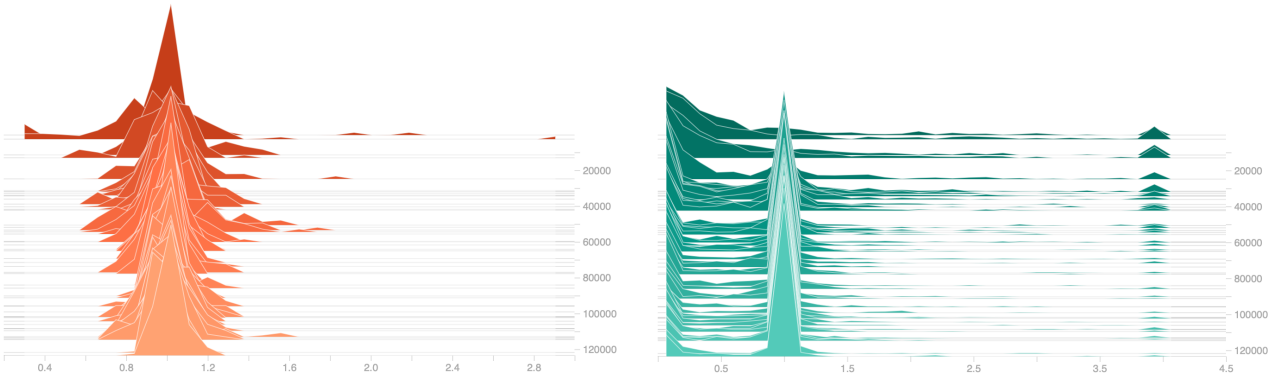
$$\nabla_\theta J(\theta) = \mathbb{E}_\mathcal{B}\Big[\rho_t \Psi^\pi(s_t, a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)\Big] \tag{3}$$

where ρ is the known as the *importance weight* and is defined as a ratio between the current policy $\pi(a|s)$ and the behaviour policy $\mathcal{B}(a|s)$ as $\rho_t = \frac{\pi(a_t|s_t)}{\mathcal{B}(a_t|s_t)}$. Unfortunately, the importance weighted gradient in Equation 3 suffers from high variance. To reduce variance, Wawrzyński [11] proposed truncating each importance weight to the interval $[0, c]$ where c is some constant. NISAC builds upon truncated importance sampling, but uses aggressive truncation of the importance weight.

Actor-Critic With Experience Replay: Actor-critic with experience replay (ACER) [19], is an off-policy actor-critic algorithm that uses experience replay. ACER builds upon the on-policy A3C algorithm [9]. Wang *et al.* [19] proposes three changes to A3C to convert it to an off-policy method: truncated importance sampling with bias correction, retrace Q-value estimation, and updates performed with TRPO [20]. As the gradients from importance sampling based updates can suffer from high variance, the quantity is usually truncated by a constant c as done by Wawrzyński [11]. However, this truncation introduces a bias so Wang *et al.* [19] propose a correction term. Therefore, the gradient for ACER is defined as:

$$\begin{aligned}\nabla_\theta J(\theta) = &\min(c, \rho_t)\big\{Q^{ret}(s_t, a_t) - V_w(s_t)\big\}\nabla_\theta \ln \pi_\theta(a_t|s_t) \\ &+ \mathbb{E}_{a\sim\pi}[\max(0, \frac{\rho_t - c}{\rho_t})(Q_w(s_t, a) - V_w(s_t)\nabla_\theta \ln \pi_\theta(a|s_t)]\end{aligned} \tag{4}$$

where Q_w and V_w are value functions predicted by the networks critic with parameters w and Q^{ret} is the estimated Q-value given by Retrace [21]. Retrace is a return-based Q-value estimation algorithm that can be used with off-policy data. The first term of Equation 4 is the truncated importance weight which reduces variance. The second term corrects the bias introduced by the first term such that ACER's policy gradient is an unbiased estimation. The implementation of ACER is quite involved, requiring several moving pieces to implement correctly. In contrast, NISAC requires fewer changes and is much simpler to implement.



(a) Truncated ρ with current policy

(b) Truncated ρ with noisy policy

Figure 1: Histograms of importance sampling weights ρ during training. The x-axis is the magnitude of $\bar{\rho}$ and the y-axis is the number of updates since start of training. Both ratios ρ are truncated between $[0, c]$, where is c the truncation value and here, $c = 4$. a) Using truncated ρ with the current policy in the numerator and b) the ratios seen during training from NISAC, which corresponds to use of the noisy policy in the numerator. In this case noise was sampled from a standard Gumbel distribution.

III. NISAC

NISAC builds off the *on-policy* actor-critic algorithm A2C. To enable *off-policy* learning NISAC uses aggressive clipping on importance sampling, additive noise, and large batches sampled from a replay memory. Psuedo-code for NISAC is provided in Algorithm 1. We begin by defining a few quantities used in the importance sampling weight ρ. In importance weighting, two policy classes exist: the current policy $\pi(a|s;\theta)$ and the behaviour policy $\mathcal{B}(a|s)$. Because replay memory is being used, the behaviour policy is simply the distribution over actions with an old parameter setting θ^*:

$$\mathcal{B}(a_t|s_t) = \pi(a_t|s_t;\theta^*) \tag{5}$$

NISAC introduces a third policy, the noisy policy $\mathcal{F}(a|s)$[1], which results from adding noise ϵ drawn from some distribution $\mathcal{D}$ to the normalized logits of the current policy and passing them through a softmax:

$$\mathcal{F}(a_t|s_t) = \text{softmax}(\log \pi(a_t|s_t;\theta) + \epsilon) \tag{6}$$

We can see, from Figure 1(b), where we truncate values to some constant c, that the addition of noise forces the ratio ρ into one of three modes instead of clumping around 1. In this work we examine the difference between noise drawn from Uniform, Gumbel, and Normal distributions. One major qualitative difference between these types of noise, with respect to the distributions shape, is that additive Gumbel noise results in less mass between modes.

Similarly to previous work, this study uses importance sampling ρ_t to weight the updates of the loss function [10], [11], [19]. However, instead of using the current policy, $\pi(\cdot|s_t)$, in the numerator, it is replaced with the noisy policy $\mathcal{F}(a_t|s_t)$:

$$\rho_t^{(i)} = \frac{\mathcal{F}(a_t^{(i)}|s_t)}{\mathcal{B}(a_t^{(i)}|s_t)} \tag{7}$$

The range of ρ_t is clipped to $[0, c]$ and this clipped importance weight is referred to as $\bar{\rho}_t$. Clipping the upper bound prevents the product of many importance weights from exploding and reduces the variance [11]. Wang *et al.* [19] notes that truncating the importance weights in this way introduces bias to the estimator. However, the value of having an unbiased algorithm is unclear, as shown by Thomas [22], and does not always correspond to improved performance. The effect of aggressively clipping ρ_t and using the noisy policy $\mathcal{F}(\cdot|s)$ in ρ_t has an interesting effect on the way the policy updates are weighted. To understand the effect of these two modifications we again refer to Figure 1, which shows the history during training of the ratios $\bar{\rho}$, truncated between $[0, c]$ to a constant c, where Figure 1(a) is plain truncated importance sampling and Figure 1(b) is truncated importance sampling with added action space noise.

From Figure 1(a), we notice that the majority of the importance weights $\bar{\rho}$ are centered around 1 resulting in off-policy samples that are weighted approximately the same. However, by using additive action space noise, in this case from the Gumbel distribution, we can see from Figure 1(b) a multi-modal distribution appears. The weights $\bar{\rho}$ form three distinct modes around $\{0, 1, c\}$, with small amounts of additional density "smeared" between modes. We hypothesize that the addition of noise, via the noisy policy $\mathcal{F}(a|s)$, has the effect of stabilising the updates to the network as the weighting assigned to each off-policy sample is near $\{0, 1, c\}$. Interestingly, each of the modes can be grouped into cases of "agreement" or "disagreement" between the noisy policy $\mathcal{F}(\cdot|s)$ and behaviour

[1]As $\mathcal{N}$ is typically used for Normal distributions, we used $\mathcal{F}$ to avoid confusion.

Algorithm 1 Pseudo-code for k-step NISAC

Initialize parameters θ and θ_v.
Initialize replay memory $\mathcal{M}$ with capacity N.
repeat
  for $i \in \{0, \cdots, k\}$ **do**
    Perform a_i according to $\pi(\cdot|s_i;\theta)$.
    Receive reward r_i and new state s_{i+1}.
    Store $(s_i, a_i, r_i, \pi(\cdot|s_t))$ in $\mathcal{M}$.
  end for
  Sample b trajectories $\{s_0, a_0, r_0, \mathcal{B}(\cdot|s_0), \cdots, s_k, a_k, r_k, \mathcal{B}(\cdot|s_k)\}$ from the replay memory $\mathcal{M}$.
  for $i \in \{0, \cdots, k\}$ **do**
    Sample ϵ from noise distribution $\mathcal{D}$.
    Compute $\pi(\cdot|s_i;\theta)$ and $\mathcal{F}(\cdot|s_i)$.
    $\bar{\rho}_i = \text{clamp}\Big(\frac{\mathcal{F}(a_i|s_i)}{\mathcal{B}(a_i|s_i)}, 0, c\Big)$
  end for
  $R \leftarrow \begin{cases} 0 & \text{for terminal } s_k \\ V(s_k;\theta_v) & \text{otherwise} \end{cases}$
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  for $i \in \{k-1, \cdots, 0\}$ **do**
    $R \leftarrow \begin{cases} 0 & \text{for terminal } s_i \\ r_i + \gamma R & \text{otherwise} \end{cases}$
    Accumulate gradients $d\theta \leftarrow d\theta + \bar{\rho}_i \nabla_\theta \log \mathcal{F}(a_i|s_i)\big\{R - V(s_i;\theta_v)\big\}$
    Accumulate gradients $d\theta_v \leftarrow d\theta_v + \nabla_{\theta_v}(R - V(s_i;\theta_v))^2$
  end for
  Perform update of θ using $d\theta$ and θ_v using $d\theta_v$.
until Max iteration or time reached.

policy $\mathcal{B}(\cdot|s)$. Where the case of agreement corresponds to ratios and the mode near 1 while disagreements correspond to ratios and modes at 0 and c. More exactly, when the noisy policy disagrees with the behaviour policy, say $\mathcal{F}(\cdot|s) \approx 1$ and $\mathcal{B}(\cdot|s) \approx 0$, the update the policy receives is at most clipped by the upper bound of our interval: c. On the other hand, when the situation is reversed, but still in disagreement, with $\mathcal{F}(\cdot|s) \approx 0$ and $\mathcal{B}(\cdot|s) \approx 1$, the policy has an importance weight of ~0.

Putting this all together the update equation for NISAC is as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathcal{B}}[\bar{\rho}_t A(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{8}$$

However, empirically we found that using the noisy policy in place of the current policy, in $\nabla \log \pi(a_t|s_t)$, produces better performance. This is empirically validated by an ablation in Section IV-B. We hypothesize that the usage of the noisy policy in place of $\pi(a|s)$ affects three changes during learning. First, by adding action space noise the resulting policy will tend towards a categorical distribution earlier in training which will affect how strongly the networks weights are updated. Second, as the noisy policy used in both the numerator of ρ_t and in place of the policy score use the same set of noise samples $\{\epsilon_i, \ldots, \epsilon_k\}$ the updates synchronize such that the policy is updated maximally in agreement. And finally, we believe the additive noise acts as strong exploration force to the policy as it can "shake-out" policies that are near deterministic, even later in training.

IV. Experiments

Our experiments focus on the Atari domain [23] as there exists a large amount of variety between environments and the states are represented as raw high-dimensional pixels. The gym software package by Brockman *et al.* [24] was used to conduct all the experiments. The network architecture and hyper-parameters, for each respective algorithm, were constant throughout all experiments.

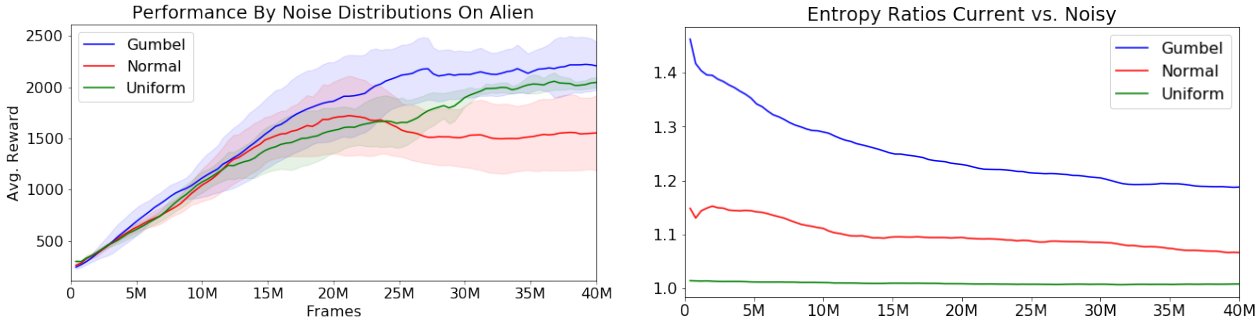
This study used the same input pre-processing and network architecture as Mnih *et al.* [9]. The network architecture consists of three convolutional layers as follows: 32 8×8 filters with stride 4, 64 4×4 filters with stride 2, and 32 3×3 filters with stride 1. The final convolutional layer feeds into a fully-connected layer with 512 units. All layers are followed by rectified non-linearity. Finally, the network outputs a softmax policy over actions and a state-value.

The experimental set-up used 16 threads running on a GPU equipped machine. As is standard in the Atari domain [1], [9], [19], all experiments are trained for 40 million frames. The optimization procedure used RMSProp [25] with a learning rate of 0.0005, policy entropy regularization weight of 0.01, and a discount factor of $\gamma = 0.99$. We estimate the gradient given in Equation 8 by uniformly sampling b trajectories of length k from a replay memory with size N. The advantage function $A(s_t, a_t) = R_t^{(k)} - V(s_t)$ is used, where $R_t^{(k)}$ is the bootstrapped k-step return for time t. A replay memory of size $N = 250000$ was kept, an update was performed every $k = 5$ steps in the environment, and a clamping coefficient of $c = 4$ was used. We sample $b = 64$ trajectories of length $k = 5$ for each update. Learning begins after we have collected $10,000$ samples in the replay memory. All experiments used the same hyperparameter settings and network architecture.

We tuned the hyperparameters and developed NISAC on the *FishingDerby* environment only; the other environments can be considered *"out of sample"*. We use the best 3 seeds and as standard we report the mean value of rewards achieved during training with 1 std. deviation as shaded areas on all graphs.

Due to limited computational resources, we were only able to evaluate NISAC on a subset of the environments and with a smaller replay memory size. Therefore, in this study, an effort was made to select environments that best showcase the performance of off-policy (truncated importance sampling) and on-policy (A2C) actor-critic methods. We note that the performance can be expected to improve with a larger replay memory, as seen with DQN and other off-policy methods using replay memory. Additionally, we focused the examination of our ablations on the *Alien* environment to reduce computational requirements.

The present work was compared with a SOTA off-policy actor-critic algorithm, ACER [19], an on-policy actor-critic algorithm, A2C, the synchronous version of A3C [9], and an off-policy actor-critic with truncated importance sampling (t-IS) [10], [11]. A2C and ACER used the *baselines* package provided by OpenAI [26]. The hyperparameters of ACER and A2C are identical to those used on the Atari environment by previous works [19] [9]. For t-IS we used a clipping constant $c = 10$ and replay memory of 250000 samples.



(a) (b)

Figure 2: Above we compare the behaviour between Gumbel, Normal, and Uniform distribution. a) Performance on the Atari game Alien under different noise distributions. b) We look at the entropy ratio between the current policy and the noisy policy $\frac{\mathbf{H}[\pi]}{\mathbf{H}[\mathcal{F}]}$.

A. Additive Noise Distribution

Within this section, we vary the noise generating distribution used in NISAC and examine the effect on performance and learning. In particular, we compare noise sampled from the standard Gumbel distribution, the standard Normal distribution, and the Uniform distribution $[0, 1]$. Our analysis looked at the performance each variant achieved on the *Alien* Atari game over 40 million frames. The experiments only adjust the noise generating distribution with no other parameter changes.

From the results in Figure 2(a), we see that the Gumbel and Normal distributions have the same initial rate of improvement but diverge roughly midway through training. The Normal distribution appears to degrade in performance before becoming stable at ~1500. The Uniform distribution, sampled between $[0, 1]$, has the same convergence characteristic as the other two distributions but instead of diverging away, similar to the Normal distribution, it continues upward before converging at ~2000. We can see that the Gumbel distribution is the most performant.

Next, in Figure 2(b), we compared the entropy $\mathbf{H}[.]$ of the current policy $\pi(a|s)$ over the noisy policy $\mathcal{F}(a|s)$. This experiment helps us understand how much each policy is exploring relative to the other as well as how the ratios ρ change over time. For Uniform noise, we see it is flat throughout training implying that the current policy and noisy policy have equivalent entropy and therefore will "explore" at the same rate and produce importance sampling weights which are roughly 1. However, when examining the Gumbel and Normal noise variants we see that the noisy policy has greater entropy. When using Normal noise we see that initially the noisy policy has +15% more entropy but this quickly decays towards the end of training, falling below a 10% difference. We would like to note the small dip of the entropy ratio roughly coincides with the peak and draw down of the Normal variant in Figure 2(a). Looking towards the Gumbel variant, we can see the noisy policy has over +40% more entropy than the current policy and declines to about +20% towards the end of training. This implies that Gumbel noise will tend to "explore" more throughout training with the effect magnified earlier on. As the behaviour policy $\mathcal{B}(a|s)$ will roughly trail behind the current policy $\pi(a|s)$ during training we can expect the ratios ρ to be at the outer modes 0 and c. Later in training, as the entropy ratio drops the likelihood that the policies disagree decreases which corresponds to ratios ρ at roughly 1. Referring back to Figure 1(b), we can indeed see the two aforementioned trends occur: ρ is mostly around 0 and c and then most of the ratios go to 1. The earlier stages coincide with greater exploration as the two policies are in disagreement more often.

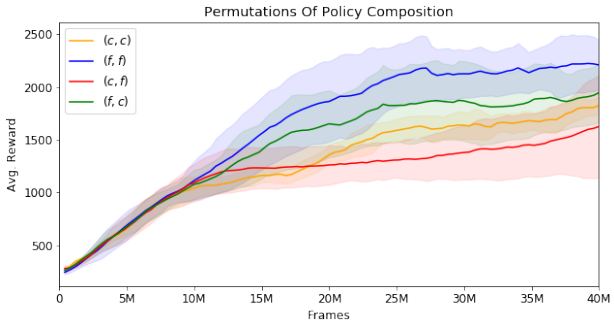


Figure 3: Variations in the choice of policy x in numerator of ρ_t and the policy y used to update the network. We see that the combination corresponding to NISAC has the highest performance and faster convergence speed. In the legend, c corresponds to current policy π and f corresponds to the noisy policy $\mathcal{F}$.

B. Noisy Policy Placement

Here, we examine the effect of using the noisy policy $\mathcal{F}(a_t|s_t)$ in the importance ratio $\bar{\rho}_t$ and as a replacement for the current policy $\pi(\cdot|s)$ in the policy gradient update. Going forward we use the Gumbel distribution, as by Section IV-A, it is the most performant. We examine different combinations of the current or noisy policy which modify Equation 8 as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_\mathcal{B}[\min(c, \frac{x}{\mathcal{B}(a_t|s_t)})A(s_t, a_t)\nabla_\theta \log y] \quad (9)$$

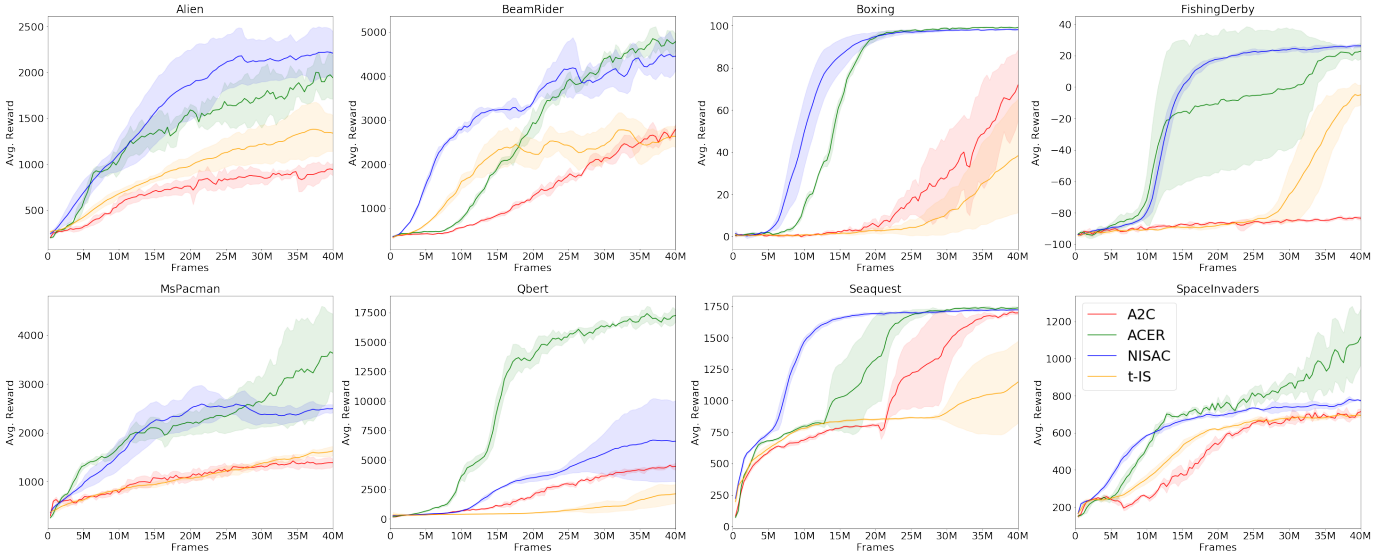


Figure 4: Training performance across 8 Atari games. We see the performance of NISAC (shown in blue) against ACER (shown in green), an off-policy actor-critic algorithm, the on-policy algorithm A2C (shown in red), and the off-policy actor-critic algorithm t-IS (shown in orange). The graphs show the average performance over 3 seeds with 1 standard deviation shown as the shaded region. NISAC matches or exceeds the performance of A2C and t-IS on all environments shown; while in all cases achieving improved convergence speed.

with each combination specified as a tuple of the form (x, y). Specifically, we look at the following combinations (π, π), $(\pi, \mathcal{F})$, $(\mathcal{F}, \pi)$, and $(\mathcal{F}, \mathcal{F})$. The combinations specified by (π, π) results in t-IS, no noise, and $(\mathcal{F}, \mathcal{F})$ in NISAC. From Figure 3 where c corresponds to current policy π and f corresponds to the noisy policy $\mathcal{F}$, we see that variant $(\mathcal{F}, \mathcal{F})$, corresponding to NISAC, has the highest performance and fastest rate of convergence. This variant outperforms $(\mathcal{F}, \pi)$ showing that there is indeed some added benefit of additional noise. Finally, we note that the lowest performing variant, $(\pi, \mathcal{F})$, uses additive noise to the policy in the importance weight ratio. As mentioned previously and shown in Section IV-A, we see that the noisy policy strongly affects how the networks weights are updated during all points of training.

C. Atari Results

To test the proposed methodology, the performance of NISAC on a subset of Atari environments was examined. In particular, the following environments were investigated: *Alien, BeamRider, Boxing, FishingDerby, MsPacman, Qbert, Seaquest,* and *SpaceInvaders*. We report the average reward every 1000 episodes over 40 million frames. As mentioned previously, environments were chosen where either the *on-policy* algorithms perform well or where there is a clear difference in performance between an *off-policy* and *on-policy* method. For NISAC, we used noise sampled from the Gumbel distribution, validated in Section IV-A, and the noisy policy in the numerator of the importance weight and the policy gradient update, as validated in Section IV-B.

From Figure 4, we see the performance of NISAC, shown in blue, in comparison to the *off-policy* ACER algorithm, shown in green, the *on-policy* A2C algorithm, shown in red, and the t-IS algorithm, shown in orange. We see that the use of replay memory and learning with *off-policy* samples significantly improves the sample efficiency of both NISAC over A2C. Qualitatively, we see that NISAC converges significantly faster than A2C while also exceeding A2C's performance on this subset of Atari environments.

The performance gap between t-IS and NISAC is also large, with NISAC showing both increased performance and sample efficiency. The difference from NISAC to t-IS is an aggressive truncation of the importance weights and the noisy policy $\mathcal{F}(\cdot|s)$ used in both the importance weight and policy. As seen from Figure 4 this gives a significant increase in performance.

Algorithm	Wallclock Time
ACER	8h47m
NISAC	6h14m
A2C	4h53m
t-IS	4h39m

Table I: Average Wallclock Time over Atari games: We can see that NISAC is 40% faster than ACER. The other baseline methods, A2C & t-IS, are faster still but are less performant across all Atari environments.

We achieve similar sample efficiency between the *off-policy* actor-critics, NISAC and ACER, across each environment. We see that NISAC sees a fast initial increase in performance across almost all environments. ACER, a SOTA algorithm, out performs NISAC on the *MsPacman*, *Qbert*, and *SpaceInvader* environments. However, we note that NISAC trains 40% faster than ACER, shown in Table I, and is significantly easier to implement with fewer modifications to the A2C algorithm,

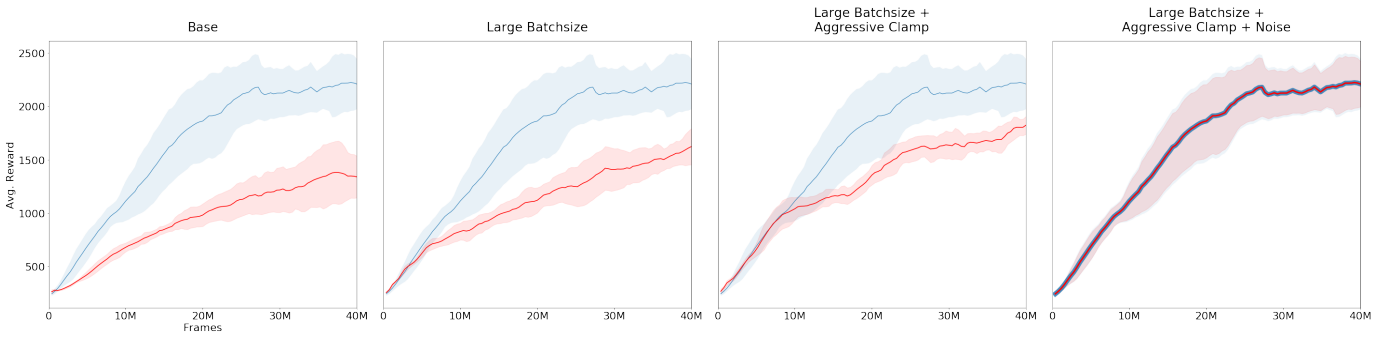


Figure 5: Ablations of NISAC. The full NISAC algorithm is shown as the blue curve while the stripped versions are shown in red. From left to right we gradually add the components onto the *base* version until we arrive at the full NISAC algorithm, shown in the last pane. The lines in the last pane are identical but with one graph stylized.

while providing better performance than both A2C and t-IS.

D. Stability

It is natural to inquire on the stability of this method, as we rely on additive noise which could cause instability after an optimal policy has been reached. To this end, we evaluate the stability of NISAC by increasing the number of training iterations such that 150 million frames are seen. The *Boxing* and *FishingDerby* environments are used for this evaluation. The environments were chosen as the policy had achieved the highest score during training, a known ceiling, and any instability would cause a divergence to a sub-optimal policy.

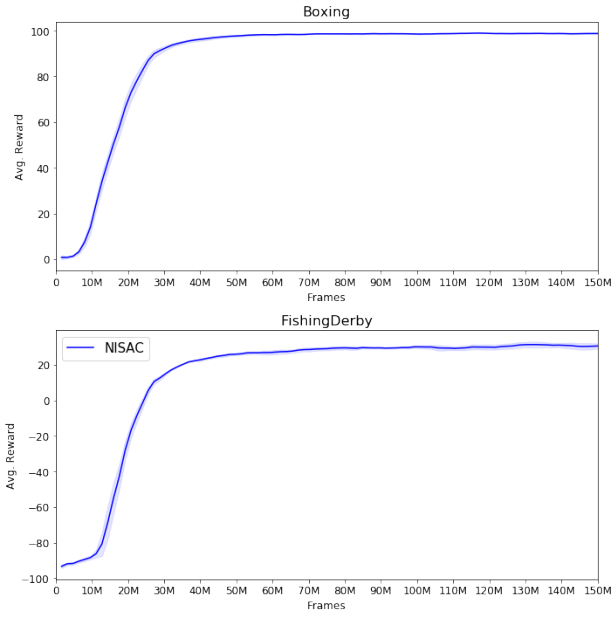


Figure 6: Stability of NISAC with extended training time. NISAC is trained for 150 million frames, ~$4x$ longer, on the *Boxing* and *FishingDerby* environments. We see that NISAC experiences little to no oscillations in performance even with continued weight updates.

From the rather uninteresting graphs, shown in Figure 6, we see that NISAC can converge to and maintain a stable policy even with continued parameter updates. To overcome the noise added by the Gumbel distribution requires the network to output a near one-hot-encoded categorical distribution.

E. Ablations Of Components

In this experiment, we performed ablations of NISAC to understand the importance of each component and impact on NISAC's performance. The results of the ablation are shown in Figure 5 with the complete NISAC algorithm, that is all the components that comprised NISAC, in blue and a stripped version as a red curve. We start with a *base* version, essentially the truncated importance sampling algorithm (t-IS) [11], shown in the left-most panel in Figure 5.

From Table II we see that the *base* version has a performance -39.39% lower than NISAC. The addition of a larger batchsize, from 16 sampled trajectories, to 64 as shown in the second panel in Figure 5 causes an increase of $+21.30\%$ over the *base* version and narrows the difference to NISAC to -26.49%. Using an aggressive clamp of 4 instead of 10 on the importance sampling ratio $\bar{\rho}$ improves performance by an additional $+12.33\%$. Finally, the addition of noise sampled closes the gap with a final increase of $+21.09\%$. It is clearly shown that *large batchsize* and *additive noise* contribute the most to the performance increase between stripped versions.

	%Δ to NISAC	%Δ from last
Base	-39.39%	N/A
+Large Batchsize	-26.49%	+21.30%
+Aggressive Clamp	-17.43%	+12.33%
+Noise	0%	+21.09%

Table II: The table provides the percent deltas between either the stripped version to NISAC or the current model to the last. We measure the change between the last 100 episodes.

Additionally, while difficult to quantify, we can see from the plots that the *aggressive clamp* and *additive noise* improve sample efficiency the most. It is clear that all the components, large batchsize, aggressive clamping, and additive policy space noise are crucial to NISAC's aggregate performance.

V. Conclusion

In this paper we have introduced Noisy Importance Sampling Actor-Critic (NISAC), a fully *off-policy* actor-critic algorithm that learns from stored off-policy trajectories. We have proven, experimentally that NISAC improves upon the performance and sample efficiency of A2C [9], an *on-policy* actor-critic, and truncated importance sampling [11], an *off-policy* algorithm. NISAC nears the performance of ACER [19], a SOTA *off-policy* actor-critic method, on several environments while completing a training session in 40% less time and being significantly easier to implement. We have provided an analysis on the effect of additive action space noise, identified the Gumbel distribution as the most performant variant, and examined where the noisy policy can be used within the importance sampling weight ρ and policy gradient update. From our analysis we have shown that additive action space noise fundamental changes the distribution of importance sample weights ρ during training. And finally, we have shown that each component in NISAC contributes to its improved performance over the baseline methods and even with additive action space noise the learned policies are stable.

VI. Future Work

One interesting future work could include finding suitable distributions to sample noise from that works with continuous actions. We also feel further investigation into possible annealing schedules for the clamping constant c could yield interesting results. Finally, if the compute is available, the evaluation of the full Atari suite across all permutations discussed would provide additional insight.

VII. Acknowledgements

We would like to thank Eder Santana for helpful discussions early in the process; Rui Wu for fruitful discussions around our methodology; and Justin Tomasi, Santiago Gomez, and Ljubisa Sehovac for proof-reading and their valuable suggestions on the paper. This research has been partially supported by the NSERC Strategic Project Grant STPGP 506840-17 at the University Of Western Ontario.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[3] OpenAI, "Openai five," https://blog.openai.com/openai-five/, 2018.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[5] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 2.

[6] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[7] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[8] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[10] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," *arXiv preprint arXiv:1205.4839*, 2012.

[11] P. Wawrzyński, "Real-time reinforcement learning by sequential actor–critics and experience replay," *Neural Networks*, vol. 22, no. 10, pp. 1484–1497, 2009.

[12] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[13] E. J. Gumbel, *Statistical theory of extreme values and some practical applications: a series of lectures*. US Government Printing Office, 1948, vol. 33.

[14] C. J. Maddison, D. Tarlow, and T. Minka, "A* sampling," in *Advances in Neural Information Processing Systems*, 2014, pp. 3086–3094.

[15] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[16] N. Meuleau, L. Peshkin, L. P. Kaelbling, and K.-E. Kim, "Off-policy policy search," 2000.

[17] T. Jie and P. Abbeel, "On a connection between importance sampling and the likelihood ratio policy gradient," in *Advances in Neural Information Processing Systems*, 2010, pp. 1000–1008.

[18] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.

[19] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[20] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[21] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, "Safe and efficient off-policy reinforcement learning," *CoRR*, vol. abs/1606.02647, 2016. [Online]. Available: http://arxiv.org/abs/1606.02647

[22] P. Thomas, "Bias in natural actor-critic algorithms," in *International conference on machine learning*, 2014, pp. 441–448.

[23] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[24] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[25] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," 2012.

[26] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," *GitHub, GitHub repository*, 2017.