SUPERVISED POLICY UPDATE

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a new sample-efficient methodology, called Supervised Policy Update (SPU), for deep reinforcement learning. Starting with data generated by the current policy, SPU formulates and solves a constrained optimization problem in the non-parameterized proximal policy space. Using supervised regression, it then converts the optimal non-parameterized policy to a parameterized policy, from which it draws new samples. The methodology is general in that it applies to both discrete and continuous action spaces, and can handle a wide variety of proximity constraints for the non-parameterized optimization problem. We show how the Natural Policy Gradient and Trust Region Policy Optimization (NPG/TRPO) problems, and the Proximal Policy Optimization (PPO) problem can be addressed by this methodology. The SPU implementation is much simpler than TRPO. In terms of sample efficiency, our extensive experiments show SPU outperforms TRPO in Mujoco simulated robotic tasks and outperforms PPO in Atari video game tasks.

1 INTRODUCTION

The policy gradient problem in deep reinforcement learning can be informally defined as seeking a parameterized policy π_θ that produces a high expected reward $J(\pi_\theta)$. The parameterized policy π_θ is realized with a neural network, and stochastic gradient descent with backpropagation is used to optimize the parameters. An issue that plagues traditional policy gradient methods is poor sample efficiency (Kakade, 2003; Schulman et al., 2015a; Wang et al., 2016b; Wu et al., 2017; Schulman et al., 2017). In algorithms such as REINFORCE (Williams, 1992), new samples are needed for every small gradient step. In environments for which generating trajectories is expensive (such as robotic environments), sample efficiency is of central concern. The sample efficiency of an algorithm is defined to be the number of calls to the environment required to attain a specified performance level (Kakade, 2003).

Thus, given the current policy π_{θ_k} and a fixed number of trajectories (samples) generated from π_{θ_k}, the goal of the sample efficiency problem is to construct a new policy with the highest performance improvement possible. In order to obtain a high performance improvement with the limited number of samples, it is desirable to limit the search to policies that are close to the original policy π_{θ_k} (Kakade, 2002; Schulman et al., 2015a; Wu et al., 2017; Achiam et al., 2017; Schulman et al., 2017; Tangkaratt et al., 2018). Intuitively, if the candidate new policy π_θ is far from the original policy π_{θ_k}, it may not perform better than the original policy because too much emphasis is being placed on the relatively small batch of new data generated by π_{θ_k}, and not enough emphasis is being placed on the relatively large amount of data and effort previously used to construct π_{θ_k}.

This guideline of limiting the search to nearby policies seems reasonable in principle, but requires a notion of closeness of two policies. One natural approach is to define a distance or divergence $\eta(\pi_\theta, \pi_{\theta_k})$ between the current policy π_{θ_k} and the candidate new policy π_θ, and then attempt to solve the constrained optimization problem:

$$\underset{\theta}{\text{maximize}} \quad \hat{J}(\pi_\theta \mid \pi_{\theta_k}, \text{new data}) \tag{1}$$

$$\text{subject to} \quad \eta(\pi_\theta, \pi_{\theta_k}) \leq \delta \tag{2}$$

where $\hat{J}(\pi_\theta \mid \pi_{\theta_k}, \text{new data})$ is an estimate of $J(\pi_\theta)$, the performance of policy π_θ, based on the previous policy π_{θ_k} and the batch of fresh data generated by π_{θ_k}. The objective (1) attempts to maximize the performance of the updated policy, and the constraint (2) ensures that the updated policy is not too far from the policy π_{θ_k} that was used to generate the data. δ is a hyper-parameter. As

discussed in the related work section, several recent papers (Kakade, 2002; Schulman et al., 2015a; 2017; Tangkaratt et al., 2018) belong to the framework (1)-(2).

We propose a new methodology, called Supervised Policy Update (SPU), for this sample efficiency problem. The methodology is general in that it applies to both discrete and continuous action spaces, and can address a wide variety of constraint types for (2). Starting with data generated by the current policy, SPU optimizes over a proximal policy space to find an optimal *non-parameterized policy*. It then solves a supervised regression problem to convert the non-parameterized policy to a parameterized policy, from which it draws new samples. We develop a general methodology for finding an optimal policy in the non-parameterized policy space, and then illustrate the methodology for three different definitions of proximity. We also show how the Natural Policy Gradient and Trust Region Policy Optimization (NPG/TRPO) problems and the Proximal Policy Optimization (PPO) problem can be addressed by this methodology. While SPU is substantially simpler than NPG/TRPO in terms of mathematics and implementation, our extensive experiments show that SPU is more sample efficient than TRPO in Mujoco simulated robotic tasks and PPO in Atari video game tasks.

2 PRELIMINARIES

We consider a Markov Decision Process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, and reward function $r(s, a)$, $s \in \mathcal{S}$, $a \in \mathcal{A}$. Let $\pi = \{\pi(a|s) : s \in \mathcal{S}, a \in \mathcal{A}\}$ denote a policy, let Π be the set of all policies, and let the expected discounted reward be:

$$J(\pi) \triangleq \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\right] \tag{3}$$

where $\gamma \in (0, 1)$ is a discount factor and $\tau = (s_0, a_0, s_1, \dots)$ is a sample trajectory. Let $A^\pi(s, a)$ be the advantage function for policy π (Silver, 2015; Levine, 2017; Schulman et al., 2015b). Deep reinforcement learning considers a set of parameterized policies $\Pi_{\text{DL}} = \{\pi_\theta | \theta \in \Theta\} \subset \Pi$, where each parameterized policy is defined by a neural network called the policy network. In this paper, we will consider optimizing over the parameterized policies in Π_{DL} as well as over the non-parameterized policies in Π. For concreteness, we assume that the state and action spaces are finite. However, the methodology also applies to continuous state and action spaces, as shown in the Appendix.

One popular approach to maximizing $J(\pi_\theta)$ over Π_{DL} is to apply stochastic gradient ascent. The gradient of $J(\pi_\theta)$ evaluated at a specific $\theta = \theta_k$ can be shown to be (Williams, 1992; Sutton et al., 2000; Schulman et al., 2015b):

$$\nabla_\theta J(\pi_{\theta_k}) = \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta_k}}\left[\sum_{t=0}^{\infty}\gamma^t \nabla_\theta \log \pi_{\theta_k}(a_t|s_t) A^{\pi_{\theta_k}}(s_t, a_t)\right]. \tag{4}$$

To obtain an estimate of the gradient, we can sample finite-length trajectories τ_i, $i = 1, \dots, N$, from π_{θ_k}, and approximate (4) as:

$$\nabla_\theta J(\pi_{\theta_k}) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T-1}\nabla_\theta \log \pi_{\theta_k}(a_{it}|s_{it}) A_{it} \triangleq g_k \tag{5}$$

where T is the length of the trajectory, and A_{it} is an approximation of $\gamma^t A^{\pi_{\theta_k}}(s_{it}, a_{it})$ obtained from a critic network. Additionally, define $d^\pi(s) \triangleq (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P_\pi(s_t = s)$ for the the future state probability distribution for policy π, and denote $\pi(\cdot|s)$ for the probability distribution over the action space $\mathcal{A}$ when in state s and using policy π. Further denote $D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s]$ for the KL divergence from distribution $\pi_{\theta_k}(\cdot|s)$ to $\pi(\cdot|s)$, and denote

$$\bar{D}_{\text{KL}}(\pi \parallel \pi_{\theta_k}) = \mathop{\mathbb{E}}_{s\sim d^{\pi_{\theta_k}}}\left[D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s]\right] \tag{6}$$

for the "aggregated KL divergence".

2.1 SURROGATE OBJECTIVES FOR THE SAMPLE EFFICIENCY PROBLEM

For the sample efficiency problem, the objective $J(\pi_\theta)$ is typically approximated using samples generated from π_{θ_k} (Schulman et al., 2015a; Achiam et al., 2017; Schulman et al., 2017). One of two

different approaches is typically used to approximate $J(\pi_\theta) - J(\pi_{\theta_k})$ using samples from π_{θ_k}. The first approach is to make a first order approximation of $J(\pi_\theta)$ in the vicinity of θ_k (Kakade, 2002; Peters & Schaal, 2008a;b; Schulman et al., 2015a):

$$J(\pi_\theta) - J(\pi_{\theta_k}) \approx (\theta - \theta_k)^T \nabla_\theta J(\pi_{\theta_k}) \approx (\theta - \theta_k)^T g_k \tag{7}$$

where g_k is the sample estimate (5). The second approach, which applies to all policies $\pi \in \Pi$ and not just to policies $\pi_\theta \in \Pi_{DL}$, is to approximate the state distribution $d^\pi(s)$ with $d^{\pi_{\theta_k}}(s)$, giving the approximation (Achiam et al., 2017; Schulman et al., 2017; Achiam, 2017):

$$J(\pi) - J(\pi_{\theta_k}) \approx \mathcal{L}_{\pi_{\theta_k}}(\pi) \triangleq \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s\sim d^{\pi_{\theta_k}}} \mathop{\mathbb{E}}_{a\sim\pi_{\theta_k}(\cdot|s)} \left[\frac{\pi(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a) \right] \tag{8}$$

To estimate the expectation in (8), as in (5), we generate trajectories of finite length T from π_{θ_k}, create estimates of the advantage values using a critic network, and then form a sample average. There is a well-known bound for the approximation (8) (Kakade & Langford, 2002; Achiam et al., 2017). Furthermore, the approximation $\mathcal{L}_{\pi_{\theta_k}}(\pi_\theta)$ matches $J(\pi_\theta) - J(\pi_{\theta_k})$ to the first order with respect to the parameter θ (Achiam et al., 2017).

3 Related Work

Natural gradient (Amari, 1998) was applied to policy gradient first by Kakade (Kakade, 2002) and then in Peters & Schaal (2008a;b); Achiam (2017); Schulman et al. (2015a). We refer to this body of literature as NPG/TRPO. The goal of NPG/TRPO is to solve the sample efficiency problem (1)-(2) with $\eta(\pi_\theta, \pi_{\theta_k}) = \bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})$, i.e., use the aggregate KL-divergence for the policy proximity constraint (2). NPG/TRPO addresses this problem in the parameter space $\theta \in \Theta$. First, it approximates $J(\pi_\theta)$ with the first-order approximation (7) and $\bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})$ using a similar second-order method. Second, it uses samples from π_{θ_k} to form estimates of these two approximations. Third, using these estimates (which are functions of θ), it solves for the optimal θ^*. The optimal θ^* is a function of g_k and of h_k, the sample average of the Hessian evaluated at θ_k. TRPO also limits the magnitude of the update to ensure $\bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k}) \leq \delta$ (i.e., ensuring the sampled estimate of the aggregated KL constraint is met *without* the second-order approximation).

SPU takes a very different approach by first (i) posing and solving the optimization problem in the non-parameterized policy space, and then (ii) solving a supervised regression problem to find a parameterized policy that is near the optimal non-parameterized policy. A recent paper, Guided Actor Critic (GAC), independently proposed a similar decomposition (Tangkaratt et al., 2018). However, GAC is much more restricted in that it considers only one specific constraint criterion (aggregated reverse-KL divergence) and applies only to continuous action spaces. Furthermore, GAC incurs significantly higher computational complexity because at each iteration, GAC inverts a damped Hessian whose complexity scales cubically in the number of action dimensions. In our experiments, the GAC implementation as provided by the authors can be up to 400 times slower than SPU in Mujoco environments with high-dimensional action spaces such as "Humanoid-v2". We also note that GAC uses past trajectories to update the policy parameters while SPU only uses the trajectories generated by the most recent policy.

Clipped-PPO (Schulman et al., 2017) takes a very different approach to TRPO. In the process of going from π_{θ_k} to $\pi_{\theta_{k+1}}$, PPO makes many gradient steps while only using the data from π_{θ_k}. Without the clipping, PPO is simply the approximation (8). The additional clipping is analogous to the constraint (2) in that it has the goal of keeping π_θ close to π_{θ_k}. Indeed, the clipping can be seen as an attempt at keeping $\pi_\theta(a_t|s_t)$ from becoming neither much larger than $(1+\epsilon)\pi_{\theta_k}(a_t|s_t)$ nor much smaller than $(1-\epsilon)\pi_{\theta_k}(a_t|s_t)$. Thus, although the clipped PPO objective does not squarely fit into the optimization framework (1)-(2), it is quite similar in spirit.

Actor-Critic using Kronecker-Factored Trust Region (ACKTR) (Wu et al., 2017) proposed using Kronecker-factored approximation curvature (K-FAC) to update both the policy gradient and critic terms, giving a more computationally efficient method of calculating the natural gradients. ACER (Wang et al., 2016a) exploits past episodes, linearizes the KL divergence constraint, and maintains an average policy network to enforce the KL divergence constraint. In future work, it would of interest to extend the SPU methodology to handle past episodes.

4 SPU FRAMEWORK

The SPU methodology has two steps. In the first step, for a given constraint criterion $\eta(\pi, \pi_{\theta_k}) \leq \delta$, we find the optimal solution to the non-parameterized problem:

$$\underset{\pi \in \Pi}{\text{maximize}} \quad \mathcal{L}_{\pi_{\theta_k}}(\pi) \tag{9}$$

$$\text{subject to} \quad \eta(\pi, \pi_{\theta_k}) \leq \delta \tag{10}$$

Note that π is not restricted to the set of parameterized policies Π_{DL}. Also note, as is common practice, we are using an approximation for the objective function. Specifically, we are using the approximation (8). However, unlike PPO/TRPO, we are not approximating the constraint (2). We will show below the optimal solution π^* for the non-parameterized problem (9)-(10) can be determined nearly in closed form for many natural constraint criteria $\eta(\pi, \pi_{\theta_k}) \leq \delta$.

In the second step of the SPU methodology, we attempt to find a policy π_θ in the parameterized space Π_{DL} that is close to the target policy π^*. Specifically, to advance from θ_k to θ_{k+1} we perform the following steps:

(i) We first sample N trajectories using policy π_{θ_k}, giving sample data (s_i, a_i, A_i), $i = 1, .., m$. Here A_i is again an estimate of the advantage value $A^{\pi_k}(s_i, a_i)$, which is obtained from an auxiliary critic network. (For notational simplicity, we henceforth index the samples with i rather than with (i, t) corresponding to the tth sample in the ith trajectory.)

(ii) For each s_i, we define the target distribution π^* to be the optimal solution to the constrained optimization problem (9)-(10) for a specific constraint η.

(iii) We then fit the policy network π_θ to the target distributions $\pi^*(\cdot|s_i)$, $i = 1, .., m$. Specifically, to find θ_{k+1}, we use the following loss function:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{m} D_{\text{KL}}(\pi_\theta \parallel \pi^*)[s_i] \tag{11}$$

For this supervised learning problem, we do not initialize with random initial weights, but instead with the weights for π_{θ_k}. In minimizing the loss function $L(\theta)$, we can use mini-batch optimization techniques such as RMSprop or Adam. Overfitting to the fresh data (s_i, a_i, A_i), $i = 1, .., m$ can be controlled by dynamic early stopping; see Chapter 7 of (Goodfellow et al., 2016). The resulting θ becomes our θ_{k+1}.

Thus SPU proceeds by solving a series of supervised learning problems, one for each policy step k.

5 SPU APPLIED TO SPECIFIC PROXIMITY CRITERIA

To illustrate the SPU methodology, for three different but natural types of proximity constraints, we solve the corresponding non-parameterized optimization problem and derive the resulting gradient for the SPU supervised learning problem.

5.1 FORWARD AGGREGATE AND DISAGGREGATE KL CONSTRAINTS

We first consider constraint criteria of the form:

$$\underset{\pi \in \Pi}{\text{maximize}} \quad \sum_s d^{\pi_{\theta_k}}(s) \underset{a \sim \pi(\cdot|s)}{\mathbb{E}} [A^{\pi_{\theta_k}}(s, a)] \tag{12}$$

$$\text{subject to} \quad \sum_s d^{\pi_{\theta_k}}(s) D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s] \leq \delta \tag{13}$$

$$D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s] \leq \epsilon \text{ for all } s \tag{14}$$

Note that this problem is equivalent to minimizing $\mathcal{L}_{\pi_{\theta_k}}(\pi)$ subject to the constraints (13) and (14). We refer to (13) as the "aggregated KL constraint" and to (14) as the "disaggregated KL constraint". These two constraints taken together restrict π from deviating too much from π_{θ_k}. Note that this problem (minus the disaggregated constraints) is analogous to the NPG/TRPO problem, which was

solved in the parameterized space with an additional approximation for the aggregated constraint. We shall refer to (12)-(14) as the forward-KL non-parameterized optimization problem.

For each $\lambda > 0$, define: $\pi^\lambda(a|s) = \frac{\pi_{\theta_k}(a|s)}{Z_\lambda(s)} e^{A^{\pi_{\theta_k}}(s,a)/\lambda}$ where $Z_\lambda(s)$ is defined so that $\pi^\lambda(\cdot|s)$ is a valid distribution. Further, for each s, let λ_s be the solution to $D_{\text{KL}}(\pi^\lambda \parallel \pi_{\theta_k})[s] = \epsilon$. Also let $\Gamma^\lambda = \{s : D_{\text{KL}}(\pi^\lambda \parallel \pi_{\theta_k})[s] \leq \epsilon\}$.

Theorem 1 *The optimal solution to the problem* (12)-(14) *is given by:*

$$\tilde{\pi}^\lambda(a|s) = \begin{cases} \pi^\lambda(a|s) & s \in \Gamma^\lambda \\ \pi^{\lambda_s}(a|s) & s \notin \Gamma^\lambda \end{cases} \tag{15}$$

where λ is chosen so that $\sum_s d^{\pi_{\theta_k}}(s) D_{KL}(\tilde{\pi}^\lambda \parallel \pi_{\theta_k})[s] = \delta$ (Proof in subsection A.1).

Equation (15) provides the structure of the optimal non-parameterized policy. As part of the SPU framework, we then seek a parameterized policy π_θ that is close to $\tilde{\pi}^\lambda(a|s)$, that is, minimizes the loss function (11). A straightforward calculation shows (Appendix B):

$$\nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \tilde{\pi}^\lambda)[s_i] = \nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_i] - \frac{1}{\tilde{\lambda}_{s_i}} \mathop{\mathbb{E}}_{a \sim \pi_\theta(\cdot|s_i)} [\nabla_\theta \log \pi_\theta(a|s_i) A^{\pi_{\theta_k}}(s_i, a)] \tag{16}$$

where $\widetilde{\lambda}_{s_i} = \lambda$ for $s_i \in \Gamma^\lambda$ and $\tilde{\lambda}_{s_i} = \lambda_{s_i}$ for $s_i \notin \Gamma^\lambda$. In practice, we will need to estimate the expectation in (16). For this, we use the sample $a_i \sim \pi_{\theta_k}(\cdot|s_i)$ and approximate $A^{\pi_{\theta_k}}(s_i, a_i)$ as A_i (obtained from the critic network), giving:

$$\nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \tilde{\pi}^\lambda)[s_i] \approx \nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_i] - \frac{1}{\widetilde{\lambda}_{s_i}} \frac{\nabla_\theta \pi_\theta(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} A_i \tag{17}$$

In order to simplify the algorithm, we slightly modify the gradient update given by (17). We replace the hyper-parameter δ with the hyper-parameter λ and tune λ rather than δ. Further, we set $\widetilde{\lambda}_{s_i} = \lambda$ for all s_i in (17) but introduce per-state acceptance to account for the disaggregated constraints, resulting in the following approximate gradient:

$$\nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \tilde{\pi}^\lambda) \approx \frac{1}{m} \sum_{i=1}^{m} [\nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_i] - \frac{1}{\lambda} \frac{\nabla_\theta \pi_\theta(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} A_i] \mathbb{1}_{D_{\text{KL}}(\pi_\theta \| \pi_{\theta_k})[s_i] \leq \epsilon} \tag{18}$$

We use (18) as our gradient for supervised training of the policy network. The equation (18) has an intuitive interpretation: the gradient represents a trade-off between the approximate performance of π_θ (as captured by $\frac{1}{\lambda} \frac{\nabla_\theta \pi_\theta(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} A_i$) and how far π_θ diverges from π_{θ_k} (as captured by $\nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_i]$). For the stopping criterion, we train until $\frac{1}{m} \sum_i D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_i] \approx \delta$.

5.2 Backward KL Constraint

In a similar manner, we can derive the structure of the optimal policy when using the reverse KL-divergence as the constraint. For simplicity, we provide the result for when there are only disaggregated constraints. We seek to find the non-parameterized optimal policy by solving:

$$\underset{\pi \in \Pi}{\text{maximize}} \quad \sum_s d^{\pi_{\theta_k}}(s) \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} [A^{\pi_{\theta_k}}(s, a)] \tag{19}$$

$$D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s] \leq \epsilon \text{ for all } s \tag{20}$$

Theorem 2 *The optimal solution to the problem* (19)-(20) *is given by:*

$$\pi^*(a|s) = \pi_{\theta_k}(a|s) \frac{\lambda(s)}{\lambda'(s) - A^{\pi_{\theta_k}}(s, a)} \tag{21}$$

where $\lambda(s) > 0$ and $\lambda'(s) > \max_a A^{\pi_{\theta_k}}(s, a)$ (Proof in subsection A.2).

Note that the structure of the optimal policy with the backward KL constraint is quite different from that with the forward KL constraint. A straight forward calculation shows (Appendix B):

$$\nabla_\theta D_{\mathrm{KL}}(\pi_\theta \parallel \pi^*)[s] = \nabla_\theta D_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s] - \underset{a\sim\pi_{\theta_k}}{E}\left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \log\left(\frac{1}{\lambda'(s) - A^{\pi_{\theta_k}}(s,a)}\right)\right] \tag{22}$$

The equation (22) has an intuitive interpretation. It increases the probability of action a if $A^{\pi_{\theta_k}}(s,a) > \lambda'(s) - 1$ and decreases the probability of action a if $A^{\pi_{\theta_k}}(s,a) < \lambda'(s) - 1$. (22) also tries to keep π_θ close to π_{θ_k} by minimizing their KL divergence.

5.3 L^∞ Constraint

In this section we show how a PPO-like objective can be formulated in the context of SPU. Recall from Section 3 that the the clipping in PPO can be seen as an attempt at keeping $\pi_\theta(a_i|s_i)$ from becoming neither much larger than $(1+\epsilon)\pi_{\theta_k}(a_i|s_i)$ nor much smaller than $(1-\epsilon)\pi_{\theta_k}(a_i|s_i)$ for $i = 1, \ldots, m$. In this subsection, we consider the constraint function

$$\eta(\pi, \pi_{\theta_k}) = \max_{i=1,\ldots,m} \frac{|\pi(a_i|s_i) - \pi_{\theta_k}(a_i|s_i)|}{\pi_{\theta_k}(a_i|s_i)} \tag{23}$$

which leads us to the following optimization problem:

$$\underset{\pi(a_1|s_1),\ldots,\pi(a_m|s_m)}{\text{maximize}} \quad \sum_{i=1}^{m} A^{\pi_{\theta_k}}(s_i, a_i)\frac{\pi(a_i|s_i)}{\pi_k(a_i|s_i)} \tag{24}$$

$$\text{subject to} \quad \left|\frac{\pi(a_i|s_i) - \pi_{\theta_k}(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)}\right| \leq \epsilon \quad i = 1, \ldots, m \tag{25}$$

$$\sum_{i=1}^{m}\left(\frac{\pi(a_i|s_i) - \pi_{\theta_k}(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)}\right)^2 \leq \delta \tag{26}$$

Note that here we are using a variation of the SPU methodology described in Section 4 since here we first create estimates of the expectations in the objective and constraints and then solve the optimization problem (rather than first solve the optimization problem and then take samples as done for Theorems 1 and 2). Note that we have also included an aggregated contraint (26) in addition to the PPO-like constraint (25), which further ensures that the updated policy is close to π_{θ_k}.

Theorem 3 *The optimal solution to the optimization problem (24-26) is given by:*

$$\pi^*(a_i|s_i) = \begin{cases} \pi_{\theta_k}(a_i|s_i)\min\{1+\lambda A_i, 1+\epsilon\} & A_i \geq 0 \\ \pi_{\theta_k}(a_i|s_i)\max\{1+\lambda A_i, 1-\epsilon\} & A_i < 0 \end{cases} \tag{27}$$

for some $\lambda > 0$ where $A_i \triangleq A^{\pi_{\theta_k}}(s_i, a_i)$ (Proof in subsection A.3).

To simplify the algorithm, we treat λ as a hyper-parameter rather than δ. After solving for π^*, we seek a parameterized policy π_θ that is close to π^* by minimizing their mean square error over sampled states and actions, i.e. by updating θ in the negative direction of $\nabla_\theta \sum_i (\pi_\theta(a_i|s_i) - \pi^*(a_i|s_i))^2$. We refer to this approach as SPU with the L^∞ constraint.

6 Experimental Results

We provide extensive experimental results which demonstrate SPU outperforms recent state-of-the-art methods for environments with continuous or discrete action spaces. We also provide ablation studies to show the importance of the different algorithmic components, and a sensitivity analysis to show that SPU's performance is relatively insensitive to hyper-parameter choices. There are two definitions of superior sample complexity performance that we use to compare two different algorithms A and B: (i) A is better than B if A takes fewer interactions with the environment to achieve a pre-defined performance threshold (Kakade, 2003); (ii) A is better than B if the averaged final performance of A is higher than that of B given the same number of interactions with the environment (Schulman et al., 2017). Implementation details are provided in Appendix D.

6.1 RESULTS ON MUJOCO

The Mujoco (Todorov et al., 2012) simulated robotics environments provided by OpenAI gym (Brockman et al., 2016) have become a popular benchmark for control problems with continuous action spaces. In terms of final performance averaged over all available ten Mujoco environments and ten different seeds, SPU with L^∞ constraint (Section 5.3) and SPU with forward KL constraints (Section 5.1) outperform TRPO by 6% and 27% respectively. Since the forward-KL approach is our best performing approach, we focus subsequent analysis on it and hereafter refer to it as SPU. Figure 1 illustrates the performance of SPU versus TRPO for each of the ten Mujoco environments. We observe that SPU clearly outperforms TRPO throughout training in the first 5 environments while being roughly the same as TRPO in the other 5. We also note that SPU significantly outperforms TRPO on the 3 most challenging Mujoco environments, which are shown in the top left corner in Figure 1. Algorithm 1 in the Appendix provides the complete description of the SPU algorithm.

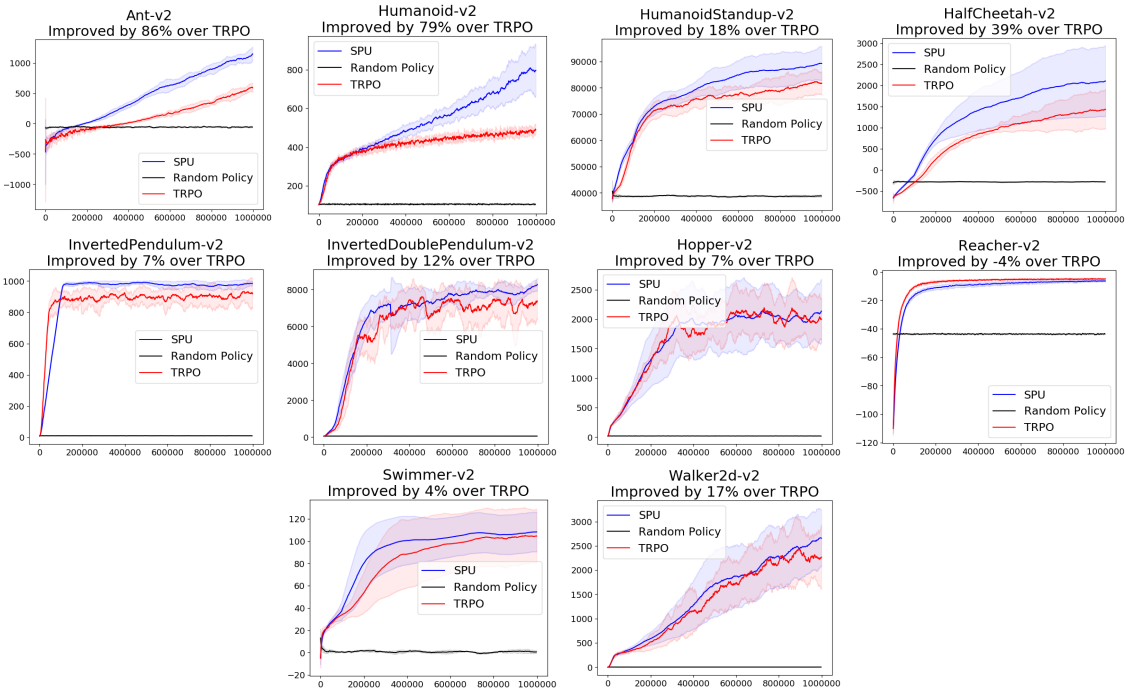

Figure 1: Performance of SPU versus TRPO on 10 Mujoco environments in 1 million timesteps. The x-axis indicates timesteps. The y-axis indicates the average episode reward of the last 100 episodes.

To ensure that SPU is not only better than TRPO in terms of performance gain early during training, we further retrain both policies for 3 million timesteps. Again here, SPU outperforms TRPO by 28%. Figure 3 in the Appendix illustrates the performance for each environment.

6.2 ABLATION STUDIES FOR MUJOCO

The indicator variable in (18) enforces the disaggregated constraint. We refer to it as *per-state acceptance*. Removing this component is equivalent to removing the indicator variable. We refer to using $\sum_i D_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_i]$ to determine the number of training epochs as *dynamic stopping*. Without this component, the number of training epochs is treated as a hyper-parameter. We also tried removing $\nabla_\theta D_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_i]$ from the gradient update step in (18). Table 1 illustrates the contribution of the different components of SPU to the overall performance. The third row shows that the term $\nabla_\theta D_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_i]$ makes a crucially important contribution to SPU. Furthermore, per-state acceptance and dynamic stopping are both also important for obtaining high performance, with the former playing a more central role. When a component is removed, the hyper-parameters are retuned to ensure that the best possible performance is obtained with the alternative (simpler) algorithm.

Table 1: Ablation study for SPU

Approach	Percentage better than TRPO	Performance vs. original algorithm
Original Algorithm	27%	0%
No grad KL	4%	- 85%
No dynamic stopping	24%	- 11%
No per-state acceptance	9%	- 67%

6.3 Sensitivity Analysis on Mujoco

To demonstrate the practicality of SPU, we show that its high performance is insensitive to hyper-parameter choice. One way to show this is as follows: for each SPU hyper-parameter, select a reasonably large interval, randomly sample the value of the hyper parameter from this interval, and then compare SPU (using the randomly chosen hyper-parameter values) with TRPO. We sampled 100 SPU hyper-parameter vectors (each vector including $\delta, \epsilon, \lambda$), and for each one determined the relative performance with respect to TRPO. First, we found that for all 100 random hyper-parameter value samples, SPU performed better than TRPO. 75% and 50% of the samples outperformed TRPO by at least 18% and 21% respectively. The full CDF is given in Figure 4 in the Appendix. We can conclude that SPU's superior performance is largely insensitive to hyper-parameter values.

6.4 Results on Atari

It has recently been observed that neural networks are not always needed to obtain high performance in many Mujoco environments (Rajeswaran et al., 2017; Mania et al., 2018). To conclusively evaluate SPU, we compare it against state-of-the-art method on the Arcade Learning Environments (Bellemare et al., 2012) exposed through OpenAI gym (Brockman et al., 2016). Using the same network architecture and hyper-parameters, we learn to play 60 Atari games from raw pixels and rewards. This is highly challenging because of the diversity in the games and the high dimensionality of the observations.

Here, we compare SPU against PPO (Schulman et al., 2017) since PPO achieves comparable performance with TRPO in our experiments on Mujoco with much faster run-time (Appendix F.3). We also argue that PPO is a high-performing algorithm since it has been used to learn dexterous movements for robot arms (Marcin Andrychowicz, 2018) and has beat top human players in DOTA (Michael Petrov, 2018). Averaged over 60 Atari environments and 20 seeds, SPU is $\mathbf{55\%}$ better than PPO in terms of averaged final performance. Figure 2 provides a high-level overview of the result, from which we can draw two conclusions: (i) In 36 environments, SPU and PPO perform roughly the same (within 20% of each other); SPU clearly outperforms PPO in 15 environments while PPO clearly outperforms SPU in 9; (ii) In those 15+9 environments, the extent to which SPU outperforms PPO is much larger than the extent to which PPO outperforms SPU. Figure 6, Figure 7 and Figure 8 in the Appendix illustrate the performance of SPU vs PPO throughout training. SPU's high performance in both the Mujoco and Atari domains demonstrates its high performance and generality.

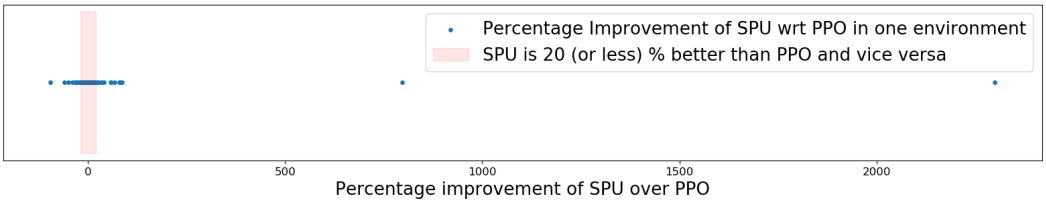

Figure 2: High-level overview of results on Atari

REFERENCES

Joshua Achiam. Advanced policy gradient methods. http://rll.berkeley.edu/deeprlcourse/f17docs/lecture_13_advanced_pg.pdf, 2017.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pp. 22–31, 2017.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL http://arxiv.org/abs/1207.4708.

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

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. https://github.com/openai/baselines, 2017.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016. URL http://arxiv.org/abs/1604.06778.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Sham Kakade. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pp. 267–274, 2002.

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

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Sergey Levine. UC Berkeley CS294 deep reinforcement learning lecture notes. http://rail.eecs.berkeley.edu/deeprlcourse-fa17/index.html, 2017.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

Maciek Chociej Rafal Jozefowicz Bob McGrew Jakub Pachocki Arthur Petron Matthias Plappert Glenn Powell Alex Ray Jonas Schneider Szymon Sidor Josh Tobin Peter Welinder Lilian Weng Wojciech Zaremba Marcin Andrychowicz, Bowen Baker. Learning dexterous in-hand manipulation. 2018. URL https://arxiv.org/abs/1808.00177.

Filip Wolski Henrique Ponde Susan Zhang Rafal Jozefowicz Jakub Pachocki Przemyslaw Debiak Jonathan Raiman Christy Dennison David Farhi Jie Tang Greg Brockman Szymon Sidor Alec Radford Larissa Schiavo Christopher Berner Diane Yoon Jonas Schneider Scott Gray Shariq Hashme Jack Clark Christopher Hesse John Schulman David Luan Quirin Fischer Paul Christiano Ilya Sutskever Eric Sigler Michael Petrov, Brooke Chan. Openai five. 2018. URL https://blog.openai.com/openai-five/.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518, 2015. URL http://dx.doi.org/10.1038/nature14236.

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008a.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008b.

Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards generalization and simplicity in continuous control. *CoRR*, abs/1703.02660, 2017. URL http://arxiv.org/abs/1703.02660.

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.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015c. URL http://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver. UCL course on reinforcement learning lecture notes. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html, 2015.

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.

Voot Tangkaratt, Abbas Abdolmaleki, and Masashi Sugiyama. Guide actor-critic for continuous control. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJk59JZ0b.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. 2012. URL https://ieeexplore.ieee.org/abstract/document/6386109/authors.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016a. URL http://arxiv.org/abs/1611.01224.

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*, 2016b.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pp. 5–32. Springer, 1992.

Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pp. 5285–5294, 2017.

Appendices

A PROOFS FOR NON-PARAMETERIZED OPTIMIZATION PROBLEMS

A.1 FORWARD KL AGGREGATED AND DISAGGREGATED CONSTRAINTS

We first show that (12)-(14) is a convex optimization. To this end, first note that the objective (12) is a linear function of the decision variables $\pi = \{\pi(a|s) \colon \; s \in \mathcal{S},\; a \in \mathcal{A}\}$. The LHS of (14) can be rewritten as: $\sum_{a\in\mathcal{A}} \pi(a|s) \log \pi(a|s) - \sum_{a\in\mathcal{A}} \pi(a|s) \log \pi_{\theta_k}(a|s)$. The second term is a linear function of π. The first term is a convex function since the second derivative of each summand is always positive. The LHS of (14) is thus a convex function. By extension, the LHS of (13) is also a convex function since it is a nonnegative weighted sum of convex functions. The problem (12)-(14) is thus a convex optimization problem. According to Slater's constraint qualification, strong duality holds since π_{θ_k} is a feasible solution to (12)-(14) where the inequality holds strictly.

We can therefore solve (12)-(14) by solving the related Lagrangian problem. For a fixed λ consider:

$$\underset{\pi\in\Pi}{\text{maximize}} \quad \sum_s d^{\pi_{\theta_k}}(s)\{\mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}[A^{\pi_{\theta_k}}(s,a)] - \lambda D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s]\} \tag{28}$$

$$\text{subject to} \quad D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s] \leq \epsilon \text{ for all } s \tag{29}$$

The above problem decomposes into separate problems, one for each state s:

$$\underset{\pi(\cdot|s)}{\text{maximize}} \quad \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}[A^{\pi_{\theta_k}}(s,a)] - \lambda D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s] \tag{30}$$

$$\text{subject to} \quad D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s] \leq \epsilon \tag{31}$$

Further consider the unconstrained problem (30) without the constraint (31):

$$\underset{\pi(\cdot|s)}{\text{maximize}} \quad \sum_{a=1}^{K} \pi(a|s) \left[A^{\pi_{\theta_k}}(s,a) - \lambda \log\left(\frac{\pi(a|s)}{\pi_{\theta_k}(a|s)}\right)\right] \tag{32}$$

$$\text{subject to} \quad \sum_{a=1}^{K} \pi(a|s) = 1 \tag{33}$$

$$\pi(a|s) \geq 0, \quad a = 1, \ldots, K \tag{34}$$

A simple Lagrange-multiplier argument shows that the opimal solution to (32)-(34) is given by:

$$\pi^{\lambda}(a|s) = \frac{\pi_{\theta_k}(a|s)}{Z_{\lambda}(s)} e^{A^{\pi_{\theta_k}}(s,a)/\lambda}$$

where $Z_{\lambda}(s)$ is defined so that $\pi^{\lambda}(\cdot|s)$ is a valid distribution. Now returning to the decomposed constrained problem (30)-(31), there are two cases to consider. The first case is when $D_{\text{KL}}(\pi^{\lambda} \parallel \pi_{\theta_k})[s] \leq \epsilon$. In this case, the optimal solution to (30)-(31) is $\pi^{\lambda}(a|s)$. The second case is when $D_{\text{KL}}(\pi^{\lambda} \parallel \pi_{\theta_k})[s] > \epsilon$. In this case the optimal is $\pi^{\lambda}(a|s)$ with λ replaced with λ_s, where λ_s is the solution to $D_{\text{KL}}(\pi^{\lambda} \parallel \pi_{\theta_k})[s] = \epsilon$. Thus, an optimal solution to (30)-(31) is given by:

$$\tilde{\pi}^{\lambda}(a|s) = \begin{cases} \dfrac{\pi_{\theta_k}(a|s)}{Z(s)} e^{A^{\pi_{\theta_k}}(s,a)/\lambda} & s \in \Gamma^{\lambda} \\ \dfrac{\pi_{\theta_k}(a|s)}{Z(s)} e^{A^{\pi_{\theta_k}}(s,a)/\lambda_s} & s \notin \Gamma^{\lambda} \end{cases} \tag{35}$$

where $\Gamma^{\lambda} = \{s : D_{\text{KL}}(\pi^{\lambda} \parallel \pi_{\theta_k})[s] \leq \epsilon\}$.

To find the Lagrange multiplier λ, we can then do a line search to find the λ that satisfies:

$$\sum_s d^{\pi_{\theta_k}}(s) D_{\text{KL}}(\tilde{\pi}^{\lambda} \parallel \pi_{\theta_k})[s] = \delta \tag{36}$$

□

A.2 BACKWARD KL CONSTRAINT

The problem (19)-(20) decomposes into separate problems, one for each state $s \in \mathcal{S}$:

$$\underset{\pi(\cdot|s)}{\text{maximize}} \quad \mathbb{E}_{a\sim\pi_{\theta_k}(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a)\right] \tag{37}$$

$$\text{subject to} \quad \mathbb{E}_{a\sim\pi_{\theta_k}(\cdot|s)}\left[\log\frac{\pi_{\theta_k}(a|s)}{\pi(a|s)}\right] \leq \epsilon \tag{38}$$

After some algebra, we see that above optimization problem is equivalent to:

$$\underset{\pi(\cdot|s)}{\text{maximize}} \quad \sum_{a=1}^{K} A^{\pi_{\theta_k}}(s,a)\pi(a|s) \tag{39}$$

$$\text{subject to} \quad -\sum_{a=1}^{K}\pi_{\theta_k}(a|s)\log\pi(a|s) \leq \epsilon' \tag{40}$$

$$\sum_{a=1}^{K}\pi(a|s) = 1 \tag{41}$$

$$\pi(a|s) \geq 0, \quad a = 1,\ldots,K \tag{42}$$

where $\epsilon' = \epsilon + entropy(\pi_{\theta_k})$. (39)-(42) is a convex optimization problem with Slater's condition holding. Strong duality thus holds for the problem (39)-(42). Applying standard Lagrange multiplier arguments, it is easily seen that the solution to (39)-(42) is

$$\pi^*(a|s) = \pi_{\theta_k}(a|s)\frac{\lambda(s)}{\lambda'(s) - A^{\pi_{\theta_k}}(s,a)}$$

where $\lambda(s)$ and $\lambda'(s)$ are constants chosen such that the disaggregegated KL constraint is binding and the sum of the probabilities equals 1. It is easily seen $\lambda(s) > 0$ and $\lambda'(s) > \max_a A^{\pi_{\theta_k}}(s,a)$ □

A.3 L^∞ CONSTRAINT

The problem (24-26) is equivalent to:

$$\underset{\pi(a_1|s_1),\ldots,\pi(a_m|s_m)}{\text{maximize}} \quad \sum_{i=1}^{m} A^{\pi_{\theta_k}}(s_i,a_i)\frac{\pi(a_i|s_i)}{\pi_k(a_i|s_i)} \tag{43}$$

$$\text{subject to} \quad 1-\epsilon \leq \frac{\pi(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} \leq 1+\epsilon \quad i = 1,\ldots,m \tag{44}$$

$$\sum_{i=1}^{m}\left(\frac{\pi(a_i|s_i) - \pi_{\theta_k}(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)}\right)^2 \leq \delta \tag{45}$$

This problem is clearly convex. $\pi_{\theta_k}(a_i|s_i), i = 1,\ldots,m$ is a feasible solution where the inequality constraint holds strictly. Strong duality thus holds according to Slater's constraint qualification. To solve (43)-(45), we can therefore solve the related Lagrangian problem for fixed λ:

$$\underset{\pi(a_1|s_1),\ldots,\pi(a_m|s_m)}{\text{maximize}} \quad \sum_{i=1}^{m}\left[A^{\pi_{\theta_k}}(s_i,a_i)\frac{\pi(a_i|s_i)}{\pi_k(a_i|s_i)} - \lambda\left(\frac{\pi(a_i|s_i) - \pi_{\theta_k}(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)}\right)^2\right] \tag{46}$$

$$\text{subject to} \quad 1-\epsilon \leq \frac{\pi(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} \leq 1+\epsilon \quad i = 1,\ldots,m \tag{47}$$

which is separable and decomposes into m separate problems, one for each s_i:

$$\underset{\pi(a_i|s_i)}{\text{maximize}} \quad A^{\pi_{\theta_k}}(s_i,a_i)\frac{\pi(a_i|s_i)}{\pi_k(a_i|s_i)} - \lambda\left(\frac{\pi(a_i|s_i) - \pi_{\theta_k}(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)}\right)^2 \tag{48}$$

$$\text{subject to} \quad 1-\epsilon \leq \frac{\pi(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} \leq 1+\epsilon \tag{49}$$

The solution to the unconstrained problem (48) without the constraint (49) is:

$$\pi^*(a_i|s_i) = \pi_{\theta_k}(a_i|s_i)\left(1 + \frac{A^{\pi_{\theta_k}}(s_i, a_i)}{2\lambda}\right)$$

Now consider the constrained problem (48)-(49). If $A^{\pi_{\theta_k}}(s_i, a_i) \geq 0$ and $\pi^*(a_i|s_i) > \pi_{\theta_k}(a_i|s_i)(1+\epsilon)$, the optimal solution is $\pi_{\theta_k}(a_i|s_i)(1+\epsilon)$. Similarly, If $A^{\pi_{\theta_k}}(s_i, a_i) < 0$ and $\pi^*(a_i|s_i) < \pi_{\theta_k}(a_i|s_i)(1-\epsilon)$, the optimal solution is $\pi_{\theta_k}(a_i|s_i)(1-\epsilon)$. Rearranging the terms gives Theorem 3. To obtain λ, we can perform a line search over λ so that the constraint (45) is binding. □

B DERIVATIONS THE GRADIENT OF LOSS FUNCTION FOR SPU

Let CE stands for CrossEntropy.

B.1 FORWARD-KL

$$\begin{aligned}
&CE(\pi_\theta||\tilde{\pi}^{\tilde{\lambda}_s})[s] \\
&= -\sum_a \pi_\theta(a|s)\log\tilde{\pi}^{\tilde{\lambda}_s}(a|s) \\
&= -\sum_a \pi_\theta(a|s)\log\left(\frac{\pi_{\theta_k}(a|s)}{Z_{\tilde{\lambda}_s(s)}}e^{A^{\pi_{\theta_k}}(s,a)/\tilde{\lambda}_s}\right) \\
&= -\sum_a \pi_\theta(a|s)\log\left(\frac{\pi_{\theta_k}(a|s)}{Z_{\tilde{\lambda}_s(s)}}\right) - \sum_a \pi_\theta(a|s)\frac{A^{\pi_{\theta_k}}(s,a)}{\tilde{\lambda}_s} \\
&= -\sum_a \pi_\theta(a|s)\log\pi_{\theta_k}(a|s) + \sum_a \pi_\theta(a|s)\log Z_{\tilde{\lambda}_s}(s) - \frac{1}{\tilde{\lambda}_s}\sum_a \pi_{\theta_k}(a|s)\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a) \\
&= CE(\pi_\theta||\pi_{\theta_k})[s] + \log Z_{\tilde{\lambda}_s}(s) - \frac{1}{\tilde{\lambda}_s}\underset{a\sim\pi_{\theta_k}(.|s)}{E}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a)\right] \\
&\Rightarrow \nabla_\theta CE(\pi_\theta||\tilde{\pi}^{\tilde{\lambda}_s})[s] = \nabla_\theta CE(\pi_\theta||\pi_{\theta_k})[s] - \frac{1}{\tilde{\lambda}_s}\underset{a\sim\pi_{\theta_k}(.|s)}{E}\left[\frac{\nabla_\theta\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a)\right] \\
&\Rightarrow \nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \tilde{\pi}^{\tilde{\lambda}_s})[s] = \nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s] - \frac{1}{\tilde{\lambda}_s}\underset{a\sim\pi_{\theta_k}(.|s)}{E}\left[\frac{\nabla_\theta\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a)\right]
\end{aligned}$$

B.2 REVERSE-KL

$$\begin{aligned}
&CE(\pi_\theta||\pi^*)[s] \\
&= -\sum_a \pi_\theta(a|s)\log\pi^*(a|s) \\
&= -\sum_a \pi_\theta(a|s)\log\left(\pi_{\theta_k}(a|s)\frac{\lambda(s)}{\lambda'(s) - A^{\pi_{\theta_k}}(s,a)}\right) \\
&= -\sum_a \pi_\theta(a|s)\log\pi_{\theta_k}(a|s) - \sum_a \pi_\theta(a|s)\log\lambda(s) + \sum_a \pi_\theta(a|s)\log(\lambda'(s) - A^{\pi_{\theta_k}}(s,a)) \\
&= CE(\pi_\theta||\pi_{\theta_k})[s] - \lambda(s) + E_{a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}\log(\lambda'(s) - A^{\pi_{\theta_k}}(s,a))\right] \\
&= CE(\pi_\theta||\pi_{\theta_k})[s] - \lambda(s) - E_{a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}\log\frac{1}{\lambda'(s) - A^{\pi_{\theta_k}}(s,a)}\right] \\
&\Rightarrow \nabla_\theta CE(\pi_\theta||\pi^*)[s] = \nabla_\theta CE(\pi_\theta||\pi_{\theta_k})[s] - E_{a\sim\pi_{\theta_k}}\left[\frac{\nabla_\theta\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}\log\frac{1}{\lambda'(s) - A^{\pi_{\theta_k}}(s,a)}\right] \\
&\Rightarrow \nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \pi^*)[s] = \nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s] - E_{a\sim\pi_{\theta_k}}\left[\frac{\nabla_\theta\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}\log\frac{1}{\lambda'(s) - A^{\pi_{\theta_k}}(s,a)}\right]
\end{aligned}$$

C EXTENSION TO CONTINUOUS STATE AND ACTION SPACES

The methodology developed in the body of this paper also applies to continuous state and action spaces. In this section, we outline the modifications that are necessary for the continuous case.

We first modify the definition of $d^\pi(s)$ by replacing $P_\pi(s_t = s)$ with $\frac{d}{ds}P_\pi(s_t \leq s)$ so that $d^\pi(s)$ becomes a density function over the state space. With this modification, the definition of $\bar{D}_{\text{KL}}(\pi \parallel \pi_k)$ and the approximation (8) are unchanged. The SPU framework described in Section 4 is also unchanged.

Consider now the non-parameterized optimization problem with aggregate and disaggregate constraints (12-14), but with continuous state and action space:

$$\underset{\pi \in \Pi}{\text{maximize}} \quad \int d^{\pi_{\theta_k}}(s) \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} [A^{\pi_{\theta_k}}(s,a)] ds \tag{50}$$

$$\text{subject to} \quad \int d^{\pi_{\theta_k}}(s) D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s] ds \leq \delta \tag{51}$$

$$D_{\text{KL}}(\pi \parallel \pi_{\theta_k})[s] \leq \epsilon \text{ for all } s \tag{52}$$

Theorem 1 holds although its proof needs to be slightly modified as follows. It is straightforward to show that (50-52) remains a convex optimization problem. We can therefore solve (50-52) by solving the Lagrangian (28-29) with the sum replaced with an integral. This problem again decomposes with separate problems for each $s \in \mathcal{S}$ giving exactly the same equations (30-31). The proof then proceeds as in the remainder of the proof of Theorem 1.

Theorem 2 and 3 are also unchanged for continuous action spaces. Their proofs require slight modifications, as in the proof of Theorem 1.

D IMPLEMENTATION DETAILS AND HYPERPARAMETERS

D.1 MUJOCO

As in (Schulman et al., 2017), for Mujoco environments, the policy is parameterized by a fully-connected feed-forward neural network with two hidden layers, each with 64 units and tanh nonlinearities. The policy outputs the mean of a Gaussian distribution with state-independent variable standard deviations, following (Schulman et al., 2015a; Duan et al., 2016). The action dimensions are assumed to be independent. The probability of an action is given by the multivariate Gaussian probability distribution function. The baseline used in the advantage value calculation is parameterized by a similarly sized neural network, trained to minimize the MSE between the sampled states TD$-\lambda$ returns and the their predicted values. For both the policy and baseline network, SPU and TRPO use the same architecture. To calculate the advantage values, we use Generalized Advantage Estimation (Schulman et al., 2015c). States are normalized by dividing the running mean and dividing by the running standard deviation before being fed to any neural networks. The advantage values are normalized by dividing the batch mean and dividing by the batch standard deviation before being used for policy update. The TRPO result is obtained by running the TRPO implementation provided by OpenAI (Dhariwal et al., 2017), commit 3cc7df060800a45890908045b79821a13c4babdb. At every iteration, SPU collects 2048 samples before updating the policy and the baseline network. For both networks, gradient descent is performed using Adam (Kingma & Ba, 2014) with step size 0.0003, minibatch size of 64. The step size is linearly annealed to 0 over the course of training. γ and λ for GAE (Schulman et al., 2015c) are set to 0.99 and 0.95 respectively. For SPU, $\delta, \epsilon, \lambda$ and the maximum number of epochs per iteration are set to 1.2, 0.05, 1.3 and 30 respectively. Training is performed for 1 million timesteps for both SPU and PPO. In the sensitivity analysis, the ranges of values for the hyper-parameters $\delta, \epsilon, \lambda$ and maximum number of epochs are $[0.05, 0.07]$, $[0.01, 0.07]$, $[1.0, 1.2]$ and $[5, 30]$ respectively.

D.2 ATARI

Unless otherwise mentioned, the hyper-parameter values are the same as in subsection D.1. The policy is parameterized by a convolutional neural network with the same architecture as described in Mnih et al. (2015). The output of the network is passed through a relu, linear and softmax layer in that order to give the action distribution. The output of the network is also passed through a different linear layer to give the baseline value. States are normalized by dividing by 255 before being fed into any network. The TRPO result is obtained by running the PPO implementation provided by OpenAI (Dhariwal et al., 2017), commit 3cc7df060800a45890908045b79821a13c4babdb. 8 different processes run in parallel to collect timesteps. At every iteration, each process collects 256 samples before updating the policy and the baseline network. Each process calculates its own update to the network's parameters and the updates are averaged over all processes before being used to update the network's parameters. Gradient descent is performed using Adam (Kingma & Ba, 2014) with step size 0.0001. In each process, random number generators are initialized with a different seed according to the formula $process_seed = experiment_seed + 10000 * process_rank$. Training is performed for 10 million timesteps for both SPU and PPO. For SPU, $\delta, \epsilon, \lambda$ and the maximum number of epochs per iteration are set to 0.02, $\delta/1.3$, 1.1 and 9 respectively.

E ALGORITHMIC DESCRIPTION FOR SPU

Algorithm 1 Algorithmic description of forward-KL non-parameterized SPU

Require: A neural net π_θ that parameterizes the policy.
Require: A neural net V_ϕ that approximates V^{π_θ}.
Require: General hyperparameters: γ, β (advantage estimation using GAE), α (learning rate), N (number of trajectory per iteration), T (size of each trajectory), M (size of training minibatch).
Require: Algorithm-specific hyperparameters: δ (aggregated KL constraint), ϵ (disaggregated constraint), λ, ζ (max number of epoch).

for k = 1, 2, ... **do**
  under policy π_{θ_k}, sample N trajectories, each of size T $(s_{it}, a_{it}, r_{it}, s_{i(t+1)})$, $i = 1, \ldots, N, t = 1, \ldots, T$
  Using any advantage value estimation scheme, estimate A_{it}, $i = 1, \ldots, N, t = 1, \ldots, T$
  $\theta \leftarrow \theta_k$
  $\phi \leftarrow \phi_k$
  for ζ epochs **do**
    Sample M samples from the N trajectories, giving $\{s_1, a_1, A_1, \ldots, s_M, a_M, A_M\}$
    $L(\phi) = \frac{1}{M}\sum_m (V^{targ}(s_m) - V_\phi(s_m))^2$
    $\phi \leftarrow \phi - \alpha \nabla_\phi L(\phi)$
    $L(\theta) = \frac{1}{M}\sum_m \left[\nabla_\theta D_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_m] - \frac{1}{\lambda}\frac{\nabla_\theta \pi_\theta(a_m|s_m)}{\pi_{\theta_k}(a_m|s_m)} A_m\right] \mathbb{1}_{D_{\mathrm{KL}}(\pi_\theta \| \pi_{\theta_k})[s_m] \leq \epsilon}$
    $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$
    if $\frac{1}{m}\sum_m D_{\mathrm{KL}}(\pi \parallel \pi_{\theta_k})[s_m] > \delta$ **then**
      Break out of for loop
  $\theta_{k+1} \leftarrow \theta$
  $\phi_{k+1} \leftarrow \phi$

F EXPERIMENTAL RESULTS

F.1 RESULTS ON MUJOCO FOR 3 MILLION TIMESTEPS

TRPO and SPU were trained for 1 million timesteps to obtain the results in section 6. To ensure that SPU is not only better than TRPO in terms of performance gain early during training, we further retrain both policies for 3 million timesteps. Again here, SPU outperforms TRPO by 28%. Figure 3 illustrates the performance on each environment.

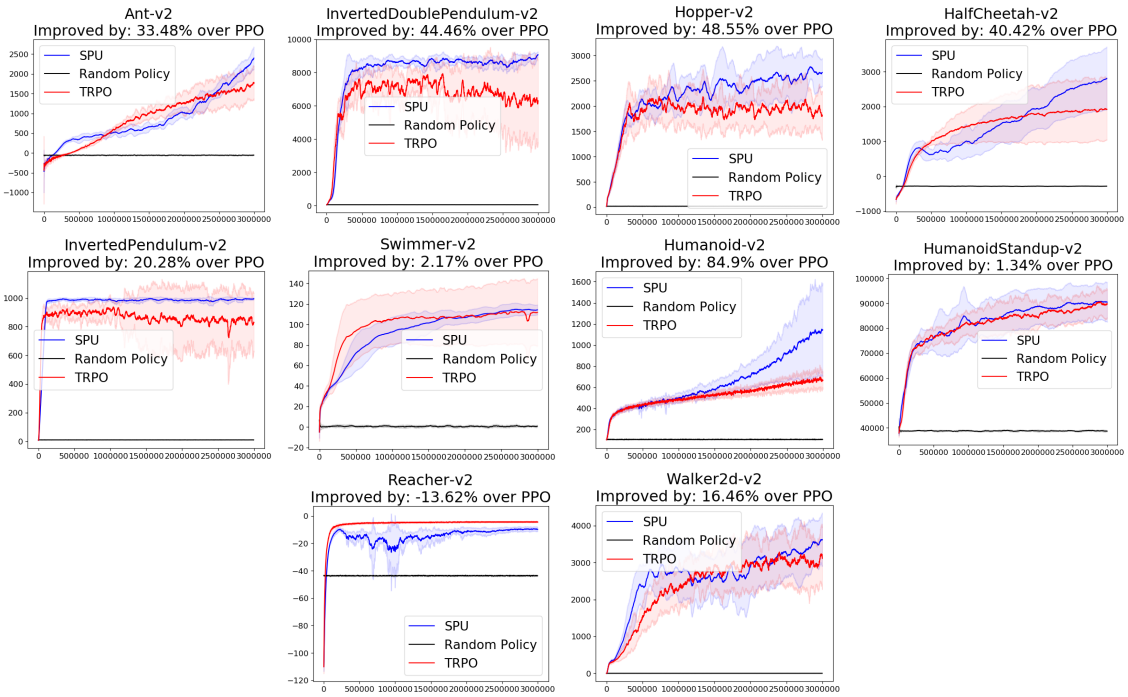

Figure 3: Performance of SPU versus TRPO on 10 Mujoco environments in 3 million timesteps. The x-axis indicates timesteps. The y-axis indicates the average episode reward of the last 100 episodes.

F.2 Sensitivity Analysis CDF for Mujoco

When values for SPU hyper-parameter are randomly sampled as is explained in subsection 6.3, the percentage improvement of SPU over TRPO becomes a random variable. Figure 4 illustrates the CDF of this random variable.

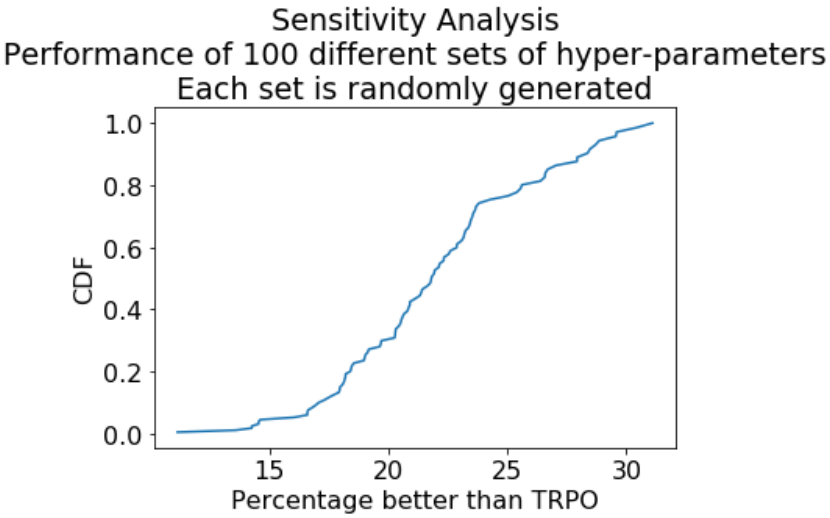

Figure 4: Sensitivity Analysis for SPU

F.3 PPO RESULT ON MUJOCO

We provides results for PPO versus TRPO to demonstrates that both have similar performance in Mujoco. In each environments, the results are averaged over 10 seeds. In terms of averaged final performance, PPO outperforms TRPO by 9%.

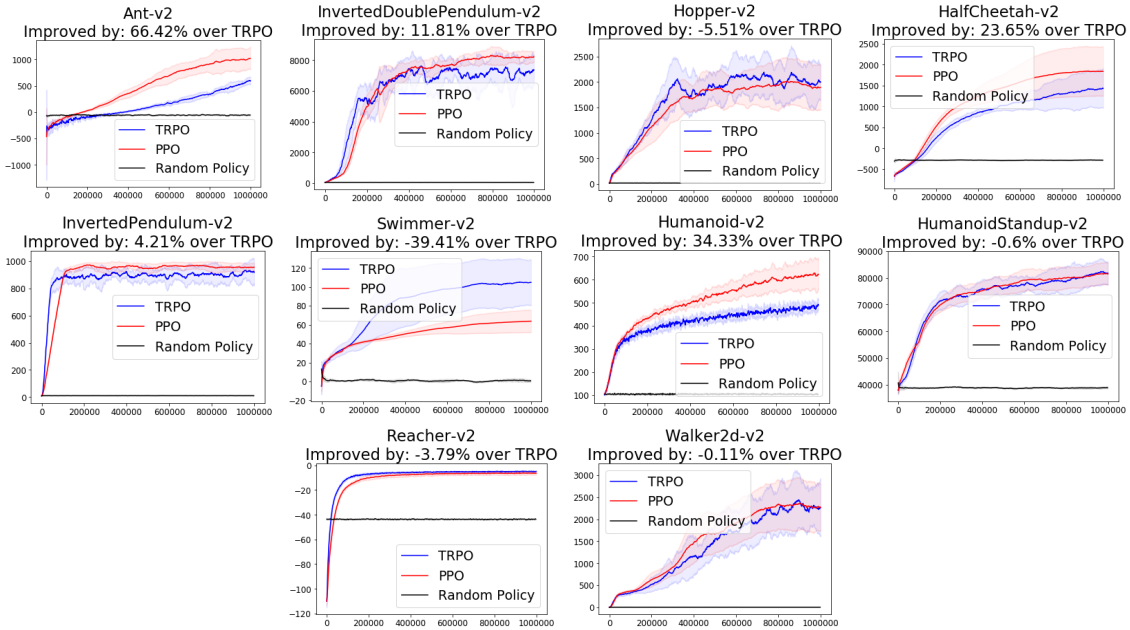

Figure 5: Performance of PPO versus TRPO on 10 Mujoco environments in 1 million timesteps. The x-axis indicates timesteps. The y-axis indicates the average episode reward of the last 100 episodes.

F.4 ATARI RESULTS

Figure 6, Figure 7 and Figure 8 illustrate the performance of SPU vs PPO throughout training in 60 Atari games.

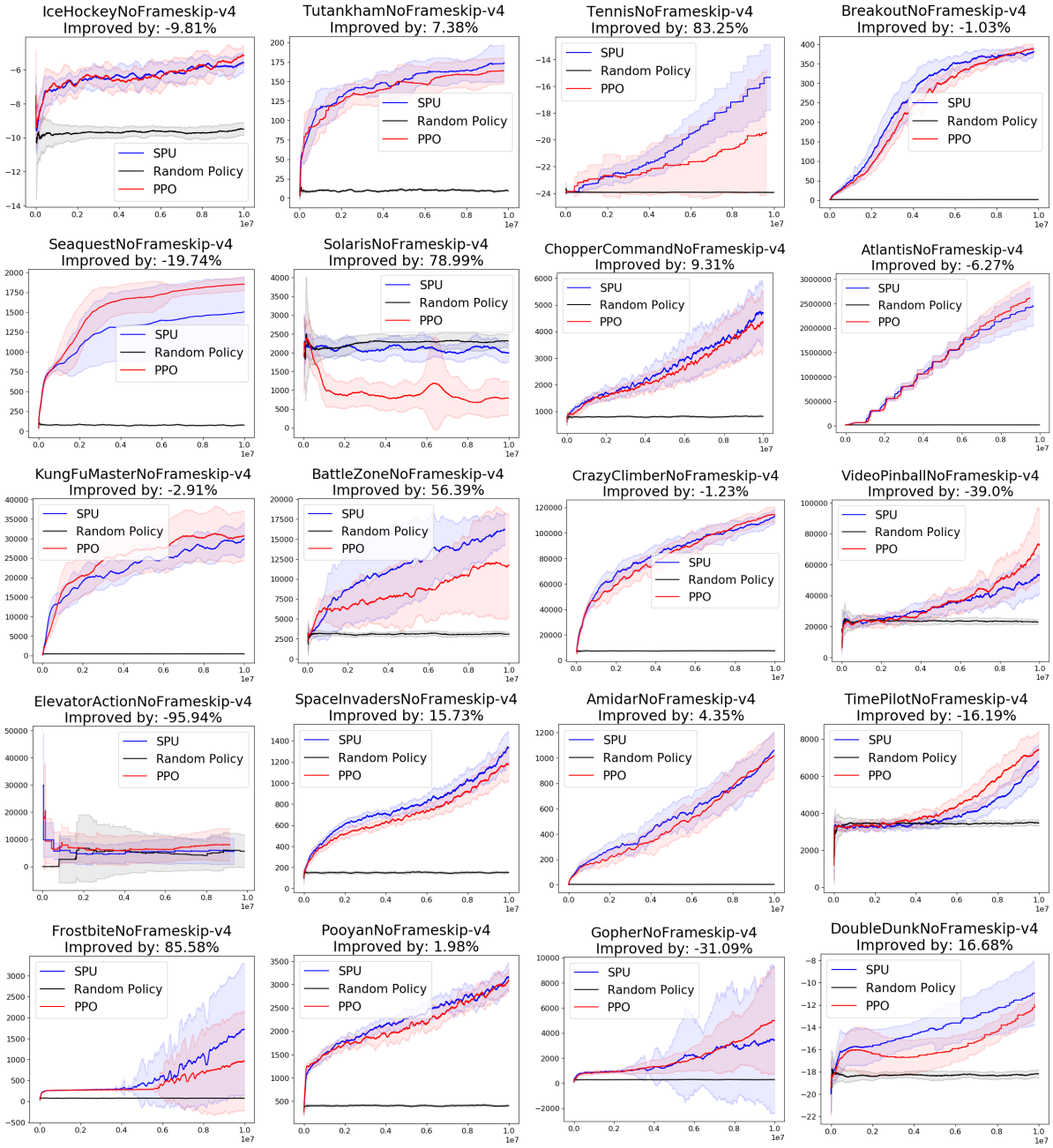

Figure 6: Atari results (Part 1) for SPU vs PPO. The x-axis indicates timesteps. The y-axis indicates the average episode reward of the last 100 episodes.

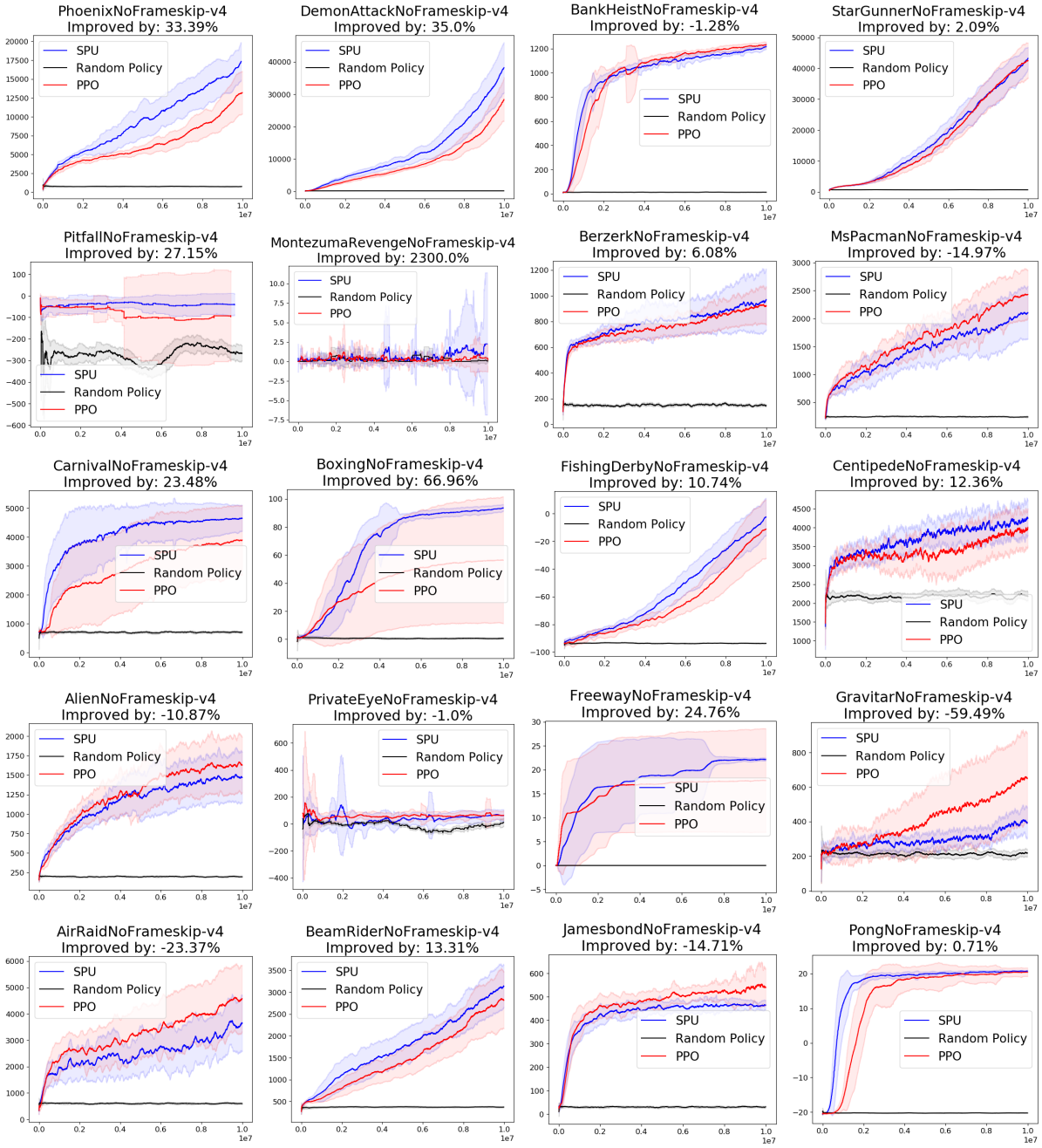

Figure 7: Atari results (Part 2) for SPU vs PPO. The x-axis indicates timesteps. The y-axis indicates the average episode reward of the last 100 episodes.

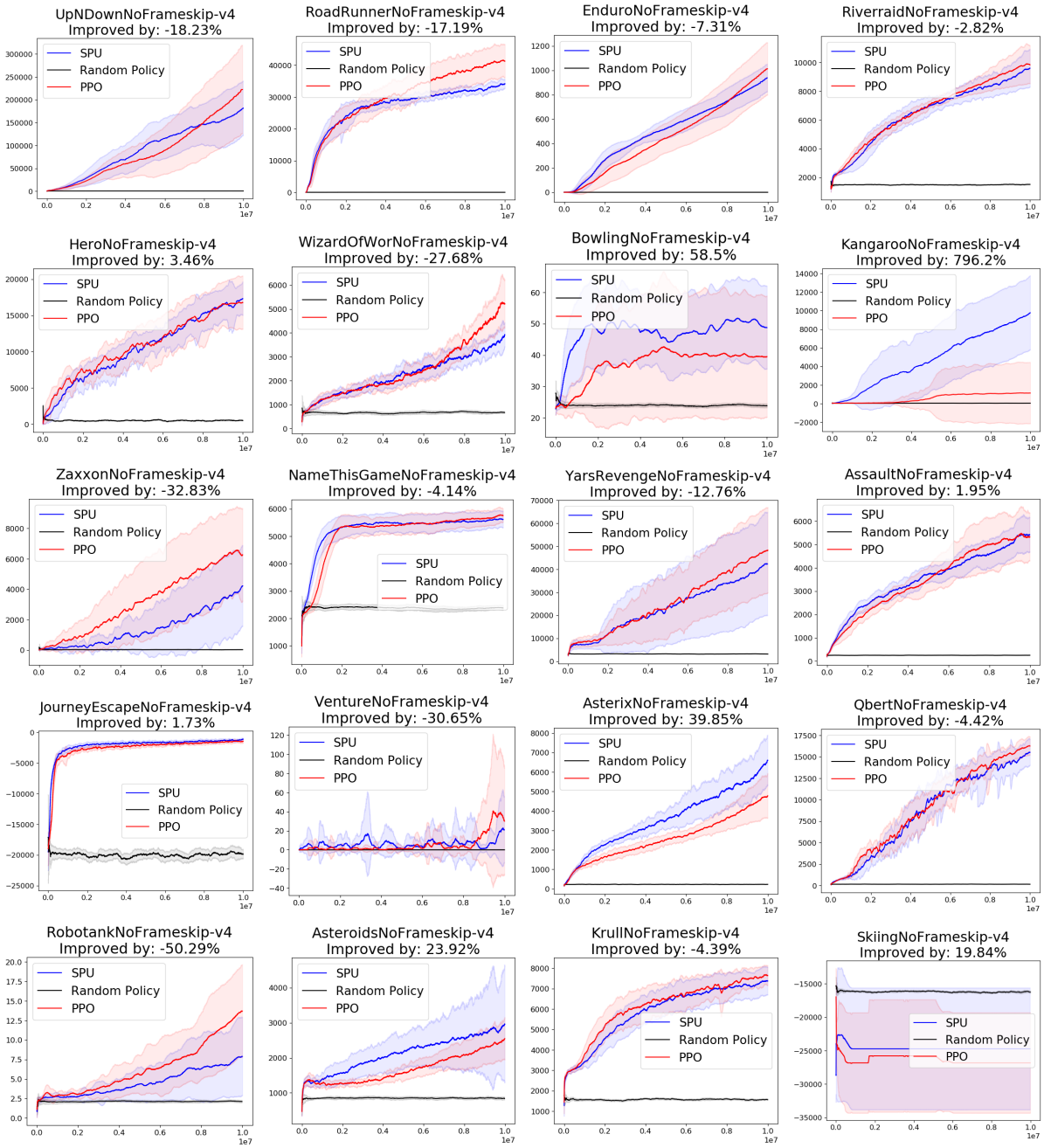

Figure 8: Atari results (Part 3) for SPU vs PPO. The x-axis indicates timesteps. The y-axis indicates the average episode reward of the last 100 episodes.