Domain Generalization in Reinforcement Learning: Analyzing TD(0), Fish, and IDGM with Polynomial Value Approximation

AP Research

Word Count: 4992

I. Introduction

Machine learning (ML), a process by which the computer uses data to mimic cognitive functions such as problem-solving or decision-making, is at the core of many recent advances in applied sciences and mathematics [1-5]. Reinforcement learning (RL) is one of the three main ML architectures to develop optimal control methods for dynamic control problems [6]. In the standard RL paradigm, an agent exists within an environment taking actions that influence its surroundings. Through trial and error, this agent learns optimal actions to accomplish a task [7,8]. The objective is communicated to the computer via rewards the agent receives at each time step [9]. More formally, this process of interacting with an environment is presented as a Markov Decision Process (MDP), where a policy π maps every state $s \in S$ (a finite set of states) to an action $a \in A(s)$ (set of all actions possible in state s), and receives some reward $r \in R$ (set of possible rewards) [8,10,11]. This process is deterministic, but can be used in stochastic control, where the state transitions are uncertain. Due to their broad definition, MDPs can be used to model complex problems, differential state transitions, etc. lending well to their use in pioneering research [12].

Rewards are a numerical signal, and actions taken may affect immediate and delayed rewards [8]. The agent aims to learn a policy that maximizes its long-term reward. This is done by considering the Bellman equation, describing the action-value function q:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \qquad (1)$$

denoting the expected value of taking an action in a given state following policy π [10]. To take future rewards into account, the value is defined as the sum of rewards the agent receives, discounted by some rate γ for how many timesteps later it was earned [8,13]. The Bellman equation can be rewritten as:

$$q_\pi(s,a) = \mathbb{E}_\pi[\sum_{k=0}^{T} \gamma^k r_{t+k+1} | S_t = s, A_t = a] \qquad (2)$$

where T is a terminal state or ∞ if the problem continues indefinitely [11]. This infinite horizon discounting is problematic to compute [14], so methods such as dynamic programming, temporal difference learning TD(λ), State-Action-Reward-State-Action (SARSA), and Q-learning have been developed with a variety of update rules to approximate this value function over a large number of episodes (iterations of the MDP to a terminal state) [6,8,9,15,16].

A policy denoted π_* is considered optimal if for every state in the environment, $v_\pi(s) \geq v_{\pi'}(s)$, meaning that $\pi \geq \pi'$ for every alternate policy π' [8]. Similarly to the way policies are evaluated, the aforementioned algorithms iteratively refine the policy over a large number of episodes, asymptotically approaching π_* in the limit [15]. For discrete, finite valued states, and stationary environments, the model should converge on the optimal policy. For control problems with continuous state variables, the number of states quickly approaches infinity, meaning an optimal policy can at best, be closely approximated [17]. Approximation methods share much in common with interpolation and regression of sampled states to a value function on unseen domains [18]. A state is represented by a feature vector, where each element corresponds to a defining characteristic of the state space. The approximate value function is given by $\hat{v}(s,w)$ for any weight vector

$$w \doteq (w_1, w_2, w_3 \ldots w_d)^\top \qquad (3)$$

where $w \in \mathbb{R}^d$ and $\hat{v}(s,w)$ represent the mapping of features onto an estimation of the value of that state [8].

Often, w is smaller than the number of states, meaning small changes in the weights affect large numbers of states [8]. This enables the model to generalize, predicting the value of states it hasn't seen before. It is important when the state space is too immense to explore every possible state, such as in continuous cases or with state-defining feature vectors of high dimensions [19]. Models that generalize are crucial to future advances. Current models are exceptional at perfecting the initial environment but fail catastrophically in marginally different situations [20]. In literature, this is referred to as domain generalization (DG), predicting the value function not just for unseen states within the possible training states, but for potential states in new

domains [21,22]. DG is similar to extrapolation, though data points are instead learned throughout training. Despite its importance, numerous recent authors have cited a lack of research on the effectiveness of regression-related approaches in handling the challenge in RL contexts [18,20,23].

II. Literature Review

Promising approaches to value function approximation fall into two broad categories: neural networks and linear function approximators. The implications of each are substantial, but this review will focus on their effectiveness in DG.

A. Deep Learning

Convolutional neural networks, multi-layer perceptrons, and other forms of deep learning for pattern recognition are some of the most recognizable forms of artificial intelligence (AI). Incorporating deep learning into RL (DRL) has shown the most promise in achieving human-level autonomy on control tasks in a vast array of applications, as well as on objectives with unexplorable numbers of states [24,25]. Models like a distance-sensitive deep-q offline RL function approximator have performed well on state-of-the-art RL benchmarks [26]. Despite this success, out-of-distribution DG remains challenging for these models. Current steps toward a generalizing model typically involve trying to estimate the gradient of the Q-function to guide generalization [27]. This approach appears successful in classification tasks but is heavily reliant on additional information to guide the generalization process, and is not suited to pressing RL tasks [28]. The challenge with DG via neural networks is estimating the value of the true gradient. This is known as the 'gradient alignment paradox,' and is currently an open problem in ML given the models' vast interworkings [29]. DRL may still see advances toward perfect DG, but considering the plethora of literature and research surrounding this field, progress toward solving such a difficult problem may be far removed.

B. Linear Function Approximators

A promising DG alternative to DRL may be linear function approximation. This method employs linear schemes of basis functions, which are easily computable functions whose combination can represent more complex behavior [30]. This method is more closely related to the weight vector introduced in the last section, as each weight is iteratively updated via the same stochastic gradient descent strategies from tabular methods with computational ease [31,32]. Using a linear superposition of fitting functions to the state feature vector reduces the problem dimensionality from the number of states to the number of weight coefficients [17]. Reduced model parameters also mean the value gradient can be easily computed using linear methods rather than with deep neural networks [32]. There are more possibilities for basis functions than data points [33], but certain bases have shown more promise in ML architectures than others. The most common are radial basis functions (RBF) and polynomial basis functions. Others, such as Fourier and proto-valued (utilizing the Laplacian to self-learn bases), also have numerous applications [30].

1) Radial Basis Functions: RBFs are a class of functions that compute the function value using proximity to known data points in the distribution. Each point acts as a centroid, with proximity evaluated using a Gaussian or inverse exponential, and the weight is determined by the input value [34]. If a is a given input, then the Gaussian, single-dimensional case is given by:

$$f_{\beta}(a) \doteq \sum_{i=1}^{N} e^{-\beta\left\|a-a_i\right\|^2} v_i \qquad (4)$$

where β is a smoothing parameter, and whose sum is computed as a weighted average to ensure a valid probability distribution [35]. RBFs are alluring for their smooth, converging interpolating properties. They rely on increasing the spans of distance shifts for increased generalization, proving more computationally expensive than simpler bases such as polynomial alternatives [36]. While these radial basis methods produce smoothly differentiable functions, they aren't well controlled near the edges of domains, because the inverse distance value relationship in unexplored regions tends the value to zero [8]. This is problematic in DG, where the behavior of unseen domains matching the function is crucial. Challenges also arise from large numbers of state space features that are exponentially more complex to compute, known as the curse of dimensionality [37]. The

combination of these issues means that RBFs are not well-suited to DG tasks.

2) Polynomial Basis Functions: A more natural choice of basis function to use is an nth-degree polynomial. This was one of the earliest linear methods examined in literature due to its simplicity, with polynomial regression methods dating back to Gauss [38]. It was also one of the first methods to be abandoned by early ML researchers for more sophisticated regression and approximation approaches to model increasingly complex situations [8]. This is partially a result of the erratic nature of higher-order fittings and inadequacy at keeping the behavior of even relatively simple interactions bounded for unseen domains [39]. Polynomial bases of lower degrees have successfully matched overall function behavior via least squares minimization, which is both computationally efficient and applicable in higher dimensional cases [40]. Despite this efficacy, polynomial fitting functions especially suffer from overfitting to the training data. This is a hurdle for any successfully generalizing architecture by representing the broad value function shape while being exact enough to behave near-optimally [20]. Numerous strategies of algorithmic and state aggregation (grouping of similar, trivial features) have been examined in depth. Due to the number of parameters, well-generalizing models are usually bound to using data augmentation techniques (artificially expanding the data set using noisy copies of the original data) [41,42]. Data augmentation is limiting as it reinforces existing biases and relies on a more thorough knowledge of the dataset. This invites the need for a model that can achieve a similar level of DG using purely source data.

The solution may come from the aforementioned gradient matching objectives across domains. This is an extension of risk extrapolation methods, that aim to minimize the inherent, and probable, inaccuracy of out-of-distribution value prediction [43]. Risk extrapolation addresses the uncertainty by minimizing the error to every possible end behavior. Due to the nature of such a prediction, it can't be perfect, but modern empirical risk minimization (ERM) algorithms are still flexible enough to outperform more complicated DG approaches [44]. A relevant method of minimizing this loss is an inter-domain gradient matching objective to target domain distributions where the gradient behavior in the test domain matches closely with the training domain gradient [21]. This can be done using the gradient inner product (GIP) to learn the invariant features between the two domains [22].

$$\mathcal{J}(\Theta) \doteq max_{\Theta} \sum_{i,j \in S}^{i \neq j} \nabla_{\Theta} \boldsymbol{\mathcal{L}}_{\mathcal{D}_i}(\Theta)^{\top} \nabla_{\Theta} \boldsymbol{\mathcal{L}}_{\mathcal{D}_j}(\theta) \qquad (5)$$

So far this approach has only been studied in supervised learning (SL). Using the GIP with polynomial basis functions may offer a more discernible alternative to less understood DL methods if used to control the function's limit to match this gradient objective. Thus, the central question driving this study: To what extent can gradient-based polynomial value function approximators generalize to domains outside of the training states used in a reinforcement learning environment?

III. Methods

To evaluate the effectiveness of various DG architectures with polynomial linear value function approximation, two pioneering SL algorithms were compared to zero-step temporal difference learning (TD(0)): Fish and Inter-Domain Gradient Matching (IDGM) [22]. Both attempt to maximize the GIP in learning domain invariant features. While the losses on the training set may not be lower, they perform significantly better on testing data [22]. Fish relies on a simpler, less accurate, first-order approximation of the 2nd derivative. In contrast, IDGM computes the complete Hessian matrix, which is computationally infeasible even for a relatively modest feature vector and state space [45]. The performance of true IDGM can still be assessed, using Newton's finite differences, which provide highly accurate estimations of the second partial derivative for small values of delta (eg. 1e-4) [32]. Both algorithms improve model performance by directly optimizing the direction of policy improvement, and traditional loss methods such as mean square error (MSE) don't directly reflect ERM convergence. ERM still serves as a good baseline for comparing DG algorithms, and the RL counterpart to ERM using stochastic gradient descent to optimize the loss is commonly TD(λ) [46]. For this study, only the next reward state was considered: TD(0). Despite n-step TD having different convergence properties (approaching Monte-Carlo sampling estimates of the value with

higher values of λ) [8], neither Fish nor IDGM optimizes these metrics. Therefore evaluating the policy convergences of the simpler TD(0) case is suitable.

As is common practice to evaluate a benchmark RL task regarding DG, a simple grid world environment and navigation objective were considered [11,20,47]. Similar to the curriculum MDP episodic task performed by Narvekar and Stone, an environment with four rooms, each with 5x5 cells and adjoining corridors, was chosen as the testbed for these algorithms [11]. The agent was trained on all tasks that lead from the room in the top-left to the bottom-right, and its DG capabilities were tested on all paths leading from the top-right to the bottom-left. This configuration with wall arrangements and narrow optimal paths introduces enough complexity to distinguish high-performance models while keeping the computational resources for training low. For the three weight update schema described (TD(0), Fish & IDGM), a grid-search of hyperparameter combinations was conducted, using a low, medium, and high value of the learning rate, outer step size, and degree polynomial used (Figure 3) [48]. The specific tunings of these variables were based on established baselines in the field and

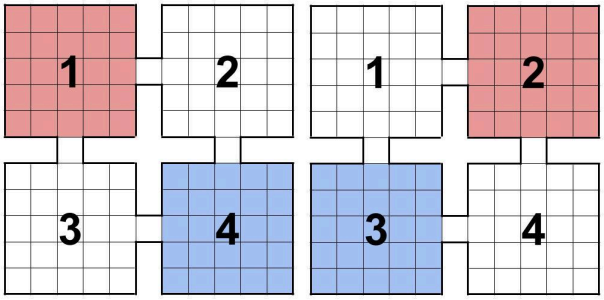


Figure 1 | The agent started in a random cell in the red room, and navigated to a designated random cell in the blue room. It was trained on all tasks from room 1 to room 4, and tested on all tasks from room 2 to room 3.

Learning Rate

TD(0)	Fish	IDGM
0.01	0.01	0.001
0.05	**0.05**	**0.005**
0.1	0.1	0.01

Outer Step Size

TD(0)	Fish	IDGM
-	0.001	0.0005
-	**0.005**	0.001
-	0.01	**0.005**

Degree

TD(0)	Fish	IDGM
2	2	2
3	3	3
4	**4**	**4**

Figure 3 | Each table shows the three hyperparameter values tested for each algorithm. Every combination of three unique values from each set was tested in the grid search of hyperparameters. Optimal tunings according to DG test performance are in bold.

Algorithm 1 Fish.

for iterations = $1, 2, \cdots$ **do**
  $\widetilde{\theta} \leftarrow \theta$
  for $\mathcal{D}_i \in \texttt{permute}(\{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_S\})$ **do**
    Sample batch $d_i \sim \mathcal{D}_i$
    $\widetilde{g}_i = \mathbb{E}_{d_i}\left[\frac{\partial l((x,y);\widetilde{\theta})}{\partial \widetilde{\theta}}\right]$ //Grad wrt $\widetilde{\theta}$
    Update $\widetilde{\theta} \leftarrow \widetilde{\theta} - \alpha \widetilde{g}_i$
  end for

  Update $\theta \leftarrow \theta + \epsilon(\widetilde{\theta} - \theta)$
end for

Algorithm 2 Direct optimization of IDGM.

for iterations = $1, 2, \cdots$ **do**
  $\widetilde{\theta} \leftarrow \theta$
  for $\mathcal{D}_i \in \texttt{permute}(\{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_S\})$ **do**
    Sample batch $d_i \sim \mathcal{D}_i$
    $g_i = \mathbb{E}_{d_i}\left[\frac{\partial l((x,y);\theta)}{\partial \theta}\right]$ //Grad wrt θ

  end for
  $\bar{g} = \frac{1}{S}\sum_{s=1}^{S} g_s, \quad \widehat{g} = \overbrace{\frac{2}{S(S-1)}\sum_{i,j\in S}^{i\neq j} g_i \cdot g_j}^{\text{GIP (batch)}}$
  Update $\theta \leftarrow \theta - \epsilon\,(\bar{g} - \gamma(\partial \widehat{g}/\partial\theta))$
end for

Figure 2 | Pseudocode for Fish and IDGM implementation as given by Yuge Shi et al. [22]

preliminary empirical testing of the algorithms within the environment. A discount factor of γ=0.9 was selected to promote long-term rewards while ensuring the policy can converge [11,20]. Each instance was run 5 times for 10,000 episodes with batch size 25 according to the start-end room conditions dictated by the DG schema from before. MSE for loss minimization and average episode length to determine policy accuracy were recorded every 100 episodes and exported at the end of the run. A run diverged and was terminated if any episode exceeded 50,000 steps to conserve compute.

The data was collected with Google Colaboratory, a cloud-based Jupyter Notebook service [49].[1] The program was developed and executed in Python 3, using libraries such as NumPy, Pandas, and MatPlotLib for calculations, data output, and visualization. All computations were performed within the cloud-based environment to ensure replicable performance. Throughout the training, the convergence of the three algorithms was tracked using average episode length, and MSE – standardized metrics for measuring policy performance and loss function minimization capabilities in episodic RL tasks [8,46]. Outcomes of the training instances were compared using the Mann-Whitney two-sample U-test[2] pairwise between Fish and TD(0), and IDGM and TD(0) [50]. This non-parametric test was employed because it doesn't make assumptions about the underlying distribution shape or changes in variance between the samples [51]. Data on the convergence of the algorithms was collected from the final 1,000 episodes, averaged over five, independent runs. The null hypothesis was defined as no significant difference in the distributions at α=0.05. Since some combinations of hyperparameter tunings were volatile and had trials in which the training diverged, those trials were omitted from the statistical analysis yet still recorded in the data tables (Appendices 2-4). Additionally, optimal and overall performance were considered to gauge hyperparameter sensitivity.

IV. Results & Discussion

The mean episode length across the final 1,000 episodes was averaged across five trials for each hyperparameter tuning. The results for each algorithm's performance across hyperparameter tunings can be found in Figures 8-13. Then, the tuning with the minimum average test environment episode length was taken as optimal for each algorithm concerning DG. The mean episode length on the training and testing environments are shown together for each algorithm in Figure 4, alongside the overall averages across all converging trials. All Mann-Whitney U-test results between pairwise combinations of algorithms for each metric are displayed in Figure 6 and data on the number of diverging trials in Figure 5. The optimal DG converging value functions for each algorithm are graphed at a constant goal position in Figure 7. Overall, the results of this study indicate that, under polynomial value function approximation, Fish and IDGM significantly outperform TD(0) in DG, while neither algorithm outperformed the other. Both DG algorithms experience greater sensitivity to hyperparameters, resulting in higher divergence rates. These findings highlight the tradeoff between generalization capability and robustness, indicating that although DG algorithms improve test performance under idealized conditions, practical applications may be hindered by both complexity and computational costs.

The influence of hyperparameter tunings is crucial in determining how well a model is at converging to a global minimum loss. Too high an initial learning rate may cause the weights to diverge, and decaying it too quickly can subdue necessary changes to the value function, causing the model to settle on a suboptimal policy. In Fish, both mean episode lengths above 100 were from the largest step sizes tested, indicating it never settled on an optimum policy, even if the trials converged. Similarly, the outer step size influences the model's tendency to prioritize domain-generalizing features. A small outer step size may lead to cautious updates and overfitting because it converges slowly, while an aggressive tuning makes the model overshoot, risking destabilizing learning. For example, every pairing of the larger outer step size for IDGM with a degree 4 polynomial eventually caused all trials to diverge. A

[1]The hardware allocated for training included an AMD EPYC 7B12 CPU, 13 GB of RAM (13290460 kB), and an NVIDIA Tesla T4 GPU with 15 GB of GPU memory and CUDA Version 12.4.

[2]Alternatively referred to as the Wilcoxon rank-sum test or Mann–Whitney–Wilcoxon test

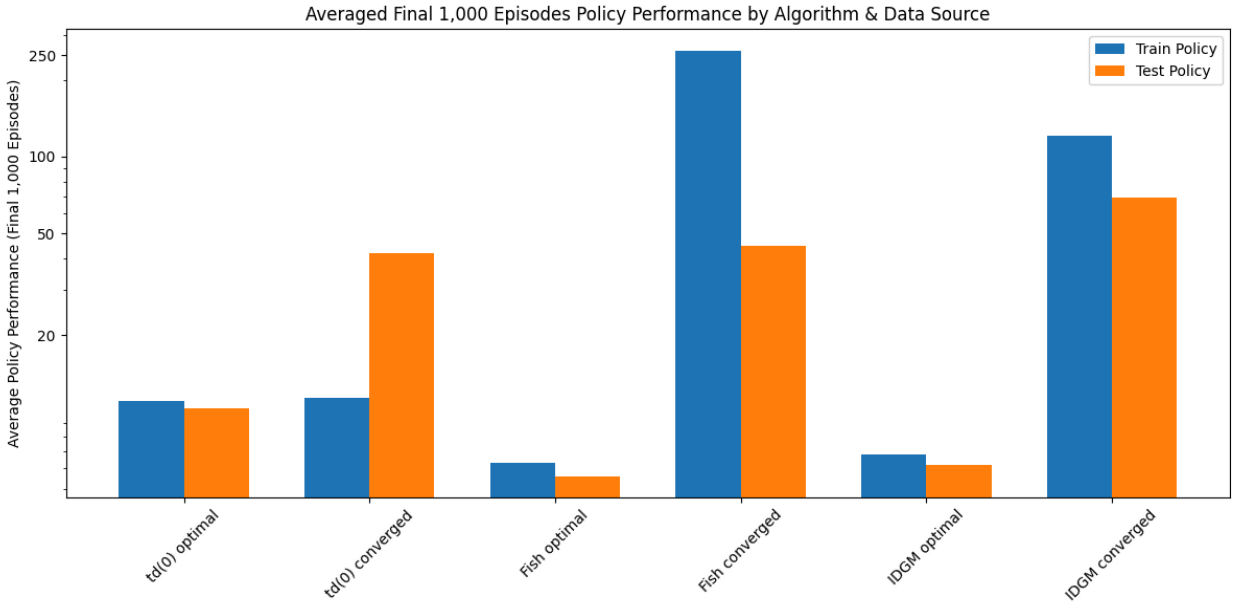


Figure 4 | The policy performance of each algorithm for both training and testing environments are grouped. The optimal DG hyperparameter tunings are depicted next to the average global performance for each algorithm. Note that the scales are logarithmic, as Fish and IDGM global performance was significantly worse even when diverging trials were excluded.

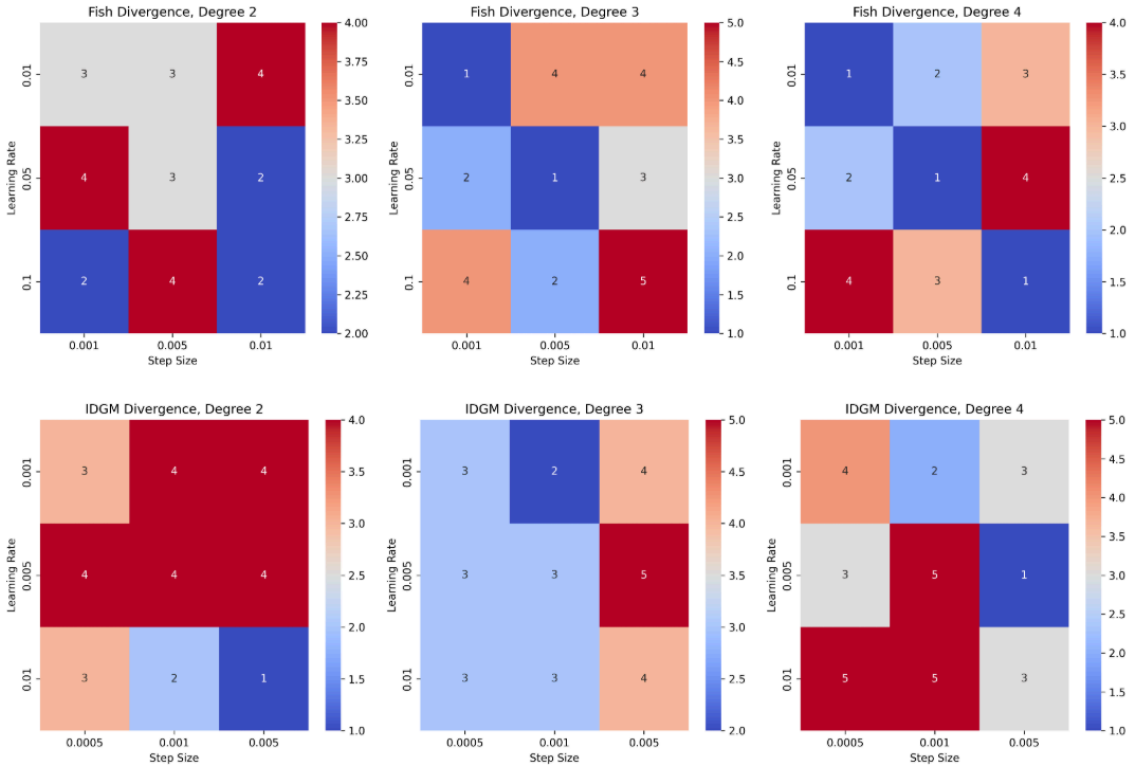


Figure 5 | Heatmap coloring of the diverging trials for each hyperparameter tuning, where warmer colors indicate a higher proportion of diverged trials, and the degree is mapped separately across three heatmaps for each algorithm

Optimal DG Policy Performance in Train Environment

Algorithms	U-statistic	P-value
TD(0) - **Fish**	5	0.0003
TD(0) - **IDGM**	8	0.00064
Fish - IDGM	57.5	0.87288

Optimal DG Policy Performance in Test Environment

Algorithms	U-statistic	P-value
TD(0) - **Fish**	8	0.00064
TD(0) - **IDGM**	1.5	0.00012
Fish - IDGM	55	0.7414

Global Policy Performance in Train Environment

Algorithms	U-statistic	P-value
TD(0) - Fish	0	$p \gg 0$
TD(0) - IDGM	0	$p \gg 0$
Fish - IDGM	54	0.69654

Global Policy Performance in Test Environment

Algorithms	U-statistic	P-value
TD(0) - Fish	59	0.9442
TD(0) - IDGM	53	0.64552
Fish - IDGM	53	0.64552

Optimal DG Average MSE in Train Environment

Algorithms	U-statistic	P-value
TD(0) - **Fish**	0	$p \gg 0$
TD(0) - **IDGM**	0	$p \gg 0$
Fish - **IDGM**	0	$p \gg 0$

Optimal DG Average MSE in Test Environment

Algorithms	U-statistic	P-value
TD(0) - Fish	0	$p \gg 0$
TD(0) - IDGM	0	$p \gg 0$
Fish - **IDGM**	2	.00014

Figure 6 | U-statistics and P-values are listed for all pairs of algorithms between optimal and global performance, for both average episode length and average MSE throughout the final 1,000 episodes. Metrics were averaged across trials before comparison. The better algorithm is bolded for statistically significant results.

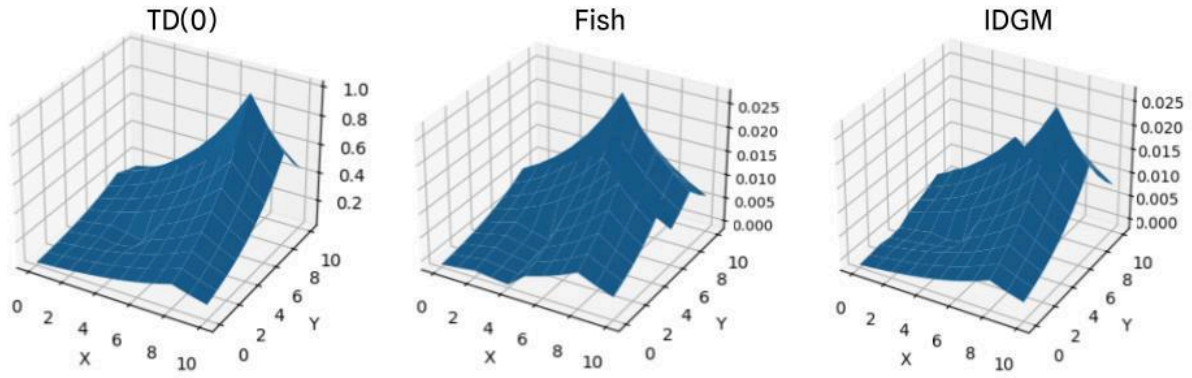


Figure 7 | The converged value functions maps for each optimal hyperparameter tuning over all states and a given goal position, where the greedy policy is argmax of cells adjacent to the agent, meaning it will always climb directly uphill towards the goal. Note that the shapes of the distributions remained largely unchanged, however Fish and IDGM had a 40x smaller spread for the value functions.

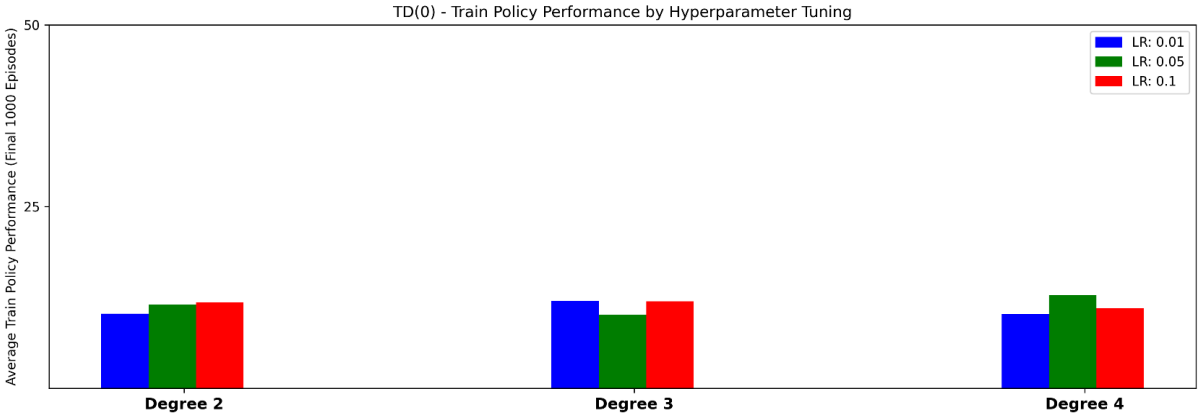


Figure 8 | Average policy performance after convergence across the last 1,000 episodes of training of all TD(0) hyperparameter tunings on the training environment

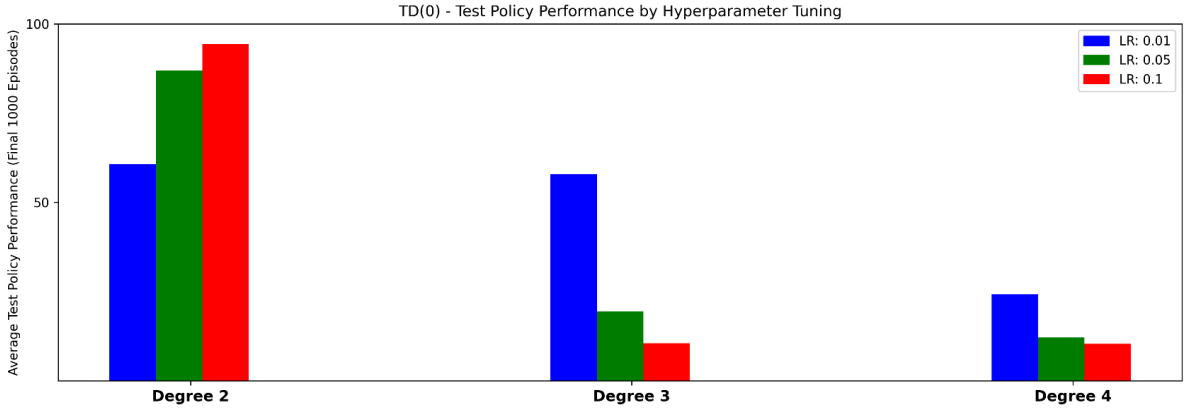


Figure 9 | Average policy performance after convergence across the last 1,000 episodes of training of all TD(0) hyperparameter tunings on the testing environment

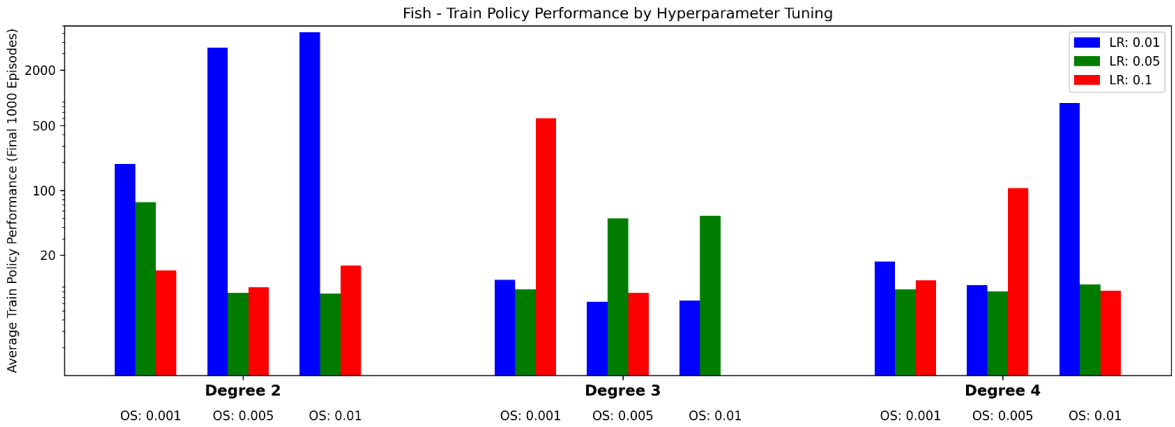


Figure 10 | Average policy performance after convergence across the last 1,000 episodes of training of all Fish hyperparameter tunings on the training environment graphed on a logarithmic scale

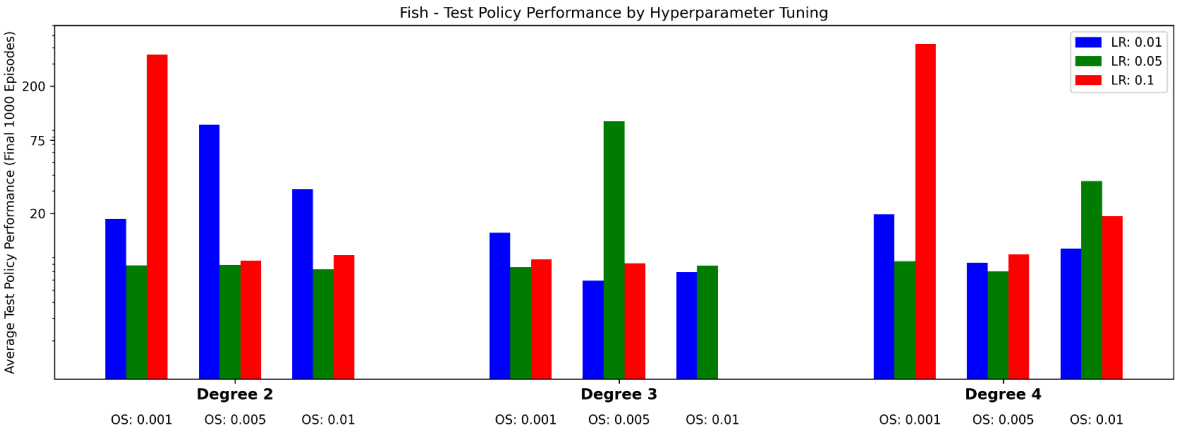


Figure 11 | Average policy performance after convergence across the last 1,000 episodes of training of all Fish hyperparameter tunings on the testing environment graphed on a logarithmic scale

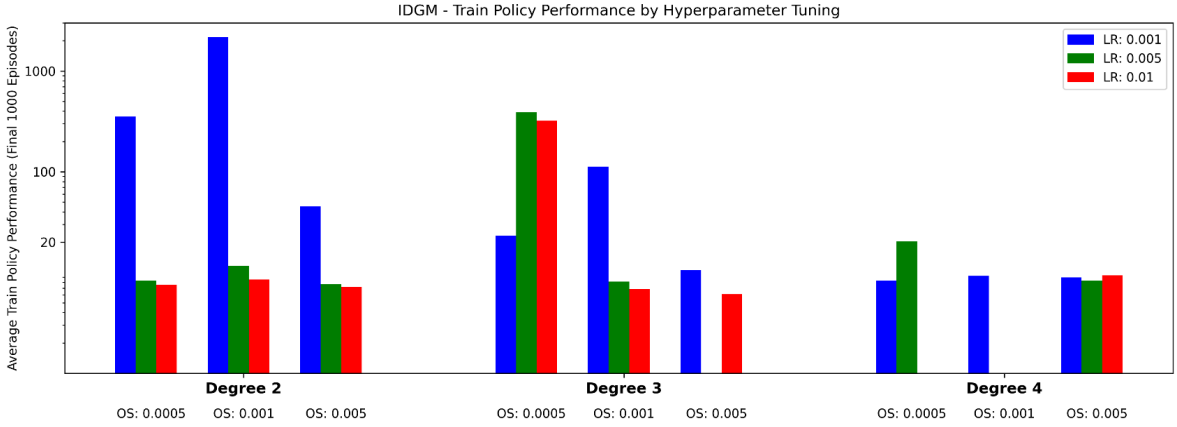


Figure 12 | Average policy performance after convergence across the last 1,000 episodes of training of all IDGM hyperparameter tunings on the training environment graphed on a logarithmic scale

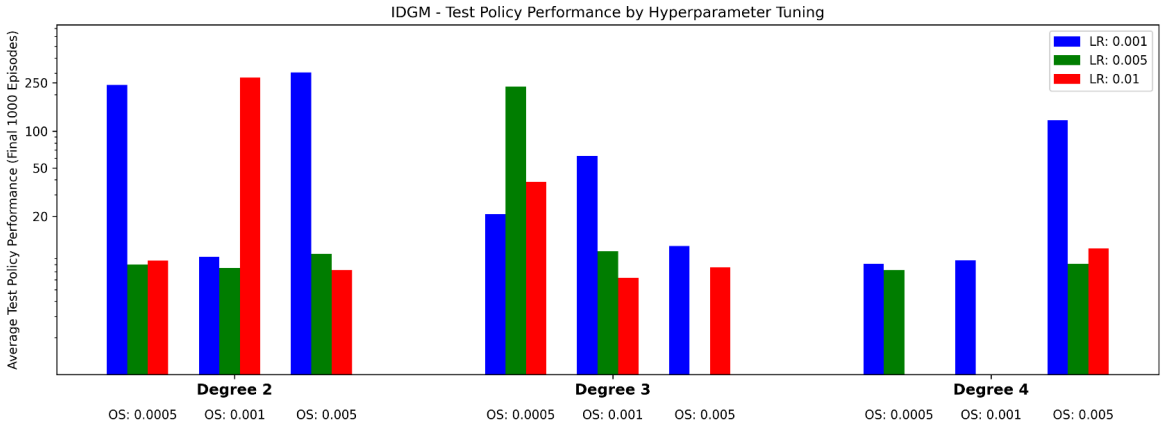


Figure 13 | Average policy performance after convergence across the last 1,000 episodes of training of all IDGM hyperparameter tunings on the testing environment graphed on a logarithmic scale

low-degree polynomial limits how expressive the value function is at guiding the agent around walls and obstacles. These areas in the feature space may have steep, highly nonlinear changes in the optimal value function in that region. These features are challenging to represent and may explain why degree 2 polynomials perform worse in all 3 algorithms. It is essential to find the optimal performing hyperparameter combination for the environment the model is trained on before comparing the effectiveness of one algorithm to another.

According to experimental values, the optimal combinations for each algorithm were: learning rate 0.1 and degree 4 for TD(0); learning rate 0.05, outer step size 0.005, and degree 4 for Fish; and learning rate 0.005, outer step size 0.005, and degree 4 for IDGM. Both Fish and IDGM were shown to outperform TD(0) when comparing optimal performance, with a p-value of .00064 and U-statistic of 8 for Fish & TD(0), and a p-value of .00012 and U-statistic of 1.5 for IDGM & TD(0), both of which are very statistically significant at a critical value of 0.05 (U-value of 30). This near-zero probability strongly favors the alternative that convergence of optimal tunings of both DG algorithms did outperform classical TD(0). Interestingly, neither Fish nor IDGM outperformed the other (p-value of 0.7414 and U-statistic of 55). This relationship also holds for performance in the training environment, with p-values of 0.0003, 0.00016, and 0.87288 and U-values of 5, 8, and 57.5 for the three tests, respectively. It can be shown that for global performance, TD(0) did not perform better than Fish or IDGM in terms of DG. Fish and TD(0) were found to have a p-value of 0.9442 and a U-value of 59, while IDGM and TD(0) had a p-value of 0.64552 and a U-value of 53. TD(0) was significantly better than both in the training environment (p-values of ~0 for each).

With regards to MSE, Fish and IDGM both converged to better mean values in training (also p-values of ~0). This is surprising because neither Fish nor IDGM directly minimizes MSE, and their graphs stay roughly constant throughout training. Analyzing the convergence graphs (Appendix 5), the most rapidly converging policies see the most significant improvement within the first 2,000 episodes. In most TD(0) convergence cases, the MSE also follows this trend. However, in optimum DG tunings, there is a clear upward trend after the initial rapid improvement. This indicates that strictly minimizing the MSE may limit generality and explains why domain invariant algorithms such as Fish and IDGM, seem to reach better testing performance despite MSE remaining constant. Minimizing MSE, cross-entropy, and other loss functions has been the target of recent efforts to develop stronger DG algorithms [52]. Proving that MSE minimization may not drive algorithm success and shouldn't be objectified points to a deeper, underlying cause requiring additional research.

Differences in the value functions converged on by Fish and IDGM may explain what enables these algorithms to generalize to new domains significantly better than TD(0). When plotting the value function each algorithm converged to, it is apparent that Fish and IDGM settle on a smaller spread of approximately 0.025. The algorithms still accurately approximated local curvature, while minimizing sharp, "discontinuous" jumps caused by amplified biases in training data. These jumps exhibited by algorithms with larger spreads may incur instability in only marginally different states – detrimental to any DG task. This problem is a prominent challenge with polynomial basis functions and was exacerbated by their use, yet was less noticeable under the traditional DL architectures. Thus, these properties have not been sufficiently reported in recent papers analyzing the DG algorithms [22]. Nguyen et al. refer to distributing the risk variance across domains, but not how lowering the variance of the environment as a whole may also provide better results [53]. This property is another important, yet seldom discussed aspect of designing successful DG algorithms that could contribute to algorithm development.

In terms of pure DG performance, both Fish and IDGM are better in simple RL tasks than traditional approaches to error minimization. Due to additional resources and compute time IDGM requires to calculate the second derivatives, the incremental gains are insufficient to merit practical use. This supports the findings of Yuge Shi, et al. in their 2021 paper, stating "Fish is an effective approximation of IDGM, with similar performance to its direct optimization," where they also note that Fish is approximately 10x faster [22]. Irrespective of DG capabilities, neither Fish nor IDGM could

outperform TD(0) on training data, and performed significantly worse when taking all combinations into account. The declining performance across all trials signifies that these models are more sensitive, and the range under which these training instances are stable is small and hard to find. Such difficulty could be a drawback in applied settings, especially under limited time or when Bayesian optimization techniques are not viable. The same conclusion can be drawn from examining the proportions of converging trials, given that all tested combinations for TD(0) converged, even if they became unstable with reduced exploration. Comparatively, many Fish and IDGM tunings averaged 2 or 3 diverging trials per combination, a definite increase in instability and sensitivity. These findings are specific to polynomial value function approximators, which are known to be more chaotic at domain limits, so these algorithms might prove more stable under different conditions and function approximators. A discrete environment with a relatively small number of states may also have contributed to this behavior. Due to the extreme nature of the divergences, it is reasonable to assume that the level of instability seen in these algorithms will leave some residual sensitivity to conditions not present in TD(0).

V. Conclusion

A. Literature Gap Analysis

This study responds to the research question and addresses the current gap in the literature in various ways. DG in ML has been a long-standing problem, dating back to the 1950-60s. Sutton and Barto cite that "Such age should give one pause," as "Perhaps there is no solution." [8]. However, the data demonstrates that Fish and IDGM are effective methods of generating value function approximations that generalize to outside domains. That result alone is a significant step in the future of high-performing DG in RL tasks indicating that effective methods of DG may not be elusive after all. While this has proven successful with polynomial linear basis functions, limitations associated with the effectiveness of these algorithms at scale mean that effective DG must be applied to DRL to revolutionize future models. A key challenge of DL models is their lack of interpretability [54]. This study expands on the theoretical understanding of the capabilities of effective DG methods such as Fish and IDGM. The polynomial function approximator, discrete state space, and low-dimensional feature vector facilitate plotting the value functions directly to analyze differences in value mappings of the same state space between the near-optimal policies of different algorithms. This highlights differences, such as how Fish and IDGM tended to minimize the spread of the value function, improving stability in DG testing environments. Both algorithms also achieved better performance without direct minimization of the MSE, indicating that future methods should focus on direct policy adjustment. Understanding where these algorithms fail, and why a significant portion of the DG methods were volatile and tended to diverge, can ultimately improve the creation of future algorithms to address these concerns, contributing to current research and algorithmic development.

B. Research Implications

These results have widespread implications extending beyond mere grid world navigation. By studying these algorithms in elementary applications, a significant theoretical insight into why these algorithms are capable of successful DG can be gained. This work and related studies will help drive advances in model complexity, and elucidate the main components of successful DG. In turn, this improves implementation into real-world problems that rely on scalable solutions to capitalize on the exponential benefits scaling model size provides in DL[3] [55]. As briefly mentioned in the introduction, the real-world uses for this technology are abounding. Autonomous control systems depend on adapting to an ever-changing environment it may have never seen before, yet still exhibiting optimal behavior [2]. Fields such as healthcare benefit from improved RL models utilizing meta-learning or domain adaptation to deal with small sample learning and data scarcity [25]. That same ability can be leveraged in many fields depending on operations research and resource allocation, such as scheduling and inventory management, finance, business management, and stock trading when insufficient historical data exists

[3]This is partly due to the Johnson-Lindenstrauss lemma and how quasi-orthogonality scales according to a power law relationship [59]

for predicting highly complex, dynamic trends in unseen market conditions [56]. Control tasks within a virtual environment including simulated systems such as flight simulators, virtual/augmented reality systems, and video games, allow the model to maintain optimal performance when real-world instances of the model do not perfectly match training. This can be advantageous when physical data collection is time-consuming, expensive, or dangerous, and limits the risk of overfitting models in high-risk domains like automated driving, healthcare, and cybersecurity [57]. Closely tied to simulated training is the field of generative AI, where the model's ability to create data is directly tied to its ability to interpret inputs outside of its training domain. Advances in reinforcement learning with human feedback (RLHF) have motivated much of the improvement from OpenAI's GPT-3 to GPT-4 models [58]. As is evident by the plethora of presently relevant systems that would be impacted by the advances this work could provide, the relevance of continuing this research by investigating these and other algorithms at scale cannot be understated. These meta-learning algorithms should be able to improve zero-shot generalization in a diverse range of domains and are desperately needed given the rapid pace the AI industry is evolving to accommodate new, diverse fields and applications.

C. Study Limitations

Despite the promising results, this study has limitations due to limited time and computational resources that need to be addressed. Chief among these were the complexity of the environment and its direct relevance to real-world systems. The idealized grid world assumes a highly structured state space, reliable reward function, lack of noise and complete feature observations, and stationary objectives. Sparsity from not assessing a wider range of hyperparameters in the grid search may have neglected ideal configurations. Furthermore, the small number of training instances in each tuning was insufficient to ensure that the sampling mean reflected the population mean. Neither batch nor discount rate was adjusted to ensure that these values did not affect the convergence of the algorithms significantly. Numerical precision inconsistency from the cloud-based development environment could have been expounded when approximating the Hessian matrix using finite differences, distorting performance. Nevertheless, it is reasonable to assert that any marginal model discrepancies do not affect the theoretical capability of IDGM concluded by this paper, and are certainly not worth the computational inefficiency of this approach. Problems arising from the environment setup may have constrained DG evaluations. There is a potential that the low-dimensional feature representation utilized in this experiment may not scale well to real-world, enigmatic problems, such as robotic control. Generalization was also measured using a fundamental environment structure that may be too easily identifiable compared to full-scale applications. Finally, the environment, reward function, and training/testing sets were all fixed meaning the model only had to optimize a stationary policy instead of being forced to adapt throughout runtime as is common in many applications.

D. Future Research Directions

The wide array of variables that could not sufficiently be investigated, and the potential influence of these algorithms invites several further research inquiries. Foremost, reevaluating the results of this experiment within a larger, continuous environment may give insight into whether instability and divergence are a feature of these algorithms or a consequence of the small, discrete state space. It is possible that the error associated with the imprecise sampling of static points forced the agent to learn from an imperfect approximation of the distribution. A study that examined this navigational task in the context of a continuous state space may be more apt to claim how stable the models are. Future work into optimizing hyperparameter tunings via Bayesian optimization methods, genetic algorithms, or meta-learning could also improve instability caused by bad combinations and reduce reliance on manual training [48]. Alternatively, investigating how these algorithms perform at scales magnitudes larger using DRL may shed light on how these algorithms behave when paired with more nonlinear approximators. This will identify the potential of Fish and IDGM on intractable optimal control problems. Related to real-world implementation, the future of these algorithms under dynamic control tasks to evaluate if these algorithms can adapt well without loss of performance should be studied before assessing

modern applications. This may pair well with Bayesian or gradient-based hyperparameter optimization so that meta-learning parameters can automatically adjust to learn the new environment. In conclusion, the observations made about factors contributing to successful DG should be accounted for to develop new state-of-the-art algorithms for handling more complex environments and resolving current limitations of both Fish and IDGM preventing widespread implementation.

References

[1] G. Cilluffo et al., "Machine learning: An overview and applications in pharmacogenetics," *Genes (Basel)*, vol. 12, no. 10, p. 1511, Sep. 2021. DOI: 10.3390/genes12101511.

[2] Y. Li, "Reinforcement learning applications," *arXiv*, preprint, arXiv:1908.06973, Aug. 2019. Available: https://arxiv.org/abs/1908.06973

[3] D. Whitley et al., "Genetic reinforcement learning for neurocontrol problems," *Mach. Learn.*, vol. 13, pp. 259–284, Nov. 1993. DOI: 10.1023/A:1022674030396

[4] V. Mnih et al., "Playing Atari with deep reinforcement learning," *arXiv*, preprint, arXiv:1312.5602, Dec. 2013. Available: https://arxiv.org/abs/1312.5602

[5] J. D. Martín-Guerrero and L. Lamata, "Reinforcement learning and physics," *Appl. Sci.*, vol. 11, no. 18, p. 8589, Sep. 2021. DOI: 10.3390/app11188589

[6] Y. Yang and A Whinston, "A survey on reinforcement learning for combinatorial optimization," in *2023 IEEE World Conf. Appl. Intell. Comput. (AIC)*, Sonbhadra, India, Jul. 2023, pp. 131-136, DOI: 10.1109/AIC57670.2023. 10263956.

[7] D. Michie, "Experiments on the mechanization of game-learning, Part I. Characterization of the model and its parameters," *The Comput. J.*, vol. 6, no. 3, pp. 232-236, Nov. 1963. DOI: 10.1093/comjnl/6.3.232

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[9] L. P. Kaelbling et al., "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, Vol. 4, No. 1, pp. 237-285, May, 1996.

[10] T. J. Walsh et al., "Transferring state abstractions between MDPs," in *ICML-06 Workshop Struct. Knowl. Transfer Mach. Learn.*, Pittsburgh, PA, USA, Jun. 2006.

[11] S. Narvekar and P. Stone, "Generalizing curricula for reinforcement learning," in *Proc. 37th Int. Conf. Mach. Learn.*, Vienna, Austria, Jul. 2020, pp. 1167-1236.

[12] D. J. White, "Real Applications of Markov Decision Processes," *Interfaces*, vol. 15, no. 6, pp. 73-83, Nov.-Dec. 1985.

[13] C. L. Lan et al., "On the generalization of representations in reinforcement learning," in *Proc. 25th Int. Conf. Artif. Intell. Stat.*, Valencia, Spain, Mar. 2022, pp. 4132-4157.

[14] O. Hernández-Lerma and J. B. Lasserre, "Infinite Horizon Discounted Cost Problems," in *Discrete-Time Markov Control Processes: Basic Optimality Criteria*, 1st ed., New York City, New York, USA: Springer, 1995, ch. 6, pp. 43-73.

[15] V. Darvariu et al., "Graph reinforcement learning for combinatorial optimization: A survey and unifying perspective," *Trans. Mach. Learn. Res.*, Aug. 2024.

[16] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Trans. Autom. Control*, vol. 42, no. 5, pp. 674-690, May, 1997. DOI: 10.1109/9.580874

[17] P. J. Schweitzer and A. Seidmann, "Generalized polynomial approximations in Markovian decision processes," *J. Math. Anal. Appl.*, vol. 110, no. 2, pp. 568-582, Sep. 1985. DOI: 10.1016/0022-247X(85)90317-8

[18] N. Ma et al., "Generalizing to unseen domains for regression," *OpenReview*, Preprint, May, 2023.

[19] E. Korkmaz, "A survey analyzing generalization in deep reinforcement learning," *arXiv*, preprint, arXiv:2401.02349v1, Jan. 2024. Available: https://arxiv.org/html/2401.02349 v1

[20] K. Cobbe et al., "Quantifying generalization in reinforcement learning," in *Proc. 36th Int. Conf. Mach. Learn.*, Long Beach, CA, USA, Jun. 2019, pp. 1282-1289.

[21] J. Wang et al., "Generalizing to unseen domains: A survey on domain generalization," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 8, May, 2022. DOI: 10.1109/TKDE.2022. 3178128

[22] Y. Shi, et al., "Gradient matching for domain generalization," in *10th Int. Conf. Learn. Represent.*, virtual, Apr. 2022.

[23] K. Wang et al., “Improving generalization in reinforcement learning with mixture regularization,” in *Proc. 34th Conf. Neural Inf. Process. Syst.*, Vancouver, Canada, Dec. 2020, pp. 7968–7978.

[24] A. H. Yahmed et al., “Deploying deep reinforcement learning systems: A taxonomy of challenges,” in *39th IEEE Int. Conf. Softw. Maint. Evol.*, Bogotá, Colombia, Oct. 2023, pp. 26-38. DOI: 10.1109/icsme58846.2023.00015

[25] C. Yu et al., “Reinforcement learning in healthcare: A survey,” *ACM Comput. Surv.*, vol. 55, no. 1, pp. 1-36, Nov. 2021. DOI: 10.1145/3477600

[26] J. Li et al., “When data geometry meets deep function: Generalizing offline reinforcement learning,” in *11th Int. Conf. Learn. Represent.*, Kigali, Rwanda, May, 2023.

[27] T. M. Mitchell and S. B. Thrun, “Explanation-based neural network learning for robot control,” in *NIPS'92: Proc. 6th Int. Conf. Neural Inf. Process. Syst.*, Denver, CO, USA, Nov. 1992, pp. 287-294. DOI: 10.5555/2987061.2987097

[28] Z. Huang et al., “Self-challenging improves cross-domain generalization,” in *Proc. 16th Eur. Conf. Comput. Vision (ECCV 2020), Glasgow, U.K., Aug. 23–28, 2020, Part II*, Aug. 2020, pp. 124-140. DOI: 10.1007/978-3-030-58536-5_8

[29] S. Srinivas and F. Fleuret, “Gradient alignment in deep neural networks,” *arXiv*, preprint, arXiv:2006.09128v1, Jun. 2020. Available: https://www.researchgate.net/publication/342229444

[30] G. Konidaris and S. Osentoski, “Value function approximation in reinforcement learning using the fourier basis,” in *Proc. 25th AAAI Conf. Artif. Intell.*, San Francisco, CA, USA, Aug. 2011, pp. 380-385. DOI: 10.1609/aaai.v25i1.7903

[31] C. Jin et al., “Provably efficient reinforcement learning with linear function approximation,” *Proc. Mach. Learn. Res.*, vol. 125, pp. 2137-2143, Jul. 2020.

[32] Y. Saad, *Iterative Methods for Sparse Linear Systems*. 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003.

[33] R. P. Adams, *Features and basis functions*, COS 324 – Elements of Machine Learning, Princeton University, Available: https://www.cs.princeton.edu/courses/archive/fall18/cos324/files/basis-functions.pdf.

[34] K. Asadi et al., “Deep radial-basis value functions for continuous control,” in *Proc. 35th AAAI Conf. Artif. Intell. (AAAI-21)*, virtual, Feb. 2021, pp. 6696-6704.

[35] S. D. Marchi, *Lectures on radial basis functions*, University of Padua. Available: https://www.math.unipd.it/~demarchi/RBF/LectureNotes.pdf

[36] R. Schaback, “Multivariate interpolation by polynomials and radial basis functions,” *Constructive Approximation*, vol. 21, no. 3, pp. 293-317, Apr. 2002. DOI: 10.1007/s00365-004-0585-2

[37] F. Alsharif, “Multilevel quasi-interpolation on Chebyshev sparse grids,” *Computation*, vol. 12, no. 7, p. 149, May, 2024. DOI: 10.3390/computation12070149

[38] H. L. Seal, “Studies in the history of probability and statistics. XV: The historical development of the gauss linear model,” *Biometrika*, vol. 54, no. 1, pp. 1-24, Jun. 1967. DOI: 10.2307/2333849

[39] T. Hestie et al., Basis expansions and regularization,” in *The Elements of Statistical Learning: Data Mining Inference and Prediction*, 2nd ed. New York City, New York, USA: Springer, 2009, ch. 5, pp. 115-163.

[40] R. Bellman et al., “Polynomial approximation - A new computational technique in dynamic programming: Allocation processes,” *Math. Comput.*, vol. 17, no. 82, pp. 155-161, Apr. 1963. DOI: 10.2307/2003635

[41] X. Ying, “An overview of overfitting and its solutions,” *J. Phys.: Conf. Ser.*, vol. 1168, no. 2, Mar. 2019. DOI: 10.1088/1742-6596/1168/2/022022

[42] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep

learning," *J. Big Data*, vol. 6, no. 60, Jul. 2019. DOI: 10.1186/s40537-019-0197-0

[43] A. Clark et al., "Domain generalization for robust model-based offline reinforcement learning," *arXiv*, preprint, arXiv:2211.14827, Nov. 2022. Available: https://arxiv.org/abs/2211.14827

[44] B. Gao et al., "Loss function learning for domain generalization by implicit gradient," in *Proc. 39th Int. Conf. Mach. Learn.*, Baltimore, MA, USA, Jul. 2022, pp. 7002-7016.

[45] W. Mustafa et al., "Input Hessian Regularization of Neural Networks," *arXiv*, preprint, arXiv:2009.06571, Sep. 2020. Available: https://arxiv.org/abs/2009.06571

[46] F. Alet et al., "Functional risk minimization," *arXiv*, preprint, arXiv:2412.21149, Dec. 2024. Available: https://arxiv.org/abs/2412.21149

[47] A. Jain et al., "Generalization to new actions in reinforcement learning," in *Proc. 37th Int. Conf. Mach. Learn.*, virtual, Jul. 2020. pp. 4661-4672. DOI: 10.5555/3524938.3525371

[48] R. Liu et al., "Optimizing the hyper-parameters for SVM by combining evolution strategies with a grid search," in *Intelligent Control and Automation*, 1st ed. D. Huang et al., Eds. Berlin, Germany: Springer Berlin, Heidelberg, 2006, ch. 87, pp. 712-721.

[49] E. Bisong, "Google Colaboratory," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, 1st ed., Berkeley, CA, USA: Apress, pp. 59-64.

[50] M. Biagiola and P. Tonella, "Testing of deep reinforcement learning agents with surrogate models," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 3, pp. 1-33, Mar. 2024. DOI: 10.1145/3631970

[51] N. Nachar, "The Mann‐Whitney U: A test for assessing whether two independent samples come from the same distribution," *Tutorials Quant. Methods Psych.*, vol. 4, no. 1, pp. 13-20, Mar. 2008. DOI: 10.20982/tqmp.04.1.p013

[52] Z. Yang et al., "Rethinking bias-variance trade-off for generalization of neural networks," in *Proc. 37th Int. Conf. Mach. Learn. (ICML)*, virtual, Jul. 2020, pp. 10767-10777. DOI: 10.5555/3524938.3525936

[53] T. Nguyen et al., "Domain generalization via risk distribution matching," in *2024 IEEE/CVF Wint. Conf. Appl. Comput. Vis. (WACV)*, Waikoloa, HI, USA, Jan. 2024, pp. 2778-2787. DOI: 10.1109/WACV57701.2024.00277

[54] Y. Zhang et al., "A survey on neural network interpretability," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 5, no. 5, pp. 726-742, Oct. 2021. DOI: 10.1109/TETCI.2021.3100641

[55] J. Hestness et al., "Deep learning scaling is predictable, empirically," *arXiv*, preprint, arXiv:1712.00409, Dec. 2017. Available: https://arxiv.org/abs/1712.00409

[56] A. L. Awad et al., "Stock market prediction using deep reinforcement learning," *Appl. Syst. Innov.*, vol. 6, no. 6, p. 106, Nov. 2023. DOI: 10.3390/asi6060106

[57] M. Xu et al., "Trustworthy reinforcement learning against intrinsic vulnerabilities: Robustness, safety, and generalizability," *arXiv*, preprint, arXiv:2209.08025, Sep. 2022. Available: https://arxiv.org/abs/2209.08025

[58] P. K. Dalvi and K. Y. Digholkar, "RLHF: Reinforcement learning using human feedback for optimization of ChatGPT," *Grenze Int. J. Eng. Technol.*, vol. 10, no. 2, pt. 4, pp. 3362-3370, Jun. 2024.

[59] P. C. Kainen and V. Kůrková, "Quasiorthogonal dimension," in *Beyond Traditional Probabilistic Data Processing Techniques: Interval, Fuzzy etc. Methods and Their Applications*, 1st ed. O. Kosheleva et al., Eds. Cham, Switzerland: Springer, 2020, pp. 615-629.

Appendix 1

Full code implementation of 3 algorithms: TD(0), Fish, & IDGM in a grid world environment. OpenAI's GPT-4o model was used to inject comments throughout the code for reader clarity, without changing code functionality.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
from mpl_toolkits.mplot3d import Axes3D
from scipy.sparse.linalg import cg
import random, math, itertools
from google.colab import auth
import gspread
from google.auth import default

# Authenticate and set up Google Sheets API
auth.authenticate_user()
creds, _ = default()

# -------------------- Global Parameters --------------------
# Hyperparameter grid: [update_rule, step_size_initial, step_size_final,
outer_step_size, polynomial_degree]
params = [['', 0, 0, 0, 0]]
sheet_names = ['']
drive_name = ''

TRIALS = 5
ROOM_SIZE = 5  # Each room is 5x5 cells
DEGREE = 0    # Polynomial degree for feature approximation
EPISODES = 10000
EVAL_INTERVAL = 100  # Evaluate policy every SAVE_EPISODE episodes (set to 100 as
per methods)
SHOW_INTERVAL = 1000  # Interval for showing plots and printing policy
NUM_TEST = 1
BATCH_SIZE = ROOM_SIZE  # Batch size for state-goal pairs generation
FD_EPSILON = 1e-4
DIVERGENCE_THRESHOLD = 50000
DISCOUNT = 0.9
SIZE = ROOM_SIZE * 2 + 1  # Gridworld size

np.set_printoptions(threshold=np.inf)

# -------------------- Main RL Loop --------------------
def main(update_rule, lr_init, lr_final, outer_lr):
   epsilon = 1.0
   step_size = lr_init
   # Initialize weight tensor for polynomial basis functions (6 dimensions:
pos(x,y), goal(x,y), distance(x,y))
```

```
    weights = np.full((DEGREE + 1,) * 6, 0.0)
    train_avg, test_avg, train_mse, test_mse = [], [], [], []
    diverged = False

    # Loop over episodes
    for episode in range(EPISODES):
        # Generate a batch of state-goal pairs from the gridworld environment
        state_goal_pairs = generate_state_goal_pairs(update_rule)
        td_adapted = weights.copy()
        gradients = [[], [], [], []]
        # Decay the outer learning rate over episodes
        current_outer_lr = outer_lr / (1 + EPISODES / 100)
        saved_features = ([], [])

        for idx, (state, goal) in enumerate(state_goal_pairs):
            dist = np.subtract(goal, state)
            # Select an action using epsilon-greedy policy
            angle = choose_action(state, goal, epsilon, weights)
            action = (round(math.cos(angle)), round(math.sin(angle)))
            new_state = move(state, action)
            new_dist = np.subtract(goal, new_state)

            # Calculate current and next features (polynomial basis)
            feat = compute_features(state, goal, dist)
            next_feat = compute_features(new_state, goal, new_dist)
            saved_features[0].append(feat)
            saved_features[1].append(next_feat)

            reward = get_reward(state, new_state, goal)
            old_val = np.sum(weights * feat)
            new_val = np.sum(weights * next_feat)

            # Inner update: update weights (or store gradient info) based on the
update rule
            weights, td_adapted, gradients = inner_update(
                step_size, reward, new_val, old_val, feat, update_rule,
                td_adapted, weights, episode, idx, gradients
            )

        # Outer update (for Fish and IDGM)
        weights = outer_update(weights, current_outer_lr, td_adapted, update_rule,
episode, state_goal_pairs, gradients, saved_features)

        # Policy evaluation and divergence check every EVAL_INTERVAL episodes
        if (episode + 1) % EVAL_INTERVAL == 0:
            train_avg, train_mse = evaluate_policy(epsilon, weights, train_avg,
train_mse, episode, test=False)
```

```
            test_avg, test_mse = evaluate_policy(epsilon, weights, test_avg,
test_mse, episode, test=True)
            if max(train_avg) > DIVERGENCE_THRESHOLD or max(test_avg) >
DIVERGENCE_THRESHOLD:
                diverged = True
                break

        # Decay exploration and learning rate gradually
        epsilon -= epsilon / ((EPISODES // 2) - 1)
        step_size = lr_init / (1 + ((lr_init - lr_final) / (lr_final * EPISODES)) *
episode)

    # Final evaluation after training (or if diverged)
    if not diverged:
        train_avg, train_mse = evaluate_policy(epsilon, weights, train_avg,
train_mse, EPISODES, test=False)
        test_avg, test_mse = evaluate_policy(epsilon, weights, test_avg, test_mse,
EPISODES, test=True)
    else:
        # If diverged, pad results to expected length
        num_evals = EPISODES // EVAL_INTERVAL + 1
        train_avg.extend([train_avg[-1]] * (num_evals - len(train_avg)))
        test_avg.extend([test_avg[-1]] * (num_evals - len(test_avg)))
        train_mse.extend([train_mse[-1]] * (num_evals - len(train_mse)))
        test_mse.extend([test_mse[-1]] * (num_evals - len(test_mse)))

    # Determine starting evaluation index based on last portion to display
    start_idx = int(min(max(np.floor(len(train_avg) * (1 - 0.2)) - 1, 0),
len(train_avg)))
    return train_avg[start_idx:], test_avg[start_idx:], train_mse[start_idx:],
test_mse[start_idx:], start_idx, update_rule, lr_init, outer_lr

# -------------------- Evaluation Function --------------------
def evaluate_policy(epsilon, weights, avg_store, mse_store, episode, test=True):
    total_steps = 0
    for _ in range(NUM_TEST):
        if test:
            state = (np.random.randint(ROOM_SIZE + 1, SIZE), np.random.randint(0,
ROOM_SIZE))
            goal = (np.random.randint(0, ROOM_SIZE), np.random.randint(ROOM_SIZE +
1, SIZE))
        else:
            state = (np.random.randint(0, ROOM_SIZE), np.random.randint(0,
ROOM_SIZE))
            goal = (np.random.randint(ROOM_SIZE + 1, SIZE),
np.random.randint(ROOM_SIZE + 1, SIZE))
        steps = 0
```

```
        dist = np.subtract(goal, state)
        while dist[0] != 0 and dist[1] != 0:
            angle = choose_action(state, goal, epsilon, weights)
            action = (round(math.cos(angle)), round(math.sin(angle)))
            state = move(state, action)
            dist = np.subtract(goal, state)
            steps += 1
        total_steps += steps
    avg = total_steps / NUM_TEST
    avg_store.append(avg)
    mse = compute_mse(test, weights)
    mse_store.append(mse)
    if (episode + 1) % SHOW_INTERVAL == 0:
        print(("Test" if test else "Train") + f" Episode: {episode + 1}\nAverage
Steps: {np.round(avg, 4)}\nGoal: {goal}\nMSE: {mse}\n")
        plot_value_function(weights, goal, avg_store, mse_store, episode, state,
test)
        print_policy(weights, goal, state)
    return avg_store, mse_store

# -------------------- Helper Functions --------------------
def generate_state_goal_pairs(update_rule):
    # Generate BATCH_SIZE random integers for positions and goals for each of 4
categories
    x_coords = [[np.random.randint(low, high) for _ in range(BATCH_SIZE)]
                for low, high in [(0, ROOM_SIZE), (ROOM_SIZE + 1, SIZE), (0,
ROOM_SIZE), (ROOM_SIZE + 1, SIZE)]]
    y_coords = [[np.random.randint(low, high) for _ in range(BATCH_SIZE)]
                for low, high in [(0, ROOM_SIZE), (0, ROOM_SIZE), (ROOM_SIZE + 1,
SIZE), (ROOM_SIZE + 1, SIZE)]]
    goal_x = [[np.random.randint(ROOM_SIZE + 1, SIZE) for _ in range(BATCH_SIZE)]
for _ in range(4)]
    goal_y = [[np.random.randint(ROOM_SIZE + 1, SIZE) for _ in range(BATCH_SIZE)]
for _ in range(4)]
    # Flatten coordinate lists to produce state and goal tuples
    states = [(x, y) for xs, ys in zip(x_coords, y_coords) for x, y in zip(xs, ys)]
    goals = [(gx, gy) for gxs, gys in zip(goal_x, goal_y) for gx, gy in zip(gxs,
gys)]
    return list(zip(states, goals))

def inner_update(step_size, reward, new_val, old_val, features, update_rule,
td_adapted, weights, episode, idx, grad_list):
    td_error = step_size * (reward + DISCOUNT * new_val - old_val) * features
    if update_rule == 'td(0)' or episode <= 0:
        weights += td_error
    elif update_rule == 'Fish':
        td_adapted -= td_error
```

```
    elif update_rule == 'IDGM':
        grad_list[idx // BATCH_SIZE].append(td_error * features)
    return weights, td_adapted, grad_list

def outer_update(weights, outer_lr, td_adapted, update_rule, episode,
state_goal_pairs, gradients, saved_features):
    if update_rule == 'Fish' and episode > 0:
        weights += outer_lr * (td_adapted - weights)
    elif update_rule == 'IDGM' and episode > 0:
        all_grads = np.array([g for sublist in gradients for g in sublist])
        if all_grads.size > 0:
            avg_grad = np.mean(all_grads, axis=0)
            weights -= outer_lr * avg_grad
    return weights

def compute_features(state, goal, dist):
    # Create a polynomial basis of order (DEGREE) for 6 features normalized by grid
size
    grids = np.indices((DEGREE + 1,) * 6)
    exponents = [grid.flatten() for grid in grids]
    norm_factors = [state[0]/SIZE, state[1]/SIZE, goal[0]/SIZE, goal[1]/SIZE,
dist[0]/SIZE, dist[1]/SIZE]
    feature_vals = np.ones_like(exponents[0], dtype=float)
    for exp, factor in zip(exponents, norm_factors):
        feature_vals *= factor ** exp
    return feature_vals.reshape((DEGREE + 1,) * 6)

def choose_action(state, goal, epsilon, weights):
    # With probability epsilon choose a random action; otherwise, select action with
highest estimated value
    if np.random.random() < epsilon:
        return np.random.choice([0, np.pi/2, np.pi, 3*np.pi/2])
    else:
        return (np.pi/2) * argmax_action(state, goal, weights)

def argmax_action(state, goal, weights):
    values = []
    for i in range(4):
        new_state = move(state, (round(math.cos(np.pi/2 * i)),
round(math.sin(np.pi/2 * i))))
        if new_state != state:
            new_dist = np.subtract(goal, new_state)
            value = np.sum(compute_features(new_state, goal, new_dist) * weights)
        else:
            value = -np.inf
        values.append(value)
    return np.argmax(values)
```

```
def move(state, action):
    new_state = [min(SIZE - 1, max(0, state[i] + action[i])) for i in range(2)]
    # Prevent moving through walls (if on corridor boundaries)
    for i in range(2):
        if new_state[0] == ROOM_SIZE and new_state[1] not in {ROOM_SIZE // 2, 2 *
ROOM_SIZE - ROOM_SIZE // 2}:
            new_state[i] = state[i]
    return tuple(new_state)

def get_reward(state, new_state, goal):
    # Reward +1 for reaching the goal; small negative reward if no move is made
    if new_state == goal:
        return 1
    elif state == new_state:
        return -1 / ROOM_SIZE
    else:
        return 0

def compute_mse(test, weights):
    # Compute mean squared error between the estimated value and an optimal value
over a sampled set of states
    pos_x, pos_y = np.indices((SIZE - 1, SIZE - 1))
    pos_x = [[x if x < ROOM_SIZE else x + 1 for x in row] for row in pos_x]
    pos_y = [[y if y < ROOM_SIZE else y + 1 for y in row] for row in pos_y]
    pos_list = list(np.array(pos_x).flatten()) + [ROOM_SIZE, SIZE - (ROOM_SIZE - 1)
/ 2 - 1, ROOM_SIZE, (ROOM_SIZE - 1) / 2]
    pos_list_y = list(np.array(pos_y).flatten()) + [(ROOM_SIZE - 1) / 2, ROOM_SIZE,
SIZE - (ROOM_SIZE - 1) / 2 - 1, ROOM_SIZE]
    # Sample a set of states and compute MSE
    all_states = []
    for gx in range(ROOM_SIZE):
        for gy in range(ROOM_SIZE):
            all_states.append((pos_list[gx], pos_list_y[gy], gx, gy, gx -
pos_list[gx], gy - pos_list_y[gy]))
    sample_states = random.sample(all_states, int(len(all_states) / SIZE))
    mse = np.mean([(np.sum(compute_features((s[0], s[1]), (s[2], s[3]), (s[4],
s[5])) * weights) - optimal_value((s[0], s[1]), (s[2], s[3]), test)) ** 2 for s in
sample_states])
    return mse

def optimal_value(state, goal, first=True, test=False):
    # Recursive estimation of optimal value (using reward and discount) for a given
state; stops when state equals goal.
    if state == goal:
        return get_reward(state, state, goal)
    if not first:
```

```
        return get_reward((-1, -1), state, goal) + DISCOUNT *
optimal_value((state[0] - 1, state[1]), goal, first=False, test=test)
    d = 0
    if test:
        state = (SIZE - state[0] - 1, state[1])
        goal = (SIZE - goal[0] - 1, goal[1])
    if state[0] < ROOM_SIZE and state[1] < ROOM_SIZE:
        i = np.argmin([
            np.sum(np.abs(np.subtract(state, (ROOM_SIZE - 1) / 2, ROOM_SIZE - 1))) +
            np.sum(np.abs(np.subtract(goal, (SIZE - ROOM_SIZE, SIZE - 1 - (ROOM_SIZE
- 1) / 2)))),
            np.sum(np.abs(np.subtract(state, (ROOM_SIZE - 1, (ROOM_SIZE - 1) / 2))))
+
            np.sum(np.abs(np.subtract(goal, (SIZE - 1 - (ROOM_SIZE - 1) / 2, SIZE -
ROOM_SIZE))))
        ])
        d = min_distance(state, goal, index=i) + ROOM_SIZE + 3
    elif state[0] > ROOM_SIZE and state[1] > ROOM_SIZE:
        d = np.sum(np.abs(np.subtract(state, goal)))
    else:
        d = min_distance(state, (SIZE - 1 - (ROOM_SIZE - 1) / 2, SIZE - ROOM_SIZE))
+ \
            min_distance(goal, (SIZE - 1 - (ROOM_SIZE - 1) / 2, SIZE - ROOM_SIZE))
    return get_reward((d - 1, 0), (d, 0), (0, 0)) + DISCOUNT * optimal_value((d - 1,
0), (0, 0), first=False, test=test)

def min_distance(state, goal, index=-1):
    if index == -1:
        return min(np.sum(np.abs(np.subtract(state, goal))),
                   np.sum(np.abs(np.subtract(state, (goal[1], goal[0])))))
    options = [
        np.sum(np.abs(np.subtract(state, ((ROOM_SIZE - 1) / 2, ROOM_SIZE - 1)))) +
        np.sum(np.abs(np.subtract(goal, (SIZE - ROOM_SIZE, SIZE - 1 - (ROOM_SIZE -
1) / 2)))),
        np.sum(np.abs(np.subtract(state, (ROOM_SIZE - 1, (ROOM_SIZE - 1) / 2)))) +
        np.sum(np.abs(np.subtract(goal, (SIZE - 1 - (ROOM_SIZE - 1) / 2, SIZE -
ROOM_SIZE))))
    ]
    return options[index]

def plot_value_function(weights, goal, avg_list, mse_list, episode, start_state,
test):
    # Generate a meshgrid for plotting the value function surface and error surface
    x, y = np.meshgrid(np.arange(SIZE), np.arange(SIZE))
    z_val = np.array([[np.sum(compute_features((i, j), goal, np.subtract(goal, (i,
j))) * weights) for j in range(SIZE)] for i in range(SIZE)])
    z_err = np.array([[(np.sum(compute_features((i, j), goal, np.subtract(goal, (i,
```

```
j))) * weights) - optimal_value((i, j), goal, test=test)) ** 2 for j in
range(SIZE)] for i in range(SIZE)])

    fig = plt.figure(figsize=(12, 8))
    gs = GridSpec(2, 2, width_ratios=[2, 1], height_ratios=[1, 1])
    ax1 = fig.add_subplot(gs[0, 0], projection="3d")
    ax1.plot_surface(x, y, z_val)
    ax1.set(title='Value Function', xlabel='X', ylabel='Y')
    ax1.view_init(elev=30, azim=-60)

    ax2 = fig.add_subplot(gs[0, 1])
    ax2.plot(np.arange(len(avg_list)) * EVAL_INTERVAL, avg_list, color='red')
    ax2.set(title='Policy Evaluation', ylim=[0, 1.2 * max(avg_list)])

    ax3 = fig.add_subplot(gs[1, 0], projection='3d')
    ax3.plot_surface(x, y, z_err)
    ax3.set(title='Error Surface', xlabel='X', ylabel='Y')
    ax3.view_init(elev=30, azim=-60)

    ax4 = fig.add_subplot(gs[1, 1])
    ax4.plot(np.arange(len(mse_list)) * EVAL_INTERVAL, mse_list, color='red')
    ax4.set(title='Mean Squared Error', ylim=[0, 1.2 * max(mse_list)])

    plt.subplots_adjust(wspace=0.25)
    plt.show()

def print_policy(weights, goal, start_state):
    # Print a grid representation of the policy using symbols for actions
    action_symbols = {0: ">", 1: "V", 2: "<", 3: "^"}
    for y in range(SIZE):
        row = ""
        for x in range(SIZE):
            if (x, y) == goal:
                row += "O "
            elif (x, y) == start_state:
                row += "X "
            elif (x == ROOM_SIZE and y not in {ROOM_SIZE // 2, 2 * ROOM_SIZE -
ROOM_SIZE // 2}) or \
                 (y == ROOM_SIZE and x not in {ROOM_SIZE // 2, 2 * ROOM_SIZE -
ROOM_SIZE // 2}):
                row += "  "
            else:
                row += action_symbols[argmax_action((x, y), goal, weights)] + " "
        print(row)
    print("\n" + "_" * 100)

def col_letter(n):
```

```
    # Convert a 1-indexed column number to an Excel-style column letter
    result = ''
    while n:
        n, rem = divmod(n - 1, 26)
        result = chr(65 + rem) + result
    return result

# -------------------- Data Export and Plotting --------------------
for p_idx, param in enumerate(params):
    train_avgs, test_avgs, train_mses, test_mses = [], [], [], []
    start_idx, update_rule = 0, ''
    DEGREE = param[4]
    for trial in range(TRIALS):
        t_avg, t_test, mse_train, mse_test, start_idx, update_rule, lr0, outer_lr =
main(param[0], param[1], param[2], param[3])
        train_avgs.append(t_avg)
        test_avgs.append(t_test)
        train_mses.append(mse_train)
        test_mses.append(mse_test)

    train_avgs = np.round(np.array(train_avgs), 4)
    test_avgs = np.round(np.array(test_avgs), 4)
    train_mses = np.round(np.array(train_mses), 4)
    test_mses = np.round(np.array(test_mses), 4)

    # Export results to Google Sheets
    gc = gspread.authorize(creds)
    worksheet = gc.open(drive_name).worksheet(sheet_names[p_idx])
    worksheet.update('A1:A2', [[param[0]], ['Episode']])
    episode_cells = worksheet.range(f"A3:A{len(train_avgs[0]) + 2}")
    for i, cell in enumerate(episode_cells):
        cell.value = EVAL_INTERVAL * i
    worksheet.update_cells(episode_cells)

    for trial in range(TRIALS):
        start_col = col_letter(trial * 5 + 3)
        end_col = col_letter(trial * 5 + 6)
        header_range = f"{start_col}1:{end_col}2"
        meta = [f"Trial {trial + 1}", f"LR = {lr0}", f"Outer = {outer_lr}", f"Degree
= {DEGREE}", "Train Avg", "Test Avg", "Train MSE", "Test MSE"]
        header_cells = worksheet.range(header_range)
        for i, cell in enumerate(header_cells):
            cell.value = meta[i]
        worksheet.update_cells(header_cells)
        data_range = f"{start_col}3:{end_col}{len(train_avgs[trial]) + 2}"
        data_cells = worksheet.range(data_range)
        data_pool = [train_avgs[trial], test_avgs[trial], train_mses[trial],
```

```
test_mses[trial]]
        for i, cell in enumerate(data_cells):
            cell.value = data_pool[i % 4][i // 4]
        worksheet.update_cells(data_cells)

    # Plot overall performance
    fig, axes = plt.subplots(1, 4, figsize=(20, 4))
    for avgs, ax, title in zip([train_avgs, test_avgs, train_mses, test_mses], axes,
['Train Policy Eval', 'Test Policy Eval', 'Train MSE', 'Test MSE']):
        for trial_data in avgs:
            ax.plot(EVAL_INTERVAL * (np.arange(len(trial_data)) + start_idx),
trial_data, color='black')
        ax.set_title(title)
        ax.set_ylim([0, 1.2 * np.max(avgs)])
    plt.subplots_adjust(wspace=0.25)
    plt.show()
```

Appendix 2

Tables containing TD(0) metrics for optimal DG trial: $\alpha = 0.1$, degree = 4

Average Episode Length on Training Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	26	170	140	12	112
100	21	100	85	26	39
200	36	139	878	24	178
300	32	77	51	37	23
400	20	115	110	43	9
500	150	80	83	136	51
600	65	133	7	31	22
700	49	89	19	67	20
800	48	177	56	67	155
900	52	37	34	34	72
1000	31	30	26	47	64
1100	39	35	18	77	92
1200	37	23	45	31	26
1300	69	57	76	9	58
1400	25	48	10	25	23
1500	21	15	25	31	47
1600	21	42	56	29	49
1700	53	37	23	34	82
1800	24	260	9	50	26
1900	52	23	23	22	17
2000	63	28	40	12	37
2100	36	53	31	10	24
2200	10	36	17	21	32
2300	9	27	12	36	10
2400	8	16	25	38	44
2500	50	18	52	28	19
2600	87	17	25	17	31
2700	28	27	36	39	18
2800	14	32	28	37	76
2900	15	8	31	19	20
3000	14	19	25	15	27
3100	27	16	11	11	26
3200	28	29	30	35	24
3300	39	23	34	20	21
3400	39	15	14	14	18
3500	27	37	15	17	20
3600	32	39	12	15	47
3700	10	24	19	25	25
3800	19	16	20	12	21
3900	20	18	18	15	23
4000	8	28	24	13	21
4100	25	14	36	21	32
4200	28	31	33	17	29
4300	18	13	15	30	12
4400	22	12	17	30	14
4500	34	28	20	32	23
4600	16	17	23	23	16

4700	18	11	19	32	23
4800	19	25	15	19	17
4900	16	17	15	25	20
5000	28	13	14	24	23
5100	22	23	13	26	25
5200	19	15	17	13	21
5300	37	15	13	31	10
5400	23	22	18	12	12
5500	14	35	16	18	13
5600	11	20	31	18	19
5700	12	19	16	26	36
5800	34	18	15	19	32
5900	30	22	16	15	19
6000	12	19	26	20	19
6100	22	21	15	21	17
6200	17	24	21	15	20
6300	16	16	21	14	26
6400	10	25	18	22	13
6500	15	21	17	16	33
6600	17	19	24	13	11
6700	15	26	19	12	16
6800	20	13	26	10	14
6900	23	29	12	22	15
7000	18	31	28	22	21
7100	13	26	31	14	17
7200	11	18	11	19	17
7300	16	14	16	15	15
7400	16	16	23	21	22
7500	19	11	19	21	24
7600	19	21	8	11	12
7700	16	10	14	16	15
7800	27	12	23	20	19
7900	30	19	17	23	15
8000	9	23	24	16	25
8100	18	17	14	10	14
8200	11	13	15	16	16
8300	21	18	22	18	18
8400	14	29	14	28	28
8500	14	12	29	21	21
8600	10	26	19	13	18
8700	20	23	21	25	20
8800	16	15	14	24	19
8900	17	17	12	12	18
9000	13	16	24	11	12
9100	11	20	9	10	11
9200	19	15	13	20	15
9300	16	12	11	16	22
9400	15	13	21	26	15
9500	13	20	13	16	20
9600	16	13	15	12	11
9700	11	22	23	16	15
9800	16	20	9	24	20
9900	17	24	21	11	13
10000	20	15	13	21	16

Average Episode Length on Testing Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	233	60	99	115	119
100	9	39	139	158	159
200	18	16	25	45	45
300	213	151	153	26	163
400	26	25	31	16	19
500	47	51	31	106	28
600	17	44	41	41	84
700	87	26	41	47	27
800	23	84	128	9	18
900	15	122	101	34	35
1000	60	16	26	25	19
1100	52	10	27	136	35
1200	40	49	69	43	34
1300	32	23	42	27	14
1400	18	57	19	13	86
1500	28	26	41	76	54
1600	51	28	18	29	31
1700	27	14	17	37	31
1800	17	51	46	32	14
1900	19	44	52	40	27
2000	11	56	25	27	36
2100	58	34	26	61	9
2200	77	18	8	9	12
2300	22	40	27	44	27
2400	27	23	13	16	16
2500	19	33	26	18	39
2600	27	10	25	24	22
2700	26	13	17	19	13
2800	35	45	37	27	31
2900	48	47	35	11	22
3000	23	20	31	24	27
3100	45	38	28	25	18
3200	26	13	13	20	35
3300	23	15	20	12	44
3400	29	21	54	19	27
3500	20	19	13	13	20
3600	18	23	41	26	16
3700	14	26	56	23	23
3800	14	25	19	26	19
3900	27	15	25	12	9
4000	20	9	23	13	19
4100	16	23	18	11	25
4200	20	23	31	28	18
4300	16	13	8	27	23
4400	17	20	23	14	17
4500	19	18	23	22	19
4600	13	13	21	12	24
4700	21	11	25	21	10
4800	18	20	21	24	28
4900	19	43	17	20	8
5000	23	11	16	28	32
5100	14	18	12	11	26

5200	32	22	11	17	21
5300	28	15	17	17	20
5400	20	16	16	20	14
5500	11	17	16	18	28
5600	17	21	15	12	20
5700	21	18	13	18	38
5800	19	16	10	17	7
5900	18	12	18	19	15
6000	15	21	19	25	17
6100	22	16	22	13	14
6200	21	16	24	20	13
6300	17	19	18	12	24
6400	12	35	14	12	17
6500	9	15	16	11	23
6600	16	18	13	18	12
6700	21	11	20	12	13
6800	14	11	14	14	17
6900	14	12	16	13	14
7000	12	13	21	18	15
7100	19	20	20	21	14
7200	16	12	11	12	19
7300	12	9	16	20	18
7400	16	11	24	10	9
7500	24	10	13	22	13
7600	9	15	13	22	22
7700	8	15	12	13	17
7800	23	14	16	20	11
7900	14	12	20	13	15
8000	12	15	13	8	12
8100	18	13	21	8	14
8200	14	17	13	23	11
8300	22	14	14	17	15
8400	14	14	19	15	15
8500	16	22	15	26	19
8600	12	13	14	18	19
8700	17	14	19	11	23
8800	15	12	17	17	24
8900	17	11	17	15	11
9000	15	17	14	16	10
9100	14	15	13	17	21
9200	17	12	15	22	17
9300	13	16	14	20	19
9400	20	16	14	12	11
9500	10	17	17	15	15
9600	17	14	13	9	12
9700	18	22	15	19	17
9800	12	14	14	18	16
9900	15	19	14	18	13
10000	9	23	8	20	11

Mean Square Error on Training Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	0.2052	0.1731	1.068	0.7073	0.1086
100	0.193	0.2181	0.5343	0.3995	0.8605
200	0.5044	0.2851	0.1367	0.2309	0.5043
300	0.6582	0.4703	0.6136	0.6526	0.3301
400	0.2413	0.4742	0.1271	0.2318	0.3247
500	0.3073	0.4887	0.4824	0.186	0.461
600	0.2197	0.3303	0.2513	0.4834	0.2455
700	0.344	0.3374	0.335	0.36	0.2608
800	0.1308	0.38	0.1762	0.3584	0.1367
900	0.1854	0.1929	0.2151	0.2183	0.5036
1000	0.1745	0.2014	0.185	0.1511	0.1268
1100	0.1503	0.2519	0.3093	0.1789	0.2794
1200	0.2681	0.1957	0.0704	0.1235	0.2925
1300	0.17	0.211	0.1645	0.096	0.2243
1400	0.26	0.263	0.2208	0.1343	0.208
1500	0.2782	0.183	0.1589	0.1645	0.1993
1600	0.1361	0.1953	0.1428	0.2506	0.1526
1700	0.227	0.2135	0.0802	0.1271	0.2158
1800	0.0978	0.2158	0.1129	0.1283	0.1783
1900	0.1161	0.099	0.1604	0.1583	0.105
2000	0.0871	0.1941	0.1174	0.1963	0.1771
2100	0.0711	0.0973	0.0961	0.0913	0.1138
2200	0.0935	0.0793	0.0931	0.074	0.1794
2300	0.11	0.0929	0.1459	0.0926	0.1168
2400	0.1178	0.0686	0.1415	0.0996	0.1273
2500	0.0497	0.084	0.1204	0.0756	0.0682
2600	0.0357	0.0502	0.1158	0.0762	0.0994
2700	0.0774	0.083	0.1986	0.088	0.0715
2800	0.0767	0.1284	0.1451	0.0686	0.081
2900	0.0653	0.0566	0.1291	0.0796	0.0605
3000	0.0664	0.0692	0.0962	0.0831	0.0585
3100	0.1005	0.0535	0.1934	0.0786	0.0793
3200	0.0617	0.1382	0.1347	0.0567	0.0782
3300	0.0647	0.0758	0.1001	0.0759	0.0511
3400	0.0696	0.067	0.0971	0.0961	0.0782
3500	0.0553	0.0788	0.07	0.0868	0.0612
3600	0.0763	0.0506	0.0882	0.054	0.0685
3700	0.095	0.0963	0.0868	0.1378	0.074
3800	0.0819	0.0692	0.0558	0.1247	0.0705
3900	0.0791	0.0491	0.0661	0.1693	0.1171
4000	0.1315	0.1044	0.0535	0.2619	0.136
4100	0.0828	0.0823	0.0729	0.2051	0.0985
4200	0.1317	0.0411	0.1583	0.1612	0.0736
4300	0.1352	0.1137	0.101	0.1782	0.1557
4400	0.1075	0.0975	0.1324	0.137	0.0882
4500	0.1667	0.1245	0.0728	0.2246	0.1156
4600	0.2178	0.0707	0.084	0.1977	0.0691
4700	0.2013	0.1815	0.1733	0.1418	0.1661
4800	0.0918	0.0607	0.2747	0.126	0.1342
4900	0.1676	0.0663	0.2318	0.2188	0.1446
5000	0.3432	0.0973	0.1496	0.1573	0.0792
5100	0.1674	0.1314	0.1313	0.243	0.1773

5200	0.0954	0.1088	0.2018	0.1478	0.1284
5300	0.2237	0.0903	0.2993	0.4216	0.2456
5400	0.1692	0.294	0.1859	0.1529	0.1705
5500	0.2578	0.2463	0.2232	0.2403	0.1177
5600	0.2334	0.2043	0.144	0.1413	0.2464
5700	0.296	0.2149	0.3206	0.2525	0.1981
5800	0.2428	0.2731	0.327	0.1657	0.1392
5900	0.2306	0.3612	0.1377	0.2482	0.2065
6000	0.2981	0.2663	0.4841	0.1683	0.2173
6100	0.1539	0.3025	0.3673	0.2547	0.2728
6200	0.4884	0.1641	0.3161	0.1562	0.4994
6300	0.2125	0.3144	0.4671	0.1667	0.3655
6400	0.1255	0.2433	0.5488	0.1607	0.1755
6500	0.2063	0.3241	0.457	0.2179	0.2109
6600	0.257	0.4374	0.3649	0.3832	0.4141
6700	0.3182	0.4191	0.3511	0.1903	0.346
6800	0.3065	0.2413	0.3481	0.261	0.2096
6900	0.2914	0.2604	0.2511	0.395	0.4203
7000	0.4491	0.4121	0.436	0.4337	0.5006
7100	0.5691	0.2225	0.6815	0.1584	0.2684
7200	0.3597	0.3367	0.5348	0.135	0.2345
7300	0.3184	0.4067	0.3233	0.2696	0.3445
7400	0.3128	0.2242	0.5719	0.2418	0.4259
7500	0.2984	0.3265	0.3263	0.4026	0.3027
7600	0.6047	0.423	0.2527	0.2449	0.313
7700	0.3169	0.2368	0.3625	0.3115	0.4956
7800	0.4357	0.3581	0.4088	0.4355	0.5202
7900	0.6679	0.3288	0.3528	0.453	0.3745
8000	0.7459	0.5234	0.2672	0.3612	0.3869
8100	0.4595	0.4651	0.4434	0.3762	0.4694
8200	0.3131	0.5313	0.4426	0.6465	0.3654
8300	0.7558	0.4049	0.4239	0.362	0.4111
8400	0.5223	0.416	0.5741	0.2983	0.5236
8500	0.5831	0.5982	0.4248	0.3582	0.4787
8600	0.6424	0.5676	0.5346	0.3294	0.4257
8700	0.6222	0.4079	0.539	0.4316	0.4763
8800	0.4744	0.4866	0.5773	0.5147	0.7522
8900	0.6539	0.5493	0.6438	0.452	0.7024
9000	0.5005	0.5007	0.6262	0.595	0.8241
9100	0.6032	0.5144	0.6441	0.4754	0.581
9200	0.4906	0.6021	0.7555	0.6279	0.7365
9300	0.6014	0.4985	0.8926	0.8074	0.5451
9400	0.5491	0.4069	0.8172	0.9006	0.5567
9500	0.5524	0.4844	0.7637	0.6744	0.4906
9600	0.525	0.6046	0.637	0.625	1.085
9700	0.6069	0.4721	0.6757	0.6478	0.8444
9800	0.551	0.466	0.483	0.6332	0.8798
9900	0.7561	0.718	0.7053	0.5578	0.6729
10000	0.9345	0.6668	0.6942	0.656	0.8301

Mean Square Error on Testing Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	0.3254	0.3482	0.7036	0.7879	0.2525
100	0.4374	0.1924	0.4661	0.6571	0.7988
200	0.3379	0.2178	0.2583	0.2816	0.4318
300	0.7376	0.3987	0.6269	0.6659	0.3218
400	0.5167	0.3067	0.1227	0.2507	0.3439
500	0.3522	0.3817	0.2337	0.4263	0.4743
600	0.2325	0.4051	0.2084	0.5442	0.3673
700	0.383	0.4673	0.3231	0.423	0.4309
800	0.1604	0.4717	0.2402	0.3214	0.3508
900	0.2675	0.2627	0.3039	0.3093	0.617
1000	0.342	0.2481	0.283	0.2984	0.3074
1100	0.4431	0.4212	0.4011	0.2828	0.6427
1200	0.4663	0.4215	0.1326	0.2002	0.5218
1300	0.265	0.3671	0.2634	0.213	0.481
1400	0.3533	0.364	0.2752	0.2436	0.6285
1500	0.2885	0.469	0.2111	0.3763	0.2896
1600	0.1934	0.3089	0.2591	0.451	0.399
1700	0.2777	0.348	0.1684	0.273	0.755
1800	0.1849	0.2773	0.1987	0.2636	0.4422
1900	0.2308	0.2298	0.2826	0.2753	0.2415
2000	0.186	0.2771	0.203	0.3767	0.3943
2100	0.1639	0.1611	0.1921	0.2277	0.2689
2200	0.1705	0.2253	0.1831	0.2507	0.3781
2300	0.1821	0.175	0.2489	0.2709	0.2627
2400	0.1968	0.212	0.3073	0.2668	0.3068
2500	0.1023	0.1363	0.2124	0.2289	0.2348
2600	0.1207	0.1152	0.2034	0.2466	0.3557
2700	0.0972	0.1577	0.3089	0.274	0.2688
2800	0.1449	0.2495	0.2148	0.3282	0.2845
2900	0.166	0.1615	0.1956	0.2892	0.3397
3000	0.1212	0.1946	0.2659	0.21	0.2085
3100	0.1664	0.1916	0.1951	0.3095	0.3173
3200	0.1811	0.1209	0.1648	0.2035	0.2916
3300	0.1417	0.1516	0.1342	0.1872	0.2393
3400	0.131	0.1603	0.1735	0.2123	0.1404
3500	0.1286	0.227	0.2107	0.291	0.2572
3600	0.1231	0.172	0.2877	0.1633	0.2329
3700	0.1149	0.1471	0.256	0.1551	0.1927
3800	0.1322	0.1326	0.1977	0.154	0.2047
3900	0.0889	0.1365	0.1747	0.1806	0.1953
4000	0.0889	0.0879	0.1779	0.1594	0.1502
4100	0.1497	0.1128	0.1737	0.2032	0.1731
4200	0.1217	0.1614	0.1071	0.2279	0.1566
4300	0.088	0.095	0.125	0.1819	0.1665
4400	0.1044	0.1143	0.1477	0.1984	0.2137
4500	0.077	0.0883	0.2667	0.1288	0.1544
4600	0.0842	0.1355	0.2038	0.1524	0.1907
4700	0.0826	0.0796	0.115	0.2032	0.1264
4800	0.1107	0.1328	0.1016	0.1608	0.1371
4900	0.0754	0.1132	0.1395	0.1281	0.1756
5000	0.0789	0.1001	0.1382	0.1087	0.185
5100	0.1087	0.1042	0.1212	0.1004	0.1192

5200	0.1399	0.0955	0.1163	0.1849	0.1811
5300	0.0987	0.1342	0.0946	0.1491	0.0864
5400	0.0961	0.1098	0.1136	0.1959	0.1286
5500	0.0839	0.1016	0.1094	0.1687	0.1075
5600	0.1414	0.1356	0.1368	0.1503	0.1224
5700	0.0985	0.1326	0.0886	0.1475	0.1474
5800	0.1053	0.1055	0.1006	0.1433	0.1508
5900	0.1425	0.0957	0.1118	0.1049	0.1336
6000	0.0712	0.1019	0.0854	0.1154	0.1256
6100	0.0967	0.0739	0.1045	0.1423	0.1262
6200	0.089	0.1314	0.1147	0.1425	0.122
6300	0.1013	0.087	0.093	0.1241	0.1145
6400	0.1188	0.1176	0.0855	0.1138	0.1835
6500	0.0821	0.0939	0.0993	0.0774	0.1549
6600	0.1039	0.109	0.097	0.0903	0.1382
6700	0.075	0.0831	0.1004	0.0808	0.1606
6800	0.0678	0.0993	0.0989	0.0971	0.1051
6900	0.0936	0.1098	0.1229	0.0969	0.1492
7000	0.08	0.1183	0.072	0.0729	0.082
7100	0.0778	0.1424	0.0771	0.0981	0.1767
7200	0.0727	0.0874	0.0608	0.0901	0.1442
7300	0.0938	0.0906	0.0814	0.0774	0.1417
7400	0.0801	0.1126	0.069	0.0837	0.1095
7500	0.0839	0.1035	0.0779	0.0792	0.1264
7600	0.0855	0.0986	0.0871	0.1248	0.0989
7700	0.0677	0.0734	0.0879	0.104	0.1144
7800	0.0809	0.1009	0.0983	0.1061	0.0977
7900	0.092	0.0953	0.1008	0.0894	0.1391
8000	0.0649	0.0572	0.0806	0.0942	0.1089
8100	0.0795	0.0868	0.1026	0.1253	0.0947
8200	0.0941	0.075	0.1182	0.1022	0.1248
8300	0.085	0.0666	0.0874	0.1006	0.1133
8400	0.1082	0.0841	0.0853	0.124	0.0883
8500	0.0925	0.0767	0.1072	0.1182	0.093
8600	0.0754	0.069	0.0647	0.0899	0.1003
8700	0.0782	0.0686	0.0691	0.0771	0.0965
8800	0.0948	0.0677	0.0955	0.0761	0.1087
8900	0.0948	0.0934	0.0674	0.1083	0.0749
9000	0.0634	0.0734	0.0889	0.088	0.0734
9100	0.0607	0.058	0.0663	0.0805	0.1154
9200	0.0696	0.083	0.0628	0.0647	0.1007
9300	0.094	0.0841	0.0639	0.0539	0.0968
9400	0.0663	0.0653	0.065	0.0726	0.0904
9500	0.0734	0.0744	0.0842	0.095	0.078
9600	0.0752	0.0682	0.0716	0.1025	0.0956
9700	0.0699	0.0747	0.0597	0.0784	0.077
9800	0.1119	0.0686	0.0473	0.0898	0.1048
9900	0.0812	0.0692	0.0512	0.0883	0.1101
10000	0.0727	0.0509	0.0634	0.0822	0.0862

Appendix 3

Tables containing Fish metrics for optimal DG trial: $\alpha = 0.05$, $\beta = 0.005$, degree = 4

Average Episode Length on Training Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	49	23	39	35	92
100	12	139	48	153	101
200	130	14	43	97	68
300	71	84	44	16	44
400	546	13	37	138	48
500	321	27	19	49	21
600	114	160	49	96	42
700	43	64	51	25	39
800	431	20	114	37	51
900	421	31	10	17	13
1000	314	31	15	28	19
1100	226	69	31	46	9
1200	111	14	47	26	23
1300	280	35	28	24	28
1400	46	38	28	32	38
1500	1029	38	33	36	43
1600	71	15	22	19	23
1700	22	12	56	16	12
1800	47	13	16	14	14
1900	316	29	42	29	18
2000	1000	25	9	19	22
2100	15	30	21	26	16
2200	193	20	10	40	28
2300	481	19	8	8	30
2400	3035	27	18	16	25
2500	849	11	41	14	19
2600	18150	34	14	19	16
2700	2071	30	19	23	20
2800	5707	49	27	16	15
2900	9189	16	12	21	13
3000	1683	11	8	14	10
3100	5287	19	30	9	12
3200	227	13	15	19	24
3300	12684	23	15	32	22
3400	52135	16	18	18	11
3500	52135	15	43	23	8
3600	52135	28	25	15	17
3700	52135	18	10	20	18
3800	52135	18	21	14	9
3900	52135	20	17	21	18
4000	52135	18	32	9	28
4100	52135	20	16	14	13
4200	52135	20	14	14	13
4300	52135	18	17	12	14
4400	52135	19	14	17	15
4500	52135	23	11	25	19

4600	52135	16	12	25	22
4700	52135	15	18	12	15
4800	52135	21	16	14	19
4900	52135	16	20	11	16
5000	52135	21	14	31	7
5100	52135	14	25	15	16
5200	52135	22	9	24	17
5300	52135	12	14	13	13
5400	52135	15	16	12	11
5500	52135	15	29	16	10
5600	52135	15	26	18	10
5700	52135	15	18	7	24
5800	52135	13	19	11	13
5900	52135	13	15	23	14
6000	52135	13	12	12	13
6100	52135	27	16	15	10
6200	52135	20	17	12	11
6300	52135	13	12	17	10
6400	52135	9	8	19	16
6500	52135	9	21	8	8
6600	52135	18	8	10	12
6700	52135	8	16	19	10
6800	52135	13	24	16	16
6900	52135	21	10	8	17
7000	52135	17	10	19	16
7100	52135	12	18	9	8
7200	52135	14	11	14	12
7300	52135	20	13	20	10
7400	52135	18	20	13	13
7500	52135	15	12	12	12
7600	52135	19	11	12	22
7700	52135	12	15	15	12
7800	52135	13	15	9	10
7900	52135	17	11	10	11
8000	52135	10	11	15	9
8100	52135	11	13	15	13
8200	52135	9	12	17	12
8300	52135	16	17	20	12
8400	52135	9	15	15	10
8500	52135	16	17	12	13
8600	52135	17	21	10	12
8700	52135	12	13	17	11
8800	52135	11	12	14	15
8900	52135	18	11	11	10
9000	52135	11	12	22	8
9100	52135	15	11	9	8
9200	52135	15	14	11	13
9300	52135	21	8	13	12
9400	52135	16	21	16	10
9500	52135	13	13	8	12
9600	52135	15	16	14	12
9700	52135	11	15	15	12
9800	52135	10	13	9	13
9900	52135	16	14	17	16
10000	52135	12	14	11	12

Average Episode Length on Testing Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	223	61	10	103	228
100	257	94	64	79	237
200	50	130	129	11	80
300	122	49	41	71	35
400	25	24	17	14	81
500	104	107	73	103	58
600	145	60	56	37	30
700	40	48	41	94	38
800	89	26	54	15	25
900	17	57	14	31	17
1000	37	76	32	20	11
1100	23	76	39	32	10
1200	30	14	66	49	31
1300	40	35	17	18	44
1400	14	46	9	31	25
1500	127	15	19	23	13
1600	32	26	13	25	26
1700	60	47	10	12	33
1800	23	24	36	17	24
1900	45	10	10	14	16
2000	14	23	24	16	24
2100	16	31	22	13	13
2200	25	11	10	16	16
2300	101	17	14	7	20
2400	58	15	20	14	8
2500	72	22	37	12	30
2600	48	13	20	21	11
2700	56	11	29	19	16
2800	12	38	24	22	13
2900	40	23	33	22	11
3000	28	22	11	20	15
3100	94	23	24	15	9
3200	25	24	17	29	20
3300	19	16	26	12	9
3400	30	14	10	18	14
3500	30	26	9	11	18
3600	30	16	22	14	14
3700	30	32	23	12	9
3800	30	34	13	21	22
3900	30	15	14	21	14
4000	30	10	20	11	13
4100	30	13	15	22	11
4200	30	9	16	15	32
4300	30	12	29	19	10
4400	30	26	17	19	16
4500	30	15	18	8	18
4600	30	22	19	12	15
4700	30	21	14	16	8
4800	30	11	18	12	12
4900	30	19	10	18	18
5000	30	34	12	9	13
5100	30	14	15	10	10

5200	30	23	15	10	20
5300	30	15	13	14	14
5400	30	20	14	11	9
5500	30	25	12	11	16
5600	30	15	10	10	18
5700	30	21	11	10	19
5800	30	24	14	18	13
5900	30	18	10	12	14
6000	30	15	11	20	11
6100	30	18	11	14	9
6200	30	13	14	21	11
6300	30	20	14	14	8
6400	30	18	12	12	14
6500	30	14	25	11	12
6600	30	20	17	16	10
6700	30	16	17	9	13
6800	30	15	17	17	11
6900	30	11	10	12	10
7000	30	16	15	13	8
7100	30	16	13	11	7
7200	30	16	17	13	18
7300	30	10	14	18	18
7400	30	16	19	17	12
7500	30	14	15	11	11
7600	30	16	11	7	17
7700	30	16	10	11	7
7800	30	12	9	8	16
7900	30	17	7	19	15
8000	30	17	11	8	13
8100	30	9	9	13	10
8200	30	11	13	16	17
8300	30	11	11	11	14
8400	30	11	11	10	14
8500	30	14	11	11	11
8600	30	12	9	12	12
8700	30	19	14	10	11
8800	30	9	11	11	7
8900	30	14	13	14	7
9000	30	15	13	13	18
9100	30	10	9	8	10
9200	30	13	12	14	11
9300	30	13	10	11	13
9400	30	13	15	9	9
9500	30	13	12	16	7
9600	30	13	15	11	13
9700	30	10	13	9	10
9800	30	13	14	13	13
9900	30	13	11	10	14
10000	30	8	14	12	13

Mean Square Error on Training Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	0.8771	0.1701	0.5096	0.7215	0.4539
100	0.8136	0.1138	0.4778	0.7147	0.4705
200	0.8592	0.1242	0.6769	0.727	0.4802
300	0.8773	0.1742	0.3798	0.683	0.5218
400	0.8281	0.1186	0.6164	0.6664	0.4756
500	0.8433	0.1684	0.6802	0.6874	0.4982
600	0.8618	0.1339	0.5455	0.7075	0.4929
700	0.8142	0.1275	0.5504	0.7271	0.5128
800	0.983	0.1157	0.5483	0.7161	0.5013
900	0.898	0.1286	0.6735	0.7054	0.4775
1000	0.868	0.1774	0.5824	0.7036	0.4832
1100	0.833	0.1646	0.7066	0.7284	0.4921
1200	0.8206	0.1533	0.4895	0.687	0.5083
1300	0.8324	0.1456	0.8426	0.705	0.4882
1400	0.8337	0.1091	0.3343	0.7079	0.4658
1500	0.8343	0.1885	0.3731	0.7287	0.5027
1600	0.8324	0.1316	0.4589	0.7673	0.4829
1700	0.871	0.1443	0.4263	0.7354	0.5079
1800	0.8784	0.118	0.4308	0.6861	0.4938
1900	0.8047	0.1491	0.5395	0.7125	0.5075
2000	0.8591	0.1616	0.7331	0.7032	0.4946
2100	0.8666	0.105	0.5647	0.7137	0.4952
2200	0.8711	0.1036	0.5785	0.752	0.4648
2300	0.864	0.1004	0.3331	0.7065	0.4968
2400	0.8519	0.1395	0.5323	0.6984	0.5206
2500	0.9095	0.1183	0.4093	0.7239	0.4562
2600	0.8738	0.1205	0.599	0.6933	0.4853
2700	0.8511	0.1366	0.586	0.7155	0.4976
2800	0.8946	0.1344	0.6588	0.6937	0.4584
2900	0.8834	0.1366	0.6215	0.6896	0.4894
3000	0.8587	0.142	0.4987	0.7283	0.4813
3100	0.8483	0.1642	0.5896	0.7097	0.5078
3200	0.8801	0.1611	0.5167	0.7138	0.504
3300	0.8483	0.1484	0.3776	0.7252	0.4885
3400	0.8321	0.1321	0.5015	0.7193	0.4839
3500	0.8321	0.1361	0.5878	0.7349	0.4761
3600	0.8321	0.121	0.7373	0.6956	0.4852
3700	0.8321	0.1405	0.7369	0.7456	0.5163
3800	0.8321	0.1179	0.5783	0.7413	0.4952
3900	0.8321	0.1352	0.5404	0.753	0.5036
4000	0.8321	0.1529	0.6613	0.7138	0.5193
4100	0.8321	0.1718	0.5321	0.7423	0.5295
4200	0.8321	0.2028	0.5456	0.7307	0.4996
4300	0.8321	0.1537	0.6794	0.7133	0.5186
4400	0.8321	0.118	0.4545	0.703	0.4582
4500	0.8321	0.1163	0.8301	0.7234	0.5161
4600	0.8321	0.1602	0.7218	0.7448	0.4826
4700	0.8321	0.1576	0.7281	0.7388	0.5256
4800	0.8321	0.1337	0.6435	0.726	0.5167
4900	0.8321	0.1123	0.402	0.7422	0.4789
5000	0.8321	0.1107	0.4005	0.7065	0.5421
5100	0.8321	0.1163	0.5706	0.7352	0.4925

5200	0.8321	0.1571	0.5686	0.7469	0.5138
5300	0.8321	0.1791	0.4773	0.7144	0.5176
5400	0.8321	0.1458	0.6679	0.7101	0.4956
5500	0.8321	0.1565	0.5043	0.6622	0.4928
5600	0.8321	0.1373	0.5942	0.6991	0.4932
5700	0.8321	0.1557	0.5697	0.7398	0.5184
5800	0.8321	0.1213	0.5325	0.7386	0.5277
5900	0.8321	0.0993	0.5516	0.7275	0.4973
6000	0.8321	0.1504	0.5465	0.7304	0.5165
6100	0.8321	0.1538	0.6021	0.7317	0.5359
6200	0.8321	0.1041	0.3183	0.7153	0.4925
6300	0.8321	0.164	0.6732	0.7233	0.4768
6400	0.8321	0.188	0.5435	0.7262	0.4789
6500	0.8321	0.1287	0.8099	0.6993	0.4792
6600	0.8321	0.1928	0.5497	0.7224	0.5194
6700	0.8321	0.1228	0.6188	0.7433	0.497
6800	0.8321	0.1363	0.6192	0.7578	0.4866
6900	0.8321	0.1384	0.5501	0.718	0.4837
7000	0.8321	0.1596	0.7778	0.7265	0.5065
7100	0.8321	0.1684	0.8692	0.7678	0.4915
7200	0.8321	0.1508	0.6721	0.7381	0.5246
7300	0.8321	0.1906	0.5011	0.7039	0.4987
7400	0.8321	0.2009	0.4951	0.6951	0.4954
7500	0.8321	0.1553	0.4833	0.7124	0.5242
7600	0.8321	0.1511	0.8114	0.721	0.5032
7700	0.8321	0.1267	0.64	0.7362	0.5153
7800	0.8321	0.1605	0.5798	0.7163	0.5144
7900	0.8321	0.1395	0.6508	0.7075	0.5252
8000	0.8321	0.168	0.533	0.7656	0.5063
8100	0.8321	0.1333	0.5153	0.7113	0.5108
8200	0.8321	0.1616	0.4684	0.7241	0.4688
8300	0.8321	0.1751	0.6334	0.7868	0.5022
8400	0.8321	0.1359	0.3904	0.7314	0.4875
8500	0.8321	0.1375	0.5765	0.714	0.4753
8600	0.8321	0.1525	0.7161	0.7592	0.5044
8700	0.8321	0.1668	0.6487	0.7476	0.4853
8800	0.8321	0.1041	0.4781	0.7641	0.4839
8900	0.8321	0.1247	0.5875	0.7069	0.5024
9000	0.8321	0.1147	0.5708	0.7384	0.5105
9100	0.8321	0.1667	0.7408	0.7522	0.5118
9200	0.8321	0.1629	0.7975	0.7129	0.4863
9300	0.8321	0.1277	0.8286	0.7399	0.4779
9400	0.8321	0.1714	0.6119	0.7181	0.5096
9500	0.8321	0.1216	0.6197	0.705	0.4906
9600	0.8321	0.1465	0.5024	0.7491	0.4698
9700	0.8321	0.1077	0.4307	0.7268	0.4783
9800	0.8321	0.185	0.4534	0.7125	0.4959
9900	0.8321	0.1618	0.6308	0.7268	0.5295
10000	0.8321	0.1375	0.7696	0.7539	0.5041

Mean Square Error on Testing Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	0.483	0.1144	0.1829	0.4349	0.357
100	0.4888	0.1181	0.1995	0.4105	0.3314
200	0.4771	0.0982	0.2185	0.4139	0.3628
300	0.476	0.1176	0.1606	0.4282	0.3542
400	0.4998	0.1253	0.1887	0.4288	0.3637
500	0.4919	0.1262	0.1817	0.4096	0.3441
600	0.4788	0.1256	0.2161	0.4086	0.3606
700	0.4802	0.099	0.1647	0.4256	0.3252
800	0.4833	0.0978	0.1749	0.4253	0.3561
900	0.4687	0.1438	0.175	0.4515	0.3273
1000	0.4608	0.1273	0.2239	0.4063	0.3504
1100	0.4938	0.1285	0.2007	0.4293	0.3494
1200	0.4694	0.1163	0.2082	0.4226	0.3363
1300	0.4864	0.1317	0.1632	0.4431	0.352
1400	0.4661	0.1287	0.189	0.4305	0.3193
1500	0.5136	0.1311	0.1792	0.4226	0.3276
1600	0.4771	0.1288	0.2047	0.4085	0.3506
1700	0.4628	0.1288	0.1721	0.4325	0.3432
1800	0.4898	0.1241	0.2046	0.4291	0.3486
1900	0.4726	0.1324	0.2013	0.4014	0.3396
2000	0.4762	0.1256	0.164	0.3997	0.3442
2100	0.4832	0.1169	0.172	0.4039	0.3435
2200	0.4546	0.1185	0.2042	0.3917	0.36
2300	0.4466	0.1362	0.1925	0.4082	0.339
2400	0.5076	0.1053	0.1951	0.3982	0.3504
2500	0.4664	0.1186	0.1898	0.4287	0.3578
2600	0.4981	0.1243	0.19	0.4403	0.3435
2700	0.4913	0.1103	0.2074	0.4414	0.3601
2800	0.4653	0.1281	0.2005	0.4347	0.3536
2900	0.4454	0.1089	0.1935	0.4278	0.3317
3000	0.4733	0.1186	0.2095	0.4304	0.3587
3100	0.4792	0.1294	0.209	0.4163	0.3288
3200	0.5035	0.1243	0.1822	0.4301	0.3499
3300	0.4576	0.0983	0.1901	0.4255	0.3483
3400	0.4916	0.1378	0.217	0.4009	0.3435
3500	0.4916	0.1232	0.1934	0.4208	0.3472
3600	0.4916	0.125	0.1716	0.4075	0.3441
3700	0.4916	0.1377	0.1894	0.4041	0.3554
3800	0.4916	0.0982	0.2009	0.4005	0.3375
3900	0.4916	0.1163	0.2129	0.4024	0.3402
4000	0.4916	0.1164	0.1713	0.414	0.3463
4100	0.4916	0.1397	0.1815	0.4305	0.3475
4200	0.4916	0.1269	0.2025	0.4169	0.3472
4300	0.4916	0.1351	0.1721	0.4038	0.3522
4400	0.4916	0.1001	0.1837	0.4331	0.3403
4500	0.4916	0.1216	0.1779	0.4198	0.3672
4600	0.4916	0.1249	0.1577	0.3913	0.3505
4700	0.4916	0.1124	0.2013	0.4266	0.3443
4800	0.4916	0.1386	0.1594	0.4326	0.361
4900	0.4916	0.1171	0.2182	0.4204	0.3455
5000	0.4916	0.1186	0.188	0.4279	0.346
5100	0.4916	0.1291	0.1739	0.3924	0.3273

5200	0.4916	0.1221	0.1636	0.4083	0.347
5300	0.4916	0.1141	0.1871	0.4311	0.3442
5400	0.4916	0.1387	0.1666	0.4051	0.3568
5500	0.4916	0.1044	0.1616	0.4268	0.3531
5600	0.4916	0.1186	0.198	0.4165	0.3516
5700	0.4916	0.1301	0.1631	0.4293	0.3594
5800	0.4916	0.1261	0.2081	0.4166	0.3577
5900	0.4916	0.1276	0.1988	0.4394	0.3627
6000	0.4916	0.1051	0.2016	0.41	0.3451
6100	0.4916	0.1178	0.185	0.4348	0.356
6200	0.4916	0.1234	0.1656	0.3984	0.3478
6300	0.4916	0.1409	0.1933	0.4244	0.3477
6400	0.4916	0.1089	0.1973	0.4271	0.3379
6500	0.4916	0.0979	0.1899	0.4283	0.364
6600	0.4916	0.1076	0.1821	0.4268	0.3513
6700	0.4916	0.1334	0.2231	0.3907	0.3456
6800	0.4916	0.1354	0.1589	0.4349	0.3484
6900	0.4916	0.1114	0.1735	0.4434	0.3496
7000	0.4916	0.1386	0.1625	0.4176	0.3601
7100	0.4916	0.1444	0.1557	0.3992	0.3241
7200	0.4916	0.1146	0.174	0.4067	0.3436
7300	0.4916	0.1429	0.1809	0.4047	0.3206
7400	0.4916	0.1198	0.1741	0.3911	0.3454
7500	0.4916	0.1156	0.2005	0.4031	0.3402
7600	0.4916	0.114	0.2343	0.3972	0.3146
7700	0.4916	0.1105	0.2192	0.4327	0.3508
7800	0.4916	0.1238	0.1963	0.3984	0.3611
7900	0.4916	0.129	0.2009	0.3893	0.3414
8000	0.4916	0.115	0.1909	0.417	0.3303
8100	0.4916	0.1389	0.1959	0.4194	0.3501
8200	0.4916	0.1156	0.1706	0.4461	0.3229
8300	0.4916	0.1294	0.1836	0.4244	0.3537
8400	0.4916	0.1427	0.1744	0.3966	0.3573
8500	0.4916	0.1374	0.1718	0.4052	0.3439
8600	0.4916	0.1404	0.1545	0.4765	0.3618
8700	0.4916	0.1192	0.1903	0.3934	0.3559
8800	0.4916	0.1163	0.1676	0.4384	0.3344
8900	0.4916	0.1179	0.1946	0.4232	0.3334
9000	0.4916	0.126	0.1767	0.4374	0.3369
9100	0.4916	0.1305	0.2128	0.4066	0.3403
9200	0.4916	0.1403	0.1869	0.4065	0.3359
9300	0.4916	0.1207	0.1765	0.4183	0.3571
9400	0.4916	0.146	0.1795	0.4097	0.3567
9500	0.4916	0.1075	0.1948	0.4122	0.3611
9600	0.4916	0.121	0.1744	0.4287	0.3392
9700	0.4916	0.1339	0.1943	0.4192	0.359
9800	0.4916	0.1431	0.1796	0.4123	0.3325
9900	0.4916	0.1047	0.1743	0.405	0.3485
10000	0.4916	0.1222	0.1836	0.4455	0.3614

Appendix 4

Tables containing IDGM metrics for optimal DG trial: $\alpha = 0.005$, $\beta = 0.005$, degree = 4

Average Episode Length on Training Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	91	121	246	64	54
100	94	99	177	55	53
200	258	123	107	87	74
300	49	77	80	19	23
400	28	70	144	40	228
500	176	12	25	63	481
600	81	19	115	9	306
700	49	109	87	44	18
800	27	37	21	16	82
900	41	50	48	29	132
1000	26	11	43	23	196
1100	29	149	37	19	54
1200	15	11	43	29	40
1300	20	35	22	56	129
1400	26	26	17	15	513
1500	84	45	28	31	42
1600	23	66	57	16	213
1700	30	56	17	26	129
1800	27	32	15	35	25
1900	19	12	31	40	108
2000	55	35	23	33	557
2100	28	21	10	8	462
2200	12	22	52	36	44
2300	50	9	23	15	161
2400	26	21	13	20	815
2500	25	19	27	22	249
2600	26	44	33	9	722
2700	33	19	15	24	2803
2800	13	22	18	13	43
2900	22	28	18	10	668
3000	29	18	24	22	1055
3100	21	18	7	19	3034
3200	8	13	17	27	952
3300	22	34	27	17	547
3400	30	30	12	32	67
3500	26	19	17	13	144
3600	18	15	20	18	197
3700	12	22	31	18	881
3800	16	20	13	14	4382
3900	24	19	19	11	254
4000	11	29	14	25	205
4100	9	10	12	14	10325
4200	30	20	15	13	6653
4300	14	11	19	23	183
4400	16	7	18	17	276
4500	14	15	16	18	2441
4600	18	13	15	11	2290

4700	26	16	28	17	5529
4800	13	23	9	12	764
4900	16	24	8	13	543
5000	15	9	16	11	694
5100	12	11	14	12	5048
5200	23	15	13	14	4247
5300	21	13	11	12	4633
5400	9	10	17	12	1613
5500	13	19	15	14	2313
5600	27	11	12	14	7773
5700	18	21	20	16	42034
5800	20	8	16	11	81806
5900	12	21	15	16	81806
6000	24	11	22	19	81806
6100	21	13	8	12	81806
6200	12	19	15	9	81806
6300	20	12	18	17	81806
6400	17	9	18	15	81806
6500	10	14	13	11	81806
6600	9	27	27	10	81806
6700	22	15	15	15	81806
6800	19	12	17	14	81806
6900	13	15	12	15	81806
7000	17	11	14	15	81806
7100	12	14	11	11	81806
7200	31	11	17	14	81806
7300	15	13	14	18	81806
7400	20	13	14	16	81806
7500	16	11	12	11	81806
7600	21	20	17	12	81806
7700	17	13	12	16	81806
7800	34	17	13	13	81806
7900	16	8	14	10	81806
8000	13	15	12	11	81806
8100	20	22	12	13	81806
8200	18	17	13	16	81806
8300	12	15	10	11	81806
8400	15	21	14	12	81806
8500	15	9	14	21	81806
8600	13	12	14	15	81806
8700	14	10	12	19	81806
8800	8	10	7	15	81806
8900	15	13	11	12	81806
9000	16	8	13	17	81806
9100	18	11	12	12	81806
9200	17	18	17	14	81806
9300	11	9	12	12	81806
9400	12	9	12	12	81806
9500	10	12	12	13	81806
9600	13	11	15	12	81806
9700	13	11	13	11	81806
9800	12	13	17	15	81806
9900	13	17	19	21	81806
10000	12	15	11	13	81806

Average Episode Length on Testing Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	56	103	163	69	75
100	100	190	26	35	4
200	39	66	32	25	42
300	59	93	67	183	27
400	166	104	131	506	111
500	35	156	97	52	101
600	39	22	27	41	24
700	31	96	78	59	24
800	37	38	27	53	41
900	26	50	73	20	9
1000	19	40	14	125	92
1100	35	79	23	165	66
1200	28	75	34	20	133
1300	30	77	18	26	42
1400	26	22	149	85	26
1500	66	65	92	35	17
1600	67	41	53	26	79
1700	29	121	22	70	54
1800	41	26	16	30	27
1900	10	20	43	18	100
2000	21	35	34	28	79
2100	23	45	32	32	13
2200	13	35	16	14	66
2300	14	47	23	23	25
2400	38	20	12	26	22
2500	58	52	31	42	14
2600	12	24	33	23	16
2700	37	45	8	52	10
2800	16	16	32	25	12
2900	18	16	25	13	18
3000	13	14	54	20	13
3100	12	31	13	12	18
3200	11	19	17	23	5
3300	28	13	14	22	23
3400	16	17	28	13	5
3500	25	18	22	22	35
3600	11	15	17	29	199
3700	35	35	31	18	4
3800	10	12	23	22	6
3900	13	15	16	14	17
4000	28	16	18	39	8
4100	13	16	11	16	44
4200	16	13	17	17	83
4300	15	18	29	23	33
4400	22	16	36	19	15
4500	27	8	8	19	17
4600	21	11	13	26	380
4700	23	25	19	12	7
4800	15	20	15	21	15
4900	12	30	24	18	8
5000	20	23	11	14	15
5100	29	19	12	25	13

5200	22	11	19	12	9
5300	12	33	11	15	9
5400	17	27	14	27	15
5500	25	28	20	30	11
5600	12	14	19	16	21
5700	14	14	17	22	7
5800	15	12	21	19	12
5900	14	22	18	16	12
6000	22	25	23	15	12
6100	15	9	28	15	12
6200	17	14	20	12	12
6300	22	17	22	15	12
6400	10	11	11	22	12
6500	14	15	17	11	12
6600	25	11	16	14	12
6700	13	17	21	21	12
6800	11	13	16	11	12
6900	14	21	11	16	12
7000	14	18	19	11	12
7100	15	15	15	20	12
7200	13	17	8	10	12
7300	14	16	9	9	12
7400	10	15	12	18	12
7500	20	13	12	16	12
7600	12	16	16	20	12
7700	16	10	15	22	12
7800	12	15	13	15	12
7900	9	21	19	25	12
8000	12	19	15	10	12
8100	14	9	20	13	12
8200	20	16	12	12	12
8300	9	20	12	13	12
8400	13	17	8	19	12
8500	16	14	23	23	12
8600	15	16	13	12	12
8700	11	16	16	20	12
8800	17	15	14	14	12
8900	10	19	11	14	12
9000	12	17	11	10	12
9100	12	18	11	14	12
9200	8	13	25	15	12
9300	10	16	10	9	12
9400	9	14	15	12	12
9500	9	12	14	15	12
9600	15	15	11	11	12
9700	11	15	12	15	12
9800	14	11	16	15	12
9900	12	11	9	17	12
10000	15	13	14	14	12

Mean Square Error on Training Environment

Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	0.297	0.2313	0.1763	0.1701	0.2981
100	0.2363	0.2237	0.1748	0.1759	0.2834
200	0.2731	0.216	0.1866	0.1793	0.3123
300	0.2726	0.2259	0.1747	0.1733	0.3142
400	0.2407	0.217	0.1652	0.1781	0.3065
500	0.2915	0.2078	0.1639	0.169	0.3207
600	0.2659	0.2233	0.1814	0.1646	0.2872
700	0.2853	0.2125	0.1887	0.1731	0.2993
800	0.2797	0.2181	0.1834	0.1782	0.3213
900	0.3006	0.2392	0.1807	0.1744	0.2897
1000	0.2921	0.2276	0.1817	0.1769	0.3154
1100	0.2598	0.207	0.159	0.1697	0.3117
1200	0.2697	0.2056	0.1738	0.1838	0.3112
1300	0.2926	0.2224	0.1953	0.1801	0.3069
1400	0.2541	0.1939	0.1865	0.1783	0.3042
1500	0.2647	0.2286	0.1894	0.1823	0.2955
1600	0.2852	0.2255	0.1744	0.1708	0.306
1700	0.2749	0.232	0.1469	0.1842	0.3003
1800	0.2749	0.2018	0.1828	0.1669	0.3133
1900	0.278	0.2033	0.1746	0.1745	0.3106
2000	0.2532	0.2032	0.1633	0.1706	0.3057
2100	0.2637	0.2475	0.1876	0.1756	0.283
2200	0.2708	0.2211	0.182	0.1782	0.3082
2300	0.2743	0.2216	0.1779	0.1752	0.3093
2400	0.2668	0.217	0.1701	0.1767	0.2972
2500	0.267	0.2138	0.1878	0.1809	0.2913
2600	0.2599	0.2203	0.1847	0.1698	0.2752
2700	0.2506	0.2159	0.1746	0.1741	0.3169
2800	0.2619	0.2195	0.1906	0.1815	0.295
2900	0.2811	0.2223	0.1789	0.1737	0.293
3000	0.2494	0.2209	0.1801	0.1717	0.298
3100	0.2619	0.2103	0.1864	0.1768	0.2986
3200	0.286	0.2098	0.1738	0.1722	0.2889
3300	0.2572	0.2192	0.1796	0.1687	0.3135
3400	0.2905	0.2408	0.172	0.1733	0.3029
3500	0.2631	0.226	0.1772	0.1788	0.2952
3600	0.2651	0.2183	0.17	0.1853	0.2876
3700	0.2744	0.2136	0.1655	0.1665	0.3058
3800	0.306	0.2203	0.1878	0.1831	0.3175
3900	0.2862	0.228	0.1661	0.1609	0.291
4000	0.2681	0.2254	0.1661	0.1593	0.321
4100	0.2629	0.2214	0.1741	0.1775	0.3186
4200	0.2526	0.2209	0.1976	0.187	0.288
4300	0.2637	0.2033	0.1935	0.1815	0.315
4400	0.2747	0.231	0.1838	0.167	0.3105
4500	0.2752	0.2434	0.1754	0.1729	0.3123
4600	0.2744	0.2223	0.1714	0.1702	0.3072
4700	0.2747	0.226	0.185	0.1697	0.3038
4800	0.2596	0.214	0.1671	0.174	0.2894
4900	0.2661	0.2169	0.1644	0.177	0.3188
5000	0.2667	0.2365	0.1677	0.1794	0.2943
5100	0.2617	0.2089	0.1778	0.1723	0.3279

5200	0.2746	0.225	0.1841	0.1754	0.2889
5300	0.2827	0.236	0.1811	0.1707	0.3059
5400	0.2654	0.2132	0.1883	0.1729	0.2979
5500	0.2813	0.2252	0.184	0.1903	0.3095
5600	0.2899	0.2293	0.1569	0.1834	0.3073
5700	0.2772	0.2086	0.179	0.1759	0.2967
5800	0.2714	0.2081	0.1641	0.1695	0.2878
5900	0.2723	0.2128	0.17	0.178	0.2878
6000	0.264	0.2173	0.1913	0.1695	0.2878
6100	0.2712	0.2073	0.1764	0.1939	0.2878
6200	0.2791	0.2169	0.1805	0.1732	0.2878
6300	0.2973	0.2112	0.1714	0.1734	0.2878
6400	0.2825	0.2234	0.1635	0.175	0.2878
6500	0.2645	0.223	0.1737	0.1821	0.2878
6600	0.2819	0.209	0.1699	0.1747	0.2878
6700	0.2662	0.2375	0.1974	0.1717	0.2878
6800	0.2803	0.2374	0.1683	0.1935	0.2878
6900	0.2821	0.2262	0.1788	0.1818	0.2878
7000	0.2952	0.2356	0.1801	0.183	0.2878
7100	0.2852	0.2234	0.1792	0.1651	0.2878
7200	0.2851	0.218	0.1759	0.1666	0.2878
7300	0.2825	0.2191	0.1794	0.1746	0.2878
7400	0.2865	0.2232	0.1736	0.1718	0.2878
7500	0.2542	0.2413	0.1698	0.1695	0.2878
7600	0.2972	0.2307	0.1869	0.1847	0.2878
7700	0.2622	0.2476	0.1722	0.1845	0.2878
7800	0.2772	0.2344	0.1915	0.1684	0.2878
7900	0.2661	0.2103	0.1782	0.1677	0.2878
8000	0.2877	0.2082	0.2024	0.1873	0.2878
8100	0.269	0.2234	0.1752	0.1706	0.2878
8200	0.2835	0.2183	0.1861	0.179	0.2878
8300	0.2704	0.2321	0.1724	0.1725	0.2878
8400	0.2926	0.2197	0.1761	0.189	0.2878
8500	0.2841	0.2421	0.1845	0.1683	0.2878
8600	0.2696	0.2311	0.1604	0.1729	0.2878
8700	0.2898	0.2106	0.1847	0.1756	0.2878
8800	0.2766	0.2032	0.1833	0.1787	0.2878
8900	0.2962	0.234	0.1683	0.1702	0.2878
9000	0.2882	0.2215	0.1825	0.1635	0.2878
9100	0.2617	0.2018	0.1713	0.1828	0.2878
9200	0.2809	0.2058	0.1674	0.1846	0.2878
9300	0.2959	0.2002	0.1901	0.1607	0.2878
9400	0.2575	0.204	0.1958	0.1638	0.2878
9500	0.2825	0.2516	0.1799	0.1814	0.2878
9600	0.2899	0.2282	0.1722	0.1718	0.2878
9700	0.2665	0.2065	0.17	0.1781	0.2878
9800	0.295	0.2177	0.1557	0.1788	0.2878
9900	0.2819	0.2221	0.1835	0.1773	0.2878
10000	0.2813	0.2205	0.206	0.1678	0.2878

Mean Square Error on Testing Environment

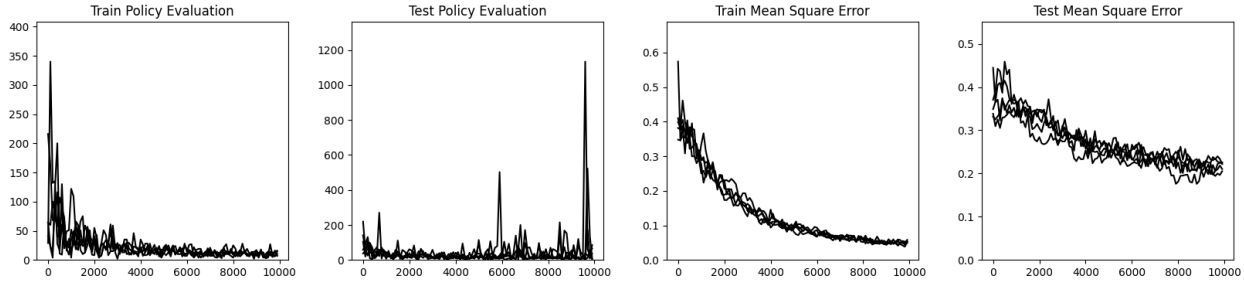
Episode	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	0.2704	0.2314	0.2223	0.2337	0.2989
100	0.27	0.2255	0.2268	0.2323	0.2651
200	0.2499	0.2577	0.2175	0.2388	0.2885
300	0.251	0.2755	0.244	0.2321	0.281
400	0.2754	0.2392	0.2123	0.2147	0.3075
500	0.2851	0.2778	0.2337	0.2347	0.2872
600	0.2555	0.2399	0.2125	0.2505	0.2704
700	0.2434	0.2666	0.219	0.2392	0.2935
800	0.2973	0.2296	0.2121	0.2344	0.2839
900	0.2532	0.2303	0.2297	0.1947	0.2792
1000	0.2615	0.247	0.2212	0.212	0.2887
1100	0.266	0.2689	0.2416	0.2248	0.2902
1200	0.2734	0.2507	0.2333	0.2297	0.267
1300	0.2514	0.2489	0.2364	0.2439	0.2701
1400	0.2678	0.2255	0.2212	0.2432	0.275
1500	0.2579	0.2566	0.2058	0.2516	0.2814
1600	0.2754	0.2561	0.2094	0.2126	0.2585
1700	0.2876	0.2375	0.2205	0.2395	0.2895
1800	0.2605	0.2409	0.2144	0.2267	0.2712
1900	0.277	0.2475	0.2295	0.2167	0.3164
2000	0.268	0.2539	0.2364	0.2316	0.2888
2100	0.2657	0.2523	0.235	0.2433	0.2933
2200	0.2408	0.241	0.1941	0.2229	0.2886
2300	0.256	0.2764	0.2424	0.2151	0.2962
2400	0.2824	0.2674	0.2446	0.2493	0.2821
2500	0.2602	0.2423	0.2106	0.231	0.2695
2600	0.2616	0.2536	0.2246	0.2479	0.2932
2700	0.2872	0.2567	0.2334	0.2187	0.2535
2800	0.2738	0.2393	0.2418	0.2378	0.2931
2900	0.275	0.2592	0.2481	0.2212	0.2845
3000	0.2705	0.2491	0.2445	0.2393	0.2758
3100	0.2787	0.2488	0.234	0.2428	0.2678
3200	0.2621	0.2541	0.2183	0.2259	0.279
3300	0.2639	0.2376	0.2272	0.2306	0.2735
3400	0.2807	0.2423	0.2258	0.2348	0.3037
3500	0.253	0.2565	0.2015	0.2332	0.2776
3600	0.256	0.2263	0.2137	0.2451	0.2613
3700	0.2822	0.2432	0.2218	0.224	0.2803
3800	0.2522	0.2521	0.2429	0.2171	0.3006
3900	0.2717	0.2524	0.2229	0.2267	0.2813
4000	0.2592	0.2667	0.219	0.2311	0.2772
4100	0.2434	0.2582	0.2381	0.2382	0.2826
4200	0.269	0.2563	0.2246	0.219	0.2451
4300	0.2745	0.2353	0.2386	0.227	0.2819
4400	0.2528	0.2454	0.2437	0.232	0.2728
4500	0.2586	0.246	0.2402	0.2277	0.2847
4600	0.2789	0.2231	0.2244	0.2308	0.2744
4700	0.2762	0.263	0.2242	0.232	0.2841
4800	0.2403	0.2385	0.2146	0.2242	0.2838
4900	0.2616	0.2401	0.2084	0.2313	0.2982
5000	0.2643	0.2637	0.2427	0.2122	0.2617
5100	0.286	0.2598	0.2158	0.2236	0.2835

5200	0.2749	0.2695	0.2262	0.235	0.2983
5300	0.264	0.2591	0.2267	0.2269	0.2696
5400	0.2719	0.2408	0.214	0.2267	0.2828
5500	0.2664	0.2395	0.2307	0.2412	0.2898
5600	0.2503	0.2268	0.2249	0.2209	0.2764
5700	0.2718	0.2712	0.2181	0.2223	0.283
5800	0.2696	0.2807	0.2421	0.2108	0.2886
5900	0.2693	0.2665	0.2223	0.2337	0.2886
6000	0.284	0.2576	0.2211	0.246	0.2886
6100	0.2676	0.2361	0.2087	0.2443	0.2886
6200	0.2489	0.2525	0.2522	0.2385	0.2886
6300	0.2836	0.2606	0.2305	0.2276	0.2886
6400	0.2662	0.2498	0.227	0.2275	0.2886
6500	0.2734	0.233	0.2282	0.2479	0.2886
6600	0.2725	0.2322	0.2099	0.2505	0.2886
6700	0.2754	0.2719	0.2432	0.2194	0.2886
6800	0.2805	0.2645	0.2336	0.2558	0.2886
6900	0.2744	0.2516	0.2241	0.2037	0.2886
7000	0.2747	0.2602	0.2388	0.2198	0.2886
7100	0.2674	0.2453	0.2404	0.2393	0.2886
7200	0.2759	0.2646	0.2225	0.2348	0.2886
7300	0.2749	0.2547	0.2439	0.2154	0.2886
7400	0.2774	0.2637	0.2506	0.2449	0.2886
7500	0.2636	0.24	0.2166	0.217	0.2886
7600	0.2792	0.2665	0.2153	0.2215	0.2886
7700	0.2623	0.2445	0.218	0.2221	0.2886
7800	0.2534	0.2603	0.2072	0.2271	0.2886
7900	0.2686	0.2313	0.2148	0.2004	0.2886
8000	0.2607	0.2684	0.2221	0.2227	0.2886
8100	0.2841	0.2552	0.2263	0.2337	0.2886
8200	0.2816	0.2513	0.2448	0.2231	0.2886
8300	0.2876	0.2425	0.2287	0.2489	0.2886
8400	0.2733	0.252	0.2123	0.2459	0.2886
8500	0.259	0.248	0.2354	0.2104	0.2886
8600	0.2826	0.254	0.2205	0.256	0.2886
8700	0.2621	0.263	0.2245	0.2236	0.2886
8800	0.2767	0.2419	0.2343	0.2382	0.2886
8900	0.2496	0.2506	0.2182	0.2208	0.2886
9000	0.2606	0.2554	0.2466	0.2436	0.2886
9100	0.2793	0.2645	0.2126	0.2283	0.2886
9200	0.271	0.2447	0.2151	0.249	0.2886
9300	0.2653	0.2572	0.2205	0.2391	0.2886
9400	0.248	0.2329	0.2249	0.2099	0.2886
9500	0.2641	0.2594	0.2316	0.2418	0.2886
9600	0.2886	0.2442	0.2458	0.2192	0.2886
9700	0.2885	0.2642	0.2377	0.213	0.2886
9800	0.2699	0.2625	0.2431	0.2081	0.2886
9900	0.2665	0.2574	0.2469	0.2147	0.2886
10000	0.25	0.2747	0.2207	0.25	0.2886

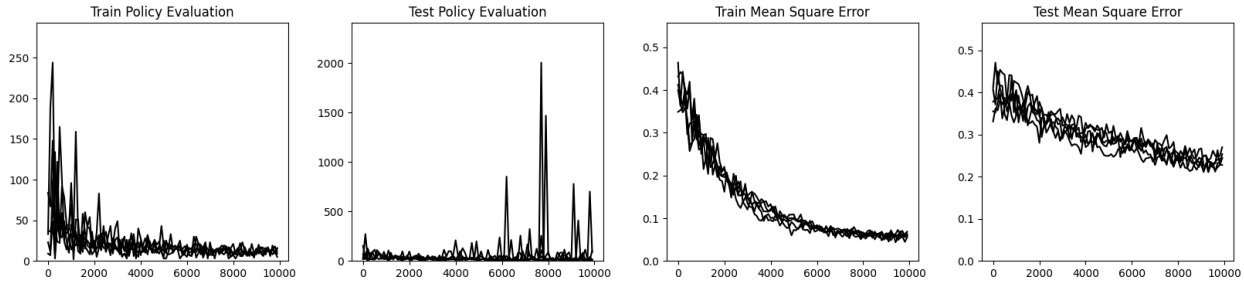
Appendix 5

Algorithm: TD(0)

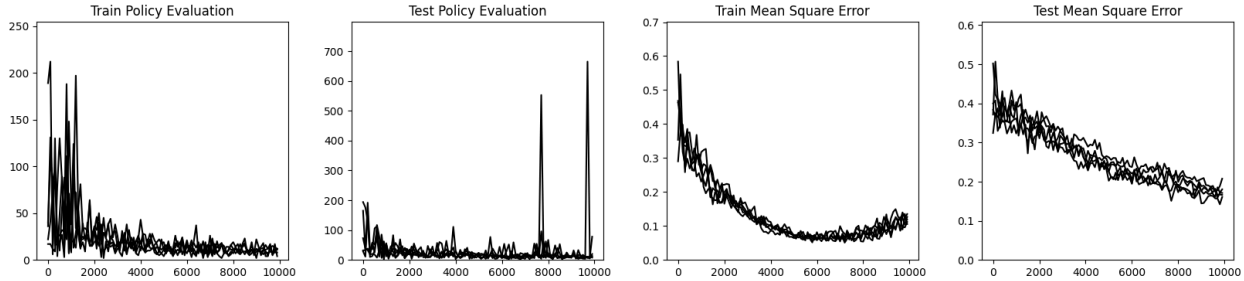
Learning Rate = 0.01, Degree = 2, 0 Diverging Trials



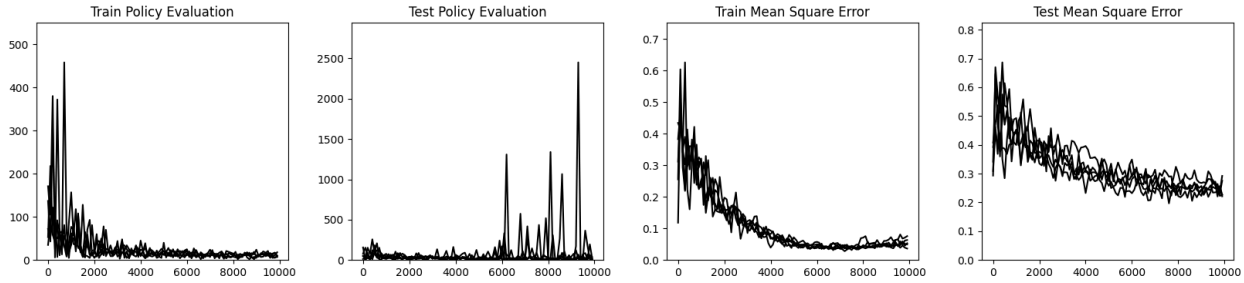
Learning Rate = 0.01, Degree = 3, 0 Diverging Trials



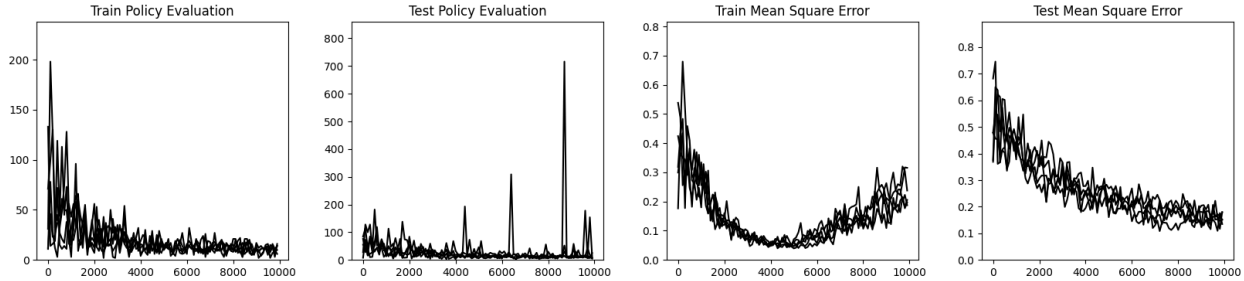
Learning Rate = 0.01, Degree = 4, 0 Diverging Trials



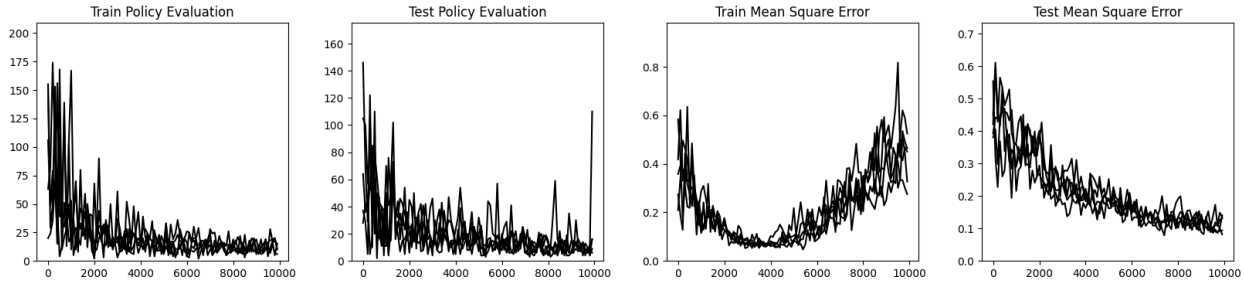
Learning Rate = 0.05, Degree = 2, 0 Diverging Trials



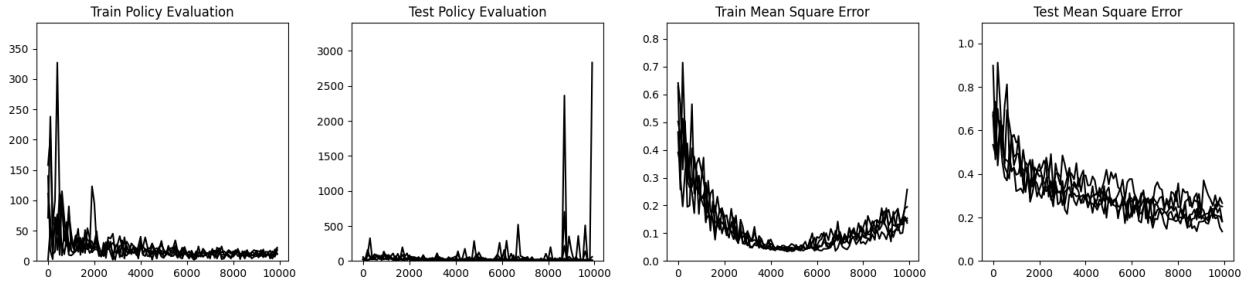
Learning Rate = 0.05, Degree = 3, 0 Diverging Trials



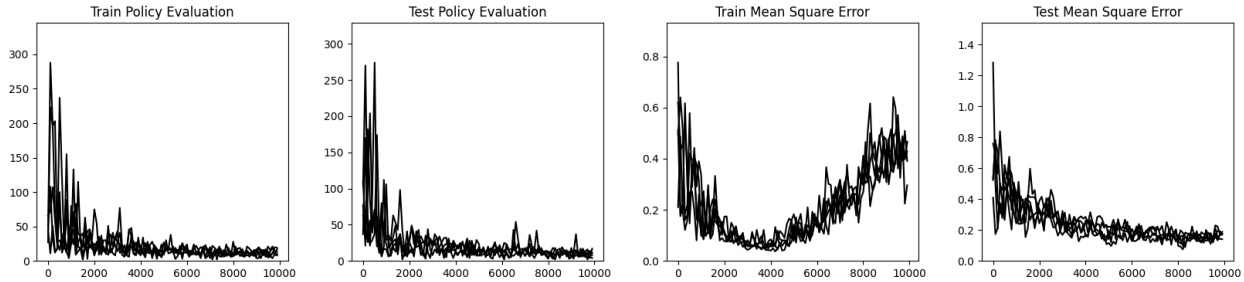
Learning Rate = 0.05, Degree = 4, 0 Diverging Trials



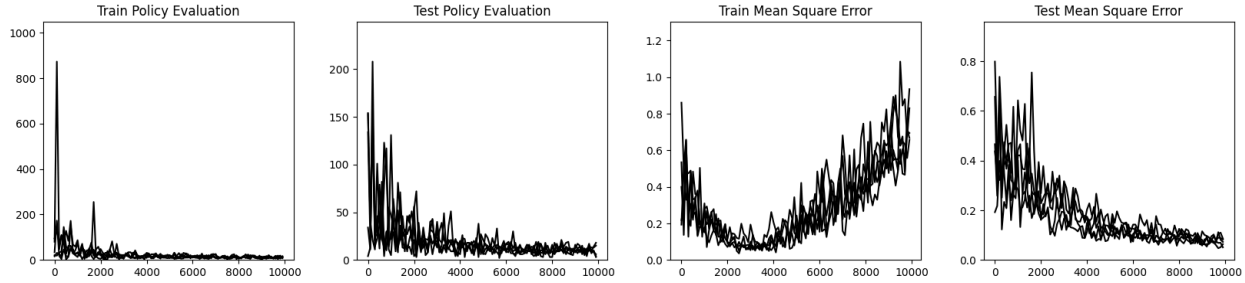
Learning Rate = 0.1, Degree = 2, 0 Diverging Trials



Learning Rate = 0.1, Degree = 3, 0 Diverging Trials

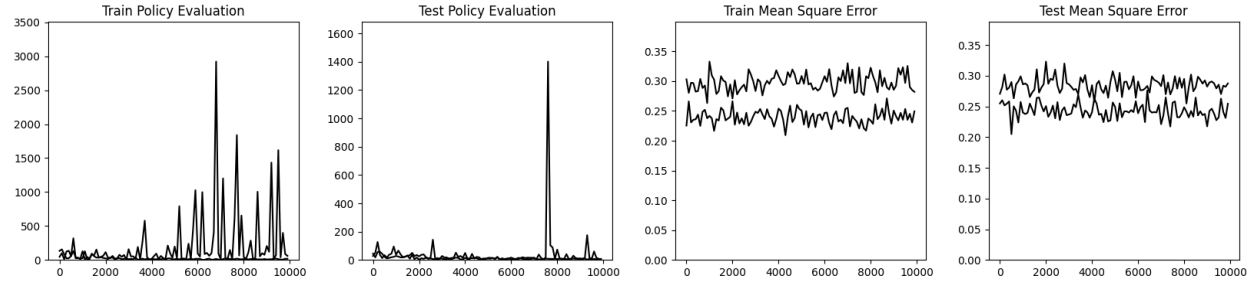


Learning Rate = 0.1, Degree = 4, 0 Diverging Trials

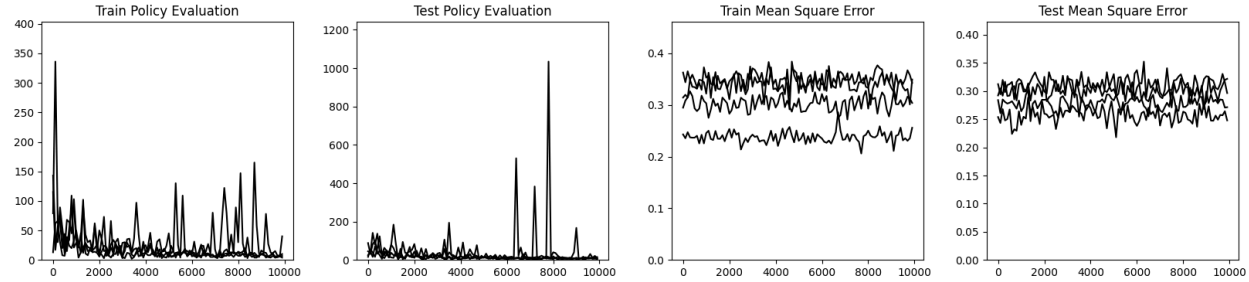


Algorithm: Fish

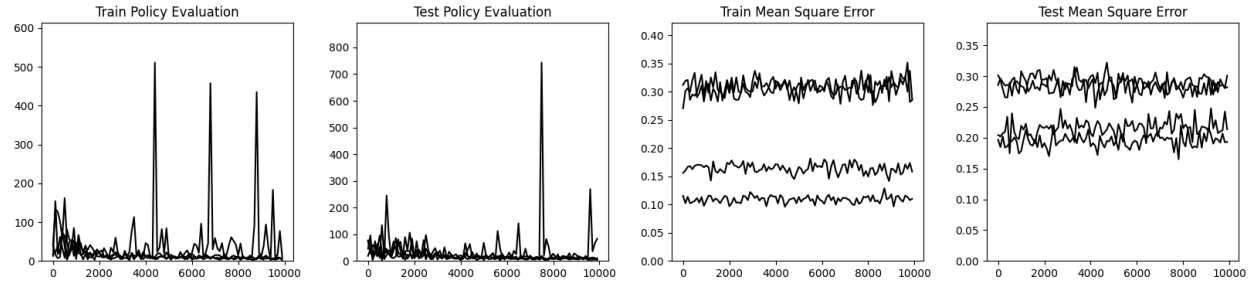
Learning Rate = 0.01, Outer Step Size = 0.001, Degree = 2, 3 Diverging Trials



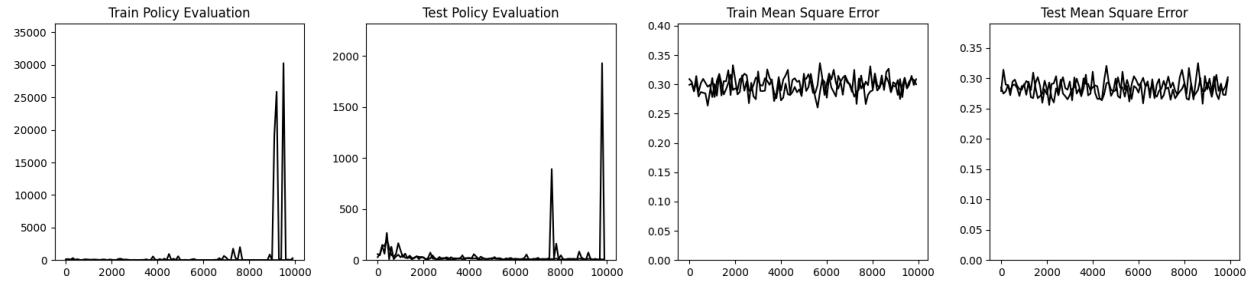
Learning Rate = 0.01, Outer Step Size = 0.001, Degree = 3, 1 Diverging Trial



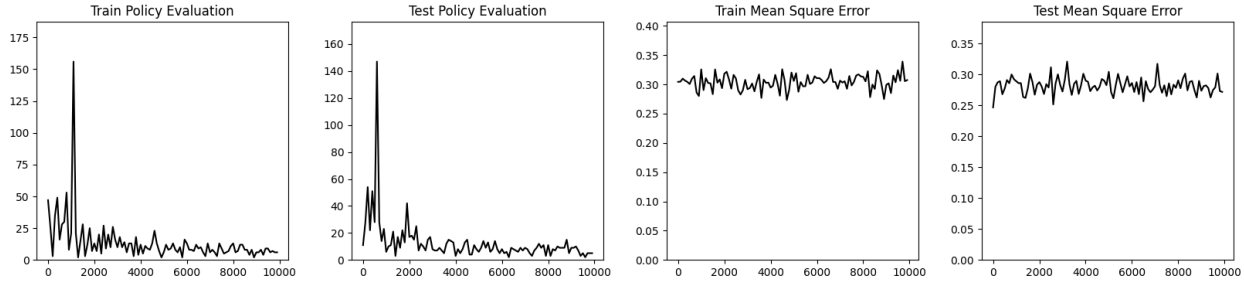
Learning Rate = 0.01, Outer Step Size = 0.001, Degree = 4, 1 Diverging Trial



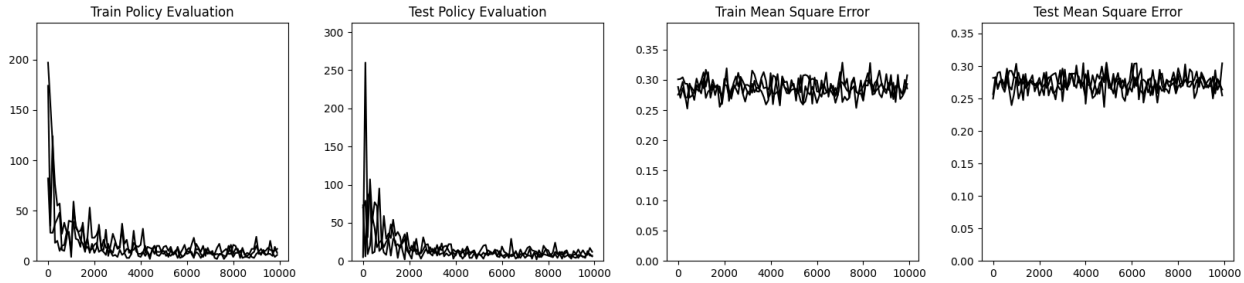
Learning Rate = 0.01, Outer Step Size = 0.005, Degree = 2, 3 Diverging Trials



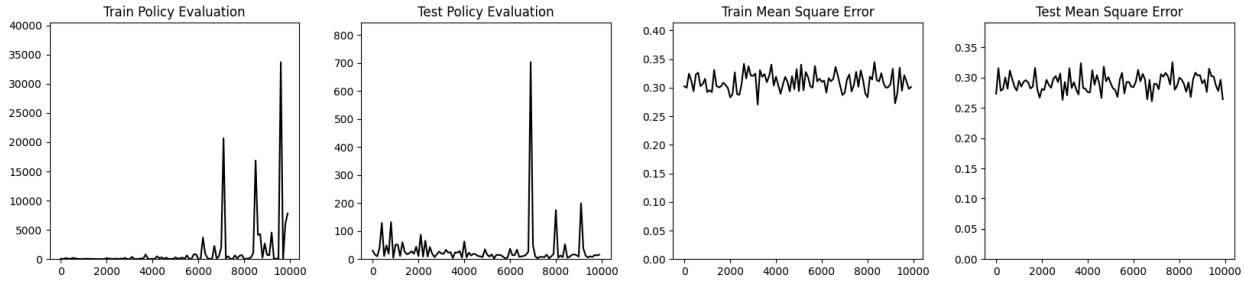
Learning Rate = 0.01, Outer Step Size = 0.005, Degree = 3, 4 Diverging Trials



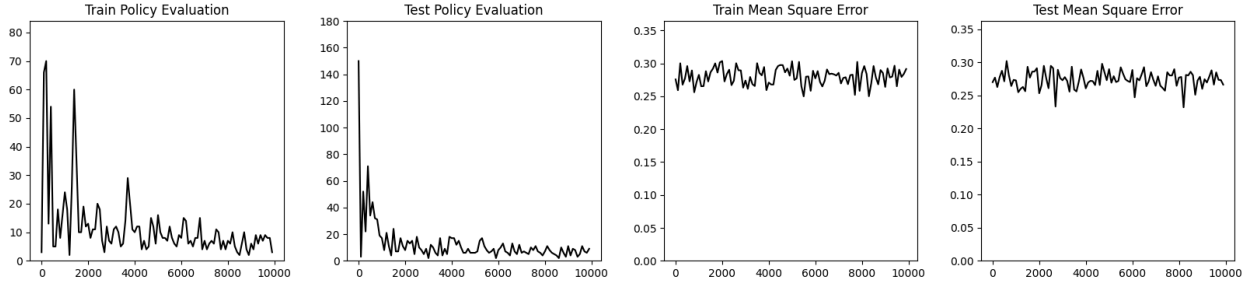
Learning Rate = 0.01, Outer Step Size = 0.005, Degree = 4, 2 Diverging Trials



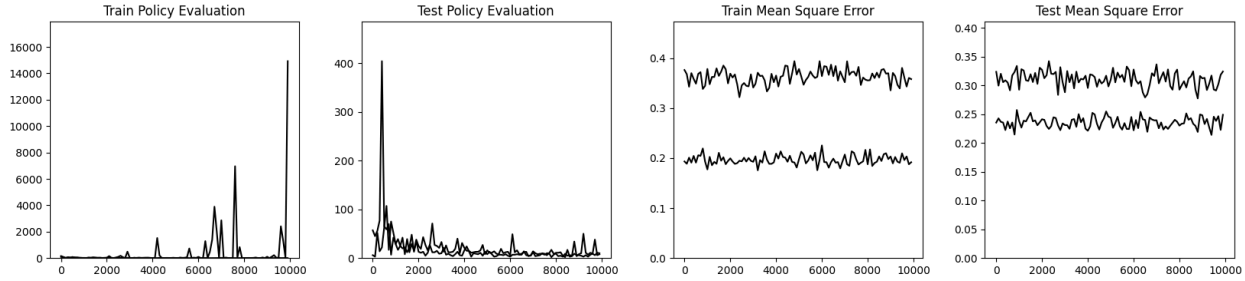
Learning Rate = 0.01, Outer Step Size = 0.01, Degree = 2, 4 Diverging Trials



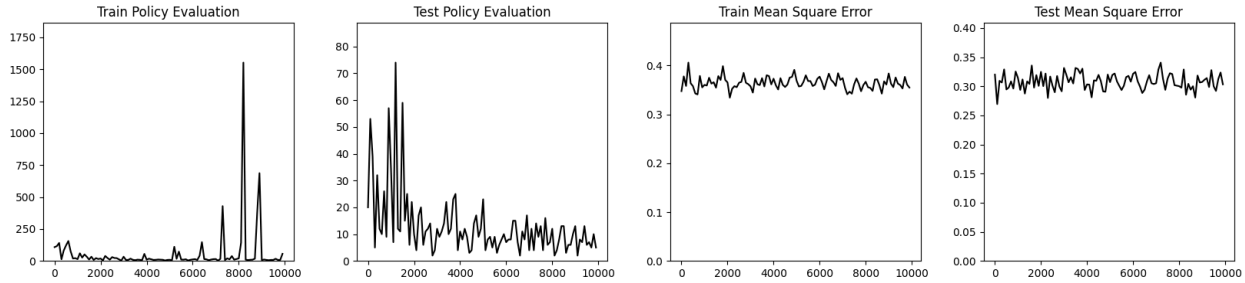
Learning Rate = 0.01, Outer Step Size = 0.01, Degree = 3, 4 Diverging Trials



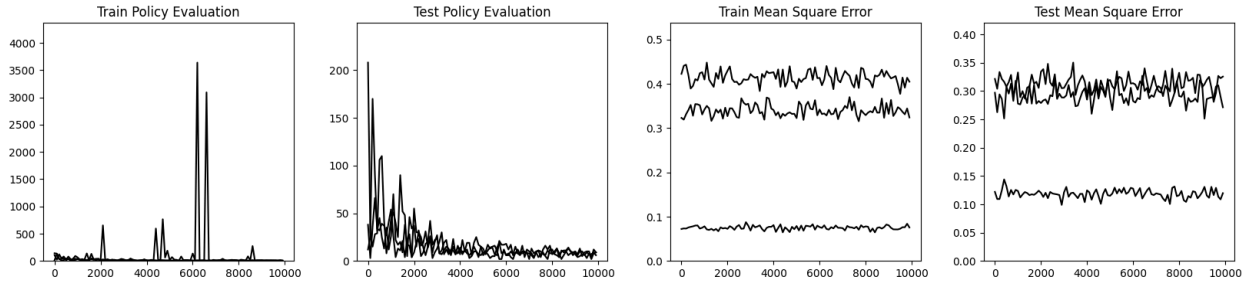
Learning Rate = 0.01, Outer Step Size = 0.01, Degree = 4, 3 Diverging Trials



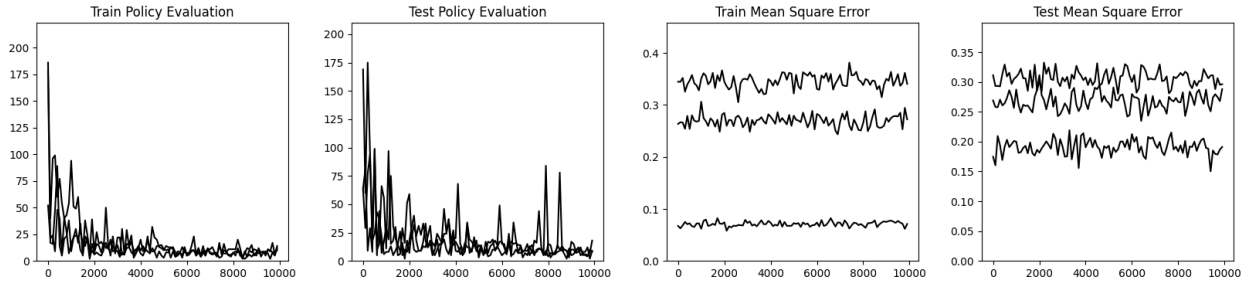
Learning Rate = 0.05, Outer Step Size = 0.001, Degree = 2, 4 Diverging Trials



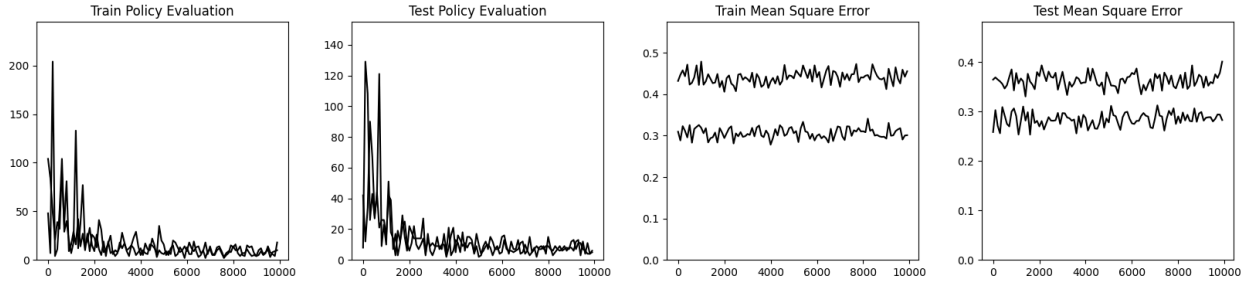
Learning Rate = 0.05, Outer Step Size = 0.001, Degree = 3, 2 Diverging Trials



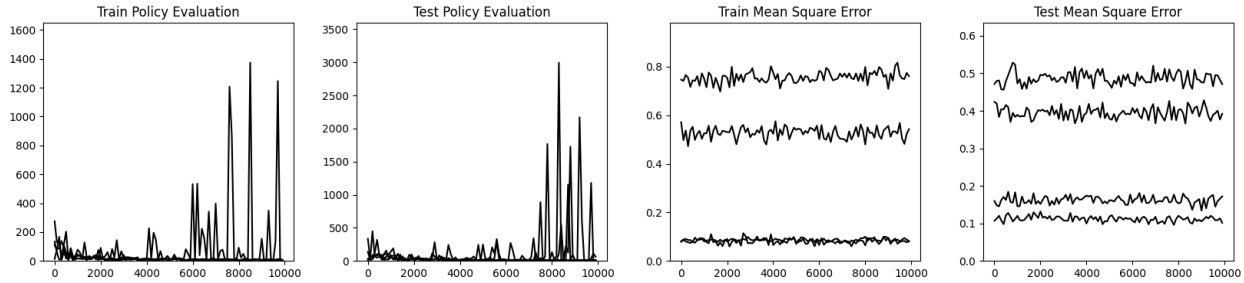
Learning Rate = 0.05, Outer Step Size = 0.001, Degree = 4, 2 Diverging Trials



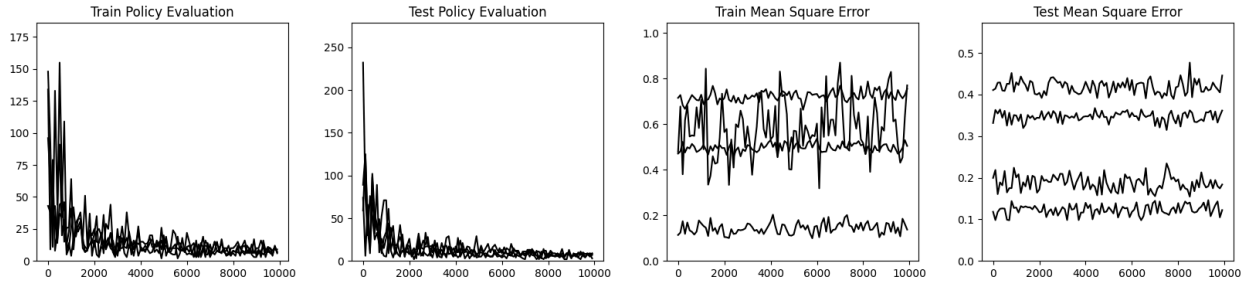
Learning Rate = 0.05, Outer Step Size = 0.005, Degree = 2, 3 Diverging Trials



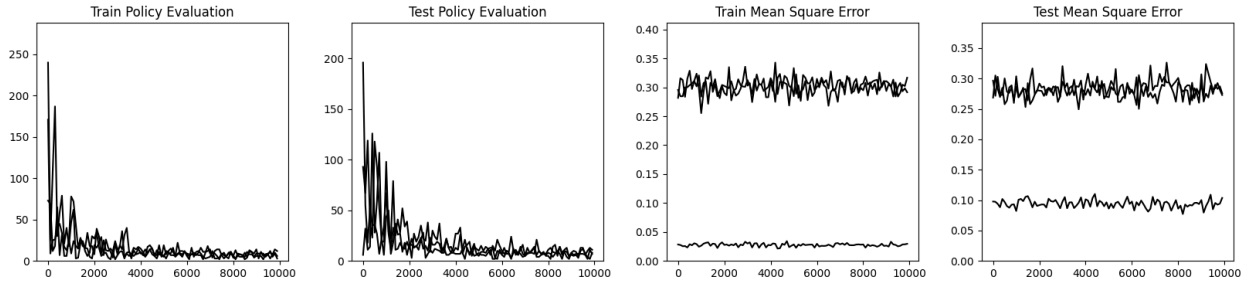
Learning Rate = 0.05, Outer Step Size = 0.005, Degree = 3, 1 Diverging Trial



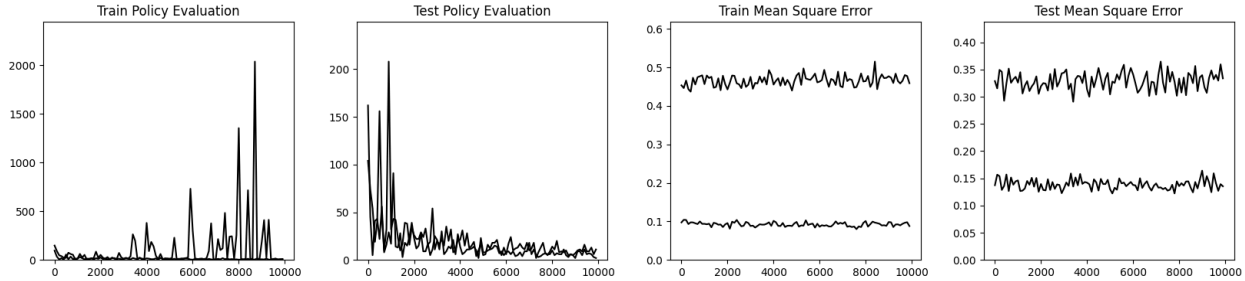
Learning Rate = 0.05, Outer Step Size = 0.005, Degree = 4, 1 Diverging Trial



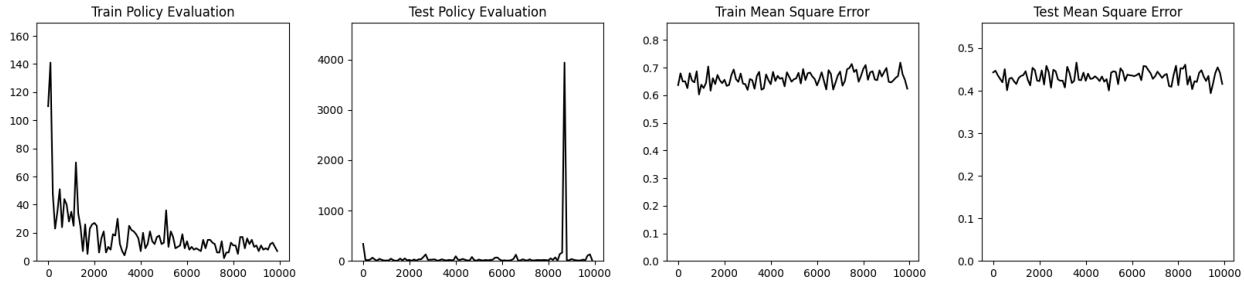
Learning Rate = 0.05, Outer Step Size = 0.01, Degree = 2, 2 Diverging Trials



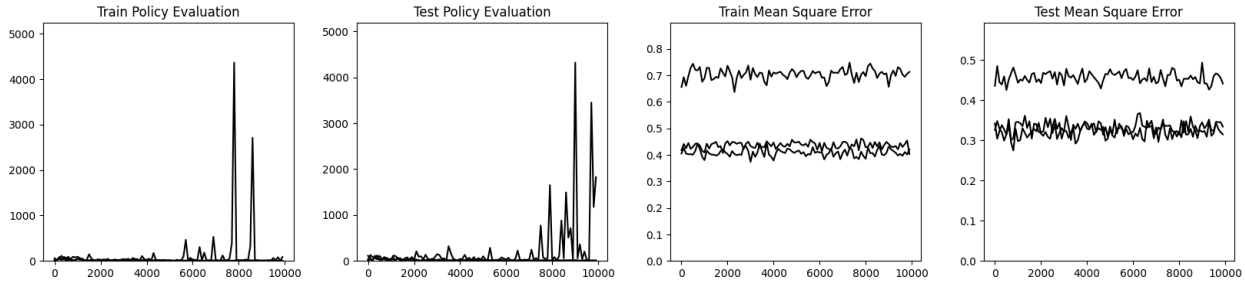
Learning Rate = 0.05, Outer Step Size = 0.01, Degree = 3, 3 Diverging Trials



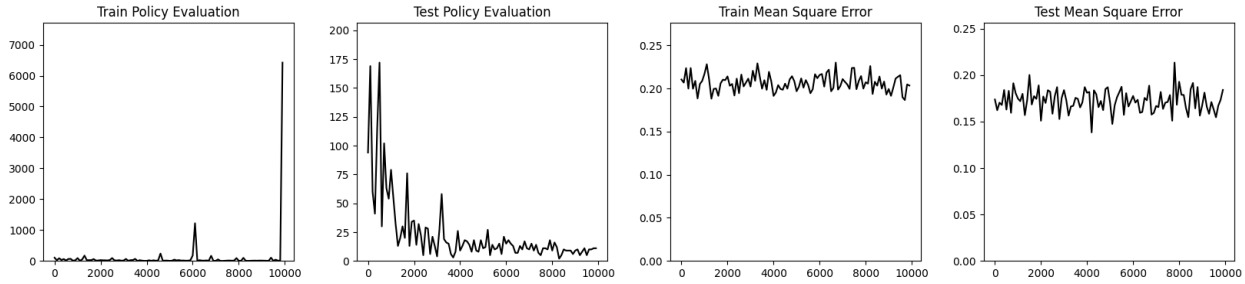
Learning Rate = 0.05, Outer Step Size = 0.01, Degree = 4, 4 Diverging Trials



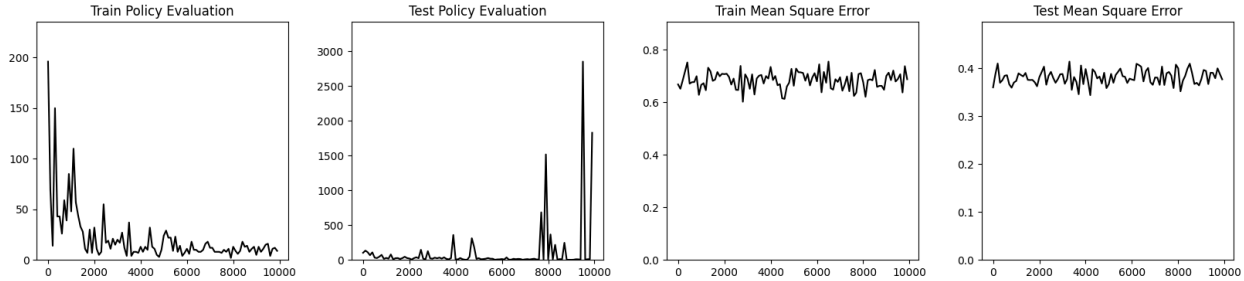
Learning Rate = 0.1, Outer Step Size = 0.001, Degree = 2, 2 Diverging Trials



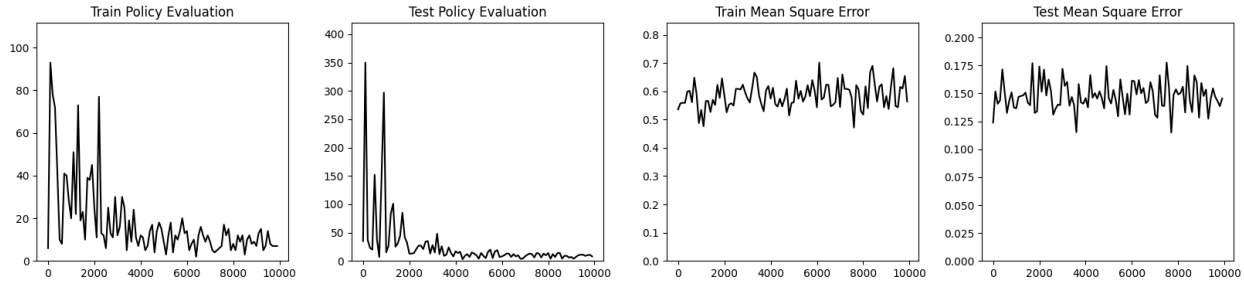
Learning Rate = 0.1, Outer Step Size = 0.001, Degree = 3, 4 Diverging Trials



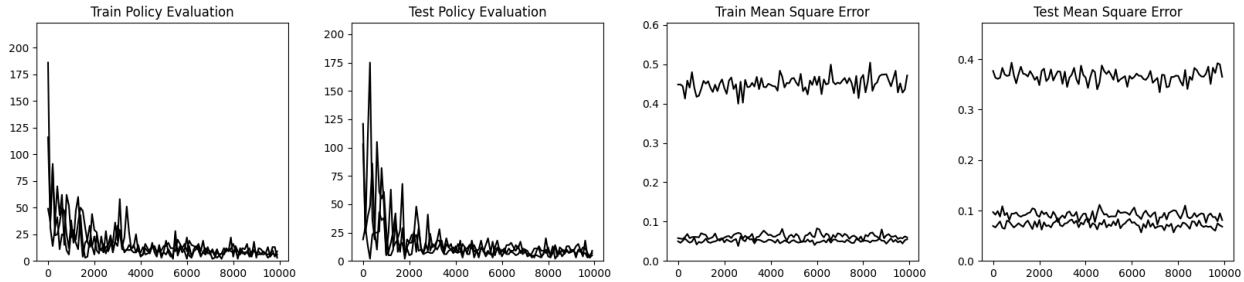
Learning Rate = 0.1, Outer Step Size = 0.001, Degree = 4, 4 Diverging Trials



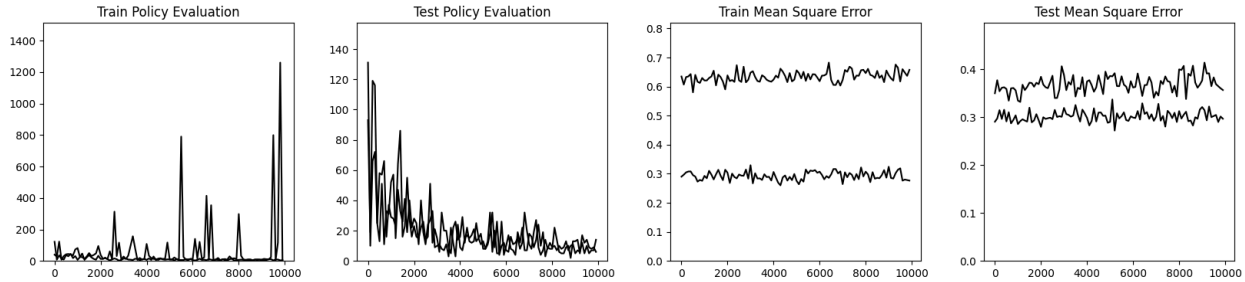
Learning Rate = 0.1, Outer Step Size = 0.005, Degree = 2, 4 Diverging Trials



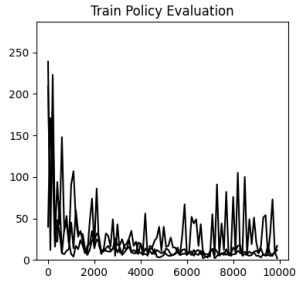
Learning Rate = 0.1, Outer Step Size = 0.005, Degree = 3, 2 Diverging Trials

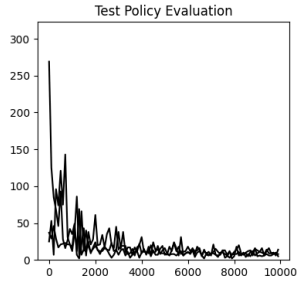


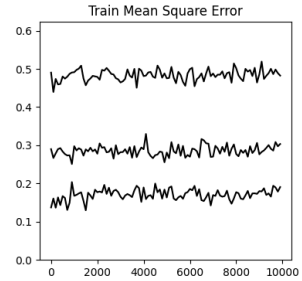
Learning Rate = 0.1, Outer Step Size = 0.005, Degree = 4, 3 Diverging Trials

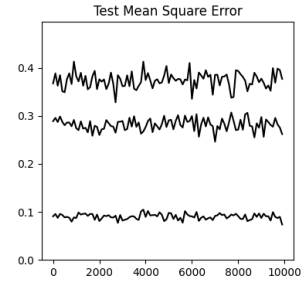


Learning Rate = 0.1, Outer Step Size = 0.01, Degree = 2, 2 Diverging Trials

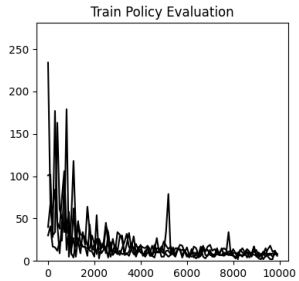


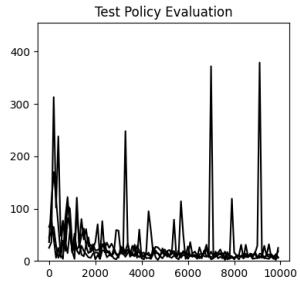


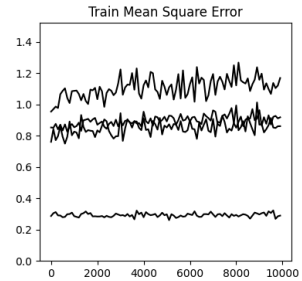




Learning Rate = 0.1, Outer Step Size = 0.01, Degree = 4, 1 Diverging Trial

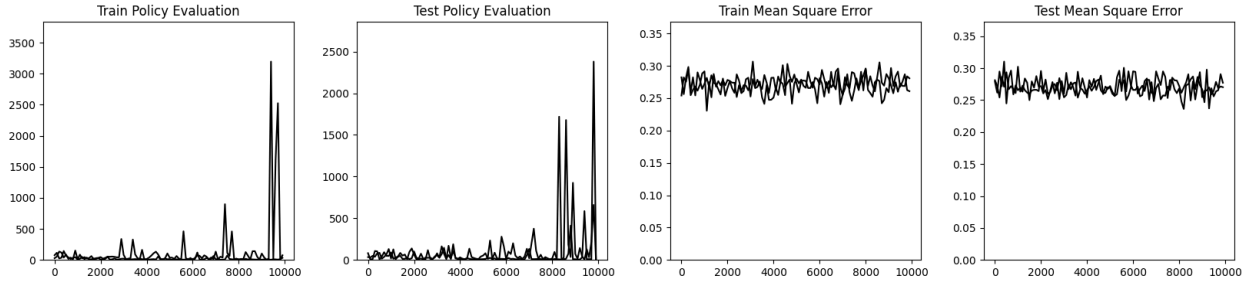




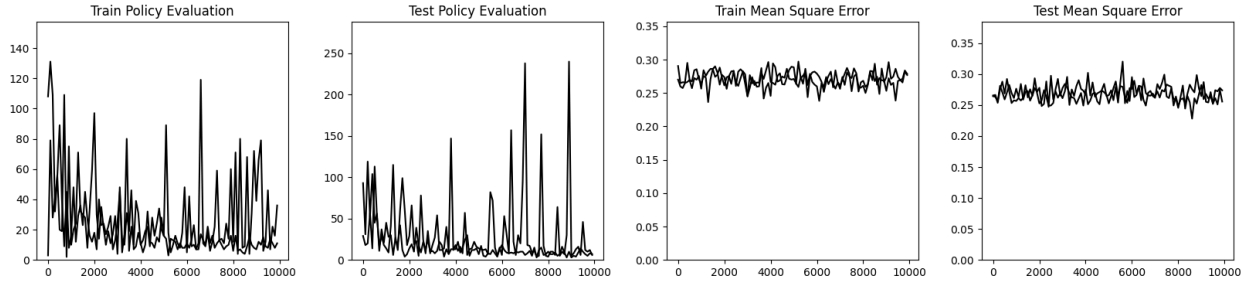


Algorithm: IDGM

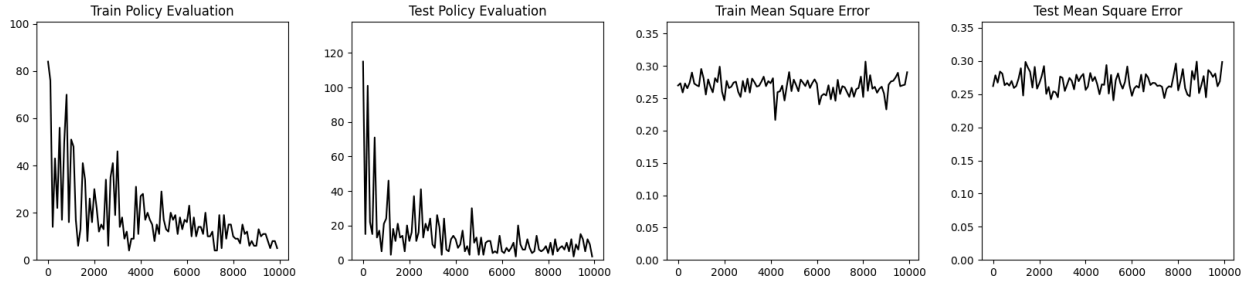
Learning Rate = 0.001, Outer Step Size = 0.0005, Degree = 2, 3 Diverging Trials



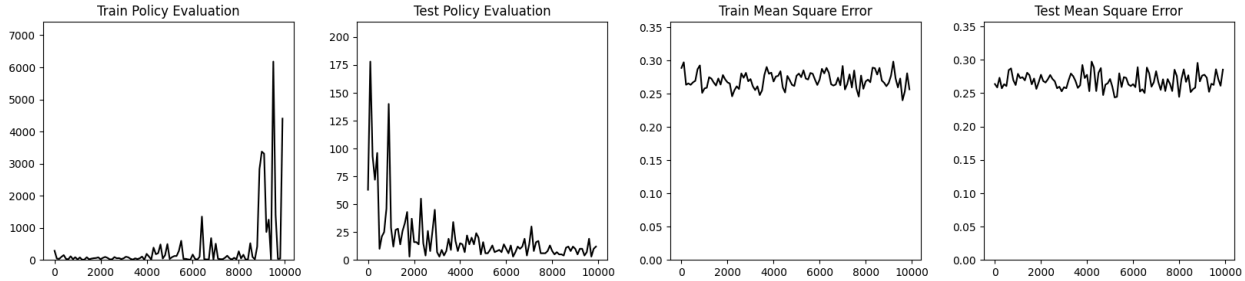
Learning Rate = 0.001, Outer Step Size = 0.0005, Degree = 3, 3 Diverging Trials



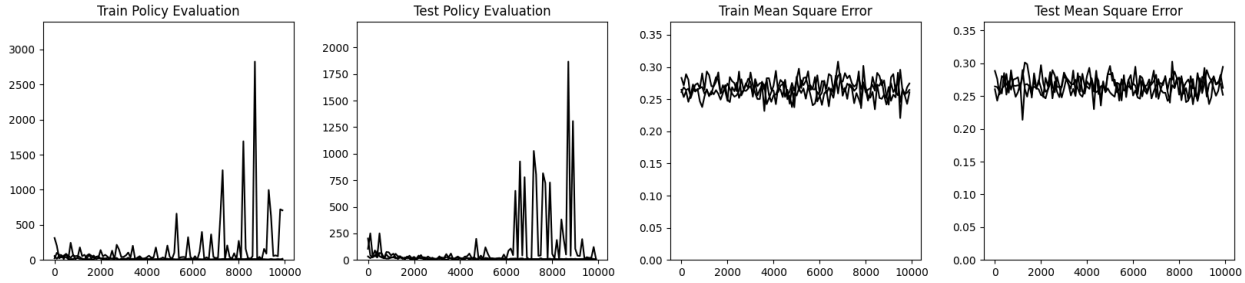
Learning Rate = 0.001, Outer Step Size = 0.0005, Degree = 4, 4 Diverging Trials



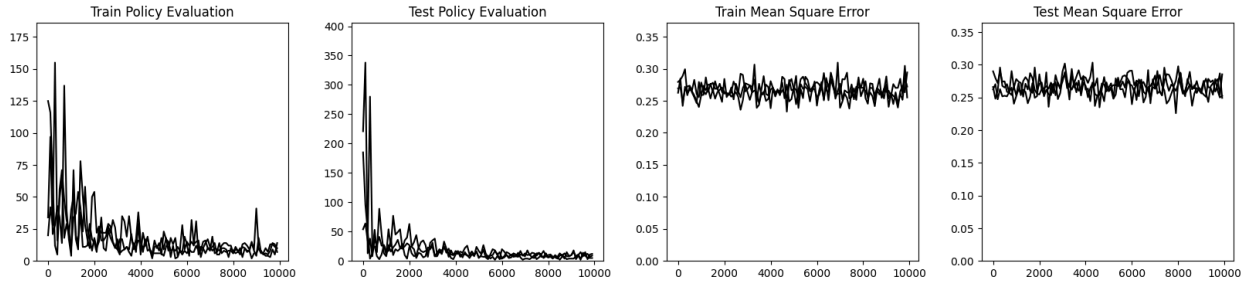
Learning Rate = 0.001, Outer Step Size = 0.001, Degree = 2, 4 Diverging Trials



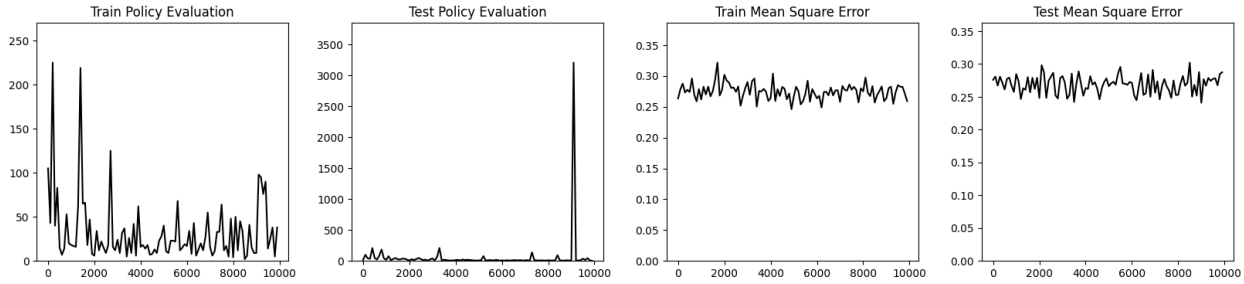
Learning Rate = 0.001, Outer Step Size = 0.001, Degree = 3, 2 Diverging Trials



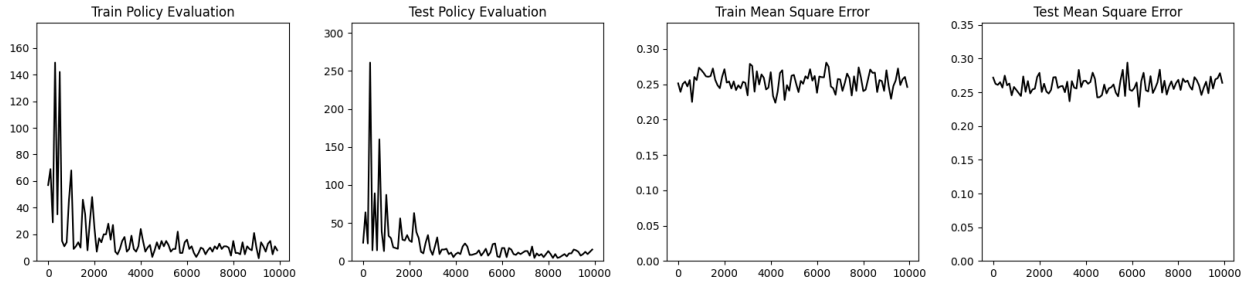
Learning Rate = 0.001, Outer Step Size = 0.001, Degree = 4, 2 Diverging Trials



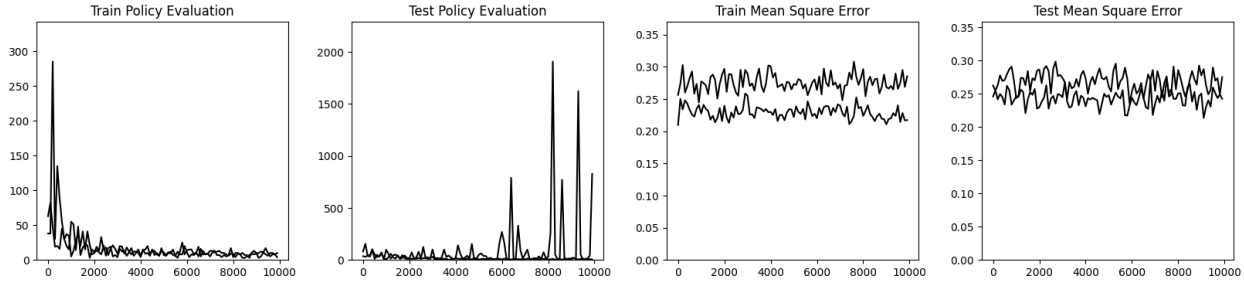
Learning Rate = 0.001, Outer Step Size = 0.005, Degree = 2, 4 Diverging Trials



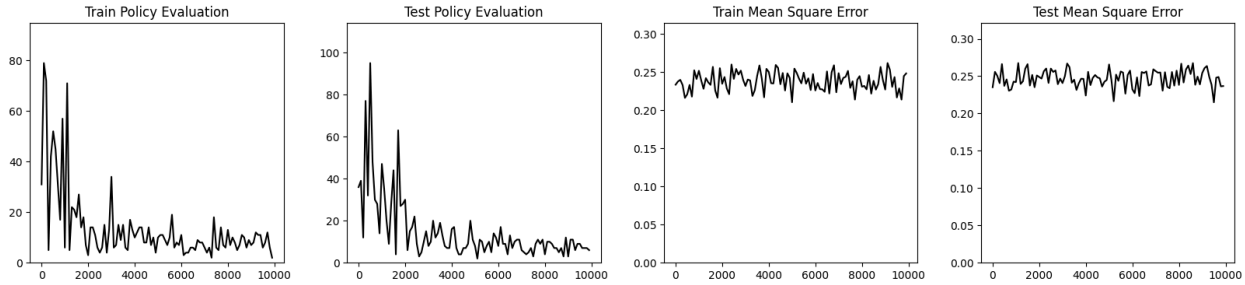
Learning Rate = 0.001, Outer Step Size = 0.005, Degree = 3, 4 Diverging Trials



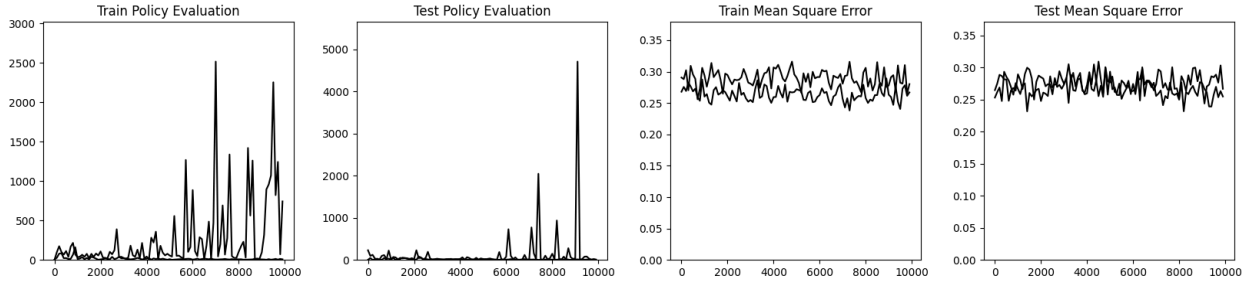
Learning Rate = 0.001, Outer Step Size = 0.005, Degree = 4, 3 Diverging Trials



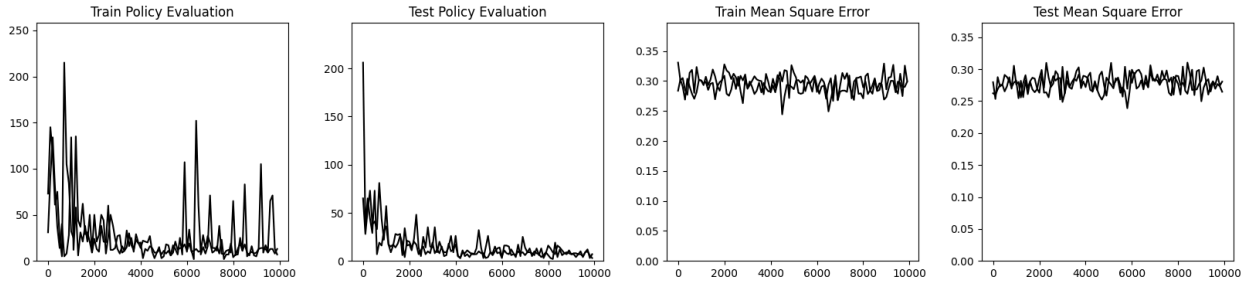
Learning Rate = 0.005, Outer Step Size = 0.0005, Degree = 2, 4 Diverging Trials



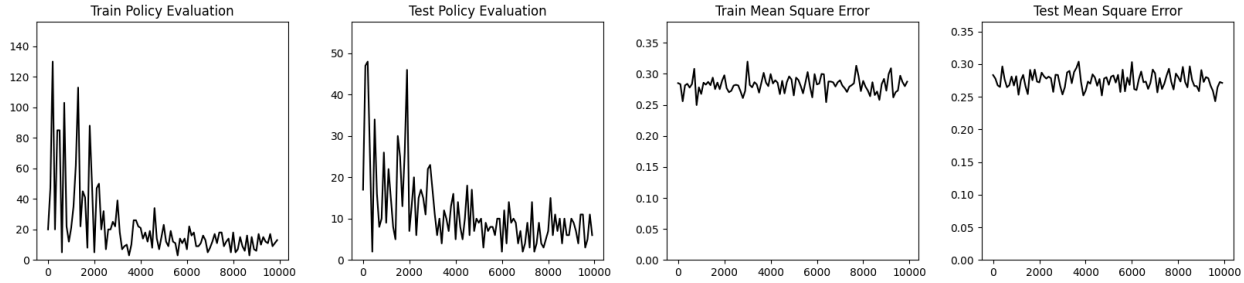
Learning Rate = 0.005, Outer Step Size = 0.0005, Degree = 3, 3 Diverging Trials



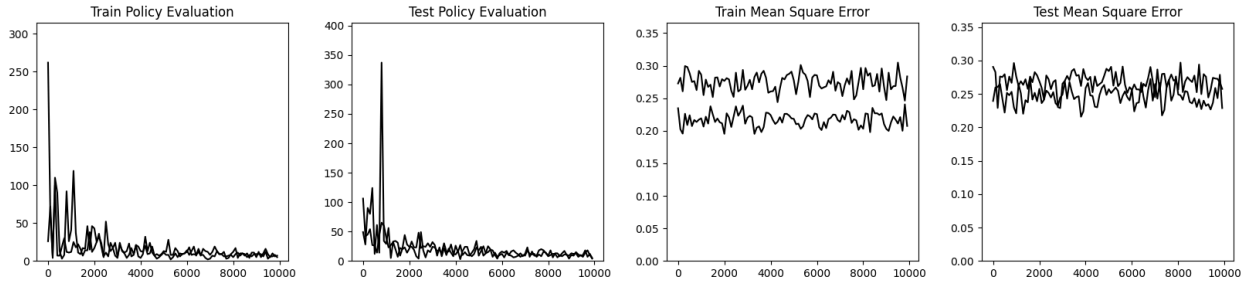
Learning Rate = 0.005, Outer Step Size = 0.0005, Degree = 4, 3 Diverging Trials



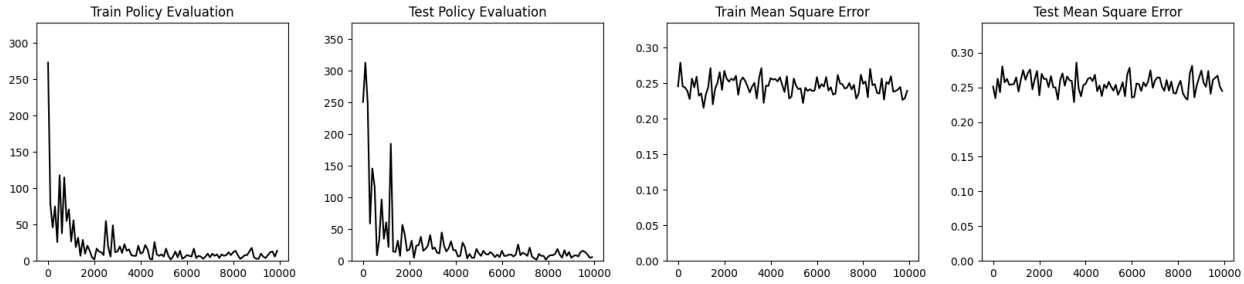
Learning Rate = 0.005, Outer Step Size = 0.001, Degree = 2, 4 Diverging Trials



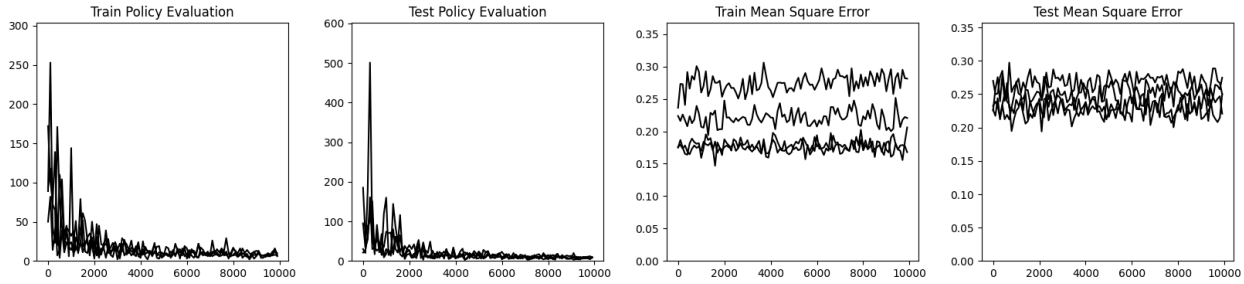
Learning Rate = 0.005, Outer Step Size = 0.001, Degree = 3, 3 Diverging Trials



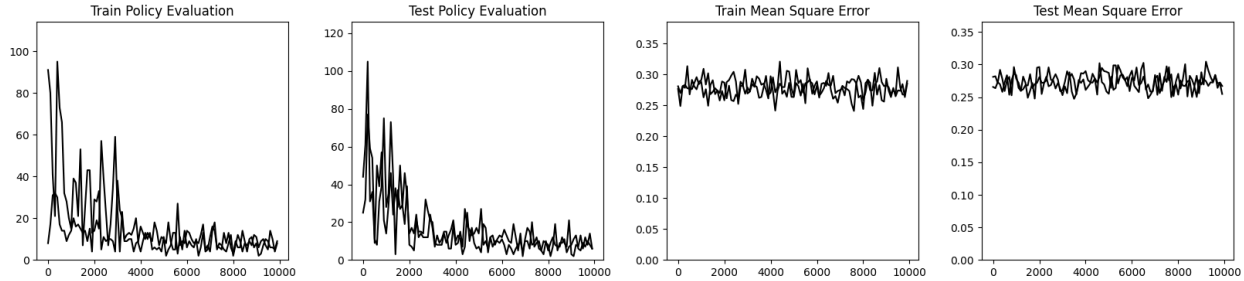
Learning Rate = 0.005, Outer Step Size = 0.005, Degree = 2, 4 Diverging Trials



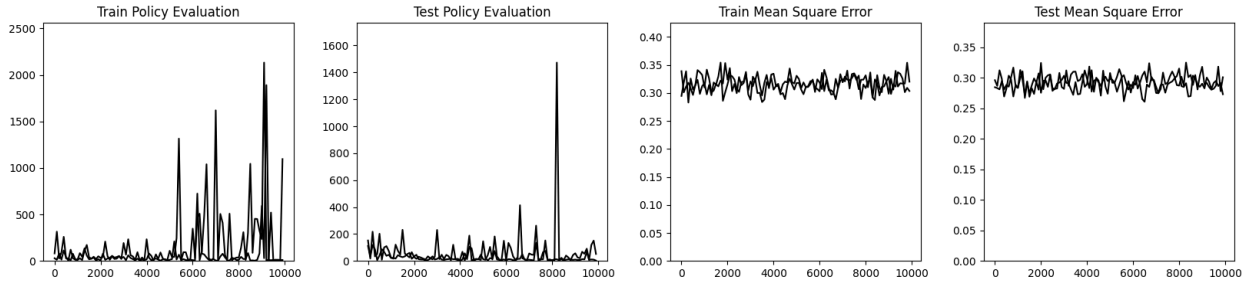
Learning Rate = 0.005, Outer Step Size = 0.005, Degree = 4, 1 Diverging Trial



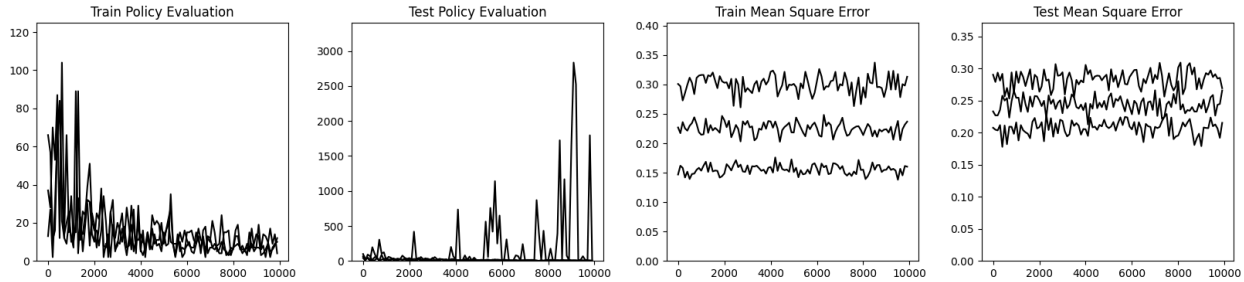
Learning Rate = 0.01, Outer Step Size = 0.0005, Degree = 2, 3 Diverging Trials



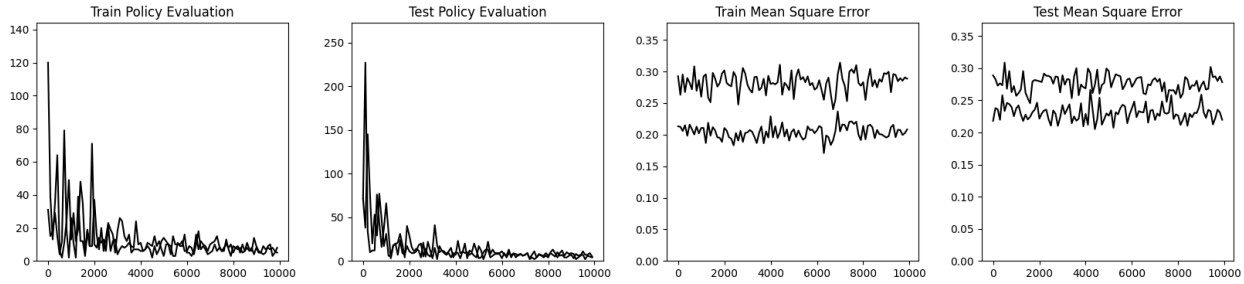
Learning Rate = 0.01, Outer Step Size = 0.0005, Degree = 3, 3 Diverging Trials



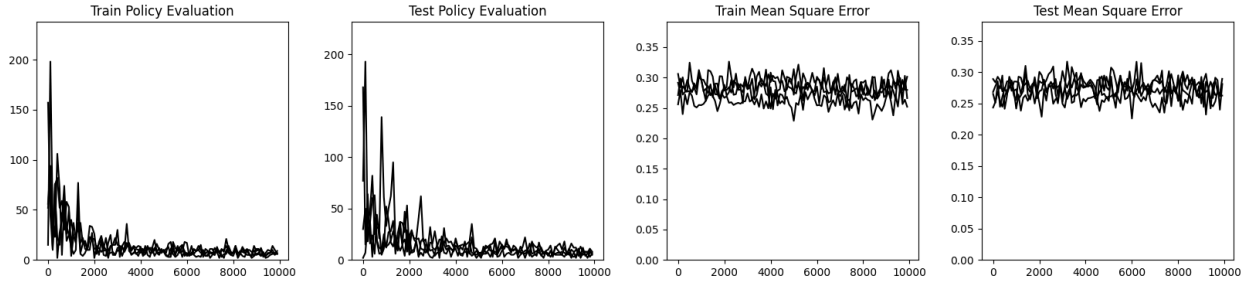
Learning Rate = 0.01, Outer Step Size = 0.001, Degree = 2, 2 Diverging Trials



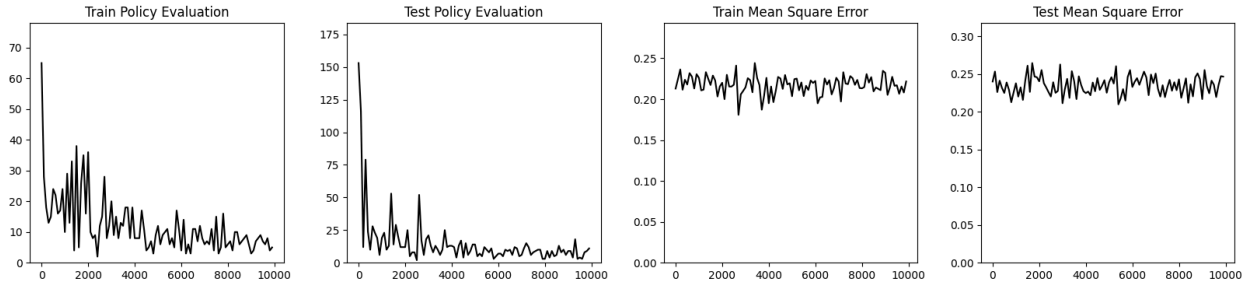
Learning Rate = 0.01, Outer Step Size = 0.001, Degree = 3, 3 Diverging Trials



Learning Rate = 0.01, Outer Step Size = 0.005, Degree = 2, 1 Diverging Trial



Learning Rate = 0.01, Outer Step Size = 0.005, Degree = 3, 4 Diverging Trials



Learning Rate = 0.01, Outer Step Size = 0.005, Degree = 4, 3 Diverging Trials