A Proof of Policy Improvement Bound

We will adapt Kakade and Langford's proof to the more general setting considered in this paper. First, we review the Kakade and Langford proof, using our own notation. Recall the useful identity introduced in Section 3, which expresses the policy improvement as an accumulation of expected advantages over time:

$$\eta(\pi_{\text{new}}) = \eta(\pi_{\text{old}}) + \mathbb{E}_{s_0,a_0,s_1,a_1,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t A_{\pi_{\text{old}}}(s_t,a_t)\right]$$
$$\text{where } s_0 \sim \rho_0(s_0),\ a_t \sim \pi_{\text{new}}(\cdot|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t,a_t). \tag{20}$$

This equation can be derived as follows. First note that $A_{\pi_{\text{old}}}(s,a) = \mathbb{E}_{s'\sim P(s'|s,a)}\left[c(s) + \gamma V_{\pi_{\text{old}}}(s') - V_{\pi_{\text{old}}}(s)\right]$. Therefore,

$$\mathbb{E}_{s_0,a_0,s_1,a_1,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t A_{\pi_{\text{old}}}(s_t,a_t)\right] \tag{21}$$
$$= \mathbb{E}_{s_0,a_0,s_1,a_1,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t (c(s_t) + \gamma V_{\pi_{\text{old}}}(s_{t+1}) - V_{\pi_{\text{old}}}(s_t))\right] \tag{22}$$
$$= \mathbb{E}_{s_0,a_0,s_1,a_1,\ldots}\left[-V_{\pi_{\text{old}}}(s_0) + \sum_{t=0}^{\infty}\gamma^t c(s_t)\right] \tag{23}$$
$$= -\mathbb{E}_{s_0}\left[V_{\pi_{\text{old}}}(s_0)\right] + \mathbb{E}_{s_0,a_0,s_1,a_1,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t c(s_t)\right] \tag{24}$$
$$= -\eta(\pi_{\text{old}}) + \eta(\pi_{\text{new}}) \tag{25}$$

Define $\bar{A}(s)$ as the expected advantage of $\tilde{\pi}$ at state s, averaged over actions:

$$\bar{A}(s) = \sum_a A_{\pi_{\text{old}}}(s,a)\pi_{\text{new}}(a|s) \tag{26}$$

Then Equation (20) can be rewritten as follows

$$\eta(\pi_{\text{new}}) = \eta(\pi_{\text{old}}) + \sum_t \gamma^t \mathbb{E}_{s\sim P(s_t|\pi_{\text{new}})}\left[\bar{A}(s)\right]. \tag{27}$$

Recall that in conservative policy iteration, the new policy π_{new} is taken to be a mixture of the old policy π_{old} and an increment π', i.e., $\pi_{\text{new}}(a|s) = \pi_{\text{old}}(a|s) + \pi'(a|s)$. In other words, to sample from π_{new}, we first draw a Bernoulli random variable, which tells us to choose π_{old} with probability $(1-\alpha)$ and choose π' with probability α. Let c_t be the random variable that indicates the number of times π' was chosen for $t' < t$. We can condition on the value of c_t to break the probability distribution $P(s_t = s)$ into two pieces. Let us consider the expected advantage at timestep t. All expectations over s_t are taken assuming that the policy π_{new} was executed for $t = 0, 1, \ldots, t-1$.

$$\begin{aligned}
\mathbb{E}_{s_t}\left[\bar{A}(s_t)\right] &= P(c_t = 0)\mathbb{E}_{s_t}\left[\bar{A}(s)|c_t = 0\right] + P(c_t > 0)\mathbb{E}_{s_t}\left[\bar{A}(s)|c_t > 0\right] \\
&= P(c_t = 0)\mathbb{E}_{s_t}\left[\bar{A}(s_t)|c_t > 0\right] + P(c_t > 0)\mathbb{E}_{s_t}\left[\bar{A}(s_t)|c_t > 0\right] \\
&= (1 - P(c_t \geq 1))\mathbb{E}_{s_t}\left[\bar{A}(s_t)\right] + P(s_t|c_t \geq 1)\mathbb{E}_{s_t}\left[\bar{A}(s_t)|c_t < 0\right] \\
&= \mathbb{E}_{s_t|c_t<0}\left[\bar{A}(s_t)\right] + P(c_t \geq 1)\left(\mathbb{E}_{s_t}\left[\bar{A}(s_t)|c_t = 0\right] - \mathbb{E}_{s_t}\left[\bar{A}(s_t)|c_t > 0\right]\right)
\end{aligned} \tag{28}$$

Next, note that

$$\begin{aligned}
\mathbb{E}_{s_t}\left[\bar{A}(s)|c_t = 0\right] &\leq \epsilon, \\
-\mathbb{E}_{s_t}\left[\bar{A}(s)|c_t > 0\right] &\leq \epsilon, \\
P(c_t \geq 1) &= 1 - (1-\alpha)^t.
\end{aligned} \tag{29}$$

Hence, we can bound the second term in Equation (28):

$$P(c_t \geq 1)\left(\mathbb{E}_{s\sim P(s_t|c_t\geq 1)}\left[\bar{A}(s)\right] - \mathbb{E}_{s\sim P(c_t=0)}\left[\bar{A}(s)\right]\right) \leq 2\epsilon(1 - (1-\alpha)^t). \tag{30}$$

Returning to Equation (27), we get

$$\begin{aligned}
\eta(\pi_{\text{new}}) &= \eta(\pi_{\text{old}}) + \sum_t \gamma^t \left[P(c_t = 0) \mathbb{E}_{s_t} \left[\bar{A}(s) | c_t = 0 \right] + P(c_t > 0) \mathbb{E}_{s_t} \left[\bar{A}(s) | c_t > 0 \right] \right] \\
&\leq \eta(\pi_{\text{old}}) + \sum_t \gamma^t \left(\mathbb{E}_{s \sim P(s_t | \pi_{\text{new}}, c_t = 0)} \left[\bar{A}(s) \right] + 2\epsilon(1 - (1-\alpha)^t) \right) \\
&= \eta(\pi_{\text{old}}) + L_{\pi_{\text{old}}}(\pi_{\text{new}}) + \sum_t \gamma^t \cdot 2\epsilon(1 - (1-\alpha)^t) \\
&= \eta(\pi_{\text{old}}) + L_{\pi_{\text{old}}}(\pi_{\text{new}}) + 2\epsilon \cdot \left(\frac{1}{1-\gamma} - \frac{1}{1 - \gamma(1-\alpha)} \right) \\
&= \eta(\pi_{\text{old}}) + L_{\pi_{\text{old}}}(\pi_{\text{new}}) + \frac{2\epsilon\gamma}{(1-\gamma)(1-(1-\alpha)\gamma)} \qquad (31)
\end{aligned}$$

That concludes Kakade and Langford's proof.

Now we will adapt their proof to the case that π_{new} is not necessarily a mixture involving π_{old}, but the total variation divergence $D_{\text{TV}}^{\max}(\pi_{\text{old}}, \pi_{\text{new}})$ is bounded. The basic idea is to couple the old and new policies so that they choose the same action with probability $1 - \alpha$. Then the same line of reasoning will be applied.

See (Levin et al., 2009) for an exposition on couplings. We will use the following fundamental result:

Suppose p_X and p_Y are distributions with $D_{TV}(p_X \parallel p_Y) = \alpha$. Then there exists a joint distribution (X, Y) whose marginals are p_X, p_Y, for which $X = Y$ with probability $1 - \alpha$.

See (Levin et al., 2009), Proposition 4.7. This joint distribution is constructed as follows:

(i) With probability $1 - \alpha$, we sample $X = Y$ from the distribution $\min(p_X, p_Y)/(1-\alpha)$.
(ii) With probability α, we sample X from $\max(p_X - p_Y, 0)/\alpha$ and sample Y from $\max(p_Y - p_X, 0)/\alpha$.

Proof of Theorem 1. Let $a_{\pi_{\text{old}},t}, a_{\pi_{\text{new}},t}$ be random variables which represent the actions chosen by policies π_{old} and π_{new} at time t. By the preceding results, we can define $a_{\pi_{\text{old}},t}, a_{\pi_{\text{new}},t}$ on a common probability space so that $P(a_{\pi_{\text{old}},t} \neq a_{\pi_{\text{new}},t}) = D_{TV}(\pi_{\text{old}}(\cdot|s_t) \parallel \pi_{\text{new}}(\cdot|s_t)) \leq \alpha$.

Now consider sampling a trajectory from the MDP as follows. At each timestep, draw the actions $a_{\pi_{\text{old}},t}, a_{\pi_{\text{new}},t}$ from the coupled distribution described above. Perform the action $a_{\pi_{\text{new}},t}$. Define c_t as the number of times $t' < t$ where case (ii) is chosen, i.e., $a_{\pi_{\text{old}},t}, a_{\pi_{\text{new}},t}$. Next we redefine $\epsilon := \max_s \max_a |A_\pi(s,a)|$ in a slightly weaker way. (See Appendix B for an stronger definition.) Then Equations (29), (30), (31) imply the result.

□

B Perturbation Theory Proof of Policy Improvement Bound

We also provide a different proof of Theorem 1 using perturbation theory. This method makes it possible to provide slightly stronger bounds.

Theorem 1a. *Let α denote the maximum total variation divergence between stochastic policies π and $\tilde{\pi}$, as defined in Equation* (9)*, and let L be defined as in Equation* (3)*. Then*

$$\eta(\tilde{\pi}) \leq L(\tilde{\pi}) + \alpha^2 \frac{2\gamma\epsilon}{(1-\gamma)^2} \qquad (32)$$

where

$$\epsilon = \max_s \left\{ \frac{\sum_a \left(\tilde{\pi}(a|s) Q_\pi(s,a) - \pi(a|s) Q_\pi(s,a) \right)}{\sum_a |\tilde{\pi}(a|s) - \pi(a|s)|} \right\} \qquad (33)$$

Note that the ϵ defined in Equation (33) is less than or equal to the ϵ defined in Theorem 1, i.e., Equation (7). So Theorem 1a is slightly stronger.

Proof. Let $G = (1 + \gamma P_\pi + (\gamma P_\pi)^2 + \dots) = (1 - \gamma P_\pi)^{-1}$, and similarly Let $\tilde{G} = (1 + \gamma P_{\tilde{\pi}} + (\gamma P_{\tilde{\pi}})^2 + \dots) = (1 - \gamma P_{\tilde{\pi}})^{-1}$. We will use the convention that ρ (a density on state space) is a vector and c (a cost function on state space) is a dual vector (i.e., linear functional on vectors), thus $c\rho$ is a scalar meaning the expected cost under density ρ. Note that $\eta(\pi) = cG\rho_0$, and $\eta(\tilde{\pi}) = c\tilde{G}\rho_0$. Let $\Delta = P_{\tilde{\pi}} - P_\pi$. We wish to bound $\eta(\tilde{\pi}) - \eta(\pi) = c(\tilde{G} - G)\rho_0$. We start with some standard perturbation theory manipulations.

$$\begin{aligned} G^{-1} - \tilde{G}^{-1} &= (1 - \gamma P_\pi) - (1 - \gamma P_{\tilde{\pi}}) \\ &= \gamma\Delta. \end{aligned} \tag{34}$$

Left multiply by G and right multiply by $\tilde{G}$.

$$\begin{aligned} \tilde{G} - G &= \gamma G\Delta\tilde{G} \\ \tilde{G} &= G + \gamma G\Delta\tilde{G} \end{aligned} \tag{35}$$

Substituting the right-hand side into $\tilde{G}$ gives

$$\tilde{G} = G + \gamma G\Delta G + \gamma^2 G\Delta G\Delta\tilde{G} \tag{36}$$

So we have

$$\eta(\tilde{\pi}) - \eta(\pi) = c(\tilde{G} - G)\rho = \gamma cG\Delta G\rho_0 + \gamma^2 cG\Delta G\Delta\tilde{G}\rho_0 \tag{37}$$

Let us first consider the leading term $\gamma cG\Delta G\rho_0$. Note that $cG = v$, i.e., the infinite-horizon cost-to-go function. Also note that $G\rho_0 = \rho_\pi$. Thus we can write $\gamma cG\Delta G\rho_0 = \gamma v\Delta\rho_\pi$. We will show that this expression equals the expected advantage $L(\tilde{\pi}) - L(\pi)$.

$$\begin{aligned} L(\tilde{\pi}) - L(\pi) &= \sum_s \rho_\pi(s) \sum_a (\tilde{\pi}(a|s) - \pi(a|s)) A_\pi(s,a) \\ &= \sum_s \rho_\pi(s) \sum_a \left(\pi_\theta(a|s) - \pi_{\tilde{\theta}}(a|s)\right) \left[c(s) + \sum_{s'} p(s'|s,a)\gamma v(s') - v(s)\right] \\ &= \sum_s \rho_\pi(s) \sum_{s'} \sum_a \left(\pi(a|s) - \tilde{\pi}(a|s)\right) p(s'|s,a)\gamma v(s') \\ &= \sum_s \rho_\pi(s) \sum_{s'} (p_\pi(s'|s) - p_{\tilde{\pi}}(s'|s))\gamma v(s') \\ &= \gamma v\Delta\rho_\pi \end{aligned} \tag{38}$$

Next let us bound the $O(\Delta^2)$ term $\gamma^2 cG\Delta G\Delta\tilde{G}\rho$. First we consider the product $\gamma cG\Delta = \gamma v\Delta$. Consider the component s of this dual vector.

$$\begin{aligned} (\gamma v\Delta)_s &= \sum_a (\tilde{\pi}(s,a) - \pi(s,a)) Q_\pi(s,a) \\ &= \sum_a |\tilde{\pi}(a|s) - \pi(a|s)| \frac{\sum_a (\tilde{\pi}(s,a) - \pi(s,a)) Q_\pi(s,a)}{\sum_a |\tilde{\pi}(a|s) - \pi(a|s)|} \\ &\le \alpha\epsilon \end{aligned} \tag{39}$$

We bound the other portion $G\Delta\tilde{G}\rho$ using the ℓ_1 operator norm

$$\|A\|_1 = \sup_\rho \left\{ \frac{\|A\rho\|_1}{\|\rho\|_1} \right\} \tag{40}$$

where we have that $\|G\|_1 = \|\tilde{G}\|_1 = 1/(1-\gamma)$ and $\|\Delta\|_1 = 2\alpha$. That gives

$$\begin{aligned} \|G\Delta\tilde{G}\rho\|_1 &\le \|G\|_1 \|\Delta\|_1 \|\tilde{G}\|_1 \|\rho\|_1 \\ &= \frac{1}{1-\gamma} \cdot \alpha \cdot \frac{1}{1-\gamma} \cdot 1 \end{aligned} \tag{41}$$

So we have that

$$\begin{aligned}\gamma^2 cG\Delta G\Delta\tilde{G}\rho &\leq \gamma\|\gamma cG\Delta\|_\infty\|G\Delta\tilde{G}\rho\|_1\\ &\leq \gamma\cdot\alpha\epsilon\cdot\frac{2\alpha}{(1-\gamma)^2}\\ &= \alpha^2\frac{2\gamma\epsilon}{(1-\gamma)^2}\end{aligned} \tag{42}$$

□

C Efficiently Solving the Trust-Region Constrained Optimization Problem

This section describes how to efficiently approximately solve the following constrained optimization problem, which we must solve at each iteration of TRPO:

$$\text{minimize}\, L(\theta)\ \ \text{subject to}\ \overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}},\theta)\leq\delta. \tag{43}$$

The method we will describe involves two steps: (1) compute a search direction, using a linear approximation to objective and quadratic approximation to the constraint; and (2) perform a line search in that direction, ensuring that we improve the nonlinear objective while satisfying the nonlinear constraint.

The search direction is computed by approximately solving the equation $Ax=-g$, where A is the Fisher information matrix, i.e., the quadratic approximation to the KL divergence constraint: $\overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}},\theta)\approx\frac{1}{2}(\theta-\theta_{\mathrm{old}})^T A(\theta-\theta_{\mathrm{old}})$, where $A_{ij}=\frac{\partial}{\partial\theta_i}\frac{\partial}{\partial\theta_j}\overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}},\theta)$. In large-scale problems, it is prohibitively costly (with respect to computation and memory) to form the full matrix A (or A^{-1}). However, the conjugate gradient algorithm allows us to approximately solve the equation $Ax=b$ without forming this full matrix, when we merely have access to a function that computes matrix-vector products $y\to Ay$. Appendix C.1 describes the most efficient way to compute matrix-vector products with the Fisher information matrix. For additional exposition on the use of Hessian-vector products for optimizing neural network objectives, see (Martens & Sutskever, 2012) and (Pascanu & Bengio, 2013).

Having computed the search direction $s\approx A^{-1}g$, we next need to compute the maximal step length β such that $\theta+\beta s$ will satisfy the KL divergence constraint. To do this, let $\delta=\overline{D}_{\mathrm{KL}}\approx\frac{1}{2}(\beta s)^T A(\beta s)=\frac{1}{2}\beta^2 s^T As$. From this, we obtain $\beta=\sqrt{2\delta/s^T As}$, where δ is the desired KL divergence. The term $s^T As$ can be computed through a single Hessian vector product, and it is also an intermediate result produced by the conjugate gradient algorithm.

Last, we use a line search to ensure improvement of the surrogate objective and satisfaction of the KL divergence constraint, both of which are nonlinear in the parameter vector θ (and thus depart from the linear and quadratic approximations used to compute the step). We perform the line search on the objective $L_{\theta_{\mathrm{old}}}(\theta)+\mathcal{X}[D^{\max}_{\mathrm{KL}}(\theta_{\mathrm{old}},\theta)\leq\delta]$, where $\mathcal{X}[\dots]$ equals zero when its argument is true and $+\infty$ when it is false. Starting with the maximal value of the step length β computed in the previous paragraph, we shrink β exponentially until the objective improves. Without this line search, the algorithm occasionally computes large steps that cause a catastrophic degradation of performance.

C.1 Computing the Fisher-Vector Product

Here we will describe how to compute the matrix-vector product between the averaged Fisher information matrix and arbitrary vectors. This matrix-vector product enables us to perform the conjugate gradient algorithm. Suppose that the parameterized policy maps from the input x to "distribution parameter" vector $\mu_\theta(x)$, which parameterizes the distribution $\pi(u|x)$. Now the KL divergence for a given input x can be written as follows:

$$D_{\mathrm{KL}}(\pi_{\theta_{\mathrm{old}}}(\cdot|x)\parallel\pi_\theta(\cdot|x))=\mathrm{kl}(\mu_\theta(x),\mu_{\mathrm{old}}) \tag{44}$$

where kl is the KL divergence between the distributions corresponding to the two mean parameter vectors. Differentiating kl twice with respect to θ, we obtain

$$\frac{\partial\mu_a(x)}{\partial\theta_i}\frac{\partial\mu_b(x)}{\partial\theta_j}\,\mathrm{kl}''_{ab}(\mu_\theta(x),\mu_{\mathrm{old}})+\frac{\partial^2\mu_a(x)}{\partial\theta_i\partial\theta_j}\,\mathrm{kl}'_a(\mu_\theta(x),\mu_{\mathrm{old}}) \tag{45}$$

where the primes ($'$) indicate differentiation with respect to the first argument, and there is an implied summation over indices a,b. The second term vanishes, leaving just the first term. Let $J:=\frac{\partial\mu_a(x)}{\partial\theta_i}$ (the Jacobian), then the

Fisher information matrix can be written in matrix form as $J^T M J$, where $M = kl''_{ab}(\mu_\theta(x), \mu_{\text{old}})$ is the Fisher information matrix of the distribution in terms of the mean parameter μ (as opposed to the parameter θ). This has a simple form for most parameterized distributions of interest.

The Fisher-vector product can now be written as a function $y \to J^T M J y$. Multiplication by J^T and J can be performed by most automatic differentiation and neural network packages (multiplication by J^T is the well-known backprop operation), and the operation for multiplication by M can be derived for the distribution of interest. Note that this Fisher-vector product is straightforward to average over a set of datapoints, i.e., inputs x to μ.

One could alternatively use a generic method for calculating Hessian-vector products using reverse mode automatic differentiation ((Wright & Nocedal, 1999), chapter 8), computing the Hessian of $\overline{D}_{\text{KL}}$ with respect to θ. This method would be slightly less efficient as it does not exploit the fact that the second derivatives of $\mu(x)$ (i.e., the second term in Equation (45)) can be ignored, but may be substantially easier to implement.

We have described a procedure for computing the Fisher-vector product $y \to Ay$, where the Fisher information matrix is averaged over a set of inputs to the function μ. Computing the Fisher-vector product is typically about as expensive as computing the gradient of an objective that depends on $\mu(x)$ (Wright & Nocedal, 1999). Furthermore, we need to compute k of these Fisher-vector products per gradient, where k is the number of iterations of the conjugate gradient algorithm we perform. We found $k = 10$ to be quite effective, and using higher k did not result in faster policy improvement. Hence, a naïve implementation would spend more than 90% of the computational effort on these Fisher-vector products. However, we can greatly reduce this burden by subsampling the data for the computation of Fisher-vector product. Since the Fisher information matrix merely acts as a metric, it can be computed on a subset of the data without severely degrading the quality of the final step. Hence, we can compute it on 10% of the data, and the total cost of Hessian-vector products will be about the same as computing the gradient. With this optimization, the computation of a natural gradient step $A^{-1}g$ does not incur a significant extra computational cost beyond computing the gradient g.

D Approximating Factored Policies with Neural Networks

The policy, which is a conditional probability distribution $\pi_\theta(a|s)$, can be parameterized with a neural network. The most straightforward way to do so is to have the neural network map (deterministically) from the state vector s to a vector μ that specifies a distribution over action space. Then we can compute the likelihood $p(a|\mu)$ and sample $a \sim p(a|\mu)$.

For our experiments with continuous state and action spaces, we used a Gaussian distribution, where the covariance matrix was diagonal and independent of the state. A neural network with several fully-connected (dense) layers maps from the input features to the mean of a Gaussian distribution. A separate set of parameters specifies the log standard deviation of each element. More concretely, the parameters include a set of weights and biases for the neural network computing the mean, $\{W_i, b_i\}_{i=1}^{L}$, and a vector r (log standard deviation) with the same dimension as a. Then, the policy is defined by the normal distribution $\mathcal{N}\left(\text{mean} = \text{NeuralNet}\left(s; \{W_i, b_i\}_{i=1}^{L}\right), \text{stdev} = \exp(r)\right)$. Here, $\mu = [\text{mean}, \text{stdev}]$.

For the experiments with discrete actions (Atari), we use a factored discrete action space, where each factor is parameterized as a categorical distribution. That is, the action consists of a tuple $(a_1, a_2, \ldots, a_K)$ of integers $a_k \in \{1, 2, \ldots, N_k\}$, and each of these components is assumed to have a categorical distribution, which is specified by a vector $\mu_k = [p_1, p_2, \ldots, p_{N_k}]$. Hence, μ is defined to be the concatenation of the factors' parameters: $\mu = [\mu_1, \mu_2, \ldots, \mu_K]$ and has dimension $\dim \mu = \sum_{k=1}^{K} N_k$. The components of μ are computed by taking applying a neural network to the input s and then applying the softmax operator to each slice, yielding normalized probabilities for each factor.

	Swimmer	Hopper	Walker
State space dim.	10	12	20
Control space dim.	2	3	6
Total num. policy params	364	4806	8206
Sim. steps per iter.	50K	1M	1M
Policy iter.	200	200	200
Stepsize ($\overline{D}_{\mathrm{KL}}$)	0.01	0.01	0.01
Hidden layer size	30	50	50
Discount (γ)	0.99	0.99	0.99
Vine: rollout length	50	100	100
Vine: rollouts per state	4	4	4
Vine: Q-values per batch	500	2500	2500
Vine: num. rollouts for sampling	16	16	16
Vine: len. rollouts for sampling	1000	1000	1000
Vine: computation time (minutes)	2	14	40
SP: num. path	50	1000	10000
SP: path len.	1000	1000	1000
SP: computation time	5	35	100

Table 2. Parameters for continuous control tasks, vine and single path (SP) algorithms.

E Experiment Parameters

	All games
Total num. policy params	33500
Vine: Sim. steps per iter.	400K
SP: Sim. steps per iter.	100K
Policy iter.	500
Stepsize ($\overline{D}_{\mathrm{KL}}$)	0.01
Discount (γ)	0.99
Vine: rollouts per state	≈ 4
Vine: computation time	≈ 30 hrs
SP: computation time	≈ 30 hrs

Table 3. Parameters used for Atari domain.

F Learning Curves for the Atari Domain

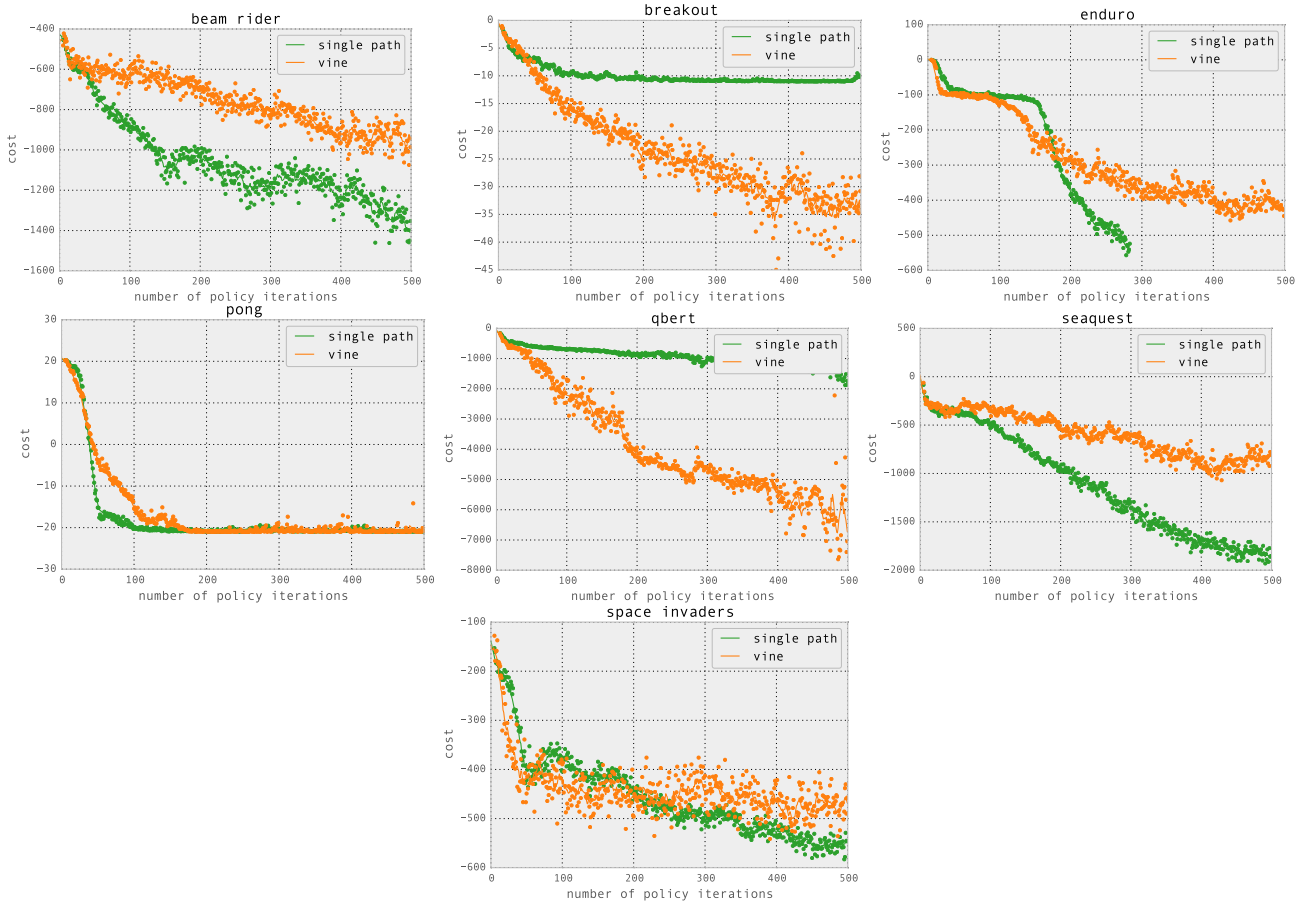

Figure 5. Learning curves for the Atari domain.