Escaping Local Maxima in Reinforcement Learning through Landscape Modified Langevin Dynamics

By

Tan Etai Adam

Supervisor:

Michael Choi

DSA4199 Honours Project in Data Science and Analytics

National University of Singapore

2022/2023

Contents

1 Abstract **4**

2 Introduction **5**

3 Background **6**

3.0.1 Experiment Environment 6

3.1 Gradient Ascent Descent (GAD) 7

3.2 Adaptive Moment Estimation with Extrapolation (Extra-Adam) 8

3.3 Mixed Nash Equilibirum via Langevin Dynamics (MixedNE-LD/SGLD) 9

3.4 MixedNE-LD with Landscape Modification (MixedNE-LDLM) 10

3.4.1 Adaptive Reward Threshold 12

4 Methodology **13**

4.1 Varying f 13

4.2 Varying α 13

4.3 Varying Where to Apply Landscape Modification 14

5 Main Result **14**

5.1 Minimizing Actor Only 15

5.1.1 Minimizing Actor with Arctan and Square 18

5.2 Maximizing Adversary Only 21

5.2.1 Maximizing Adversary with Arctan and Square 23

5.3 Minimizing Actor and Maximizing Adversary 26

6 Limitations **28**

7 Conclusion **29**

8 References **30**

1 Abstract

In reinforcement learning (RL) algorithms, an intelligent agent in an environment learns what actions to take in order to maximize the amount of reward it receives. Usually this is done through gradient descent, where the algorithm follows the gradient of the reward function in order to reach the peak. One common problem these algorithms face is getting stuck in local maxima, where they reach a local peak in reward that is less than the global maxima. Gradient descent reads the gradient at these local maxima as close to 0, thus they are unable to escape this local maxima to find the global peak in reward. We propose using landscape modification to augment Stochastic Gradient Langevin Dynamics so as to allow the agent to continue iterating to find a better maxima. Experiments were done on OpenAI Gym utilising the MuJoCo environment achieving great to limited effects depending on the task.

2 Introduction

In reinforcement learning, there is always a tradeoff between exploration and exploitation. How often should an algorithm keep trying to improve what it already knows, and how often should it try to search elsewhere for a better solution? This problem can be represented by local maxima vs global maxima. When an algorithm gets stuck in a stationary point, a local maximum/minimum, it stops exploring and is just continually trying to exploit it's current situation, to not much benefit. Rather than being stuck in a stationary point, it is usually a better use of resources to keep searching for a potential better solution.

There have been varied attempts to deal with the problem of local maxima. [1] and [2] inject noise into gradient descent in different ways. [3] shows that even without noise, random initialization avoids saddle points. Saddle points are different from local minima, but the method of escaping from the two can intuitively be thought of as similar.

In practice, choosing the right strategies and learning rate schedule can help to alleviate this problem. However, as [4] argues, saddle points are flanked by plateaus of the same error, which makes it very difficult for gradient descent methods to escape.

In this paper, we propose using Landscape Modification to inject this noise in varied amounts depending on the distance from the observed maxima. With additional noise, it should be considerably harder for the algorithm to get stuck in any local maxima. This is because when it oscillates around a maxima, the oscillations will likely be larger than usual, allowing it more opportunities to escape the local maxima. It will be able to explore the space to a greater extent, revealing any superior solutions.

3 Background

We will be building on previous work done by [5] by using their their algorithms as a baseline, then modifying their algorithm for mixed nash equilibrium via Langevin dynamics to include landscape modification.

3.0.1 Experiment Environment

We will be using a few different tasks for testing, but all of them have the same general properties. We will have an actor and an adversary. The actor will receive the normal cumulative reward from maximizing the objective in RL. The adversary will receive the negative of that value. At each time step, the actor and adversary will be given the current state, then each take their actions. The actor controls the "player" in the environment, while the adversary controls environmental factors. This can include variables such as the mass of items or the friction in the environment.

This creates a two-player game between the actor and adversary. Usually, these robust RL problems are tackled by maximizing the actor's reward and minimizing the adversary's reward. This can be through methods such as Gradient Ascend Descent or Adaptive Moment Estimation with Extrapolation. [5] tested an alternative method of using Mixed Nash Equilibrium via Langevin Dynamics to reach a solution for the problem.

3.1 Gradient Ascent Descent (GAD)

Algorithm 1 Gradient Ascent Descent

```
Require: $K$=number of steps, $\eta$=step size, $\theta$=variables of actor, $\omega$=variables of adversary, $h$=reward function, $\nabla_\theta$=gradient with respect to $\theta$
Ensure: $k = 0$, Initialize $\theta_0, \omega_0$ randomly
  while $k < K$ do
    $\theta_{k+1} \leftarrow \theta_k + \eta \nabla_\theta h(\theta_k, \omega_k)$
    $\omega_{k+1} \leftarrow \omega_k - \eta \nabla_\omega h(\theta_k, \omega_k)$
    $k \leftarrow k + 1$
  end while

Output: $\theta_K, \omega_K$
```

A straightforward algorithm that moves the actor towards the maximum by iteratively adding the gradient of the values in the actor. The adversary similarly moves towards the minimum by iteratively subtracting the gradient of the values in the adversary.

3.2 Adaptive Moment Estimation with Extrapolation (Extra-Adam)

Algorithm 2 Adaptive Moment Estimation with Extrapolation (Extra-Adam)

```
Require: $K$=number of steps, $\eta$=step size, $\theta$=variables of actor, $\omega$=variables of adversary, $h$=reward function, $\nabla_\theta$=gradient with respect to $\theta$, $\{\beta_1, \beta_2\}$=decay rates for moment estimates, $g$=gradient, $\{m, v\}$={first, second} moment of gradient, $P_\Omega[.]$=projection onto constraint set $\Omega$, $e$=extrapolated version of variable
Ensure: Initialize $\theta_0, \omega_0$ randomly, $k, m_{-1}, v_{-1} = 0$,
  while $k < K$ do
      if $ge_k$ exists then
          $g_k \leftarrow ge_k$
      else
          $g_k \leftarrow \nabla_\omega h(\omega_k)$
      end if
      $me_k \leftarrow \beta_1 m_{k-1} + (1 - \beta_1) g_k$
      $ve_k \leftarrow \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$                ▷ Updating extrapolated moments
      $\hat{me}_k \leftarrow me_k / (1 - \beta_1^{2k-1})$
      $\hat{ve}_k \leftarrow ve_k / (1 - \beta_2^{2k-1})$                    ▷ Correcting bias of extrapolated moments
      $\omega e_k \leftarrow P_\Omega[\omega_k - \eta(\hat{me}_k / \sqrt{\hat{ve}_k} + \epsilon)]$          ▷ Extrapolating
      $ge_k \leftarrow \nabla_\omega h(\omega e_k)$
      $m_k \leftarrow \beta_1 me_k + (1 - \beta_1) ge_k$
      $v_k \leftarrow \beta_2 ve_k + (1 - \beta_2) ge_k^2$                    ▷ Updating moments
      $\hat{m}_k \leftarrow m_k / (1 - \beta_1^{2k})$
      $\hat{v}_k \leftarrow v_k / (1 - \beta_2^{2k})$                          ▷ Correcting bias of moments
      $\omega_{k+1} \leftarrow P_\Omega[\omega_k - \eta(\hat{m}_k / \sqrt{\hat{v}_k} + \epsilon)]$
      $k \leftarrow k + 1$
  end while

Output: $\omega_K$
```

Extra-Adam as explained by [6], gives "stability" to the calculated gradients. It builds on the ideas of using the first and second moment of the gradient from Adam by using extrapolation to improve estimates of the moments.

The algorithm above only shows the case where ω is trying to be minimized, but the concept is the same for also maximizing θ.

3.3 Mixed Nash Equilibirum via Langevin Dynamics (MixedNE-LD/SGLD)

Algorithm 3 Mixed Nash Equilibrium via Langevin Dynamics

```
Require: $K$=number of steps, $\eta$=step size, $\theta$=variables of actor, $\omega$=variables of adversary, $h$=reward function, $\nabla_\theta$=gradient with respect to $\theta$, $\epsilon_t$=thermal noise (small number that decreases with additional steps), $\xi$=standard normal random variable
Ensure: $k = 0$, Initialize $\theta_0, \omega_0$ randomly
  while $k < K$ do
    $\theta_{k+1} \leftarrow \theta_k + \eta \nabla_\theta h(\theta_k, \omega_k) + \sqrt{2\eta}\epsilon\xi$
    $\omega_{k+1} \leftarrow \omega_k - \eta \nabla_\omega h(\theta_k, \omega_k) + \sqrt{2\eta}\epsilon\xi$
    $k \leftarrow k + 1$
  end while

Output: $\theta_K, \omega_K$
```

This is the main idea from [5]. It builds upon GAD by adding "practical relaxations", which turns out to be adding in an additional noise term. Their goal was to find a stabler training algorithm, but we feel that this noise term could be taken futher.

3.4 MixedNE-LD with Landscape Modification (MixedNE-LDLM)

Algorithm 4 MixedNE-LD with Landscape Modification

```
Require: $K$=number of steps, $\eta$=step size, $\theta$=variables of actor, $\omega$=variables of adversary, $h$=reward function, $\nabla_\theta$=gradient with respect to $\theta$, $\epsilon_t$=thermal noise (small number that decreases with additional steps), $\xi$=standard normal random variable,
    $f$=any function where $\forall x \geq 0,\ f(x) \geq 0$,
    $(x)_+$=function where if $x \leq 0, (x)_+ = 0$ and if $x > 0, (x)_+ = x$,
    $c$=reward threshold, $\alpha$=variable to control size of additional noise
Ensure: $k = 0$, Initialize $\theta_0, \omega_0$ randomly
  while $k < K$ do
    $\theta_{k+1} \leftarrow \theta_k + \eta \nabla_\theta h(\theta_k, \omega_k) + \sqrt{2\eta}\sqrt{\alpha f[(c_1 - h(\theta_k, \omega_k))_+] + \epsilon^2}\xi$
    $\omega_{k+1} \leftarrow \omega_k - \eta \nabla_\omega h(\theta_k, \omega_k) + \sqrt{2\eta}\sqrt{\alpha f[(h(\theta_k, \omega_k) - c_2)_+] + \epsilon^2}\xi$
  end while

Output: $\theta_K, \omega_K$
```

In MixedNE-LD, or Stochastic Gradient Langevin Dynamics, the noise term allows the actor and adversary to escape from local maxima and minima in order to find a possible better global maxima and minima. In this new MixedNE-LDLM, we scale this noise up, depending on how far away it is from a reward threshold level. This is where the "Landscape Modification" takes place.

The distance from the reward threshold level is measured by $f[(h(\theta_k, \omega_k) - c)_+]$. f is any function where $\forall x \geq 0,\ f(x) \geq 0$, which means non-negative values of x return non-negative values of $f(x)$. $(x)_+$ is a function where if $x \leq 0, (x)_+ = 0$ and if $x > 0, (x)_+ = x$, which turns all negative numbers into 0. This means that when the reward is within the reward threshold level, no extra noise is added. We can also test different functions of f to observe if there are any differences.

Let us use a simple example where we try and apply MixedNE-LDLM. To minimize the adversary, we will try to find the global minima of the negative reward function.

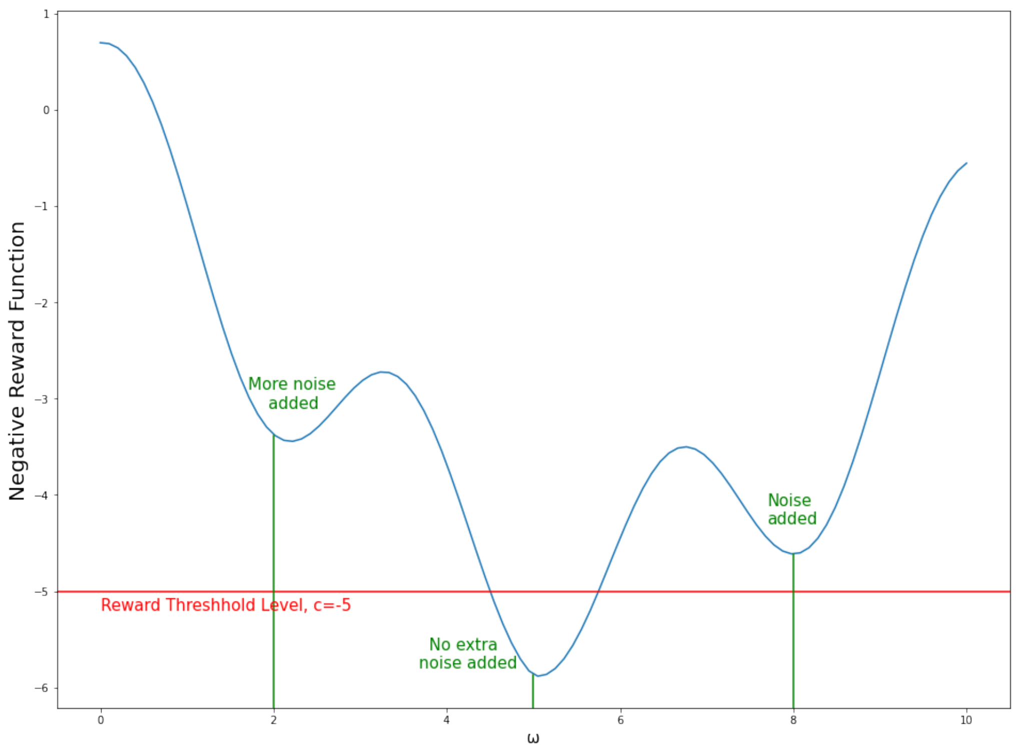

Figure 1: Example of MixedNE-LDLM trying to find global minima

When $\omega = 8$, the negative reward function is about -4.6. This is greater than the reward threshold level, thus more noise is added.

When $\omega = 2$, the negative reward function is about -3.4. This is even greater than the reward threshold level, thus even more noise is added.

When $\omega = 5$, the negative reward function is about -5.9. This is lesser than the reward threshold level, thus no additional noise is added.

This same logic also applies when trying to find maxima.

3.4.1 Adaptive Reward Threshold

Algorithm 5 MixedNE-LDLM with adaptive threshold (SGLMLD)

```
Require: $K$=number of steps, $\eta$=step size, $\theta$=variables of actor, $\omega$=variables of adversary, $h$=reward function, $\nabla_\theta$=gradient with respect to $\theta$, $\epsilon_t$=thermal noise (small number that decreases with additional steps), $\xi$=standard normal random variable, $f$=any function where $f(x) \geq 0, \forall x \geq 0$, $(x)_+$=function where if $x \leq 0, (x)_+ = 0$ and if $x > 0, (x)_+ = x$, $c$=reward threshold, $\alpha$=variable to control size of additional noise
Ensure: $k = 0,$ Initialize $\theta_0, \omega_0$ randomly
  while $k < K$ do
    $\theta_{k+1} \leftarrow \theta_k + \eta\nabla_\theta h(\theta_k, \omega_k) + \sqrt{2\eta}\sqrt{\alpha f[(c_1 - h(\theta_k, \omega_k))_+] + \epsilon^2}\xi$
    $\omega_{k+1} \leftarrow \omega_k - \eta\nabla_\omega h(\theta_k, \omega_k) + \sqrt{2\eta}\sqrt{\alpha f[(h(\theta_k, \omega_k) - c_2)_+] + \epsilon^2}\xi$
    $c_1 \leftarrow max(h(\theta_k, \omega_k), c_1)$
    $c_2 \leftarrow min(h(\theta_k, \omega_k), c_2)$
  end while

Output: $\theta_K, \omega_K$
```

Finally, we also add in an adaptive reward threshold. One issue with having a static reward threshold is that the range of possible reward values has to be known beforehand. This can be alleviated by using an adaptive reward threshold. The reward threshold is updated to the value of the best performance so far. If the current performance is worse than the best performance, then it is impossible for the current performance to be a global minima. Thus noise is added to lower the chance of getting stuck in local maxima, in order to potentially find a global maxima.

4 Methodology

Each experiment was repeated at least 3 times to reduce any random errors. From here on, we refer to "minimizing the loss of the reward function of the actor" as simply "minimizing actor" and "minimizing the loss of the negative reward function of the adversary" as simply "maximizing adversary".

4.1 Varying f

As explained in 3.4, f determines how the noise is shaped depending on how far away it is from the reward threshold level. We only care about the values of x that are positive as we do not apply any extra noise when the reward is within the reward threshold level. For the different functions of f, we tested using $f(x) = x, f(x) = arctan(x)$ and $f(x) = x^2$. These values of f were chosen for their different properties.

$f(x) = x$ scales linearly with x, thus the penalty given to how far away it is from the reward threshold level increases at a regular pace.

$f(x) = arctan(x)$ initially increases faster than $f(x) = x$, but it then quickly plateaus. Thus the penalty given does not considerably increase after a certain point.

$f(x) = x^2$ increases at an increasing pace. The penalty given continually worsens as the distance increases.

We want to see how these different values of f, that affect how the algorithm penalizes distance, change how the agent performs.

4.2 Varying α

We also varied the values of α as first seen in Algorithm 4. This is a variable that affects the scale of the extra noise added. This is an important variable as the scale of the extra noise can turn itself from not affecting the agent at all, to making the agent completely diverge. For each case, we tested a few values of α to determine the optimal value.

4.3 Varying Where to Apply Landscape Modification

Finally, we also used SGLMLD instead of MixedNE-LD (SGLD) only on the actor, the adversary, and then both. The reward that the actor and adversary receives are the negative of each other. Thus it is interesting to find out if applying more noise would affect each the actor and adversary equally, and how they would interact when combined.

5 Main Result

Each line of the graph represents the reward the agent receives when it is placed in the environment for a certain number of timesteps. A larger value of reward indicates that the agent is performing better. For ease of reading the graphs, we will take SGLD to be the main baseline that we compare the results to.

The total number of timesteps for different environments differ from each other due to constraints in computational power. From the learning curve graphs obtained from [5], we limited the number of timesteps to where it seemed like there were already clear differences in the reward function.

5.1 Minimizing Actor Only

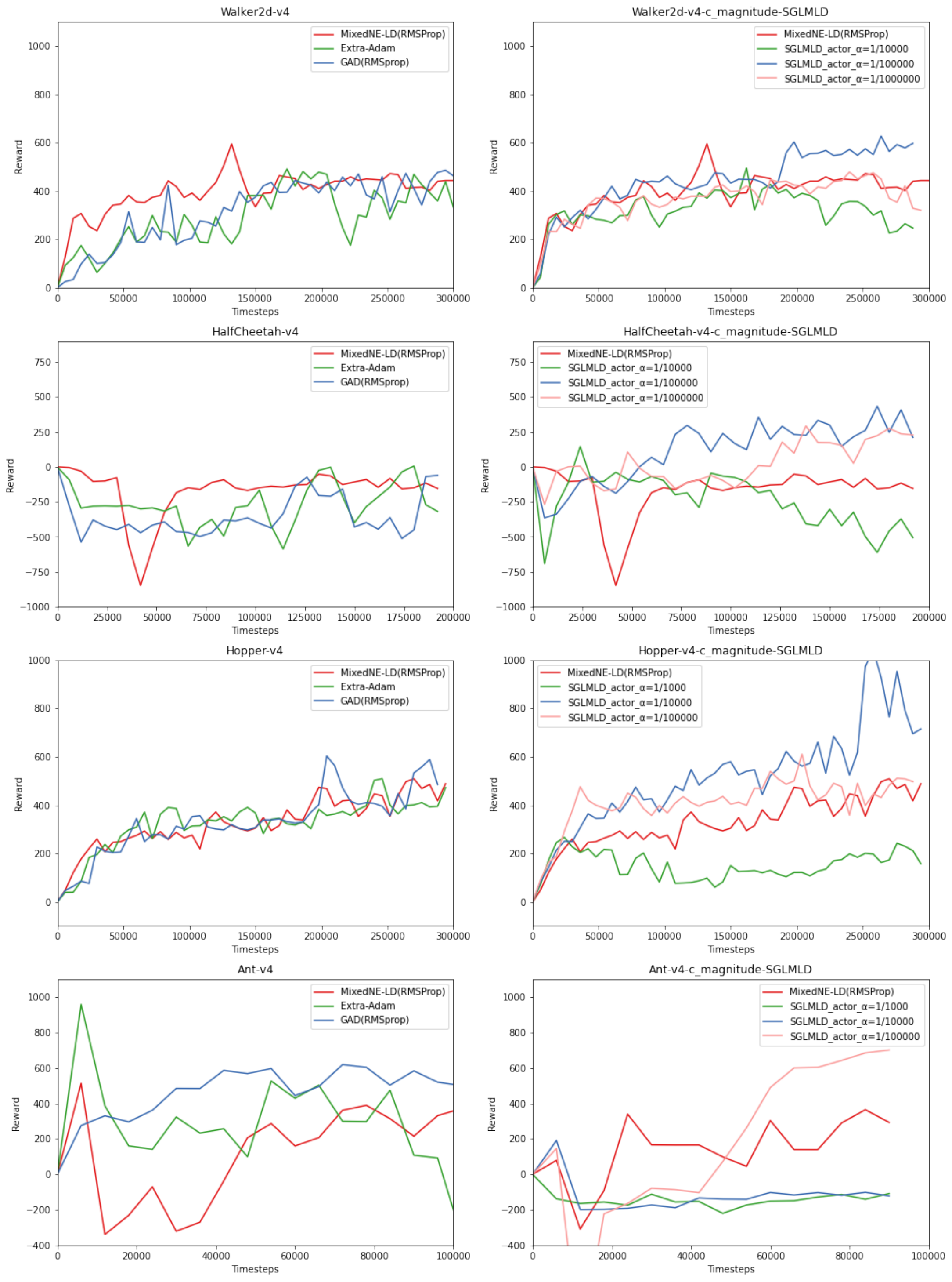

Figure 2: Varying α, $f(x) = x$, only minimizing actor (1/2)

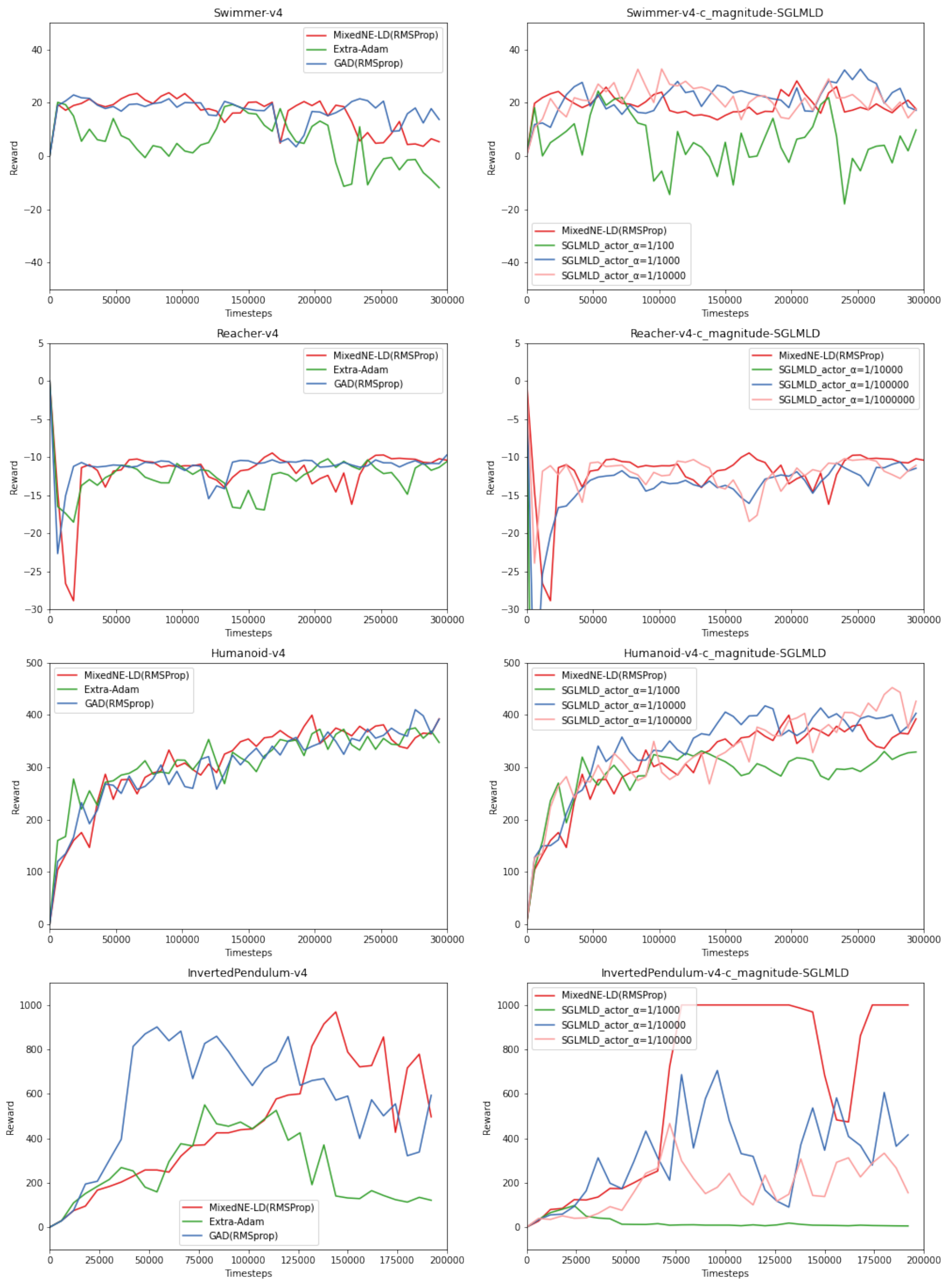

Figure 3: Varying α, $f(x) = x$, only minimizing actor (2/2)

Graph Explanation: On the left side, we have the baselines. On the right side, we have SGLD being compared to SGLMLD with only the actor being minimized, $f(x) = x$ and α being varied.

Optimal Value of α: $\alpha = 1/100,000$ was optimal for Walker2d, HalfCheetah and Ant. $\alpha = 1/10,000$ was optimal for Hopper. The other graphs were not definitive enough to draw similar conclusions.

Observations: On the left, the baselines generally perform similarly to each other. On the right, there seemed to always be an α that would improve the performance of SGLMLD over SGLD. It is only in Reacher and InvertedPendulum where this was not the case.

Insights: There seem to be cases that perform really well compared to the rest of the field. Walker2d with $\alpha = 1/100,000$ and Hopper with $\alpha = 1/10,000$ in particular. This might be the case where just the right amount of noise was introduced that led to that specific agent being able to discover something that the other agents could not.

5.1.1 Minimizing Actor with Arctan and Square

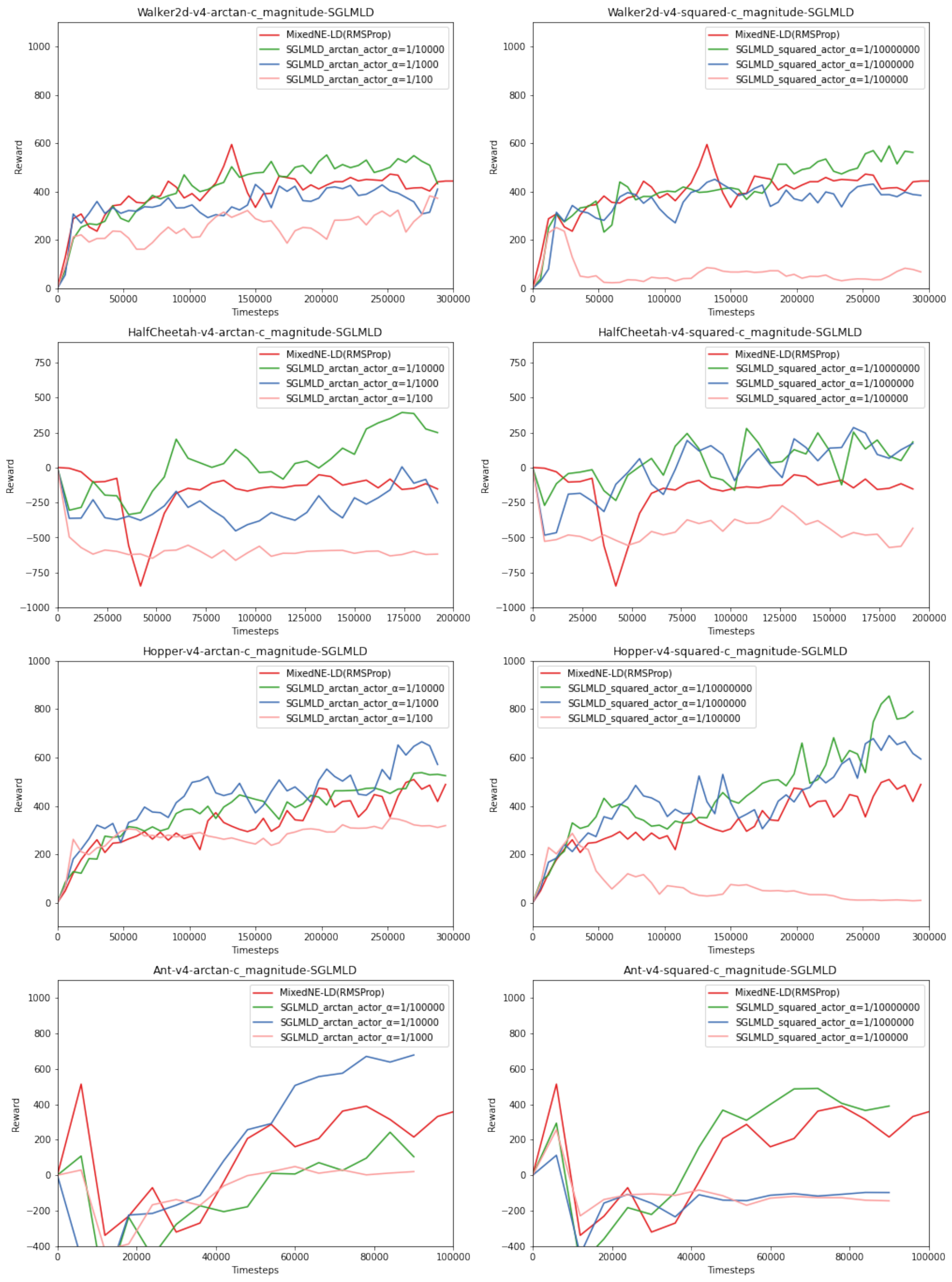

Figure 4: Varying α, only minimizing actor, using arctan and square (1/2)

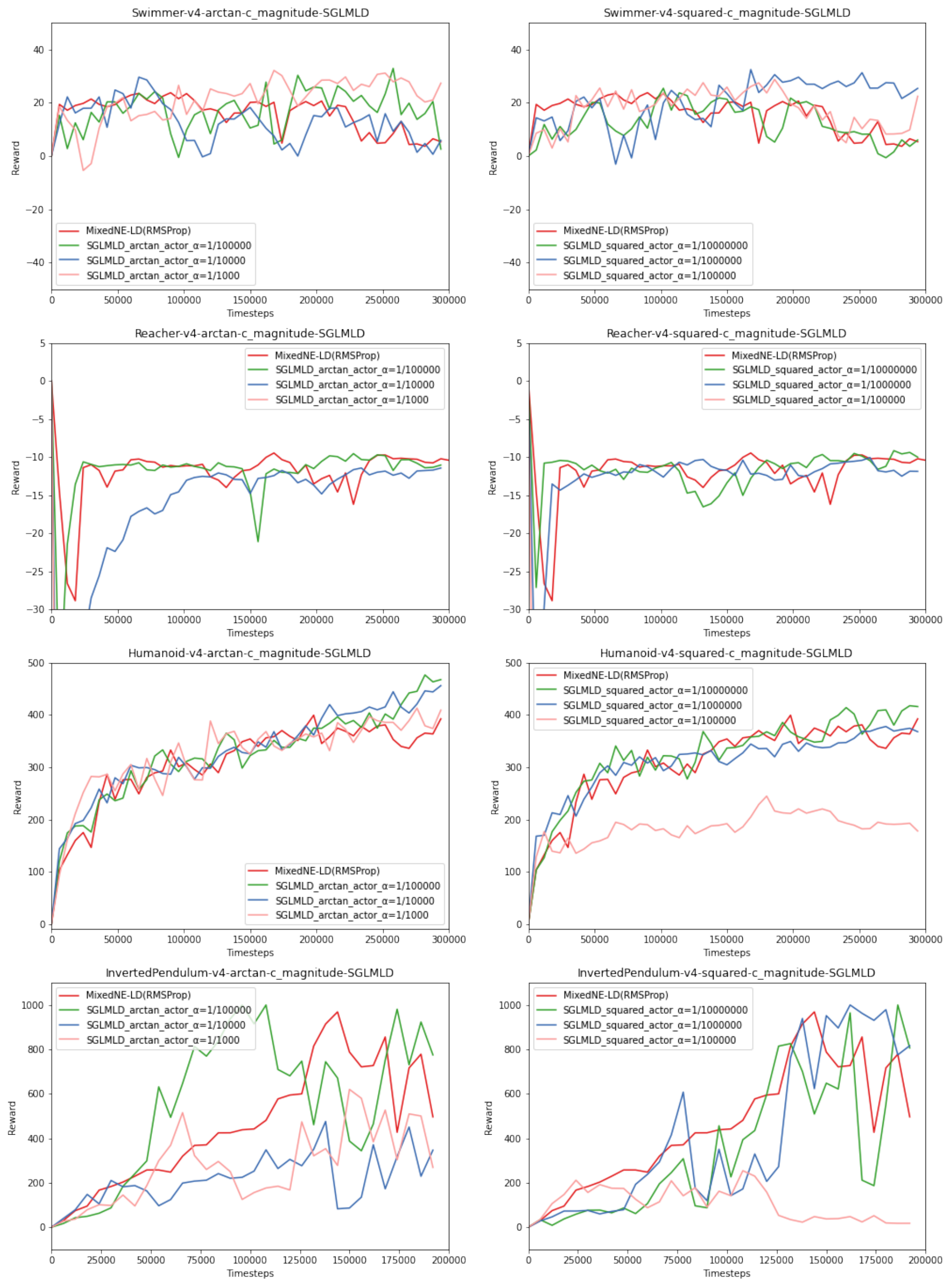

Figure 5: Varying α, only minimizing actor, using arctan and square (2/2)

Graph Explanation: On the left side, we have SGLD being compared to SGLMLD with only the actor being minimized, $f(x) = arctan(x)$ and α being varied. On the right side, we have the same thing except with $f(x) = x^2$.

Optimal Value of α: For $f(x) = arctan(x)$, $\alpha = 1/1,000$ was optimal for Hopper and Swimmer and $\alpha = 1/10,000$ was optimal for Walker2d, HalfCheetah and Ant. The other graphs were not definitive enough to draw similar conclusions.
For $f(x) = x^2$,$\alpha = 1,000,000$ was optimal for Swimmer and $\alpha = 1/10,000,000$ was optimal for Walker2d, HalfCheetah, Hopper, Ant and Humanoid. The other graphs were not definitive enough to draw similar conclusions.

Observations: Generally, there seemed to always be an α that would improve the performance of SGLDLM over SGLD. It is only Reacher and InvertedPendulum and $f(x) = x^2$ for Humanoid where this was not the case.

Insights: These graphs show how crucial it is to find the right α. The largest values of α performed worse than the baseline for both $f(x) = arctan(x)$ and $f(x) = x^2$ in Walker2d, HalfCheetah, Hopper and Reacher (where it did not manage to even appear on the graph). Humanoid and InvertedPendulum for $f(x) = x^2$ also behaved similarly.
To add on, sometimes, the middle value of α performs better than the smaller and larger values. This is true in $f(x) = arctan(x)$ for Hopper and Ant and $f(x) = x^2$ for Swimmer.
Additionally, when we divide α by 10 to go to the next smallest size, sometimes the behaviour appears to be the same as the baseline, hinting that the additional noise is not affecting the reward values meaningfully. However, when we further divide α by 10, we sometimes improve the reward values. This is present in both for Walker2d and $f(x) = arctan(x)$ for HalfCheetah.

5.2 Maximizing Adversary Only

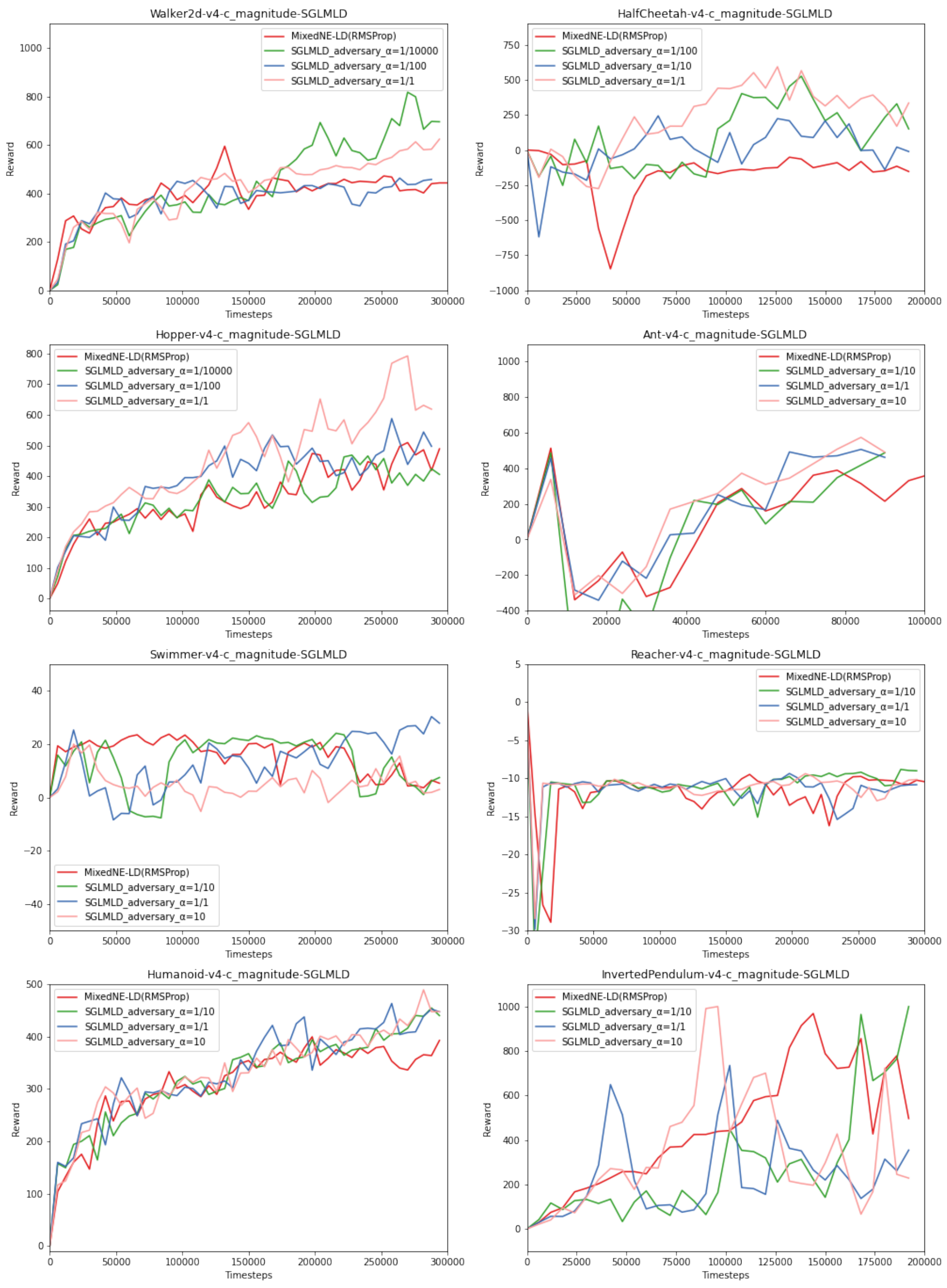

Figure 6: Varying α, only maximizing adversary

Graph Explanation: Now we have SGLD being compared to SGLMLD with only the adversary being maximized, $f(x) = x$ and α being varied.

Optimal Value of α: Ant, Reacher and InvertedPendulum did not have a clear value of α that outperformed the baseline. In all other environments, $\alpha = 1$ performed well.

Observations: Generally, there seemed to always be an α that would improve the performance of SGLDLM over SGLD. It is only in Ant, Reacher and InvertedPendulum where this was not the case.

Insights: When maximizing the adversary, the values of α needed to improve performance seemed to be higher than when compared to minimizing the actor. It could be the case where there are more gradients to optimize in the actor compared to the adversary. For the actor, adding a bit of noise to each gradient might add up a significant amount of noise. Thus to have an impact on the adversary that could have less gradients, a larger scale of noise needs to be included.
The value of α did not seem to have an impact on Humanoid and Ant, despite having an improvement over SGLD. This could be the case where just any noise would allow the agent to improve itself, where the actual scale of the noise is less important.

5.2.1 Maximizing Adversary with Arctan and Square

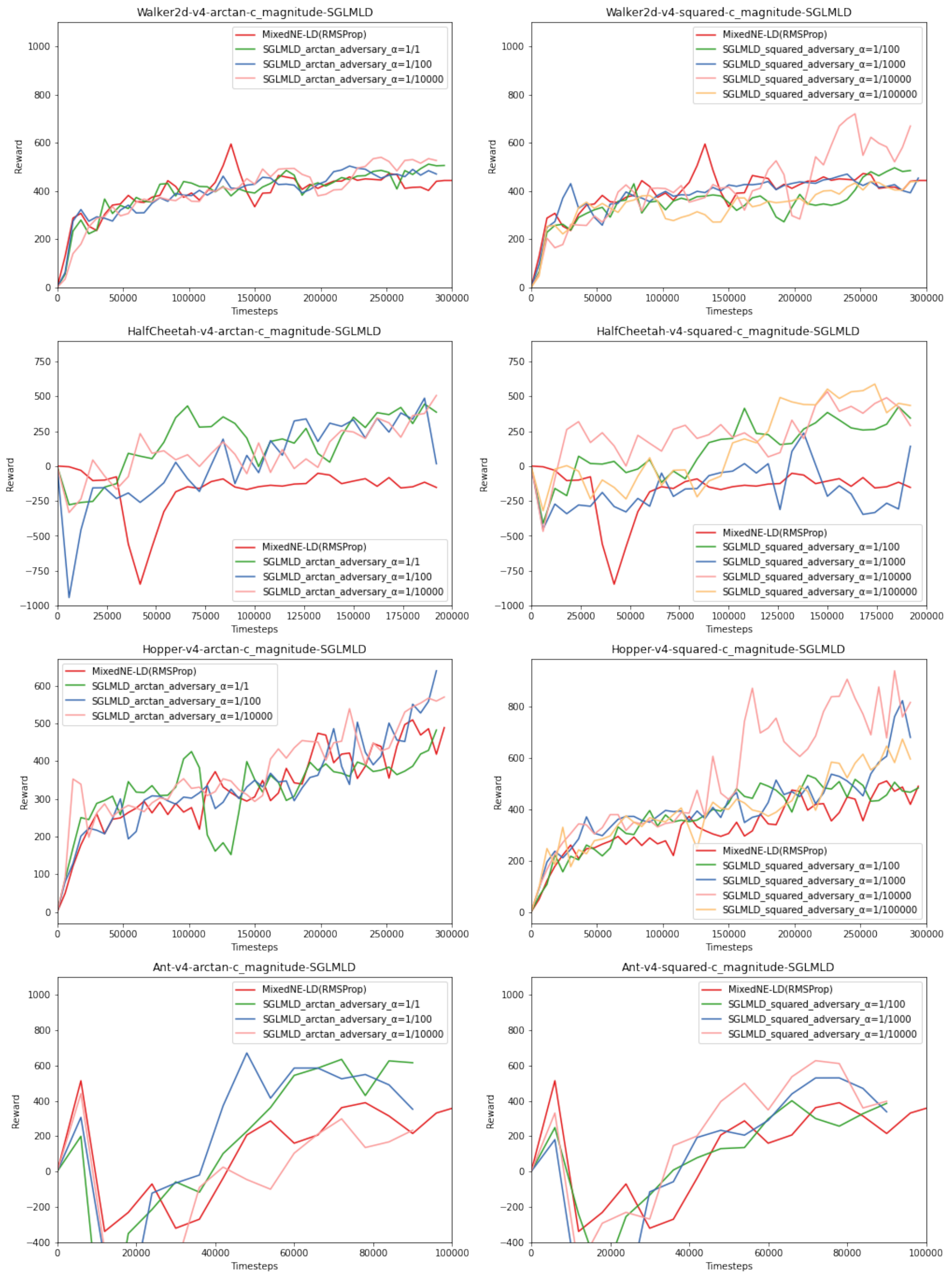

Figure 7: Varying α, only maximizing adversary, using arctan and square(1/2)

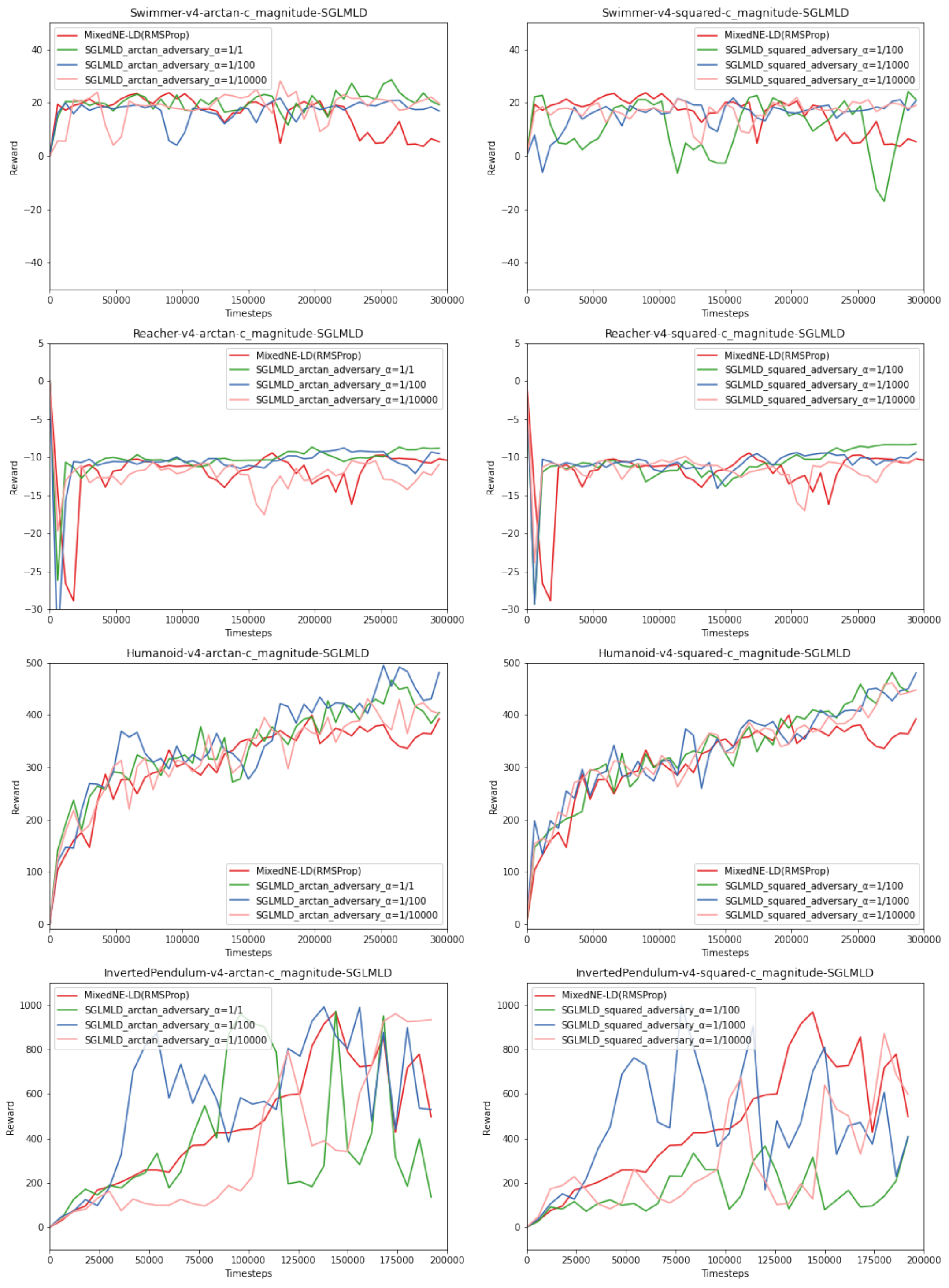

Figure 8: Varying α, only maximizing adversary, using arctan and square(2/2)

Graph Explanation: On the left side, we have SGLD being compared to SGLMLD with only the adversary being maximized, $f(x) = arctan(x)$ and α being varied. On the right side, we have the same thing except with $f(x) = x^2$.

Optimal Value of α: Reacher, InvertedPendulum and $f(x) = arctan(x)$ for Walker2d did not have a clear value of α that outperformed the baseline. For $f(x) = arctan(x)$, $\alpha = 1/100$ performed well for HalfCheetah, Hopper, Ant, Swimmer and Humanoid. For $f(x) = x^2$, $\alpha = 1/10,000$ performed well for Walker2d, HalfCheetah, Hopper, Ant, Swimmer, and Humanoid.

Observations: Generally, there seemed to always be an α that would improve the performance of SGLDLM over SGLD. It is only in Reacher, InvertedPendulum and $f(x) = arctan(x)$ for Walker2d where this was not the case.

Insights: Across these graphs, we can see that there is not much of a difference between the shape of the graphs regardless of the f used. As long as the value of α is appropriate, then it seems like how you vary f to penalize the distance from the reward threshold level is not important.

5.3 Minimizing Actor and Maximizing Adversary

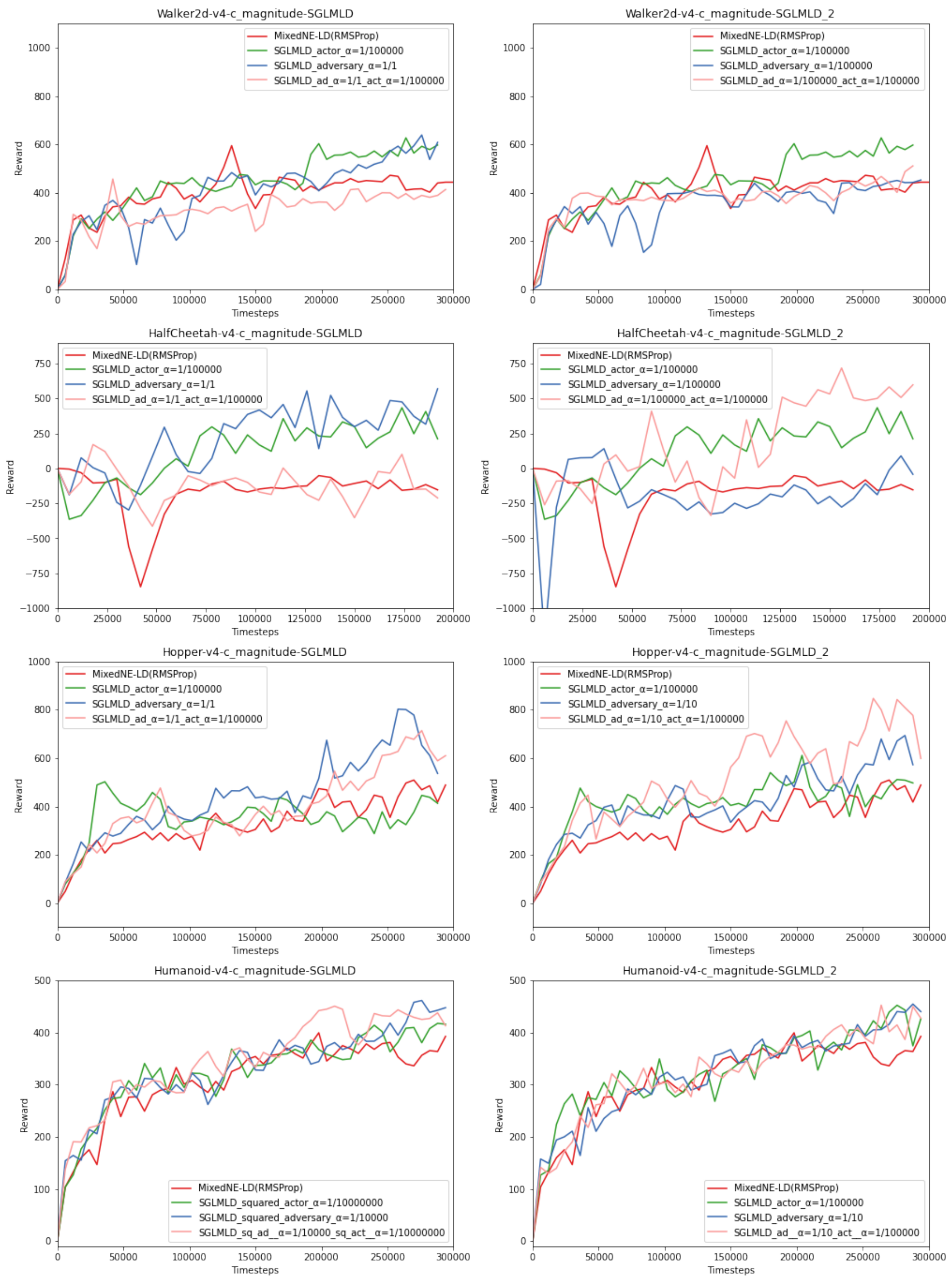

Figure 9: Varying α, minimizing actor and maximizing adversary

Graph Explanation: Now we attempt to use SGLMLD instead of SGLD to minimize the actor and maximize the adversary at the same time. We show 2 cases from Walker2d, HalfCheetah, Hopper and Humanoid. We will refer to the graph on the left as 1 and the right as 2.

For Walker2d 1, HalfCheetah 1 and Hopper 1, we show the the case where we minimize the actor with $f(x) = x, \alpha = 1/100,000$, the case where we maximize the adversary with $f(x) = x, \alpha = 1$, and the case where we do both at the same time.

For Walker2d 2 and Half Cheetah 2, we show the the case where we minimize the actor with $f(x) = x, \alpha = 1/100,000$, the case where we maximize the adversary with $f(x) = x, \alpha = 1/100,000$, and the case where we do both at the same time.

For Hopper 2, we show the the case where we minimize the actor with $f(x) = x, \alpha = 1/100,000$, the case where we maximize the adversary with $f(x) = x, \alpha = 1/10$, and the case where we do both at the same time.

For Humanoid 1, we show the the case where we minimize the actor with $f(x) = x^2, \alpha = 1/10,000,000$, the case where we maximize the adversary with $f(x) = x^2, \alpha = 1/10,000$, and the case where we do both at the same time.

For Humanoid 2, we show the the case where we minimize the actor with $f(x) = x, \alpha = 1/100,000$, the case where we maximize the adversary with $f(x) = x, \alpha = 1/10$, and the case where we do both at the same time.

Observations: Sometimes when only minimizing actor/maximizing adversary, the performance is better than the baseline, but when combining the two, the performance reduces to the same as the baseline. This is true for Walker2d 1, Walker2d 2 and HalfCheetah 1.

Sometimes performance is about the same as when either minimizing actor or maximizing adversary. This is true for Hopper 1, Humanoid 1 and Humanoid 2.

Sometimes performance is better than when either minimizing actor or maximizing adversary. This is true for HalfCheetah 2 and Hopper 2.

Insights: Both the actor and adversary are working together to optimize basically the same function, but when noise is added to them at the same time, a variety of effects can occur.

It seems that when using values of α and f that already work, it is unlikely that the performance will be worse than the baseline. This might be the case where the optimized amounts of noise that are added to both the actor and adversary are not enough to negatively affect the performance to that extent. However, they still make the agent more unstable, which degrades performance.

When the combined performance improves, it might be the case where the added noise works out to be more suitable for that particular environment.

When the combined performance does not show much improvement, it might be a combination of the above 2 situations that cancels each other out.

6 Limitations

One big limitation is the lack of computing power. Ideally, a lot more cases would have been tested with a larger sample size. Additionally, there was no time to do a grid search to adjust hyperparameters. Thus the same hyperparameters as [5] was used. The newer versions of the environments I used may have different optimal hyperparameters. With more computing power, a clearer picture might have been painted. However, the larger inferences should remain the same.

7 Conclusion

Generally, there usually was a value of α that outperformed the baseline. However, some environments were consistently unable to be optimized to be better than the baseline. This is namely Reacher and InvertedPendulum. This could be due to some environments not being suited to the way that I was adding noise.

The choice of f also seemed limited in effect. However, choosing the right α was usually crucial. The exception is in a few cases where it seemed like as long as you inject any additional noise, an improvement was able to be made.

Focusing on either minimizing the actor or maximizing the adversary did not seem to make a considerable difference. When combining the two, the performance was mixed between improving, staying the same, or reverting back to the baseline. It did not perform worse than the baseline, so it might be reasonable to always use SGLMLD over SGLD.

The noise we add in a scaling fashion through landscape modification seems to be a useful tool in increasing the range of possible values that the next iteration is able to take. This allows improvement especially when the current reward is far away from the previously achieved best reward.

As we deepen our knowledge of machine learning methods, we discover many different ways of approaching the same problem. The best method is probably still unknown to us, but we are slowly striving towards continued improvement.

Source Code: `https://github.com/Englishificational/DSA4199-FYP`

8 References

References

[1] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan, "How to escape saddle points efficiently," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1724–1732. [Online]. Available: https://proceedings.mlr.press/v70/jin17a.html

[2] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle points - online stochastic gradient for tensor decomposition," *CoRR*, vol. abs/1503.02101, 2015. [Online]. Available: http://arxiv.org/abs/1503.02101

[3] J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht, "Gradient descent converges to minimizers," 2016. [Online]. Available: https://arxiv.org/abs/1602.04915

[4] Y. N. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *CoRR*, vol. abs/1406.2572, 2014. [Online]. Available: http://arxiv.org/abs/1406.2572

[5] P. Kamalaruban, Y.-T. Huang, Y.-P. Hsieh, P. Rolland, C. Shi, and V. Cevher, "Robust reinforcement learning via adversarial training with langevin dynamics," 2020. [Online]. Available: https://arxiv.org/abs/2002.06063

[6] G. Gidel, H. Berard, P. Vincent, and S. Lacoste-Julien, "A variational inequality perspective on generative adversarial nets," *CoRR*, vol. abs/1802.10551, 2018. [Online]. Available: http://arxiv.org/abs/1802.10551