SEERL: SAMPLE EFFICIENT ENSEMBLE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Ensemble learning is a very prevalent method employed in machine learning. The relative success of ensemble methods is attributed to its ability to tackle a wide range of instances and complex problems that require different low-level approaches. However, ensemble methods are relatively less popular in reinforcement learning owing to the high sample complexity and computational expense involved. We present a new training and evaluation framework for model-free algorithms that use ensembles of policies obtained from a single training instance. These policies are diverse in nature and are learned through directed perturbation of the model parameters at regular intervals. We show that learning an adequately diverse set of policies is required for a good ensemble while extreme diversity can prove detrimental to overall performance. We evaluate our approach to challenging discrete and continuous control tasks and also discuss various ensembling strategies. Our framework is substantially sample efficient, computationally inexpensive and is seen to outperform various baseline methods, including other ensemble approaches.

1 INTRODUCTION

Deep reinforcement learning over the years has made considerable advancements with applications across a variety of domains – from learning to play Atari 2600 suite from raw visual inputs (Mnih et al. (2015)), mastering the game of Go (Silver et al. (2017)), learning locomotion skills for robotics (Schulman et al. (2015b;a); Lillicrap et al. (2017)) and most recently, the development of Alpha Fold (Evans et al. (2018)) to predict the 3D structure of a protein solely based on its genetic sequence.
However, creating a single agent that performs well, is sample efficient and robust, is a challenging task. To improve the performance on a task, one requires more samples which leads to reduced sample efficiency. On the other hand, increasing robustness is associated with reducing the variance in performance, which is difficult owing to the lack of repeatability when using deep neural networks. However, without deep neural networks, the required performance and generalization on many tasks cannot be achieved. We look towards ensemble learning as a method to overcome the above problems.
We exploit the well-known concept of ensemble learning and adapt it for reinforcement learning in a novel way. Traditionally, the idea of using ensembles in reinforcement learning settings is associated with combining multiple value functions or policies from different models. These models could be the same algorithm trained across different hyper-parameter settings or different algorithms altogether. Training multiple such models is an approach that cannot be used in practice owing to high sample complexity and computational expense.
We tackle the problem by creating sufficiently diverse policies from a single training instance. These policies are obtained by directed perturbation of the model parameters at regular intervals. The directed perturbation is induced by sudden and sharp variations in the learning rate, and for doing so, we employ cyclical learning rates (Loshchilov & Hutter (2016)). When the model parameters are perturbed using more significant learning rates, the directed motion along the gradient direction prevents the optimizer from settling in any sharp basins. It tends to move into the general vicinity of the local minima. Lowering the learning rates at such a time leads the optimizer to converge to some final local minima. We leverage the diversity of the policies learned at these different local minima for the ensemble. We show through experimentation that directed perturbation and not ran-

dom perturbation is necessary for obtaining diverse policies. We outperform SOTA in a number of environments, as shown in Figure 1(b).

Additionally, we develop a framework that selects the best subset of policies for including in the ensemble. Again, our approach does not require additional samples to find this subset.
Since we use models from a single training instance instead of training M different models independently from scratch, we refer to our approach as Sample Efficient Ensemble Reinforcement Learning (SEERL) To summarize, our main contributions are:

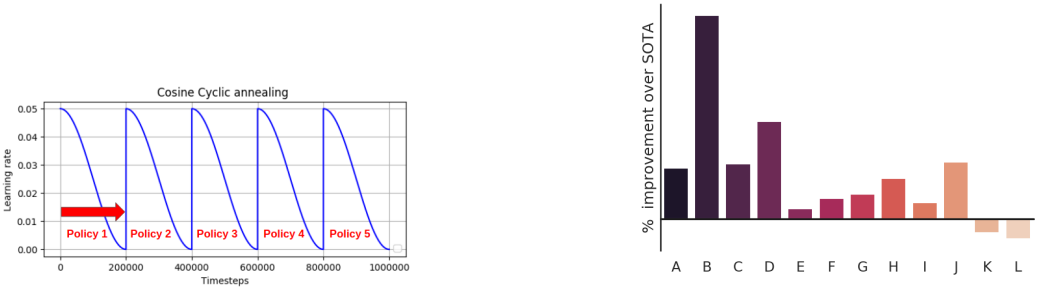
- A novel sample efficient framework for learning M diverse models from a single training instance with little to no additional cost.
- An optimization framework to select the best policies, from a diverse set, for ensemble
- Evaluation of various ensemble strategies for discrete and continuous action spaces.

2 RELATED WORK

There has recently been a small body of work on using ensembles for reinforcement learning which is associated with ensembles during the training phase to reduce the variance and improve robustness of the policy.
(Anschel et al. (2017)) trains multiple Q networks in parallel with different weight initialization and averages the Q values from all the different networks to reduce variance. It results in learning policies that are much more stable. However, the approach requires training multiple networks simultaneously and a possibility that the model might diverge if either of the Q values being averaged is biased.
Another interesting work falls under the umbrella of model-based reinforcement learning (Kurutach et al. (2018)). Multiple neural networks are initialized to learn a model of the environment using samples from the real system. Although the framework is sample efficient, it is costly to train multiple models. Using our framework, we could obtain multiple models from a single training instance at no additional computation costs.
Earlier works (Wiering & Van Hasselt (2008); Duell & Udluft (2013); Faußer & Schwenker (2015a;b)) explore the idea of value function ensembles and policy ensembles during evaluation phase. However, value function ensembles from different algorithms trained independently could degrade performance as they tend to converge to different fixed points and thereby have different bias and variance.(Marivate & Littman (2013)) tries to tackle this problem by having a meta-learner linearly combine the value functions from different algorithms during training time to adjust for the inherent bias and variance. Although training multiple algorithms in parallel is sample efficient, it is still far away from being a practical approach and tends to reduce diversity.
Our method combines the best of both approaches and improves the performance of the algorithm by balancing sample complexity with the computational expense. (Smith (2015); Loshchilov & Hutter (2016); Huang et al. (2017)) show that cycling learning rates are effective for training deep neural network architectures and ensembling in supervised learning settings. The authors show that in each cycle, the models obtained are comparable with those learned using traditional learning rate schedules. Even though the model is seen to degrade in performance temporarily, the new model surpasses the previous one, as the learning rate anneals.

3 PRELIMINARIES

Reinforcement learning is associated with sequential decision making and involves the interaction of an agent with an environment. In this paper, we consider a discrete-time finite-horizon Markov Decision Process(MDP) defined by (S, A, P, ρ_t, r, γ), where S denotes the set of states, A denotes the set of actions, $P : S \times A \rightarrow$ S' the transition function, ρ_t, the probability distribution over the initial states, $r : S \times S' \times A \rightarrow$ R, the reward function and γ the discount factor. Policy dictates the behavior of an agent at a particular state in an environment. More formally, a policy is defined by $\pi(s) : S \rightarrow P(A)$ where P(A) denotes the probability distribution over actions $a \, \epsilon \, A$ in a state $s \, \epsilon \, S$. The objective of the agent is to maximize the discounted return $R_t = \Sigma_{i=t}^{T} \, \gamma^{i-t} \, r(s_i,a_i,s_{i+1})$, where $r(s_i,a_i,s_{i+1})$ is the reward function.



(a) (b)

Figure 1: (a) Cyclical cosine annealing learning rate schedule. α_0 is set at 0.05, number of models $M = 5$ and training timesteps $T = 1000000$ (b) State of the art (SOTA) results in SEERL. A-Breakout, B-Skiing,C-Amidar,D-Solaris, E-Pong, F-Freeway,G-Robotank, H-Tutankham, I-Asteroids, J-berzerk, K-centipede, L-crazy climber

4 SEERL

SEERL is an ensemble of diverse policies obtained from a single training instance. Unlike supervised learning, where reusing the dataset for different training models is possible, training multiple agents independently for an ensemble suffers from high sample complexity. Though training agents in parallel is a possible solution to tackle sample complexity, it is computationally expensive and limits the diversity among the learned policies, since every policy observes the same state sequence. Our approach saves policies during training at periodic intervals when the learning rate anneals to a small value. It is then used as an ensemble during the evaluation phase.

4.1 Learning policies

To learn multiple policies from a single training instance, we perturb the parameters of the model at regular intervals. Instead of random perturbations, we employ directed perturbation in the direction of the gradient steps. We use cosine cyclical annealing learning rate schedule (Loshchilov & Hutter (2016)) to introduce these directed perturbations, as shown in Figure 1. Depending on the number of time-steps needed to train the agent, and the number of models needed for the ensemble, the learning rate schedule can be calculated.
As the learning rate anneals to a small value, the model converges to a local minimum, and we obtain the first policy. By increasing the learning rate, the model is perturbed along the gradient direction and dislodged from its local minima. In other words, if M models are required, we split the training process into M different training cycles wherein each cycle the model starts at a high learning rate and anneals to a small value. The high learning rate is significant as it provides energy to the policy to escape the local minima and the small learning rate traps it into a well behaved local minima. The formulation is as follows:

$$\alpha(t) = \frac{\alpha_0}{2}\left(\cos\left(\frac{\pi \mod (t-1, \lceil T/M \rceil)}{\lceil T/M \rceil}\right) + 1\right) \tag{1}$$

where α_0 is the initial learning rate, t is the time-step, and T is the total number of time steps for which the agent is trained, and M is the number of models.

4.2 Policy selection

The policies obtained during training are diverse and of different performance capabilities. Ideally, the best m policies should be selected for the ensemble to avoid bias from the poorer policies. At the same time, the policies also need to be diverse so as to obtain good performance in different parts of the state space when used in an ensemble. Near identical policies would not yield much improvement in an ensemble.
We propose an optimization framework to select the best subset of policies. The formulation has two parts, a model error term and a KL divergence term indicative of diversity. Only optimizing for

the model error term would result in the selection of policies with good performance. The addition of the KL divergence term helps balance the requirements of performance and diversity. We have a hyper-parameter $\beta \in [1, 2)$, that balances between diversity and performance. The KL divergence is calculated based on the action distribution between the two polices over the sampled states. The formulation is as follows :

$$J(w) = \sum_{s \in S} P(s) \left[\sum_{m \in M} w_m \left(\sum_{a \in A} L(s,a) - \frac{\beta}{M-1} \sum_{k \in M, k \neq m} KL(\pi_m(a|s) || \pi_k(a|s)) \right) \right]^2 \quad (2)$$

with the following constraints $\sum_{m \in M} w_m = 1, w_m \geq 0 \ \ \forall m$. S is the set of states and $P(s)$ is the probability of observing a particular state. $L(s,a)$ is the weighted error associated with the model and is indicative of the performance of the model. We weigh the loss in a manner that we give more weight when the loss is above a certain threshold value, and the action taken by the model, a, matches the action taken by the ensemble a_e. We formalize $L(s,a)$ as follows :

$$L(s,a) = \begin{cases} 1, & \text{if } |L'(s,a)| \geq T_{err} \textbf{ and } a_e - \epsilon \leq a \leq a_e + \epsilon, (\text{or } a = a_e, \text{if discrete}) \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

$L'(s,a)$ is the total error for a particular state action pair. T_{err} is the threshold value that is used to distinguish the good actions from the bad. If $L'(s,a)$ is above the threshold error and the actions match, then we would like to consider that action from the policy as a bad one. For example, in the case of A2C, the total error for a state action pair is the weighted sum of the policy gradient loss and the value function loss. Let $V_m(s)$ be the value function at state s obtained by using model m. From the Markov chain, we can obtain the reward, $r(s,a,s')$ after taking action a at that state and transitioning to s'.

$$V_{loss}(s) = r(s,a,s') + \gamma V(s') - V_m(s) \quad (4)$$

$$\pi_{loss}(a|s) = \nabla log(\pi(a|s) A(s) \quad (5)$$

$$L'(s,a) = \pi_{loss}(a|s) + V_{loss}(s) * C_v \quad (6)$$

γ is the discount factor, C_v is the coefficient used for weighting the value function.
In a discrete action space, it is relatively easy to determine if the action taken by the model and ensemble are the same. However, in continuous action space, the final ensemble action and the action taken by the policy might never coincide. Therefore, we introduce a ϵ bound on the ensembled action. If the action from the model is within a ϵ distance from the ensembled action, we consider it as a match. ϵ ranges between 0.005 to 0.01, depending on the environment.
We use a squared loss formulation to capture the inter-dependencies among the policies. Instead, if the degree were 1, the objective function would be the weighted sum of the loss, and the one with the lowest error would be the best policy. By having a higher degree, we are capturing the dependencies among the policies.
By minimizing this objective function, we obtain the values of w_m, the Lagrange multipliers to this optimization framework. To choose the best ensemble set, we arrange the values of w_m in decreasing order and select the top m values.
In order to run this optimization, we select multiple trajectories from the training samples and use the states from them. It is much more efficient reuse of data in comparison to evaluating the policies directly in the environment. This is also more efficient than searching through all possible combination of selecting m policies.

4.3 ENSEMBLE TECHNIQUES

Once we choose the m policies, selecting the optimal ensemble strategy is a challenging task. Depending on the complexity of the action space, discrete or continuous, there are multiple strategies to ensemble the actions in the environment. When building ensembles, all m policies are loaded in parallel and provided with an observation from the environment. Based on the observation, every policy outputs an action. The ensemble strategy decides which action to select based on the available set of actions. We divide the ensemble strategy into two categories, based on the complexity of the action space, into strategies for discrete and continuous action spaces.

Algorithm 1 SEERL

Input: Initialize a policy π_θ, training timesteps T,evaluation timesteps T', number of policies M, maximum learning rate α_0, number of policies to ensemble m, ensemble strategy E, D to store the policies
Output: Average reward during evaluation
Training

while $t \leq T$ **do**
  Calculate the learning rate, based on the inputs to the cosine annealing learning rate schedule f
  $\alpha(t) = f(\alpha_0, t, T, M)$
  Train the agent and optimize using $\alpha(t)$
  if t $\bmod$ (T/M) **then**
    Save policy π_θ^i in D for $i = 1, 2 \ldots, M$
  end if
end while

Evaluation

Select the m policies from D using the Policy selection algorithm
Select an ensemble strategy E
while $t \leq T'$ **do**
  Collect actions from the m policies, $a_1, a_2 \ldots, a_m$ for environment state s_t
  Find the optimal action, a^* using E
  Perform action a^* on the environment
  Obtain cumulative reward for the episode, r_t and the next state s_{t+1}
end while
return Average reward obtained during evaluation

4.3.1 Ensemble in discrete action spaces

In discrete action spaces, we consider majority voting as a good solution. Due to different fixed point convergences of value functions of algorithms trained independently, it is not possible to compare actions by their Q values.

$$\pi(a|s) = \underset{a \in A(s)}{\operatorname{argmax}} \left[\sum_{m \in M} N_m(s, a) \right] \tag{7}$$

where $N_m(s, a)$ is one if the agent m takes action a in state s, else zero. It the case of a tie, random action is chosen among the set of actions having the tie.

4.3.2 Ensemble in continuous action spaces

In continuous action spaces, (Duell & Udluft (2013)) proposes multiple strategies to find the optimal action. However, the performance comparison of the strategies is not provided and environments considered are too simple. The different strategies are as follows:

- Averaging: We take the average of all the actions as part of the ensemble. This strategy could fail in settings where one or more of the actions are extremely biased and thereby shifts the calculated value away from the true mean value.
- Binning: This is the equivalent of majority voting in a continuous action space setting. We discretize the action space into multiple bins of equal size and average the bin with the most number of actions. The average value obtained is the optimal action to take. Through this method, we have discretized the action space, sorted the bins based on its bin-count, and calculated the mean of the bin with the highest bin-count. The only parameter to be specified here is the number of bins to be specified. We use five bins in our experiments
- Density-based Selection(DBS): This approach tries to search for the action with the highest density in the action space. Given M action vectors, a, each of k dimensions to be ensem-

bled, we calculate the density of each action vector using Parzen windows as follows:

$$d_i = \sum_{j=1}^{M} exp - \frac{\sum_{l=1}^{k}(a_{il} - a_{jl})^2}{r^2} \tag{8}$$

The action with the highest density, d_i, is selected as the final action. The only parameter to be specified is r, and we have chosen $r = 0.0001$ in our experiments.

- Selection through Elimination(STE): This approach eliminates action based on the Euclidean distance. We calculate the mean of the action vectors and compute the euclidean distance to each action from the mean. The action with the largest euclidean distance is eliminated, and the mean is re-computed. The process is repeated until two actions remain. The final action is chosen as the average of the two actions.

5 EXPERIMENTS

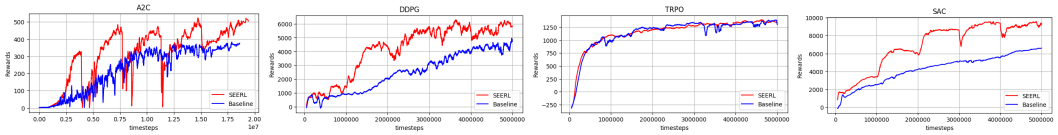
We consider the environments from the Atari 2600 game suite and the robotic environments from Mujoco (Todorov et al. (2012)). The Atari 2600 is considered for its discrete state and action space, whereas Mujoco is used for its continuous state and action space. We conduct our experiments on the following algorithms, A2C (Mnih et al. (2016)), ACER (Wang et al. (2016)), ACTKR (Wu et al. (2017)), (DDPG Lillicrap et al. (2017)), (SAC Schulman et al. (2017)) and TRPO (Schulman et al. (2015a))
We use three baseline models to compare with SEERL during the ensemble. The first baseline, B1, ensembles policies that are trained independently from one algorithm, Eg. Five models of A2C, each of which has been trained for 20 million time steps. The second, B2, ensembles policies that are trained independently from different algorithms, Eg. two models of A2C, two models of ACER and one model from ACKTR, each of which has been trained for 20 million time-steps. The third and final baseline, B3, uses policies generated from random perturbation of model parameters at regular intervals. Eg. Five models of A2C, each of which has been obtained by perturbing at regular intervals and saving the parameters. We provide more details regarding the experiments and ablation studies in the Appendix.

Through our experiments, we answer the following questions:

- How does SEERL compare against traditional ensembles in terms of sample complexity and final performance?
- How the diversity among policies contributes to the final performance?
- How are the policies obtained from SEERL any different from those obtained through random perturbation?

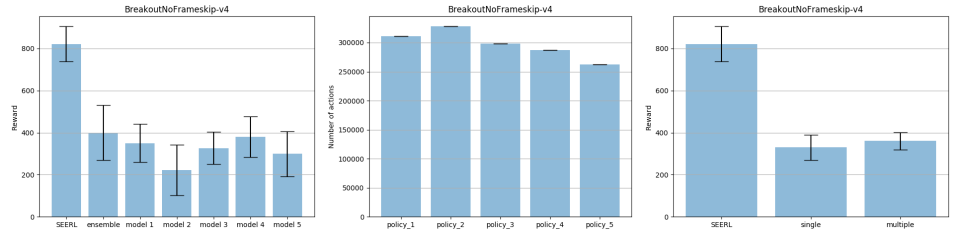
5.1 TRAINING AND EVALUATION

We train both SEERL and single-agent models with the same hyper-parameter configurations. The ambiguity that SEERL will lead to poor convergence as a result of shifting from zero to the maximum learning rate multiple times is mitigated through our results. SEERL performance during training is at least at par or better than the baseline, as shown in Figure 2. We train the models across different values of M ranging from 3 to 9.
The baseline models are trained using the set of hyper-parameters seen in their original implementation. In order to create diverse baseline models, we train the models on five different seeds with different learning rates and learning rate schedules. We consider two learning rate schedules, constant and linearly decreasing. Using our policy selection framework, we then select m policies for the ensemble. For the majority of our experiments, we consider $m = 5$. Details regarding the training parameters for different algorithms can be found in the Appendix We perform evaluation of SEERL in comparison with our three baseline models. It is observed that SEERL outperforms all the baselines and in some environments, it outperforms the state of the art score. Each of the methods is evaluated in the true environment for 100 episodes and the mean average reward is shown in Figure 3(f)

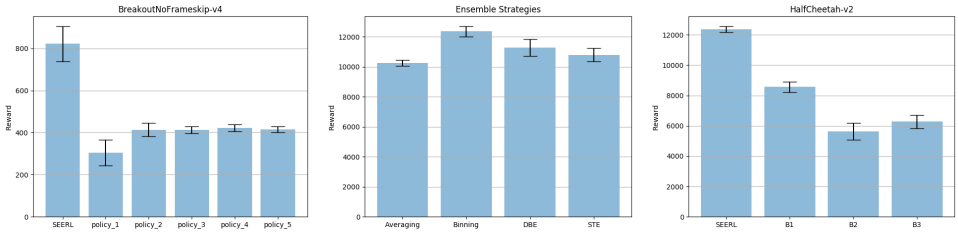


(a) (b) (c) (d)

Figure 2: Comparison of training performance between SEERL and Baseline. (a) A2C on Breakout (b) DDPG on Half Cheetah (c) TRPO on Half Cheetah (d) SAC on Half Cheetah



(a) (b) (c)



(d) (e) (f)

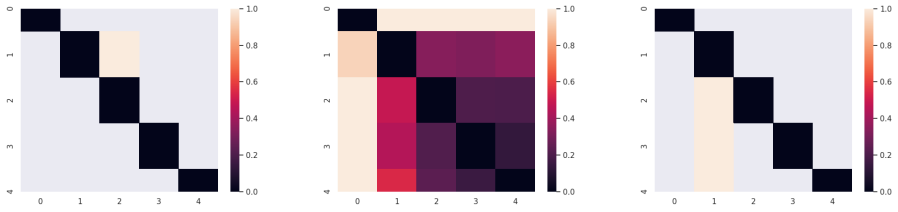
Figure 3: (a) Comparison between SEERL, B1, and the individual policies of B1 (b) Comparing the dominance of the policies used in SEERL.(c) Comparison between SEERL and two versions of B3, during the evaluation on Breakout using A2C (d) Comparison between SEERL and its individual policies (e) The performance comparison of various ensembling strategies in continuous action space. (f) Comparison between SEERL and baseline ensembles B1, B2 and B3 for Half Cheetah using SAC

5.2 Analysis

We try to understand why and how SEERL gives such superior performance in comparison to baselines. We analyze the individual performance of the SEERL policies, the dominance among policies in the ensemble, the diversity among the policies, and finally, the comparison between a randomly perturbed model and SEERL.

5.2.1 Performance of individual policies

The analysis of the individual policies helps us to understand if the perturbations degrade the model during training. If the perturbations tend to harm the model, the performance of the individual policies should decrease as training progresses. We see in Figure 3(d) that the performance of the policies does not degrade during training and that new policies may be better than others. However, for the baseline model, the individual policies which have been trained independently, show different performance levels. This could indicate that some policies might have converged to bad local minima while others were able to settle in good ones. The performance of the individually trained models have high variance with respect to each other and supports our motivation of using ensembles instead of a single model.



(a) (b) (c)

Figure 4: (a) Divergence between independently trained policies used in the baseline, B1, for Breakout using A2C. (b) : Divergence between SEERL policies for Breakout using A2C. (c) : Divergence between policies obtained using random perturbations for Breakout using A2C.

5.2.2 Dominance of Policies in SEERL

In order to understand if any particular policy is dominating in the ensemble of SEERL, we calculate the number of times the action taken by a policy matches the action taken by the ensemble. We do this evaluation for 500000 samples and observe that no particular policy is dominating in the ensemble, as shown in Figure 3(b). All of the policies contribute almost equally to the ensemble.

5.2.3 Random perturbation vs. SEERL

To emphasize that any perturbing the parameters of the model will not lead to better models, we show the comparison between SEERL and a randomly perturbed model in Figure 5(c). This model has been perturbed with random values at regular intervals similarly to SEERL. The perturbation is done by backpropagating from random values of gradients instead of the true values. We see that doing a single perturbation or, multiple ones sequentially for a small period, does not lead to better performance. With this, we can establish that just perturbation is not sufficient, but directed perturbation along the gradient direction is necessary to obtain a better model. In this experiment, all the hyper-parameters have been kept identical to that used in SEERL and baselines.

5.2.4 Diversity of Policies

The performance of SEERL depends on the diversity of the individual policies being learned. We have shown earlier, the performance of the individual policies being learned and the diverse nature in their performance. We hope to establish more concretely the diversity of the individual policies by understanding the action distribution across states for each policy. We compute the KL divergence between the policies based on the action distribution across a diverse number of states. The greater the KL divergence between the policies, the more diverse the policies are. From Figure 4(b), we can observe that the SEERL policies are diverse, and diversity continues to exist as new models are formed. Conversely, for the baseline models (Figure 4(a) and Figure 4(c)), the KL divergence between the policies is substantial. This observation can be used to explain why the baseline ensembles failed to perform. The policies did not have much overlap in the action space, and hence ensemble techniques such as majority voting were unable to find a good action. We can, therefore, conclude that SEERL can generate policies with sufficient diversity for a good ensemble.

6 Conclusion and Future work

In this paper, we introduce SEERL, a framework to ensemble multiple policies obtained from a single training instance. Through our experiments, we show that the policies learned at the different local minima are diverse in their performance and hence are well suited for the ensemble. SEERL outperforms three baseline methods in complex environments having discrete and continuous action spaces. We show our results using various reinforcement learning algorithms and therefore show that it is not limited to its performance in any particular setting. Future work will explore how to combine the learned policies during training time as a growing ensemble to stabilize training and increase diversity.

REFERENCES

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 176–185. JMLR. org, 2017.

Siegmund Duell and Steffen Udluft. Ensembles for continuous actions in reinforcement learning. In *ESANN*, 2013.

R Evans, J Jumper, J Kirkpatrick, L Sifre, TFG Green, C Qin, A Zidek, A Nelson, A Bridgland, H Penedones, S Petersen, K Simonyan, S Crossan, D Jones, D Silver, K Kavukcuoglu, D Hassabis, and A Senior. De novo structure prediction with deeplearning based scoring. *Annu Rev Biochem*, 77:363–382, 2018.

Stefan Faußer and Friedhelm Schwenker. Neural network ensembles in reinforcement learning. *Neural Processing Letters*, 41(1):55–69, 2015a.

Stefan Faußer and Friedhelm Schwenker. Selective neural network ensembles in reinforcement learning: taking the advantage of many agents. *Neurocomputing*, 169:350–357, 2015b.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Timothy Paul Lillicrap, Jonathan James Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daniel Pieter Wierstra. Continuous control with deep reinforcement learning, January 26 2017. US Patent App. 15/217,758.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Vukosi Ntsakisi Marivate and Michael Littman. An ensemble of linearly combined reinforcement-learning agents. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

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

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL `http://arxiv.org/abs/1602.01783`.

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, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

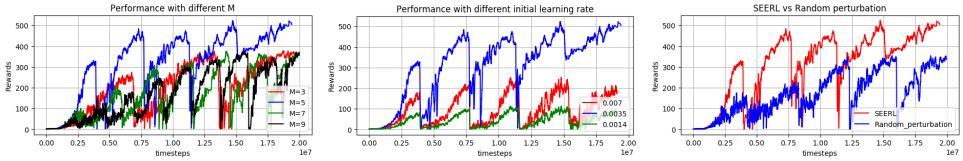
Leslie N Smith. No more pesky learning rate guessing games. *arXiv preprint arXiv:1506.01186*, 2015.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

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, 2016. URL http://arxiv.org/abs/1611.01224.

Marco A Wiering and Hado Van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):930–936, 2008.

Yuhuai Wu, Elman Mansimov, Shun Liao, Roger B. Grosse, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *CoRR*, abs/1708.05144, 2017. URL http://arxiv.org/abs/1708.05144.



(a) (b) (c)

Figure 5: (a) Training performance of SEERL as M varies between 3 to 9. (b) Training performance of SEERL as maximum learning rate α_0 varies. (c) Training performance between SEERL and randomly perturbed model, with multiple perturbations in a sequence

A Appendix

A.1 Ablation studies

A.1.1 Effect of varying the number of cycles

The performance of SEERL is affected by the selection of M. For a fixed training budget, if the value of M chosen to be is very large, the performance is seen to degrade. With larger M, the training cycle for each policy is reduced, thereby reducing the chance for the policy to settle to a good local minimum before it is perturbed again. In practice, we find that setting the value of M between 3 to 7 works reasonably well. Figure 4(d) compares the performance of SEERL with varying M values between 3 and 9

A.1.2 Effect of varying maximum learning rate value

The maximum leaning rate value influences the performance of the policies and therefore affects the performance of SEERL. It directly impacts the perturbation of the local minima and hence, the diversity of the policies being learned. In practice, we have seen that having a more significant value tends to perform better, owing to the strong perturbation it causes at different local minima leading to reasonably different policies. We have used values ranging between 0.01 to 0.001 throughout our experiments. Figure 4(e) compares the performance of SEERL with different values of α_0 with $M = 5$

A.2 Training Results on Atari environments using A2C

In order to evaluate the scalability of our framework, we test it across 40 Atari games and analyze the performance. Each environment is trained for 20 million time-steps and evaluated for 100 episodes. The baselines are trained using the hyper-parameters from the original implementation. SEERL is also trained using the same hyper-parameters, but the maximum learning rate chosen is three to seven times the baseline value depending on environments. We outperform all the three baselines across 35 environments and also outperform the state of the art scores in 10 environments.

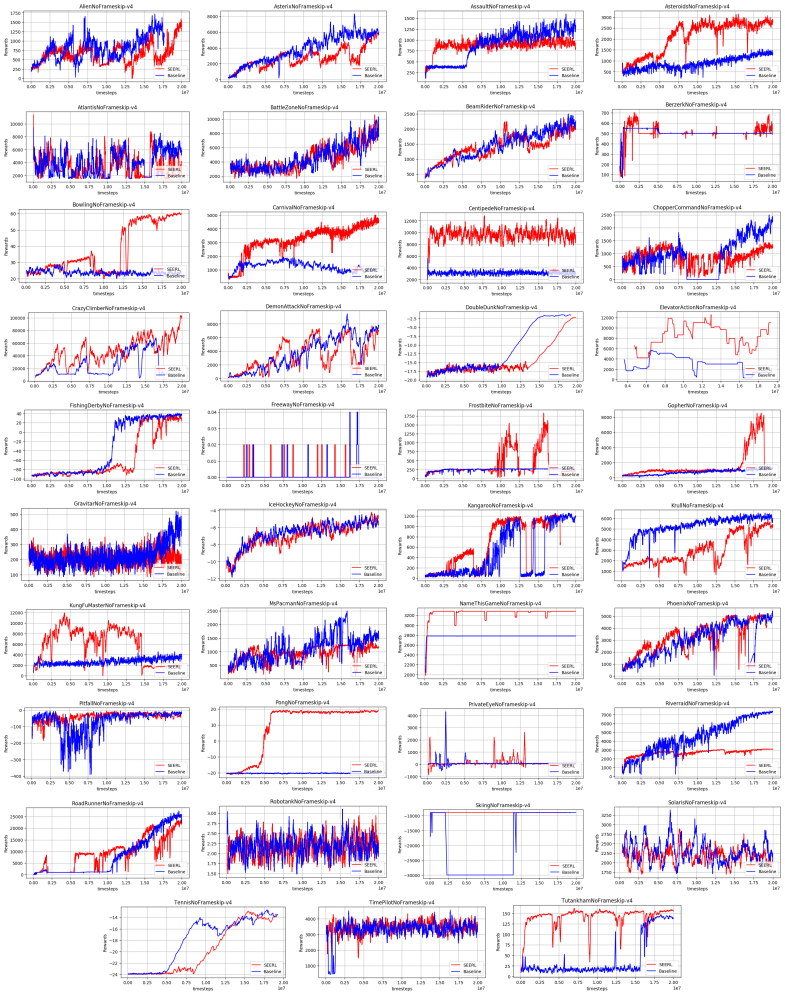


Figure 6: Training performance of SEERL across different Atari games using A2C

A.3 TRAINING RESULTS ON MUJOCO USING DDPG

We evaluate the training performance of SEERL on Mujoco environments using DDPG

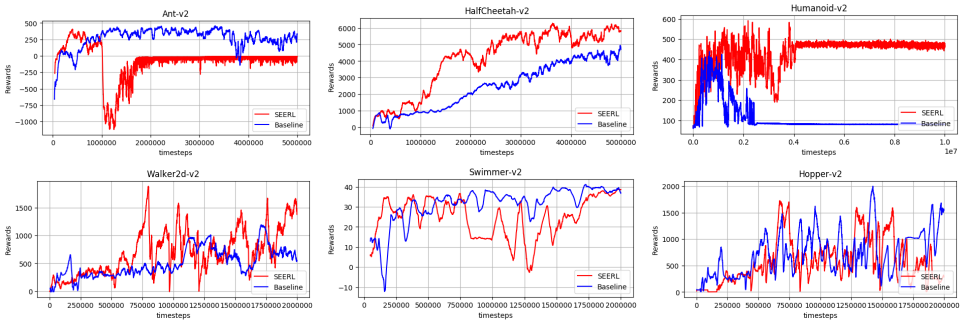


Figure 7: Training performance of SEERL across different Mujoco environments using DDPG

A.4 TRAINING RESULTS ON MUJOCO USING TRPO

We evaluate the training performance of SEERL on Mujoco environments using TRPO

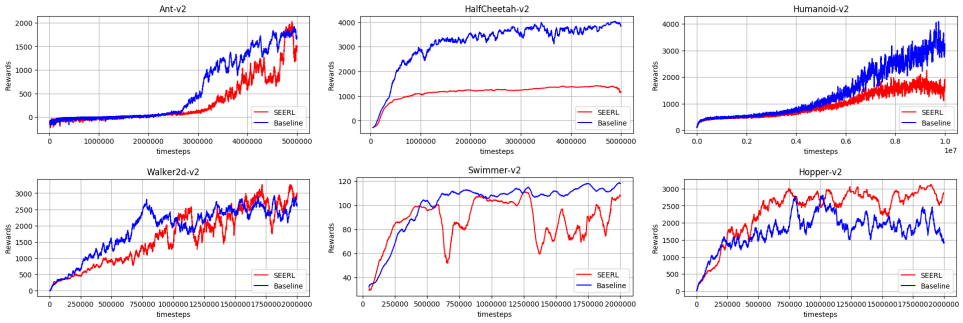


Figure 8: Training performance of SEERL across different Mujoco environments using TRPO

A.5 TRAINING RESULTS ON MUJOCO USING PPO

We evaluate the training performance of SEERL on Mujoco environments using PPO

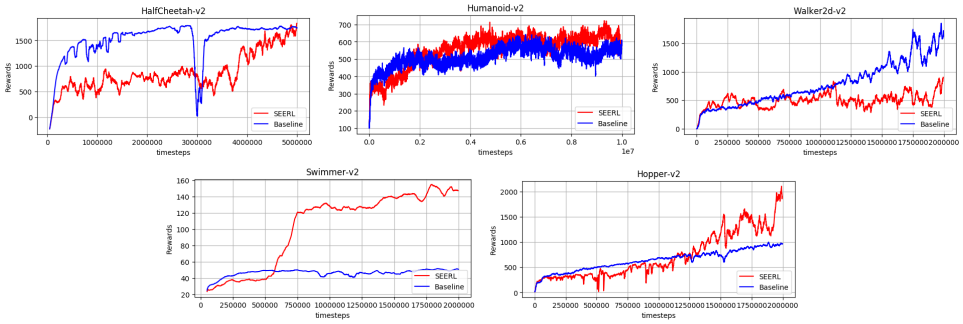


Figure 9: Training performance of SEERL across different Mujoco environments using PPO

A.6 TRAINING RESULTS ON MUJOCO USING SAC

We evaluate the training performance of SEERL on Mujoco environments using SAC

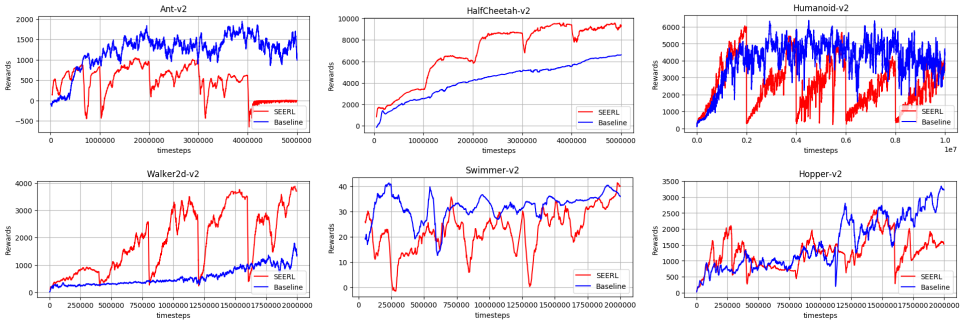


Figure 10: Training performance of SEERL across different Mujoco environments using SAC