GUMBELCLIP: OFF-POLICY ACTOR-CRITIC USING EXPERIENCE REPLAY

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper presents GumbelClip, a set of modifications to the actor-critic algorithm, for off-policy reinforcement learning. GumbelClip uses the concepts of truncated importance sampling along with additive noise to produce a loss function enabling the use of off-policy samples. The modified algorithm achieves an increase in convergence speed and sample efficiency compared to on-policy algorithms and is competitive with existing off-policy policy gradient methods while being *significantly* simpler to implement. The effectiveness of GumbelClip is demonstrated against existing on-policy and off-policy actor-critic algorithms on a subset of the Atari domain.

1 INTRODUCTION

Recent advances in reinforcement learning (RL) have enabled the extension of long-standing methods to complex and large-scale tasks such as Atari (Mnih et al., 2015), Go (Silver et al., 2016), and DOTA (OpenAI, 2018). 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) (LeCun et al., 2015; Mnih et al., 2015). 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 (Sutton et al., 1998).

The most famous off-policy method is *Q*-Learning (Watkins & Dayan, 1992) 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. (2015) and used various modifications, such as experience replay, for stable convergence. Within DQN, experience replay (Lin, 1992) is often motivated as a technique for reducing sample correlation.

Unfortunately, all 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 is costly for large action spaces. To overcome these limitations, one could use policy gradient algorithms (Sutton et al., 2000), such as actor-critic methods, 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 (Degris et al., 2012) or by combining several techniques together, as in ACER (Wang et al., 2016). However, the resulting methods are quite complex and require many modifications to existing algorithms.

This paper, proposes a set of adjustments to A2C (Mnih et al., 2016), a parallel on-policy actor-critic algorithm, enabling off-policy learning from stored trajectories. Therefore, our contributions are as follows:

- GumbelClip, a fully *off-policy* actor-critic algorithm, the result of a small set of simple adjustments, in under 10 lines of code (LOC)[1], to the A2C algorithm.
- GumbelClip has increased sample efficiency and overall performance over *on-policy* actor-critic algorithms, such as A2C.
- GumbelClip performs similarily to other *off-policy* actor-critic algorithms, such as ACER, while being *significantly* simpler to implement.

The paper is organized as follows: Section 2 covers background information, Section 3 describes the GumbelClip algorithm, Section 4 details the experiments along with results, discussion, and ablations of our methodology. Section 5 discusses possible future work, and finally Section 6 provides concluding remarks.

2 Background and Notation

2.1 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 $P : \mathcal{S} \times \mathcal{A} \to (\mathcal{S} \to [0, 1])$, and a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Within this work, discrete actions and time steps are assumed. A *policy* is a probability distribution over actions conditioned on states, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$.

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 $Q^\pi(s_t, a_t) = \mathbb{E}_{a\sim\pi}[G_t|S_t = s, A_t = a]$.

2.2 Policy gradient methods

To optimize the parameters θ of the stochastic policy π the policy gradient theorem (Sutton et al., 2000) is used, which provides an expression for the gradient of the discounted reward objectives with respect to parameter θ. Therefore, the parameters θ of the differentiable stochastic policy $\pi_\theta(a_t|s_t)$ are updated as:

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

where $\Psi^\pi(s_t, a_t)$, as shown by Schulman et al. (2015b), can be replaced with quantities such as: the total reward of the trajectory, the TD 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.

2.3 Gumbel-Softmax Distribution

The Gumbel-Softmax distribution (GSD) (Jang et al., 2016) is a continuous distribution used to approximate samples from a categorical distribution. The GSD uses standard Gumbel noise to sample directly from the softmax distribution. On its own, the Gumbel distribution is typically used to model the maximum of a set of independent samples. Given categorical class probabilities $\pi_1, \pi_2, ..., \pi_k$ the Gumbel-Softmax distribution is defined as:

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \qquad \text{for } i = 1, ..., k \tag{2}$$

[1] Assuming code for replay memory is available.

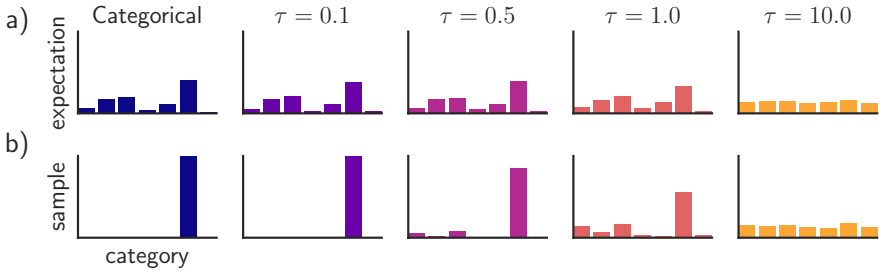


Figure 1: *Gumbel-Softmax distribution*: The temperature parameter τ controls the interpolation between discrete one-hot-encoded categorical distributions and continuous categorical densities. As $\tau \to 0$ samples from the Gumbel-Softmax distribution are identical to those from a categorical distribution. If $\tau \to \infty$ then the samples tend towards a uniform distribution. At $\tau = 1$ the samples wash out the smaller values and accentuate high density areas. *Figure used with permission of Jang et al. (2016).*

where $g_1 ... g_k$ are i.i.d. samples drawn from Gumbel$(0, 1)$ and τ is a temperature hyperparameter. The temperature hyperparameter τ, interpolates between a one-hot-encoded categorical distribution and a continuous categorical distribution. In this work, $\tau = 1$. Additional choices of τ and their effects are shown in Figure 1.

2.4 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* (Degris et al., 2012; Meuleau et al., 2000; Jie & Abbeel, 2010; Levine & Koltun, 2013).

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 $\mathcal{B}(a|s)$:

$$\rho_t = \frac{\pi(a_t|s_t)}{\mathcal{B}(a_t|s_t)} \tag{4}$$

Unfortunately, the importance weighted gradient in Equation 3 suffers from high variance. To reduce variance, Wawrzyński (2009) proposed truncating each importance weight to the interval $[0, c]$ where c is some constant.

3 GumbelClip

GumbelClip builds off the *on-policy* actor-critic algorithm A2C. To enable *off-policy* learning GumbelClip uses clipped importance sampling, policy forcing through Gumbel noise, and large batches sampled from a replay memory. Psuedo-code for GumbelClip is provided in Algorithm 1. We begin by defining a few quantities.

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^*:

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

GumbelClip introduces a third policy, the forced policy $\mathcal{F}(a|s)$ using the Gumbel-Softmax distribution, which results from adding noise $\epsilon^{(i)}$ sampled from a standard Gumbel distribution to the

normalized logits of the current policy:

$$\mathcal{F}(a_t^{(i)}|s_t) = \frac{\exp\left(\log \pi(a_t^{(i)}|s_t;\theta) + \epsilon^{(i)}\right)}{\sum_{j=0} \exp\left(\log \pi(a_t^{(j)}|s_t;\theta) + \epsilon^{(j)}\right)} \tag{6}$$

As the name implies, adding Gumbel noise has the effect of "forcing" the sampled policy distribution to be more categorical such that one action contains most of the probability mass. As Equation 6 is identical to the Gumbel-Softmax distribution, with temperature $\tau = 1$, we can refer to Figure 1b) to understand the characteristics of the resulting sampled distribution.

Algorithm 1 Pseudo-code for GumbelClip

Initialize parameters θ and θ_v.
Initialize replay memory $\mathcal{D}$ 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 $\mathcal{D}$.
  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 $\mathcal{D}$.
  for $i \in \{0, \cdots, k\}$ **do**
    Compute $\pi(\cdot|s_i;\theta)$ and $\mathcal{F}(\cdot|s_i)$.
    $\bar{\rho}_i = \text{clamp}\left(\frac{\mathcal{F}(a_i|s_i)}{\mathcal{B}(a_i|s_i)}, 0, c\right)$
  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.

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

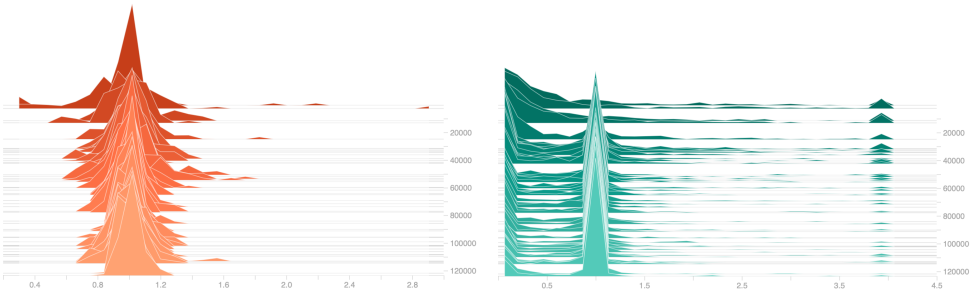
$$\rho_t^{(i)} = \frac{\mathcal{F}(a_t^{(i)}|s_t)}{\mathcal{B}(a_t^{(i)}|s_t)} \tag{7}$$

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 (Wawrzyński, 2009). Putting this all together the update equation for GumbelClip is as follows:

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

The gradient is estimated by uniformly sampling b trajectories of length k from a replay memory with size N. In addition, the network parameters are updated using the forced policy $\mathcal{F}(a|s)$. 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.

As the forced policy in the numerator tends towards a categorical distribution, it became evident that the importance weights had the habit of clumping near the boundaries of the interval and often near ~1. Following from Figure 2, we can see the distribution of the ratio between GumbelClip and a typical application of truncated importance sampling to A2C as might be suggested by Wawrzyński (2009). Figure 2(a) shows this behaviour if one were to implement truncated importance sampling



(a) Truncated ρ with current policy (b) Truncated ρ with forced policy

Figure 2: Histograms of ratios. The x-axis is the magnitude of $\bar{\rho}$ and the y-axis is the number of updates since start of training. a) Using truncated ρ with the current policy in the numerator and b) the ratios seen during training from GumbelClip, which corresponds to use of the forced policy in the numerator.

under the current policy $\pi(a|s;\theta)$; while Figure 2(b) shows the resulting distribution if one uses the forced policy $\mathcal{F}(a|s)$.

The effect of clipping and Gumbel noise has an interesting effect on the way the policy is updated. From Figure 2(b), we see that three modes exist, which roughly correspond to the cases of agreement and disagreement. 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 forced 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.

4 Experiments

Our experiments focus on the Atari domain (Bellemare et al., 2013) 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. (2016) was used to conduct all the experiments. All experiments used the same algorithm, network architecture, and hyper-parameters to learn to play Atari games using only raw pixel observations. This study used the same input pre-processing and network architecture as Mnih et al. (2016). 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. All experiments trained for 40 million frames. 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. The optimization procedure used RMSProp (Hinton et al., 2012) with a learning rate of $5e-4$, entropy regularization of 0.01, and a discount factor of $\gamma = 0.99$. We sample 64 trajectories of length 5 for each update. Learning begins after we have collected $10,000$ samples in the replay memory. We tuned the hyperparameters and developed GumbelClip on the *FishingDerby* environment only; the other environments can be considered "out of sample". All experiments used the same hyperparameter settings and network architecture. We use the best 3 of 4 seeds and report the mean value with 1 standard deviation as shaded areas on all graphs. As GumbelClip uses larger batchsizes we see an improvement in total run time per seed of 6-7 hours over A2C which takes 9-10 hours per seed.

Due to limited computational resources, we were only able to evaluate GumbelClip 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 (ACER) and on-policy (A2C)

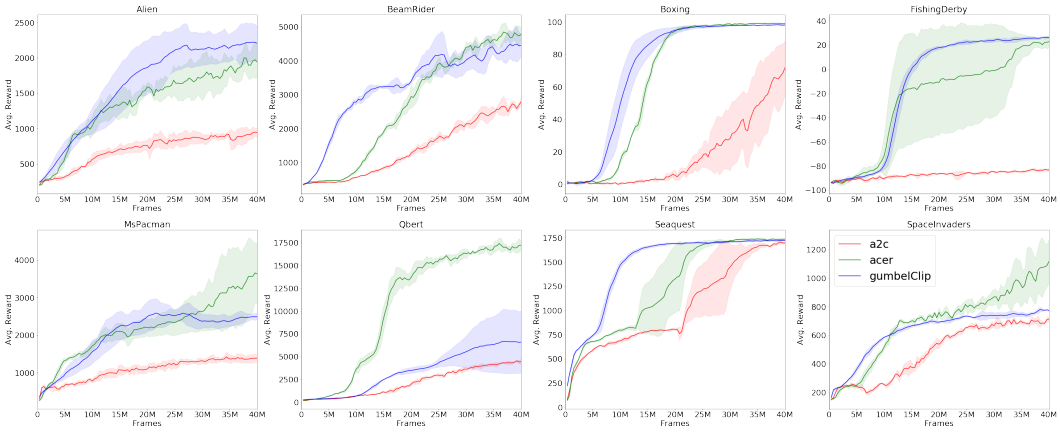


Figure 3: Training performance across 8 Atari games. We see the performance of GumbelClip (shown in blue) against ACER (shown in green), an off-policy actor-critic algorithm, and the on-policy algorithm A2C (shown in red). The graphs show the average performance over 3 seeds with 1 standard deviation shown as the shaded region. GumbelClip matches or exceeds the performance of the on-policy algorithm on all environments shown; while in all cases achieving improved convergence speed. It also shows a respectable performance when compared to the ACER algorithm on many of the environments.

actor-critic methods. We note that the performance can be expected to improve with a larger replay memory, as seen with DQN and other methods using replay memory. Additionally, we focused the examination of our ablations on the *Alien* environment to reduce computational requirements.

GumbelClip is based on a modified version of the open-source A2C implementation by Kostrikov (2018). The present work was compared with an off-policy actor-critic algorithm, ACER (Wang et al., 2016), and an on-policy actor-critic algorithm, A2C, the synchronous version of A3C (Mnih et al., 2016). Both baseline models, A2C and ACER, used the *baselines* package provided by OpenAI (Dhariwal et al., 2017). ACER used a replay ratio of 4, trust region updates (Schulman et al., 2015a), a discount factor of $\gamma = 0.99$, entropy regularization of 0.01, updated every 20 steps, and a 50000 capacity replay memory per thread.

4.1 Experimental Results

To test the proposed methodology, the performance of GumbelClip 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.

From Figure 3, we see the performance of GumbelClip, shown in blue, in comparison to the *on-policy* ACER algorithm, shown in green, and the *off-policy* A2C algorithm. We see that the use of replay memory and learning with *off-policy* samples significantly improves the sample efficiency of GumbelClip over A2C. Across the board, we see that GumbelClip converges significantly faster than A2C while also exceeding A2C's performance.

We achieve similar sample efficiency between the *off-policy* actor-critics, GumbelClip and ACER, across each environment. The additive noise and aggressive clamping seem to have both positive and negative effects. We see that GumbelClip sees a faster initial increase in performance across almost all environments, even outperforming ACER in some cases. However, clamping has been noted by Wang et al. (2016) to have the possibility of introducing bias. We hypothesize this might be the cause of poor performance on the *Qbert* environment, which while better than A2C, is much lower when compared to ACER. The benefit of GumbelClip comes from its simplicity, requiring a small number of easy changes to the A2C algorithm, while still providing better performance than other *on-policy* actor-critic algorithms.

4.2 WILL OTHER DISTRIBUTIONS WORK FOR NOISE GENERATION?

Here we examine the distribution used to draw the additive noise $\epsilon^{(i)}$ for use in the forced policy $\mathcal{F}(a|s)$. We compare the performance of noise sampled from the standard Gumbel distribution to that of the standard Normal distribution and the Uniform distribution $[0, 1]$.

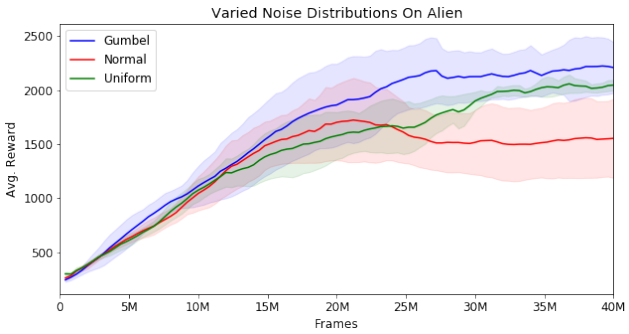


Figure 4: Variations in distributions used to sample additive noise. We compare the impact of noise drawn from the standard Gumbel distribution to the standard Normal distribution and Uniform distribution between $[0, 1]$.

The experiments only adjust the sampling distribution with no other parameter changes. We investigate the performance on the *Alien* Atari game over 40 million frames. From the results in Figure 4, 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. From Figure 4, we see that the Gumbel distribution is the most performant.

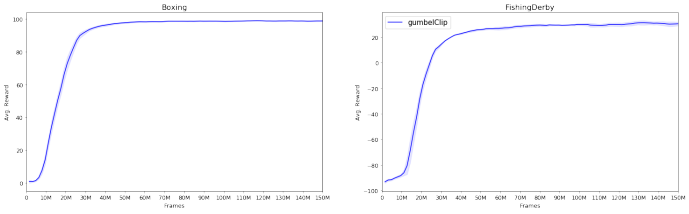


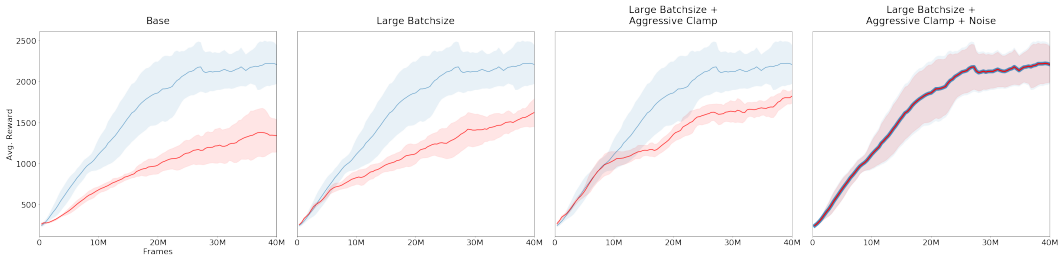
Figure 5: Stability of GumbelClip with extended training time. GumbelClip is trained for 150M frames, ~$4x$ longer, on the *Boxing* and *FishingDerby* environments. We see that GumbelClip experiences little to no oscillations in performance even with continued weight updates.

4.3 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 GumbelClip 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. From the rather uninteresting graphs, shown in Figure 5, we see that GumbelClip 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.

4.4 ABLATIONS

In our final set of experiments, we performed ablations over the various components of GumbelClip to tease apart the cause of improvements in performance. The results of the ablation are shown in Figure 6(a) with the complete GumbelClip algorithm in blue and each stripped version of GumbelClip as a red curve. We start with a *base* version, which is the application of truncated importance sampling and replay memory to the *off-policy* algorithm A2C; this base version is shown in the left-most panel in Figure 6(a). From Table 6(b) we see that the *base* version has a performance -39.39%



(a)

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

(b)

Figure 6: Ablations of GumbelClip. We gradually introduce each component of GumbelClip to a *base* version of an *off-policy* actor-critic A2C algorithm. a) The full GumbelClip 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 GumbelClip algorithm, shown in the last pane. The lines in the last pane are identical with one graph is stylized. b) The table provides the percent deltas between either the stripped version to GumbelClip or the current model to the last. We measure the change between the last 100 episodes.

lower than GumbelClip. The simple addition of a larger batchsize, from 16 sampled trajectories, to 64 as shown in the second panel in Figure 6(a) causes an increase of $+21.30\%$ over the *base* version and narrows the difference to GumbelClip 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 from the Gumbel distribution closes the gap with a final increase of $+21.09\%$. The addition of noise changes Equation 8 from being in terms of $\pi(a|s)$, in both the policy being optimized and the numerator in $\bar{\rho}$, to $\mathcal{F}(a|s)$. It is clearly shown that the *large batchsize* and *additive noise* contribute the most to the performance increase between stripped versions. Additionally, while difficult to quantify, we can see from the plots that the *aggressive clamp* and *additive noise* improve sample efficiency the most.

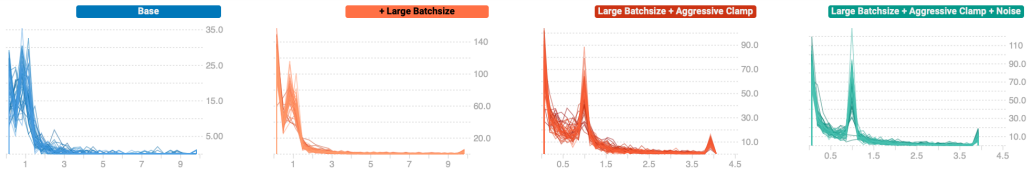


Figure 7: Change in distributions over $\bar{\rho}$ as components are added, shown as histograms. The x-axis corresponds to the magnitude of $\bar{\rho}$ and the y-axis is the number of occurrences. From left to right we see the distribution over $\bar{\rho}$ when using the *base* version. As components are added we see three modes, at approximately 0, 1, and c, become more pronounced. The addition of Gumbel noise increases the smoothness between $0 \rightarrow 1$, creates a fatter tail from the mode $1 \rightarrow c$, and increases the density of all modes.

From Figure 7, we see the effect that each addition to the *base* version has on the ratio $\bar{\rho}$ and therefore the updates to the network. In the last two panes of Figure 7, we see three clear modes at 0, 1, and c. Addition of Gumbel noise increases the density across all modes and reduces the "noise" seen from $0 \rightarrow 1$. The modes, as discussed in Section 3, correspond to "agreement", at 1, and "disagreement" at $\{0, c\}$.

5 FUTURE WORK

As GumbelClip relies heavily on the Gumbel distribution to sample noise from, it is best suited to environments with discrete actions. Therefore, an interesting avenue for future work would be to find a suitable distribution to sample noise from that works with continuous actions.

Additionally, further investigation into possible annealing schedules for a temperature hyperparameter, in Equation 6, or around the clamping constant c could yield interesting results.

6 CONCLUSION

In this paper we have presented GumbelClip, a set of adjustments to the *on-policy* A2C algorithm, enabling full *off-policy* learning from stored trajectories in a replay memory. Our approach relies on aggressive clipping of the importance weight, large batchsize, and additive noise sampled from the Gumbel distribution. We have empirically validated the use of each component in GumbelClip through ablations and shown the stability of the algorithm.

Furthermore, we have shown that GumbelClip achieves superior performance and higher sample efficiency than A2C. GumbelClip nears the performance and sample efficiency of ACER on many of the tested environments. Our methodology requires minimal changes to the A2C algorithm, which in contrast to ACER, makes the implementation of GumbelClip straightforward.

REFERENCES

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. *GitHub, GitHub repository*, 2017.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 2012.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Tang Jie and Pieter Abbeel. On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems*, pp. 1000–1008, 2010.

Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pp. 1–9, 2013.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Nicolas Meuleau, Leonid Peshkin, Leslie P Kaelbling, and Kee-Eung Kim. Off-policy policy search. 2000.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Paweł Wawrzyński. Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks*, 22(10):1484–1497, 2009.