Mean Actor Critic

Kavosh Asadi[1*] **Cameron Allen**[1*] **Melrose Roderick**[1] **Abdel-rahman Mohamed**[2†]
George Konidaris[1] **Michael Littman**[1]

Brown University[1]
Providence, RI

Amazon[2]
Seattle, WA

Abstract

We propose a new algorithm, Mean Actor-Critic (MAC), for discrete-action continuous-state reinforcement learning. MAC is a policy gradient algorithm that uses the agent's explicit representation of all action values to estimate the gradient of the policy, rather than using only the actions that were actually executed. This significantly reduces variance in the gradient updates and removes the need for a variance reduction baseline. We show empirical results on two control domains where MAC performs as well as or better than other policy gradient approaches, and on five Atari games, where MAC is competitive with state-of-the-art policy search algorithms.

1 Introduction

Reinforcement learning (RL) algorithms generally fall into one of two categories: value-function-based methods and policy search methods. Value-function-based methods maintain an estimate of the value of performing each action in each state, and choose the actions associated with the most value in their current state (Sutton & Barto, 1998). By contrast, policy search algorithms maintain an explicit policy, and agents draw actions directly from that policy to interact with their environment (Sutton et al., 2000). A subset of policy search algorithms, policy gradient methods, represent the policy using a differentiable parameterized function approximator (for example, a neural network) and use stochastic gradient ascent to update its parameters to achieve more reward.

To facilitate gradient ascent, the agent interacts with its environment according to the current policy and keeps track of the outcomes of its actions. From these (potentially noisy) sampled outcomes, the agent estimates the gradient of the objective function. A critical question here is how to compute an accurate gradient using these samples, which may be costly to acquire, while using as few sample interactions as possible.

Actor-critic algorithms compute the policy gradient using a learned value function to estimate expected future reward (Sutton et al., 2000; Konda & Tsitsiklis, 2000). Since the expected reward is a function of the environment's dynamics, which the agent does not know, it is typically estimated by executing the policy in the environment. Existing algorithms compute the policy gradient using the value of states the agent visits, and critically, these methods take into account only the actions the agent actually executes during environmental interaction.

We propose a new policy gradient algorithm, Mean Actor-Critic (or MAC), for the discrete-action continuous-state case. MAC uses the agent's policy distribution to *average the value function over all actions*, rather than sampling the action-value for actions it actually executed. This approach significantly reduces variance in the gradient updates and removes the need for

[*]These authors contributed equally to this work. Please send correspondence to Kavosh Asadi <kavosh@brown.edu> and Cameron Allen <csal@brown.edu>.

[†]This work was completed while at Microsoft Research.

an additional variance reduction term, called a baseline, often used in policy gradient methods (Williams, 1992; Sutton et al., 2000). We implement MAC using deep neural networks, and we show empirical results on two control domains, where MAC performs as well as or better than other policy gradient approaches, and on five Atari games, where MAC is competitive with state-of-the-art policy gradient and evolutionary policy search methods.

We note that the core idea behind MAC has also been independently and concurrently explored by Ciosek and Whiteson (2017).

2 Background

In RL, we train an agent to select actions in its environment so that it maximizes some notion of long-term reward. We formalize the problem as a Markov decision process (MDP) (Puterman, 1990), which we specify by the tuple $\langle \mathcal{S}, s_0, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$, where $\mathcal{S}$ is a set of states, $s_0 \in \mathcal{S}$ is a fixed initial state, $\mathcal{A}$ is a set of discrete actions, the functions $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ respectively describe the reward and transition dynamics of the environment, and $\gamma \in [0,1)$ is a discount factor representing the relative importance of immediate versus long-term rewards.

More concretely, we denote the expected reward for performing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ as:

$$\mathcal{R}(s,a) = \mathbb{E}\left[r_{t+1} \middle| s_t = s, a_t = a\right] ,$$

and we denote the probability that performing action a in state s results in state $s' \in \mathcal{S}$ as:

$$\mathcal{T}(s,a,s') = \Pr(s_{t+1} = s' \middle| s_t = s, a_t = a) .$$

In the context of policy search methods, the agent maintains an explicit policy $\pi(a|s;\theta)$ denoting the probability of taking action a in state s under the policy π parameterized by θ. Note that for each state, the policy outputs a probability distribution over the discrete set of actions: $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. At each timestep t, the agent takes an action a_t drawn from its policy $\pi(\cdot|s_t;\theta)$, then the environment provides a reward signal r_t and transitions to the next state s_{t+1}.

The agent's goal at every timestep is to maximize the sum of discounted future rewards, or simply *return*, which we define as:

$$G_t = \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} .$$

In a slight abuse of notation, we will also denote the total return for a trajectory τ as $G(\tau)$, which is equal to G_0 for that same trajectory.

The agent's policy induces a *value function* over the state space. The expression for return allows us to define both a state value function, $V^\pi(s)$, and a state-action value function, $Q^\pi(s,a)$. Here, $V^\pi(s)$ represents the expected return starting from state s, and following the policy π thereafter, and $Q^\pi(s,a)$ represents the expected return starting from s, executing action a, and then following the policy π thereafter:

$$V^\pi(s) = \mathop{\mathbb{E}}_{\pi}\left[G_t \middle| s_t = s\right] = \sum_{a \in \mathcal{A}} \left[\pi(a|s;\theta) Q^\pi(s,a)\right] ,$$

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\pi}\left[G_t \middle| s_t = s, a_t = a\right] .$$

The agent's goal is to find a policy that maximizes the return for every timestep, so we define an objective function J that allows us to score an arbitrary policy parameter θ:

$$J(\theta) \quad = \quad \mathop{\mathbb{E}}_{\tau \sim Pr(\tau|\theta)}[G(\tau)] = \sum_{\tau} Pr(\tau|\theta) G(\tau) ,$$

where τ denotes a trajectory. Note that the probability of a specific trajectory depends on policy parameters as well as the dynamics of the environment. Our goal is to be able to compute the gradient of J with respect to the policy parameters θ:

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \sum_\tau \nabla_\theta Pr(\tau|\theta) G(\tau) \\
&= \sum_\tau Pr(\tau|\theta) \frac{\nabla_\theta Pr(\tau|\theta)}{Pr(\tau|\theta)} G(\tau) \\
&= \sum_\tau Pr(\tau|\theta) \nabla_\theta \log Pr(\tau|\theta) G(\tau) \\
&= \mathop{\mathbb{E}}_{s\sim d^\pi,\, a\sim\pi} [\nabla_\theta \log \pi(a|s;\theta) G_0] \\
&= \mathop{\mathbb{E}}_{s\sim d^\pi,\, a\sim\pi} [\nabla_\theta \log \pi(a|s;\theta) G_t] \\
&= \mathop{\mathbb{E}}_{s\sim d^\pi,\, a\sim\pi} [\nabla_\theta \log \pi(a|s;\theta) Q^\pi(s,a)|s_0] \qquad (1)
\end{aligned}
$$

where $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t Pr(s_t = s|s_0, \pi)$ is the discounted state distribution. In the second and third lines we rewrite the summation using the log-derivative trick. In the fourth line, we convert the summation to an expectation, and use the G_0 notation in place of $G(\tau)$. Next, we make use of the fact that $\mathbb{E}[G_0] = \mathbb{E}[G_t]$, given by Williams (1992). Intuitively this makes sense, since the policy for a given state should depend only on the rewards achieved after that state. Finally, we invoke the definition that $Q^\pi(s,a) = \mathbb{E}[G_t]$.

A nice property of expectation (1) is that, given access to Q^π, the expectation can be estimated through implementing policy π in the environment. Alternatively, we can estimate Q^π using the return G_t, which is an unbiased (and usually a high variance) sample of Q^π. This is essentially the idea behind the REINFORCE algorithm (Williams, 1992), which uses the following gradient estimator:

$$
\nabla_\theta J(\theta) \approx \sum_{t=1}^{T} G_t \nabla_\theta \log \pi(a_t|s_t;\theta) \qquad (2)
$$

Alternatively, we can estimate Q^π using some sort of function approximation: $\widehat{Q}(s,a;\omega) \approx Q^\pi(s,a)$, which results in variants of actor-critic algorithms. Perhaps the simplest actor critic algorithm approximates (1) as follows:

$$
\nabla_\theta J(\theta) \approx \sum_{t=1}^{T} \widehat{Q}(s_t,a_t;w) \nabla_\theta \log \pi(a_t|s_t;\theta) \qquad (3)
$$

Note that value function approximation can, in general, bias the gradient estimation (Baxter & Bartlett, 2001).

One way of reducing variance in both REINFORCE and actor-critic algorithms is to use an additive control variate as a baseline (Williams, 1992; Sutton et al., 2000; Greensmith et al., 2004). A baseline is a function that is fixed over actions, and subtracting it from either the sampled returns or the estimated Q-values does not bias the gradient estimation. We refer to techniques that use such a baseline as *advantage* variations of the basic algorithms, since they approximate the advantage $A(s,a)$ of choosing action a over some baseline representing "typical" performance for the policy in state s. The update performed by advantage REINFORCE is:

$$
\theta \leftarrow \theta + \alpha \sum_{t=1}^{T} (G_t - b) \nabla_\theta \log \pi(a_t|s_t;\theta) \,,
$$

where b is a scalar baseline measuring the performance of the policy, such as a running average of the observed return over the past few episodes of interaction.

Advantage actor-critic uses an approximation of the expected value of each state s_t as its baseline: $\widehat{V}(s_t) = \sum_a \pi(a|s_t;\theta)\widehat{Q}(s_t,a;\omega)$, which leads to the following update rule:

$$
\theta \leftarrow \theta + \alpha \sum_{t=1}^{T} \big(\widehat{Q}(s_t,a_t;\omega) - \widehat{V}(s_t)\big) \nabla_\theta \log \pi(a_t|s_t;\theta) \,.
$$

Another way of estimating the advantage function is to use the TD-error signal $\delta = r_t + \gamma V(s') - V(s)$. This approach is convenient, because it only requires estimating one set of parameters, namely

for V. However, because the TD-error is a sample of the advantage function $A(s,a) = Q^\pi(s,a) - V^\pi(s)$, this approach has higher variance (due to the environmental dynamics) than methods that explicitly compute $Q(s,a) - V(s)$. Moreover, given Q and π, V can easily be computed as $V = \sum_a \pi(a|s)Q(s,a)$, so in practice, it is still only necessary to estimate one set of parameters (for Q).

3 Mean Actor-Critic

An overwhelming majority of recent actor-critic papers have computed the policy gradient using an estimate similar to Equation (3) (Degris et al. (2012); Mnih et al. (2016); Wang et al. (2016)). This estimate samples both states and actions from trajectories executed according to the current policy in order to compute the gradient of the objective function with respect to the policy weights.

Instead of using only the sampled actions, Mean Actor-Critic (MAC) explicitly computes the probability-weighted average over all Q-values, for each state sampled from the trajectories. This eliminates the need to use the log-derivative trick to remove preference for actions that have a higher probability under π. The result is an estimate of the gradient where the variance due to action sampling is reduced to zero. This is exactly the difference between computing the sample mean (whose variance is inversely proportional to the number of samples), and calculating the mean directly (which is simply a scalar with no variance).

MAC is based on the observation that expectation (1), which we repeat here, can be simplified as follows:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathop{\mathbb{E}}_{s\sim d^\pi,\, a\sim\pi}[\nabla_\theta \log \pi(a|s;\theta) Q^\pi(s,a)|s_0] \\
&= \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi(a|s;\theta)\nabla_\theta \log \pi(a|s;\theta) Q^\pi(s,a) \\
&= \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s;\theta) Q^\pi(s,a) \\
&= \mathop{\mathbb{E}}_{s\sim d^\pi(s)} \Big[\sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s;\theta) Q^\pi(s,a) \Big] \qquad (4)
\end{aligned}$$

We can estimate (4) by sampling states from a trajectory and using function approximation:

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T-1} \sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s_t;\theta) \widehat{Q}(s_t,a;\omega) \,.$$

In our implementation, the inner summation is computed by combining two neural networks that represent the policy and state-action value function. The value function can be learned using a variety of methods, such as temporal-difference learning or Monte Carlo sampling. After performing a few updates to the value function, we update the parameters θ of the policy with the following update rule:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T-1} \sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s_t;\theta) \widehat{Q}(s_t,a;\omega) \qquad (5)$$

To improve stability, repeated updates to the value and policy networks are interleaved, as in Generalized Policy Iteration (Sutton & Barto, 1998).

In traditional actor-critic approaches, which we refer to as *sampled-action* actor-critic, the only actions involved in the computation of the policy gradient estimate are those that were actually executed in the environment. In MAC, computing the policy gradient estimate will frequently involve actions that were not actually executed in the environment. This results in a trade-off between bias and variance. In domains where we can expect accurate Q-value predictions from our function approximator, despite not actually executing all of the relevant state-action pairs, MAC results in lower variance gradient updates and increased sample-efficiency. In domains where this assumption is not valid, MAC may perform worse than sampled-action actor-critic, though such domains also tend to be pathological for function-approximation in general.

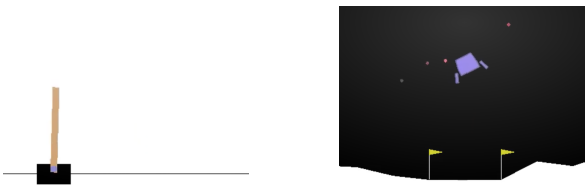

Figure 1: Screenshots of the classic control domains: Cart Pole (left) and Lunar Lander (right)

In some ways, MAC is similar to Expected Sarsa (Van Seijen et al., 2009). Expected Sarsa considers all next-actions a_{t+1}, then computes the expected TD-error, $\mathbb{E}[\delta] = r_t + \gamma\,\mathbb{E}[Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)$, and uses the resulting error signal to update the Q function. By contrast, MAC considers all current-actions a_t, and uses the corresponding $Q(s_t, a_t)$ values to update the policy directly.

It is natural to consider whether MAC could be improved by subtracting an action-independent baseline, as in sampled-action actor-critic and REINFORCE:

$$\nabla_\theta J(\theta) \quad = \quad \mathop{\mathbb{E}}_{s\sim d^\pi(s)} \Big[\sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s;\theta) \Big(Q^\pi(s,a) - V^\pi(s) \Big) \Big]$$

However, if we simplify the expectation as follows,

$$\nabla_\theta J(\theta) \quad = \quad \mathop{\mathbb{E}}_{s\sim d^\pi(s)} \Big[\sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s;\theta) Q^\pi(s,a) - V^\pi(s) \nabla_\theta \sum_{a\in\mathcal{A}} \pi(a|s;\theta) \Big]$$

we see that both $V^\pi(s)$ and the gradient operator can be moved outside of the summation, leaving just the sum of the action probabilities, which is always 1, and hence the gradient of the baseline term is always zero. This is true regardless of the choice of baseline, since the baseline cannot be a function of the actions or else it will bias the expectation. Thus, we see that subtracting a baseline is unnecessary in MAC, since it has no effect on the policy gradient estimate.

4 Experiments

This section presents an empirical evaluation of MAC across three different problem domains. We first evaluate the performance of MAC versus popular policy gradient benchmarks on two classic control problems. We then evaluate MAC on a subset of Atari 2600 games and investigate its performance compared to state-of-the-art policy search methods.

4.1 Classic Control Experiments

In order to determine whether MAC's lower variance policy gradient estimate translates to faster learning, we chose two classic control problems, namely Cart Pole and Lunar Lander, and compared MAC's performance against four standard sampled-action policy gradient algorithms. We used the open-source implementations of Cart Pole and Lunar Lander provided by OpenAI Gym (Brockman et al., 2016), in which both domains have continuous state spaces and discrete action spaces. Screenshots of the two domains are provided in Figure 1.

For each problem domain, we implemented MAC using two independent neural networks, representing the policy and Q function. We then performed a hyperparameter search to determine the best network architectures, optimization method, and learning rates. Specifically, the hyperparameter search considered: 0, 1, 2, or 3 hidden layers; 50, 75, 100, or 300 neurons per layer; ReLU, Leaky ReLU (with leak factor 0.3), or tanh activation; SGD, RMSProp, Adam, or Adadelta as the optimization method; and a learning rate chosen from 0.0001, 0.00025, 0.0005, 0.001, 0.005, 0.01, or 0.05. To find the best setting, we ran 10 independent trials for each combination of hyperparameters and chose the setting with the best asymptotic performance over the 10 trials. We terminated each episode after 200 and 1000 timesteps (in Cart Pole and Lunar Lander, respectively), regardless of the state of the agent.

We compared MAC against four standard benchmarks: REINFORCE, advantage REINFORCE, actor-critic, and advantage actor-critic. We implemented the REINFORCE benchmarks using just a

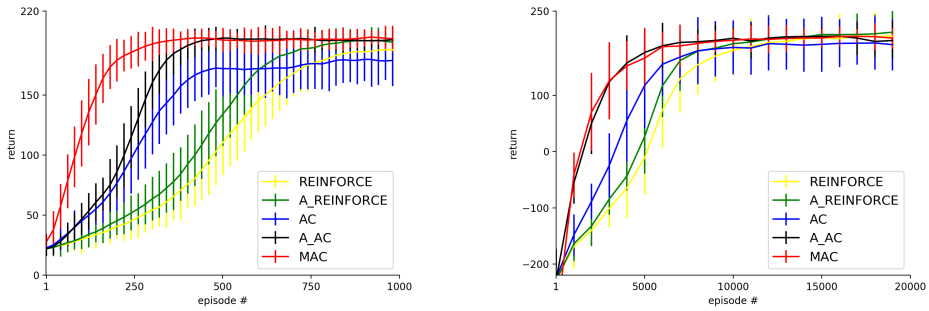

Figure 2: Performance comparison for CartPole (left) and Lunar Lander (right) of MAC vs. sampled-action policy gradient algorithms. Results are averaged over 100 independent trials.

Algorithm	**Cart Pole**	**Lunar Lander**
REINFORCE	109.5 ± 13.3	101.1 ± 10.5
Advantage REINFORCE	121.8 ± 11.2	114.7 ± 8.1
Actor-Critic	138.7 ± 13.2	124.6 ± 5.1
Advantage Actor-Critic	157.4 ± 6.4	162.8 ± 14.9
MAC	$\mathbf{178.3 \pm 7.6}$	$\mathbf{163.5 \pm 12.8}$

Table 1: Performance summary of MAC vs. sampled-action policy gradient algorithms. Scores denote the mean performance of each algorithm over all trials and episodes.

single neural network to represent the policy, and we implemented the actor-critic benchmarks using two networks to represent both the policy and Q function. For each benchmark algorithm, we then performed the same hyperparameter search that we had used for MAC.

In order to keep the variance as low as possible for the advantage actor-critic benchmark, we explicitly computed the advantage function $A(s,a) = Q(s,a) - V(s)$, where $V(s) = \sum_a \pi(a|s)Q(s,a)$, rather than sampling it using the TD-error (see Section 2).

Once we had determined the best hyperparameter settings for MAC and each of the benchmark algorithms, we then ran each algorithm for 100 independent trials. Figure 2 shows learning curves for the different algorithms, and Table 1 summarizes the results using the mean performance over trials and episodes. On Cart Pole, MAC learns substantially faster than all of the benchmarks, and on Lunar Lander, it performs competitively with the best benchmark algorithm, advantage actor-critic.

4.2 Atari Experiments

To test whether MAC can scale to larger problem domains, we evaluated it on several Atari 2600 games using the Arcade Learning Environment (ALE) (Bellemare et al., 2013) and compared MAC's performance against that of state-of-the-art policy search methods, namely, Trust Region Policy Optimization (TRPO) (Schulman et al., 2015), Evolutionary Strategies (ES) (Salimans et al., 2017), and Asyncronous Advantage Actor Critic (A3C) (Mnih et al., 2016). Due to the computational load inherent in training deep networks to play Atari games, we limited our experiments to a subset of five Atari games: Beamrider, Breakout, Pong, Seaquest and Space Invaders. These games were chosen as they are the five most common Atari games for tuning hyperparameters (Mnih et al., 2015, 2016).

The value network architecture was exactly the same as DeepMind's Deep Q-Network (DQN) (Mnih et al., 2015), and the policy network contained an additional fully-connected layer of 512 neurons with ReLU activation, followed by a linear layer and a softmax output layer. The policy and value networks used shared weights for the first four "feature" layers.

Table 2: Atari performance of MAC vs. policy search methods (random start condition). MAC vs. Best calculated as $100 \times (\text{MAC} - \text{Random}) / (\text{Best} - \text{Random})$.

Game	Random	TRPO (single)	TRPO (vine)	ES (1-hour)	**MAC**	MAC vs. Best
Beam Rider	363.9	**1425.2**	859.5	744.0	761.3	37.5%
Breakout	1.7	10.8	34.2	9.5	**160.1***	**487.38%**
Pong	-20.7	20.9	20.9	**21.0**	20.2	98.1%
Seaquest	68.4	1908.6	788.4	1390.0	**9445.3**	**509.6%**
Space Invaders	148.0	568.4	450.2	678.5	**2229.0**	**392.3%**
					Average Performance vs. Best:	**304.95%**

We trained the value network and feature layers using RMSProp and a learning rate of 0.0005, keeping all other hyperparameters the same as DQN, and using the same experience replay buffer and target network. In place of choosing actions using DQN's ϵ-greedy strategy with respect to the Q-values for a given state s, we instead sampled an action from the policy network's output distribution for state s. To compute the value targets needed for Q-learning, we multiplied the policy distribution at the next state s' by the corresponding Q-values from the target network.

After every episode, we froze the value network and feature layers, and trained the policy network using RMSProp and a learning rate of 0.0005, following the update rule given by (5). Each policy update sampled 25 independent batches, each of 32 experience tuples (s, a, r, s'), drawn uniformly from only the most recent 10,000 experiences in the replay buffer.

For each game, we trained for 50 million frames, pausing every 250,000 frames to run 125,000 evaluation episodes. We compared the mean score from each evaluation and kept the best-performing network, effectively implementing early stopping. We then evaluated the best-performing network under two types of starting conditions, random starts and human starts, in order to compare with the results from previous methods. Under random starts, the game is initialized with randomly between 0 and 30 no-ops ALE actions (Mnih et al., 2015), and we evaluated the network for 30 episodes (stopping after a maximum time of five minutes per episode). Under human starts, the game is initialized with a random game state sampled from 3 minutes of human expert play (Mnih et al., 2016), and we evaluated the network for 100 episodes (stopping after a maximum time of 30 minutes per episode, including the human expert initialization). The agent's performance is evaluated only the agent's total score after the human start (not including the human's points). Note that we used a different human expert than that of A3C, as the game initializations from these runs were not publicly available.

When evaluating our network on Breakout, we noticed an issue with the training scheme that resulted in strange behavior. We used the standard training scheme for Atari (Mnih et al., 2015), which consists of evaluation periods of a fixed number of timesteps (not a fixed number of games) at regular intervals. A consequence is that it is possible to get lucky in the first game of the evaluation period and achieve a high score, but subsequently be unable to finish a second game (e.g. failing to release the ball). If this happens, the training process will record the average completed game score (i.e. the high score), even if it is much higher than the agent's true average performance. This occurred when training MAC, and we could not evaluate the resulting network, because it could not finish enough games. We wanted to know whether the behavior was due to MAC getting stuck or MAC simply not learning anything, so we added a small amount of noise to the policy ($\epsilon = 0.005$), only during evaluation, in order to avoid getting stuck. Results in Tables 2 and 3 that were obtained using this noisy policy are denoted with an asterisk.

For the random-start condition, we compared MAC against TRPO (both single and vine) (Schulman et al., 2015), as well as ES (Salimans et al., 2017). We found that MAC performed significantly better on three out of five games, and competitively on a fourth (See Table 2). On average, MAC was three times better than the best performing benchmark algorithm.

For the human-start condition, we compared MAC against the state-of-the-art A3C algorithm (Mnih et al., 2016). In this experiment, MAC was better on two out of five experiments and worse on three (see Table 3). MAC significantly outperformed A3C on Seaquest, leading to an average performance across all five games of about three and a half times that of A3C.

Table 3: Atari performance of MAC vs. A3C (human start condition)

Game	Random	A3C FF (1-day)	**MAC**	MAC vs. A3C
Beam Rider	363.9	**13235.9**	469.8	0.8%
Breakout	1.7	**551.6**	107.4*	19.2%
Pong	-20.7	11.4	**15.6**	**113.1%**
Seaquest	68.4	551.6	**7878.0**	**1616.2%**
Space Invaders	148.0	**2214.7**	1254.9	53.6%
			Average Performance vs. A3C:	**360.6%**

Note that the performance on Beam Rider was low relative to the other games. This was due to the algorithm falling into a local optimum where it always chose to go left while shooting. Such a strategy is locally optimal, and measurably better than random play, but it prevented our agent from finding a better policy. This behavior is typical of policy gradient algorithms, and we note that MAC's performance is still competitive with TRPO (vine) and ES. A3C is able to overcome this problem by exploring different parts of the search space in parallel, an approach not always feasible in practice.

5 Discussion

At its core, MAC offers a new way of computing the policy gradient which can substantially reduce variance and increase learning speed. There are a number of orthogonal improvements to policy gradient methods, such as using natural gradients (Kakade, 2002; Peters & Schaal, 2008), off-policy learning (Wang et al., 2016; Gu et al., 2016; Asadi & Williams, 2016), second-order methods (Furmston et al., 2016), and asynchronous exploration (Mnih et al., 2016). We have not investigated how MAC performs with these extensions; however, just as these improvements were added to basic actor-critic methods, they could be added to MAC as well, and we expect they would improve its performance in a similar way.

A typical use-case for actor-critic algorithms is for problem domains with continuous actions, which are awkward for value-function-based methods (Sutton & Barto, 1998). One approach to dealing with continuous actions is to use a deterministic policy (Silver et al., 2014; Lillicrap et al., 2015) and to perform off-policy policy gradient updates. However, in settings where on-policy learning is necessary, using a deterministic policy leads to a sub-optimal behavior (Sutton & Barto, 1998), and hence a stochastic policy is typically used instead. MAC uses a stochastic policy, but it was designed for discrete-action domains. Extending MAC to continuous actions would require changing MAC's sum over actions to an integral, an operation that is more complex. Such a modification will be the subject of future work.

6 Conclusion

The basic formulation of policy gradient estimators presented here—where the gradient is estimated by averaging the state-action value function across actions—leads to a new family of actor-critic algorithms. This family has the advantage of not requiring an additional variance-reduction baseline, substantially reducing the design effort required to apply them. It is also a natural fit with deep neural network function approximators, resulting in a network architecture that is end-to-end differentiable.

Our results show that the MAC algorithm (the simplest member of the resulting family), when combined with a neural network implementation, either outperforms, or is competitive with, state-of-the-art policy search algorithms that make similar assumptions. In future work, we aim to develop this family of algorithms further, first by including typical elaborations of the basic actor-critic architecture like natural or second-order gradients, and second by adding parallel searches (such as are conducted by A3C), which would help to avoid the local optima that are inherent in policy search and which in some cases cause MAC to perform badly. Our results so far suggest that our new approach is highly promising, and that extensions to it will provide even further improvement in performance.

References

Asadi, Kavosh and Williams, Jason D. Sample-efficient deep reinforcement learning for dialog control. *arXiv preprint arXiv:1612.06000*, 2016.

Baxter, Jonathan and Bartlett, Peter L. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

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

Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. Openai gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`.

Ciosek, Kamil and Whiteson, Shimon. Expected policy gradients. *arXiv preprint arXiv:1706.05374*, 2017.

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

Furmston, Thomas, Lever, Guy, and Barber, David. Approximate newton methods for policy search in markov decision processes. *Journal of Machine Learning Research*, 17(227):1–51, 2016.

Greensmith, Evan, Bartlett, Peter L, and Baxter, Jonathan. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004.

Gu, Shixiang, Lillicrap, Timothy, Ghahramani, Zoubin, Turner, Richard E, and Levine, Sergey. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016.

Kakade, Sham M. A natural policy gradient. In *Advances in neural information processing systems*, pp. 1531–1538, 2002.

Konda, Vijay R and Tsitsiklis, John N. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy P, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

Peters, Jan and Schaal, Stefan. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008.

Puterman, Martin L. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Salimans, Tim, Ho, Jonathan, Chen, Xi, and Sutskever, Ilya. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael, and Moritz, Philipp. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1889–1897, 2015.

Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395, 2014.

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

Sutton, R.S. and Barto, A.G. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

Van Seijen, Harm, Van Hasselt, Hado, Whiteson, Shimon, and Wiering, Marco. A theoretical and empirical analysis of expected sarsa. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, pp. 177–184. IEEE, 2009.

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

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.