
Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation

Yuhuai Wu*

University of Toronto
Vector Institute
ywu@cs.toronto.edu

Elman Mansimov*

New York University
mansimov@cs.nyu.edu

Shun Liao

University of Toronto
Vector Institute
slliao3@cs.toronto.edu

Roger Grosse

University of Toronto
Vector Institute
rgrosse@cs.toronto.edu

Jimmy Ba

University of Toronto
Vector Institute
jimmy@psi.utoronto.ca

Abstract

In this work, we propose to apply trust region optimization to deep reinforcement learning using a recently proposed Kronecker-factored approximation to the curvature. We extend the framework of natural policy gradient and propose to optimize both the actor and the critic using Kronecker-factored approximate curvature (K-FAC) with trust region; hence we call our method Actor Critic using Kronecker-Factored Trust Region (ACKTR). To the best of our knowledge, this is the first scalable trust region natural gradient method for actor-critic methods. It is also a method that learns non-trivial tasks in continuous control as well as discrete control policies directly from raw pixel inputs. We tested our approach across discrete domains in Atari games as well as continuous domains in the MuJoCo environment. With the proposed methods, we are able to achieve higher rewards and a 2- to 3-fold improvement in sample efficiency on average, compared to previous state-of-the-art on-policy actor-critic methods. Code is available at <https://github.com/openai/baselines>.

1 Introduction

Agents using deep reinforcement learning (deep RL) methods have shown tremendous success in learning complex behaviour skills and solving challenging control tasks in high-dimensional raw sensory state-space [25, 18, 13]. Deep RL methods make use of deep neural networks to represent control policies. Despite the impressive results, these neural networks are still trained using simple variants of stochastic gradient descent (SGD). SGD and related first-order methods explore weight space inefficiently. It often takes days for the current deep RL methods to master various continuous and discrete control tasks. Previously, a distributed approach was proposed [18] to reduce training time by executing multiple agents to interact with the environment simultaneously, but this leads to rapidly diminishing returns of sample efficiency as the degree of parallelism increases.

Sample efficiency is a dominant concern in RL; robotic interaction with the real world is typically scarcer than computation time, and even in simulated environments the cost of simulation often dominates that of the algorithm itself. One way to effectively reduce the sample size is to use more advanced optimization techniques for gradient updates. Natural policy gradient [11] uses the technique of natural gradient descent [1] to perform gradient updates. Natural gradient methods

*Equal contribution.

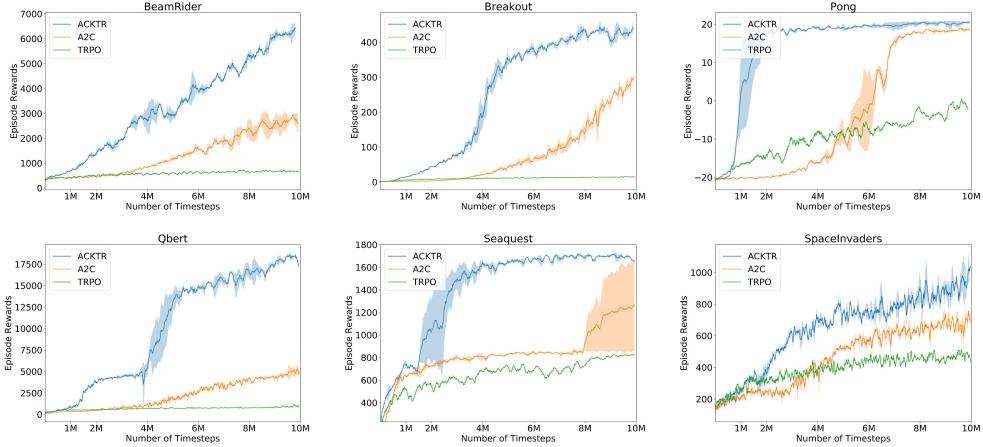


Figure 1: Performance comparisons on six standard Atari games trained for 10 million timesteps (1 timestep equals 4 frames). The shaded region denotes the standard deviation over 2 random seeds.

follow the steepest descent direction that uses the Fisher metric as the underlying metric, a metric that is based not on the choice of coordinates but rather on the manifold (i.e., the surface).

However, the exact computation of the natural gradient is intractable because it requires inverting the Fisher information matrix. Trust-region policy optimization (TRPO) [22] avoids explicitly storing and inverting the Fisher matrix by using Fisher-vector products [21]. However, it typically requires many steps of conjugate gradient to obtain a single parameter update, and accurately estimating the curvature requires a large number of samples in each batch; hence TRPO is impractical for large models and suffers from sample inefficiency.

Kronecker-factored approximated curvature (K-FAC) [16, 7] is a scalable approximation to natural gradient. It has been shown to speed up training of various state-of-the-art large-scale neural networks [2] in supervised learning by using larger mini-batches. Unlike TRPO, each update is comparable in cost to an SGD update, and it keeps a running average of curvature information, allowing it to use small batches. This suggests that applying K-FAC to policy optimization could improve the sample efficiency of the current deep RL methods.

In this paper, we introduce the actor-critic using Kronecker-factored trust region (ACKTR; pronounced ‘‘actor’’) method, a scalable trust-region optimization algorithm for actor-critic methods. The proposed algorithm uses a Kronecker-factored approximation to natural policy gradient that allows the covariance matrix of the gradient to be inverted efficiently. To best of our knowledge, we are also the first to extend the natural policy gradient algorithm to optimize value functions via Gauss–Newton approximation. In practice, the per-update computation cost of ACKTR is only 10% to 25% higher than SGD-based methods. Empirically, we show that ACKTR substantially improves both sample efficiency and the final performance of the agent in the Atari environments [4] and the MuJoCo [27] tasks compared to the state-of-the-art on-policy actor-critic method A2C [18] and the famous trust region optimizer TRPO [22].

We make our source code available online at <https://github.com/openai/baselines>.

2 Background

2.1 Reinforcement learning and actor-critic methods

We consider an agent interacting with an infinite-horizon, discounted Markov Decision Process $(\mathcal{X}, \mathcal{A}, \gamma, P, r)$. At time t , the agent chooses an action $a_t \in \mathcal{A}$ according to its policy $\pi_\theta(a|s_t)$ given its current state $s_t \in \mathcal{X}$. The environment in turn produces a reward $r(s_t, a_t)$ and transitions to the next state s_{t+1} according to the transition probability $P(s_{t+1}|s_t, a_t)$. The goal of the agent is to maximize the expected γ -discounted cumulative return $\mathcal{J}(\theta) = \mathbb{E}_\pi[R_t] = \mathbb{E}_\pi[\sum_{i \geq 0}^{\infty} \gamma^i r(s_{t+i}, a_{t+i})]$ with respect to the policy parameters θ . Policy gradient methods [30, 26] directly parameterize a policy $\pi_\theta(a|s_t)$ and update parameter θ so as to maximize the objective $\mathcal{J}(\theta)$. In its general form,

the policy gradient is defined as [23],

$$\nabla_{\theta} \mathcal{J}(\theta) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \Psi^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right],$$

where Ψ^t is often chosen to be the advantage function $A^{\pi}(s_t, a_t)$, which provides a relative measure of value of each action a_t at a given state s_t . There is an active line of research [23] on designing an advantage function that provides both low-variance and low-bias gradient estimates. As this is not the focus of our work, we simply follow the asynchronous advantage actor critic (A3C) method [18] and define the advantage function as the k -step returns with function approximation,

$$A^{\pi}(s_t, a_t) = \sum_{i=0}^{k-1} (\gamma^i r(s_{t+i}, a_{t+i}) + \gamma^k V_{\phi}^{\pi}(s_{t+k})) - V_{\phi}^{\pi}(s_t),$$

where $V_{\phi}^{\pi}(s_t)$ is the value network, which provides an estimate of the expected sum of rewards from the given state following policy π , $V_{\phi}^{\pi}(s_t) = \mathbb{E}_{\pi}[R_t]$. To train the parameters of the value network, we again follow [18] by performing temporal difference updates, so as to minimize the squared difference between the bootstrapped k -step returns \hat{R}_t and the prediction value $\frac{1}{2} \|\hat{R}_t - V_{\phi}^{\pi}(s_t)\|^2$.

2.2 Natural gradient using Kronecker-factored approximation

To minimize a nonconvex function $\mathcal{J}(\theta)$, the method of steepest descent calculates the update $\Delta\theta$ that minimizes $\mathcal{J}(\theta + \Delta\theta)$, subject to the constraint that $\|\Delta\theta\|_B < 1$, where $\|\cdot\|_B$ is the norm defined by $\|x\|_B = (x^T B x)^{\frac{1}{2}}$, and B is a positive semidefinite matrix. The solution to the constraint optimization problem has the form $\Delta\theta \propto -B^{-1}\nabla_{\theta}\mathcal{J}$, where $\nabla_{\theta}\mathcal{J}$ is the standard gradient. When the norm is Euclidean, i.e., $B = I$, this becomes the commonly used method of gradient descent. However, the Euclidean norm of the change depends on the parameterization θ . This is not favorable because the parameterization of the model is an arbitrary choice, and it should not affect the optimization trajectory. The method of natural gradient constructs the norm using the Fisher information matrix F , a local quadratic approximation to the KL divergence. This norm is independent of the model parameterization θ on the class of probability distributions, providing a more stable and effective update. However, since modern neural networks may contain millions of parameters, computing and storing the exact Fisher matrix and its inverse is impractical, so we have to resort to approximations.

A recently proposed technique called Kronecker-factored approximate curvature (K-FAC) [16] uses a Kronecker-factored approximation to the Fisher matrix to perform efficient approximate natural gradient updates. We let $p(y|x)$ denote the output distribution of a neural network, and $L = \log p(y|x)$ denote the log-likelihood. Let $W \in \mathbb{R}^{C_{out} \times C_{in}}$ be the weight matrix in the ℓ^{th} layer, where C_{out} and C_{in} are the number of output/input neurons of the layer. Denote the input activation vector to the layer as $a \in \mathbb{R}^{C_{in}}$, and the pre-activation vector for the next layer as $s = Wa$. Note that the weight gradient is given by $\nabla_W L = (\nabla_s L)a^T$. K-FAC utilizes this fact and further approximates the block F_{ℓ} corresponding to layer ℓ as \hat{F}_{ℓ} ,

$$\begin{aligned} F_{\ell} &= \mathbb{E}[\text{vec}\{\nabla_W L\} \text{vec}\{\nabla_W L\}^T] = \mathbb{E}[aa^T \otimes \nabla_s L (\nabla_s L)^T] \\ &\approx \mathbb{E}[aa^T] \otimes \mathbb{E}[\nabla_s L (\nabla_s L)^T] := A \otimes S := \hat{F}_{\ell}, \end{aligned}$$

where A denotes $\mathbb{E}[aa^T]$ and S denotes $\mathbb{E}[\nabla_s L (\nabla_s L)^T]$. This approximation can be interpreted as making the assumption that the second-order statistics of the activations and the backpropagated derivatives are uncorrelated. With this approximation, the natural gradient update can be efficiently computed by exploiting the basic identities $(P \otimes Q)^{-1} = P^{-1} \otimes Q^{-1}$ and $(P \otimes Q) \text{vec}(T) = PTQ^T$:

$$\text{vec}(\Delta W) = \hat{F}_{\ell}^{-1} \text{vec}\{\nabla_W \mathcal{J}\} = \text{vec}(A^{-1} \nabla_W \mathcal{J} S^{-1}).$$

From the above equation we see that the K-FAC approximate natural gradient update only requires computations on matrices comparable in size to W . Grosse and Martens [7] have recently extended the K-FAC algorithm to handle convolutional networks. Ba et al. [2] later developed a distributed version of the method where most of the overhead is mitigated through asynchronous computation. Distributed K-FAC achieved 2- to 3-times speed-ups in training large modern classification convolutional networks.

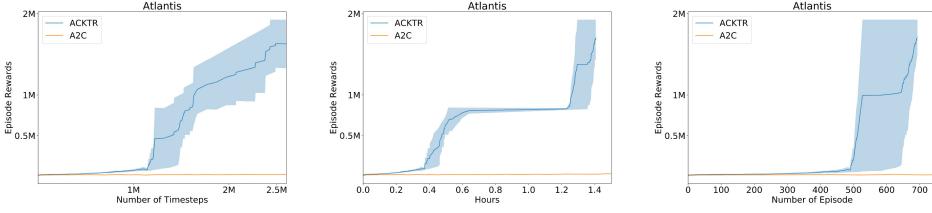


Figure 2: In the Atari game of Atlantis, our agent (ACKTR) quickly learns to obtain rewards of 2 million in 1.3 hours, 600 episodes of games, 2.5 million timesteps. The same result is achieved by advantage actor critic (A2C) in 10 hours, 6000 episodes, 25 million timesteps. ACKTR is 10 times more sample efficient than A2C on this game.

3 Methods

3.1 Natural gradient in actor-critic

Natural gradient was proposed to apply to the policy gradient method more than a decade ago by Kakade [11]. But there still doesn't exist a scalable, sample-efficient, and general-purpose instantiation of the natural policy gradient. In this section, we introduce the first scalable and sample-efficient natural gradient algorithm for actor-critic methods: the actor-critic using Kronecker-factored trust region (ACKTR) method. We use Kronecker-factored approximation to compute the natural gradient update, and apply the natural gradient update to both the actor and the critic.

To define the Fisher metric for reinforcement learning objectives, one natural choice is to use the policy function which defines a distribution over the action given the current state, and take the expectation over the trajectory distribution:

$$F = \mathbb{E}_{p(\tau)}[\nabla_{\theta} \log \pi(a_t|s_t)(\nabla_{\theta} \log \pi(a_t|s_t))^T],$$

where $p(\tau)$ is the distribution of trajectories, given by $p(s_0) \prod_{t=0}^T \pi(a_t|s_t)p(s_{t+1}|s_t, a_t)$. In practice, one approximates the intractable expectation over trajectories collected during training.

We now describe one way to apply natural gradient to optimize the critic. Learning the critic can be thought of as a least-squares function approximation problem, albeit one with a moving target. In the setting of least-squares function approximation, the second-order algorithm of choice is commonly Gauss-Newton, which approximates the curvature as the Gauss-Newton matrix $G := \mathbb{E}[J^T J]$, where J is the Jacobian of the mapping from parameters to outputs [19]. The Gauss-Newton matrix is equivalent to the Fisher matrix for a Gaussian observation model [15]; this equivalence allows us to apply K-FAC to the critic as well. Specifically, we assume the output of the critic v is defined to be a Gaussian distribution $p(v|s_t) \sim \mathcal{N}(v; V(s_t), \sigma^2)$. The Fisher matrix for the critic is defined with respect to this Gaussian output distribution. In practice, we can simply set σ to 1, which is equivalent to the vanilla Gauss-Newton method.

If the actor and critic are disjoint, one can separately apply K-FAC updates to each using the metrics defined above. But to avoid instability in training, it is often beneficial to use an architecture where the two networks share lower-layer representations but have distinct output layers [18, 28]. In this case, we can define the joint distribution of the policy and the value distribution by assuming independence of the two output distributions, i.e., $p(a, v|s) = \pi(a|s)p(v|s)$, and construct the Fisher metric with respect to $p(a, v|s)$, which is no different than the standard K-FAC except that we need to sample the networks' outputs independently. We can then apply K-FAC to approximate the Fisher matrix $\mathbb{E}_{p(\tau)}[\nabla \log p(a, v|s)\nabla \log p(a, v|s)^T]$ to perform updates simultaneously.

In addition, we use the factorized Tikhonov damping approach described by [16]. We also follow [2] and perform the asynchronous computation of second-order statistics and inverses required by the Kronecker approximation to reduce computation time.

3.2 Step-size Selection and trust-region optimization

Traditionally, natural gradient is performed with SGD-like updates, $\theta \leftarrow \theta - \eta F^{-1} \nabla_{\theta} L$. But in the context of deep RL, Schulman et al. [22] observed that such an update rule can result in large updates to the policy, causing the algorithm to prematurely converge to a near-deterministic policy. They advocate instead using a trust region approach, whereby the update is scaled down to modify the policy distribution (in terms of KL divergence) by at most a specified amount. Therefore, we adopt the trust region formulation of K-FAC introduced by [2], choosing the effective step size η to be $\min(\eta_{\max}, \sqrt{\frac{2\delta}{\Delta\theta^\top \hat{F} \Delta\theta}})$, where the learning rate η_{\max} and trust region radius δ are hyperparameters. If the actor and the critic are disjoint, then we need to tune a different set of η_{\max} and δ separately for both. The variance parameter for the critic output distribution can be absorbed into the learning rate parameter for vanilla Gauss-Newton. On the other hand, if they share representations, we need to tune one set of η_{\max} , δ , and also the weighting parameter of the training loss of the critic, with respect to that of the actor.

4 Related work

Natural gradient [1] was first applied to policy gradient methods by Kakade [11]. Bagnell and Schneider [3] further proved that the metric defined in [11] is a covariant metric induced by the path-distribution manifold. Peters and Schaal [20] then applied natural gradient to the actor-critic algorithm. They proposed performing natural policy gradient for the actor’s update and using a least-squares temporal difference (LSTD) method for the critic’s update. However, there are great computational challenges when applying natural gradient methods, mainly associated with efficiently storing the Fisher matrix as well as computing its inverse. For tractability, previous work restricted the method to using the compatible function approximator (a linear function approximator). To avoid the computational burden, Trust Region Policy Optimization (TRPO) [22] approximately solves the linear system using conjugate gradient with fast Fisher matrix-vector products, similar to the work of Martens [14]. This approach has two main shortcomings. First, it requires repeated computation of Fisher vector products, preventing it from scaling to the larger architectures typically used in experiments on learning from image observations in Atari and MuJoCo. Second, it requires a large batch of rollouts in order to accurately estimate curvature. K-FAC avoids both issues by using tractable Fisher matrix approximations and by keeping a running average of curvature statistics during training. Although TRPO shows better per-iteration progress than policy gradient methods trained with first-order optimizers such as Adam [12], it is generally less sample efficient.

Several methods were proposed to improve the computational efficiency of TRPO. To avoid repeated computation of Fisher-vector products, Wang et al. [28] solve the constrained optimization problem with a linear approximation of KL divergence between a running average of the policy network and the current policy network. Instead of the hard constraint imposed by the trust region optimizer, Heess et al. [9] and Schulman et al. [24] added a KL cost to the objective function as a soft constraint. Both papers show some improvement over vanilla policy gradient on continuous and discrete control tasks in terms of sample efficiency.

There are other recently introduced actor-critic models that improve sample efficiency by introducing experience replay [28], [8] or auxiliary objectives [10]. These approaches are orthogonal to our work, and could potentially be combined with ACKTR to further enhance sample efficiency.

5 Experiments

We conducted a series of experiments to investigate the following questions: (1) How does ACKTR compare with the state-of-the-art on-policy method and common second-order optimizer baseline in terms of sample efficiency and computational efficiency? (2) What makes a better norm for optimization of the critic? (3) How does the performance of ACKTR scale with batch size compared to the first-order method?

We evaluated our proposed method, ACKTR, on two standard benchmark platforms. We first evaluated it on the discrete control tasks defined in OpenAI Gym [5], simulated by Arcade Learning Environment [4], a simulator for Atari 2600 games which is commonly used as a deep reinforcement learning benchmark for discrete control. We then evaluated it on a variety of continuous control

Domain	Human level	ACKTR		A2C		TRPO (10 M)	
		Rewards	Episode	Rewards	Episode	Rewards	Episode
Beamrider	5775.0	13581.4	3279	8148.1	8930	670.0	N/A
Breakout	31.8	735.7	4094	581.6	14464	14.7	N/A
Pong	9.3	20.9	904	19.9	4768	-1.2	N/A
Q-bert	13455.0	21500.3	6422	15967.4	19168	971.8	N/A
Seaquest	20182.0	1776.0	N/A	1754.0	N/A	810.4	N/A
Space Invaders	1652.0	19723.0	14696	1757.2	N/A	465.1	N/A

Table 1: ACKTR and A2C results showing the last 100 average episode rewards attained after 50 million timesteps, and TRPO results after 10 million timesteps. The table also shows the episode N , where N denotes the first episode for which the mean episode reward over the N^{th} game to the $(N + 100)^{th}$ game crosses the human performance level [17], averaged over 2 random seeds.

benchmark tasks defined in OpenAI Gym [5], simulated by the MuJoCo [27] physics engine. Our baselines are (a) a synchronous and batched version of the asynchronous advantage actor critic model (A3C) [18], henceforth called A2C (advantage actor critic), and (b) TRPO [22]. ACKTR and the baselines use the same model architecture except for the TRPO baseline on Atari games, with which we are limited to using a smaller architecture because of the computing burden of running a conjugate gradient inner-loop. See the appendix for other experiment details.

5.1 Discrete control

We first present results on the standard six Atari 2600 games to measure the performance improvement obtained by ACKTR. The results on the six Atari games trained for 10 million timesteps are shown in Figure 1, with comparison to A2C and TRPO². ACKTR significantly outperformed A2C in terms of sample efficiency (i.e., speed of convergence per number of timesteps) by a significant margin in all games. We found that TRPO could only learn two games, Seaquest and Pong, in 10 million timesteps, and performed worse than A2C in terms of sample efficiency.

In Table 1 we present the mean of rewards of the last 100 episodes in training for 50 million timesteps, as well as the number of episodes required to achieve human performance [17]. Notably, on the games Beamrider, Breakout, Pong, and Q-bert, A2C required respectively 2.7, 3.5, 5.3, and 3.0 times more episodes than ACKTR to achieve human performance. In addition, one of the runs by A2C in Space Invaders failed to match human performance, whereas ACKTR achieved 19723 on average, 12 times better than human performance (1652). On the games Breakout, Q-bert and Beamrider, ACKTR achieved 26%, 35%, and 67% larger episode rewards than A2C.

We also evaluated ACKTR on the rest of the Atari games; see Appendix B for full results. We compared ACKTR with Q-learning methods, and we found that in 36 out of 44 benchmarks, ACKTR is on par with Q-learning methods in terms of sample efficiency, and consumed a lot less computation time. Remarkably, in the game of Atlantis, ACKTR quickly learned to obtain rewards of 2 million in 1.3 hours (600 episodes), as shown in Figure 2. It took A2C 10 hours (6000 episodes) to reach the same performance level.

5.2 Continuous control

We ran experiments on the standard benchmark of continuous control tasks defined in OpenAI Gym [5] simulated in MuJoCo [27], both from low-dimensional state-space representation and directly from pixels. In contrast to Atari, the continuous control tasks are sometimes more challenging due to high-dimensional action spaces and exploration. The results of eight MuJoCo environments trained for 1 million timesteps are shown in Figure 3. Our model significantly outperformed baselines on six out of eight MuJoCo tasks and performed competitively with A2C on the other two tasks (Walker2d and Swimmer).

We further evaluated ACKTR for 30 million timesteps on eight MuJoCo tasks and in Table 2 we present mean rewards of the top 10 consecutive episodes in training, as well as the number of

²The A2C and TRPO Atari baseline results are provided to us by the OpenAI team, <https://github.com/openai/baselines-results>.

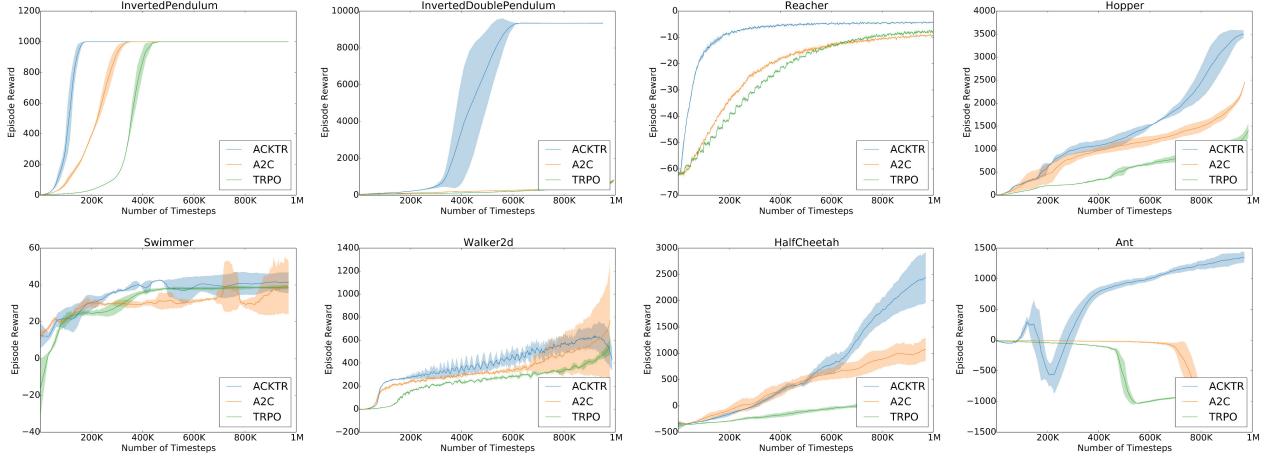


Figure 3: Performance comparisons on eight MuJoCo environments trained for 1 million timesteps (1 timestep equals 4 frames). The shaded region denotes the standard deviation over 3 random seeds.

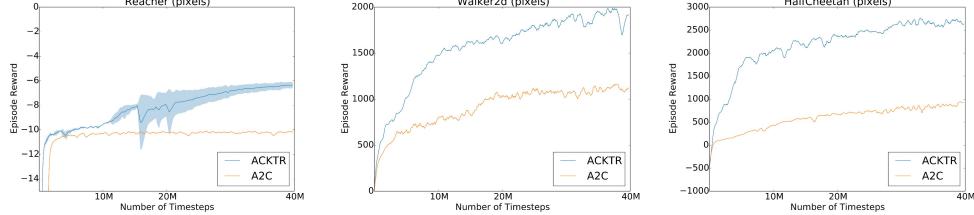


Figure 4: Performance comparisons on 3 MuJoCo environments from image observations trained for 40 million timesteps (1 timestep equals 4 frames).

episodes to reach a certain threshold defined in [8]. As shown in Table 2, ACKTR reaches the specified threshold faster on all tasks, except for Swimmer where TRPO achieves 4.1 times better sample efficiency. A particularly notable case is Ant, where ACKTR is 16.4 times more sample efficient than TRPO. As for the mean reward score, all three models achieve results comparable with each other with the exception of TRPO, which in the Walker2d environment achieves a 10% better reward score.

We also attempted to learn continuous control policies directly from pixels, without providing low-dimensional state space as an input. Learning continuous control policies from pixels is much more challenging than learning from the state space, partially due to the slower rendering time compared to Atari (0.5 seconds in MuJoCo vs 0.002 seconds in Atari). The state-of-the-art actor-critic method A3C [18] only reported results from pixels on relatively simple tasks, such as Pendulum, Pointmass2D, and Gripper. As shown in Figure 4 we can see that our model significantly outperforms A2C in terms of final episode reward after training for 40 million timesteps. More specifically, on Reacher, HalfCheetah, and Walker2d our model achieved a 1.6, 2.8, and 1.7 times greater final reward compared to A2C. The videos of trained policies from pixels can be found at <https://www.youtube.com/watch?v=gtM87w1xGoM>. Pretrained model weights are available at <https://github.com/emansim/acktr>.

5.3 A better norm for critic optimization?

The previous natural policy gradient method applied a natural gradient update only to the actor. In our work, we propose also applying a natural gradient update to the critic. The difference lies in the norm with which we choose to perform steepest descent on the critic; that is, the norm $\|\cdot\|_B$ defined in section 2.2. In this section, we applied ACKTR to the actor, and compared using a first-order method (i.e., Euclidean norm) with using ACKTR (i.e., the norm defined by Gauss-Newton) for critic optimization. Figures 5 (a) and (b) show the results on the continuous control task HalfCheetah and the Atari game Breakout. We observe that regardless of which norm we use to optimize the critic, there are improvements brought by applying ACKTR to the actor compared to the baseline A2C.

Domain	Threshold	ACKTR		A2C		TRPO (10 M)	
		Rewards	Episodes	Rewards	Episodes	Rewards	Episodes
Ant	3500 (6000)	4621.6	3660	4870.5	106186	5095.0	60156
HalfCheetah	4700 (4800)	5586.3	12980	5343.7	21152	5704.7	21033
Hopper	2000 (3800)	3915.9	17033	3915.3	33481	3755.0	39426
InvertedPendulum	950 (950)	1000.0	6831	1000.0	10982	1000.0	29267
InvertedDoublePendulum	9100 (9100)	9356.0	41996	9356.1	82694	9320.0	78519
Reacher	-7 (-3.75)	-1.5	3325	-1.7	20591	-2.0	14940
Swimmer	90 (360)	138.0	6475	140.7	11516	136.4	1571
Walker2d	3000 (N/A)	6198.8	15043	5874.9	26828	6874.1	27720

Table 2: ACKTR, A2C, and TRPO results, showing the top 10 average episode rewards attained within 30 million timesteps, averaged over the 3 best performing random seeds out of 8 random seeds. “Episode” denotes the smallest N for which the mean episode reward over the N^{th} to the $(N + 10)^{th}$ game crosses a certain threshold. The thresholds for all environments except for InvertedPendulum and InvertedDoublePendulum were chosen according to Gu et al. [8], and in brackets we show the reward threshold needed to solve the environment according to the OpenAI Gym website [5].

However, the improvements brought by using the Gauss-Newton norm for optimizing the critic are more substantial in terms of sample efficiency and episode rewards at the end of training. In addition, the Gauss-Newton norm also helps stabilize the training, as we observe larger variance in the results over random seeds with the Euclidean norm.

Recall that the Fisher matrix for the critic is constructed using the output distribution of the critic, a Gaussian distribution with variance σ . In vanilla Gauss-Newton, σ is set to 1. We experimented with estimating σ using the variance of the Bellman error, which resembles estimating the variance of the noise in regression analysis. We call this method adaptive Gauss-Newton. However, we find adaptive Gauss-Newton doesn’t provide any significant improvement over vanilla Gauss-Newton. (See detailed comparisons on the choices of σ in Appendix D).

5.4 How does ACKTR compare with A2C in wall-clock time?

We compared ACKTR to the baselines A2C and TRPO in terms of wall-clock time. Table 3 shows the average timesteps per second over six Atari games and eight MuJoCo (from state space) environments. The result is obtained with the same experiment setup as previous experiments. Note that in MuJoCo tasks episodes are processed sequentially, whereas in the Atari environment episodes are processed in parallel; hence more frames are processed in Atari environments. From the table we see that ACKTR only increases computing time by at most 25% per timestep, demonstrating its practicality with large optimization benefits.

(Timesteps/Second)	Atari			MuJoCo			
	batch size	80	160	640	1000	2500	25000
ACKTR		712	753	852	519	551	582
A2C		1010	1038	1162	624	650	651
TRPO		160	161	177	593	619	637

Table 3: Comparison of computational cost. The average timesteps per second over six Atari games and eight MuJoCo tasks during training for each algorithms. ACKTR only increases computing time at most 25% over A2C.

5.5 How do ACKTR and A2C perform with different batch sizes?

In a large-scale distributed learning setting, large batch size is used in optimization. Therefore, in such a setting, it is preferable to use a method that can scale well with batch size. In this section, we compare how ACKTR and the baseline A2C perform with respect to different batch sizes. We experimented with batch sizes of 160 and 640. Figure 5 (c) shows the rewards in number of timesteps. We found that ACKTR with a larger batch size performed as well as that with a smaller batch size. However, with a larger batch size, A2C experienced significant degradation in terms of sample efficiency. This corresponds to the observation in Figure 5 (d), where we plotted the training curve

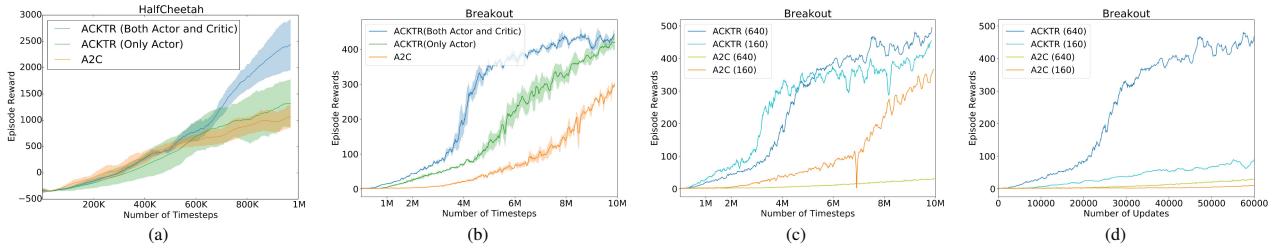


Figure 5: (a) and (b) compare optimizing the critic (value network) with a Gauss-Newton norm (ACKTR) against a Euclidean norm (first order). (c) and (d) compare ACKTR and A2C with different batch sizes.

in terms of number of updates. We see that the benefit increases substantially when using a larger batch size with ACKTR compared to with A2C. This suggests there is potential for large speed-ups with ACKTR in a distributed setting, where one needs to use large mini-batches; this matches the observation in [2].

6 Conclusion

In this work we proposed a sample-efficient and computationally inexpensive trust-region-optimization method for deep reinforcement learning. We used a recently proposed technique called K-FAC to approximate the natural gradient update for actor-critic methods, with trust region optimization for stability. To the best of our knowledge, we are the first to propose optimizing both the actor and the critic using natural gradient updates. We tested our method on Atari games as well as the MuJoCo environments, and we observed 2- to 3-fold improvements in sample efficiency on average compared with a first-order gradient method (A2C) and an iterative second-order method (TRPO). Because of the scalability of our algorithm, we are also the first to train several non-trivial tasks in continuous control directly from raw pixel observation space. This suggests that extending Kronecker-factored natural gradient approximations to other algorithms in reinforcement learning is a promising research direction.

Acknowledgements

We would like to thank the OpenAI team for their generous support in providing baseline results and Atari environment preprocessing codes. We also want to thank John Schulman for helpful discussions.

References

- [1] S. I. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.
- [2] J. Ba, R. Grosse, and J. Martens. Distributed second-order optimization using Kronecker-factored approximations. In *ICLR*, 2017.
- [3] J. A. Bagnell and J. G. Schneider. Covariant policy search. In *IJCAI*, 2003.
- [4] 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*, 47:253–279, 2013.
- [5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [6] D. A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *ICLR*, 2016.
- [7] R. Grosse and J. Martens. A Kronecker-factored approximate Fisher matrix for convolutional layers. In *ICML*, 2016.

- [8] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. In ICLR, 2017.
- [9] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver. Emergence of locomotion behaviours in rich environments. arXiv preprint arXiv:1707.02286.
- [10] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In ICLR, 2017.
- [11] S. Kakade. A natural policy gradient. In Advances in Neural Information Processing Systems, 2002.
- [12] D. Kingma and J. Ba. Adam: A method for stochastic optimization. ICLR, 2015.
- [13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In ICLR, 2016.
- [14] J. Martens. Deep learning via Hessian-free optimization. In ICML-10, 2010.
- [15] J. Martens. New insights and perspectives on the natural gradient method. arXiv preprint arXiv:1412.1193, 2014.
- [16] J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature.
- [17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.
- [18] V. Mnih, A. Puigdomenech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In ICML, 2016.
- [19] J. Nocedal and S. Wright. Numerical Optimization. Springer, 2006.
- [20] J. Peters and S. Schaal. Natural actor-critic. Neurocomputing, 71(7-9):1180–1190, 2008.
- [21] N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. Neural Computation, 2002.
- [22] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In Proceedings of the 32nd International Conference on Machine Learning (ICML), 2015.
- [23] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In Proceedings of the International Conference on Learning Representations (ICLR), 2016.
- [24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
- [25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016.
- [26] R. S. Sutton, D. A. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In Advances in Neural Information Processing Systems 12, 2000.
- [27] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.

- [28] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. In *ICLR*, 2016.
- [29] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- [30] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

A Experimental details

A.1 Discrete control

For experiments on the Atari Environment, we adopted the same input preprocessing procedure as in [17], with a slight modification to the architecture. Specifically, we used a shared network to parameterize the policy and value function: The first convolutional layer is of 32 filters of size 8×8 with stride 4 followed by another convolutional layer with 64 filters of size 4×4 and stride 2, followed by a final convolutional layer with 32 filters of size 3×3 with stride 1, followed by a fully connected layer of size 512, followed by one softmax output layer that parameterizes the policy and a linear output layer that predicts the value. We used 32 filters in the third convolutional layer because we found that it saved time for computing the Fisher matrix inverse without any degradation in performance. (One alternative would be to use the doubly factored approximation [2] with all 64 filters.) For the baseline A2C, we used the same architecture as in [17]. For TRPO, because of its high per-iteration expense, we used a smaller architecture, with 2 convolutional layers followed by a fully connected layer with 128 units. The first convolutional layer had 8 filters of size 8×8 with stride 4, followed by another convolutional layer with 16 filters of size 4×4 with stride 2.

We tuned the maximum learning rate η_{\max} using a grid search over $\{0.7, 0.2, 0.07, 0.02\}$ on the game of Breakout, with the trust region radius δ set to 0.001. We used the same hyperparameters for all Atari experiments. Both the baseline (A2C) and our method used a linear schedule for the learning rate over the course of training, and entropy regularization with weight 0.01. Following [17], the agent is trained on each game using 50 million time steps or 200 million frames. Unless otherwise stated, we used a batch size of 640 for ACKTR, 80 for A2C, and 512 for TRPO. The batch sizes were chosen to achieve better sample efficiency.

A.2 Continuous control

For experiments with low-dimensional state space as an input we used two separate neural networks with 64 hidden units per layer in a two-layer network. We used Tanh and ELU [6] nonlinearities for the policy network and value network, respectively, for all layers except the output layer, which didn't have any nonlinearity. The log standard deviation of a Gaussian policy was parameterized as a bias in a final layer of policy network that didn't depend on input state. For all experiments, we used ACKTR and A2C trained with batch sizes of 2500 and TRPO trained with a batch size of 25000. The batch sizes were chosen to be consistent with the experimental design of the results provided by the OpenAI team.

For experiments using pixels as an input we passed in a 42×42 RGB image along with the previous frame to a convolutional neural network. The two convolutional layers contained 32 filters of size 3×3 with stride of 2 followed by a fully connected layer of 256 hidden units. In contrast to our Atari experiments, we found that separating the policy network and value function into two separate networks resulted in better empirical performance in both ACKTR and A2C. We used ReLU nonlinearity for the policy network and ELU [6] nonlinearity for the value function. We also found that it is important to use orthogonal initialization for both networks, otherwise the A2C baseline failed to improve its episode reward. All models were trained with batch size of 8000. We tuned the maximum learning rate η_{\max} using a grid search over $\{0.3, 0.03, 0.003\}$ on the tasks of Reacher and Hopper, with the trust region radius δ set to 0.001. We fixed hyperparameters for all MuJoCo experiments.

Table 4: Raw scores across all games, starting with 30 no-op actions. Other scores from [29].

GAMES	HUMAN	DQN	DDQN	DUEL	PRIOR.	PRIOR. DUEL.	ACKTR
Alien	7,127.7	1,620.0	3,747.7	4,461.4	4,203.8	3,941.0	3197.1
Amidar	1,719.5	978.0	1,793.3	2,354.5	1,838.9	2,296.8	1059.4
Assault	742.0	4,280.4	5,393.2	4,621.0	7,672.1	11,477.0	10,777.7
Asterix	8,503.3	4,359.0	17,356.5	28,188.0	31,527.0	375,080.0	31,583.0
Asteroids	47,388.7	1,364.5	734.7	2,837.7	2,654.3	1,192.7	34,171.6
Atlantis	29,028.1	279,987.0	106,056.0	382,572.0	357,324.0	395,762.0	3,433,182.0
Bank Heist	753.1	455.0	1,030.6	1,611.9	1,054.6	1,503.1	1,289.7
Battle Zone	37,187.5	29,900.0	31,700.0	37,150.0	31,530.0	35,520.0	8910.0
Beamrider	16,926.5	8,627.5	13,772.8	12,164.0	23,384.2	30,276.5	13,581.4
Berzerk	2,630.4	585.6	1,225.4	1,472.6	1,305.6	3,409.0	927.2
Bowling	160.7	50.4	68.1	65.5	47.9	46.7	24.3
Boxing	12.1	88.0	91.6	99.4	95.6	98.9	1.45
Breakout	30.5	385.5	418.5	345.3	373.9	366.0	735.7
Centipede	12,017.0	4,657.7	5,409.4	7,561.4	4,463.2	7,687.5	7,125.28
Crazy Climber	35,829.4	110,763.0	117,282.0	143,570.0	141,161.0	162,224.0	150,444.0
Demon Attack	1,971.0	12,149.4	58,044.2	60,813.3	71,846.4	72,878.6	274,176.7
Double Dunk	-16.4	-6.6	-5.5	0.1	18.5	-12.5	-0.54
Enduro	860.5	729.0	1,211.8	2,258.2	2,093.0	2,306.4	0.0
Fishing Derby	-38.7	-4.9	15.5	46.4	39.5	41.3	33.73
Freeway	29.6	30.8	33.3	0.0	33.7	33.0	0.0
Gopher	2,412.5	8,777.4	14,840.8	15,718.4	32,487.2	104,368.2	47,730.8
Ice Hockey	0.9	-1.9	-2.7	0.5	1.3	-0.4	-4.2
James Bond	302.8	768.5	1,358.0	1,312.5	5,148.0	812.0	490.0
Kangaroo	3,035.0	7,259.0	12,992.0	14,854.0	16,200.0	1,792.0	3,150.0
Krull	2,665.5	8,422.3	7,920.5	11,451.9	9,728.0	10,374.4	9,686.9
Kung-Fu Master	22,736.3	26,059.0	29,710.0	34,294.0	39,581.0	48,375.0	34,954.0
Phoenix	7,242.6	8,485.2	12,252.5	23,092.2	18,992.7	70,324.3	133,433.7
Pitfall!	6,463.7	-286.1	-29.9	0.0	-356.5	0.0	-1.1
Pong	14.6	19.5	20.9	21.0	20.6	20.9	20.9
Q-bert	13,455.0	13,117.3	15,088.5	19,220.3	16,256.5	18,760.3	23,151.5
River Raid	17,118.0	7,377.6	14,884.5	21,162.6	14,522.3	20,607.6	17,762.8
Road Runner	7,845.0	39,544.0	44,127.0	69,524.0	57,608.0	62,151.0	53,446.0
Robotank	11.9	63.9	65.1	65.3	62.6	27.5	16.5
Seaquest	42,054.7	5,860.6	16,452.7	50,254.2	26,357.8	931.6	1,776.0
Solaris	12,326.7	3,482.8	3,067.8	2,250.8	4,309.0	133.4	2,368.6
Space Invaders	1,668.7	1,692.3	2,525.5	6,427.3	2,865.8	15,311.5	19,723.0
Star Gunner	10,250.0	54,282.0	60,142.0	89,238.0	63,302.0	125,117.0	82,920.0
Time Pilot	5,229.2	4,870.0	8,339.0	11,666.0	9,197.0	7,553.0	22,286.0
Tutankham	167.6	68.1	218.4	211.4	204.6	245.9	314.3
Up and Down	11,693.2	9,989.9	22,972.2	44,939.6	16,154.1	33,879.1	436,665.8
Video Pinball	17,667.9	196,760.4	309,941.9	98,209.5	282,007.3	479,197.0	100,496.6
Wizard Of Wor	4,756.5	2,704.0	7,492.0	7,855.0	4,802.0	12,352.0	702.0
Yars' Revenge	54,576.9	18,098.9	11,712.6	49,622.1	11,357.0	69,618.1	125,169.0
Zaxxon	9,173.3	5,363.0	10,163.0	12,944.0	10,469.0	13,886.0	17,448.0

B Results on the remaining Atari games

In this section we present results on the rest of the Atari games in Table 4. The score reported for our method is the mean of the last 100 episode rewards after 50 million time steps. Each episode is started with 30 no-op actions. We find that there is no result reported in A3C [18] or A2C using the same metric. Hence we compare our results with other Q-learning methods obtained from [29]. Due to limited computational resources, we were only able to evaluate ACKTR on a subset of the games. Our results are obtained with a single random seed and we have not tuned any hyperparameters. Although we use only one random seed, our results are on par with Q-learning methods, which use off-policy techniques such as experience replay. Q-learning methods usually take days to finish one training, whereas our method takes only 16 hours on a modern GPU.

C MuJoCo results with comparisons to OpenAI baselines

We compared ACKTR with the results of A2C and TRPO sent to us by the OpenAI team (<https://github.com/openai/baselines-results>). We followed their experimental protocol as closely as possible. Like our baselines, A2C and TRPO were trained with the same two-layer architecture with 64 hidden units in each layer on batch sizes of 2500 and 25000 respectively. However, in contrast to our baselines, the value function used Tanh nonlinearities and was “softly” updated by calculating the weighted average of the value function before and after the update.

Compared to our implementation of the A2C baseline, A2C implemented by OpenAI performed better on the Hopper, InvertedPendulum, Swimmer, and Walker2d tasks while performing worse on the Reacher and HalfCheetah tasks. TRPO by OpenAI performed worse than the TRPO trained by us on Hopper while achieving the same performance on the rest of the tasks. Results are shown in Figure 6.

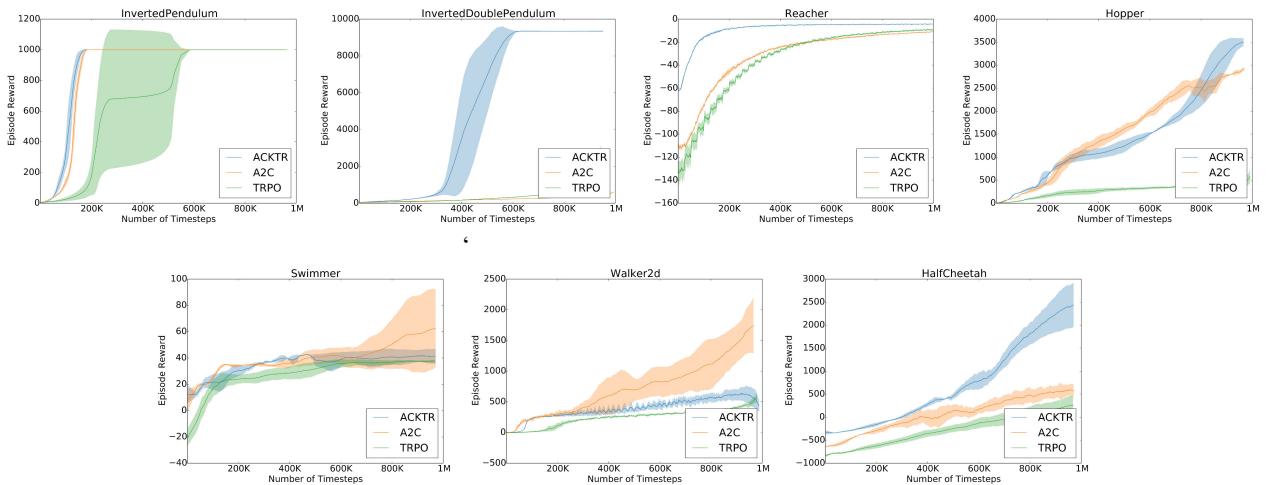


Figure 6: Performance comparisons on seven MuJoCo environments trained for 1 million timesteps (1 timestep equals 4 frames). The numbers for A2C and TRPO were provided to us by the OpenAI team. The shaded region denotes the standard deviation over 3 random seeds.

D Adaptive Gauss-Newton?

In this section we investigate whether training the critic using adaptive Gauss-Newton (i.e., keeping an estimate of the standard deviation of the Bellman error as the standard deviation of the critic output distribution) provides any improvement over vanilla Gauss-Newton (both defined in Section 3.1). We ran adaptive Gauss-Newton on all six standard Atari games and eight MuJoCo tasks. The results are shown in Figure 7 and Figure 8. We see that in Atari games, adaptive Gauss-Newton hurts the performance in terms of sample efficiency in Beamrider, Q-bert and Seaquest, and shows only a slight improvement in the game of Pong. In MuJoCo tasks, adaptive Gauss-Newton gives a slight improvement on the tasks of InvertedDoublePendulum, Swimmer, Walker2d, and Ant while performing on par on the tasks of InvertedPendulum and Reacher, and working considerably worse on the HalfCheetah task compared to vanilla Gauss-Newton.

E How well does the Kronecker-factored quadratic approximation match the exact KL?

We indirectly tested how accurate the Kronecker-factored approximation to the curvature is by measuring the exact KL changes during training, while performing trust region optimization using a Kronecker-factored quadratic model. We tested this in two Mujoco environments, HalfCheetah and Reacher. The values of approximated KL and exact KL are shown in Figure 9. From the plot we see

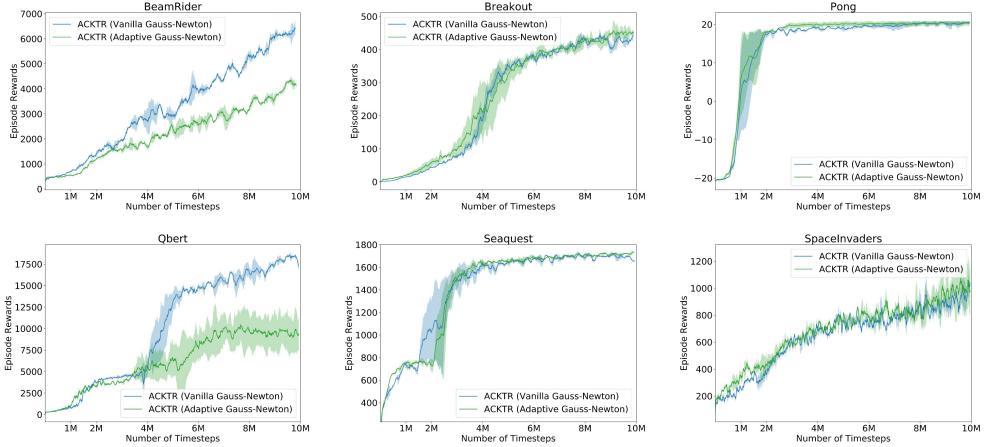


Figure 7: Performance comparisons of critic trained with adaptive Gauss-Newton and vanilla Gauss-Newton on six Atari environments trained for 10 million timesteps (1 timestep equals 4 frames). The shaded region denotes the standard deviation over 2 random seeds.

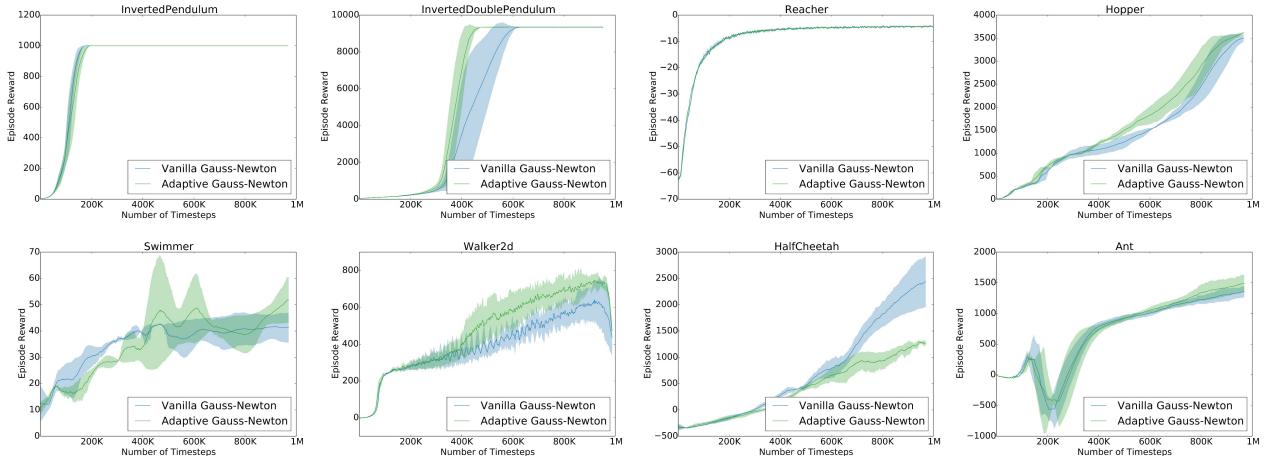


Figure 8: Performance comparisons of critic trained with adaptive Gauss-Newton and vanilla Gauss-Newton on eight MuJoCo environments trained for 1 million timesteps (1 timestep equals 4 frames). The shaded region denotes the standard deviation over 3 random seeds.

that exact KL is close to the trust region radius, showing the effectiveness of trust region optimization via Kronecker-factored approximation.

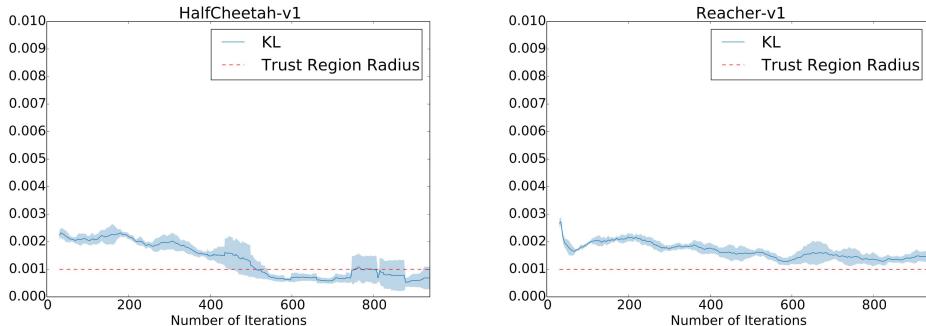


Figure 9: The plot shows the exact KL changes during training with trust region optimization using ACKTR. The actual KL is close to the trust region radius, showing the effectiveness of trust region optimization via Kronecker-factored approximation.