ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

LONG-HORIZON VALUE GRADIENT METHODS ON STIEFEL MANIFOLD

M.Sc. THESIS

Tolga Ok

Department of Computer Engineering

Computer Engineering Programme

DECEMBER 2022

<u>**ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL**</u>

LONG-HORIZON VALUE GRADIENT METHODS ON STIEFEL MANIFOLD

M.Sc. THESIS

Tolga Ok
(504181535)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Assoc. Prof. Nazım Kemal ÜRE

DECEMBER 2022

Tolga Ok, a M.Sc. student of ITU Graduate School student ID 504181535 successfully defended the thesis entitled "LONG-HORIZON VALUE GRADIENT METHODS ON STIEFEL MANIFOLD", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Nazım Kemal ÜRE** Istanbul Technical University	....
Jury Members :	**Assoc. Prof. Yusuf YASLAN** İstanbul Technical University	....
	Adjunct Prof. Volkan VURAL University of California San Diego	....
		....

Date of Submission : **29 December 2022**
Date of Defense : **29 December 2022**

To my family and friends,

FOREWORD

I am greatly appreciative of the support and goodwill of my family and friends, and I want to express my particular thanks to my advisor Professor Nazım Kemal Üre who has been incredibly supportive of me during my master studies and whose guidance contributed to the completion of this thesis.

December 2022 Tolga Ok

TABLE OF CONTENTS

ABBREVIATIONS

TD	: Temporal Difference
NN	: Neural Network
RL	: Reinforcement Learning
GPI	: Generalized Policy Iteration
PG	: Policy Gradient
PPO	: Proximal Policy Optimization
GAE	: Generalized Advantage Estimation
MBLR	: Model Based Reinforcement Learning
VG	: Value Gradient
AC	: Actor Critic
St	: Stifel manifold
DPG	: Deterministic Policy Gradient
VG-PPO	: Value Gradient Proximal Policy Optimization
SAC	: Soft Actor Critic
SVG	: Stochastic Value Gradient
SAC-SVG	: Soft Actor Critic Stochastic Value Gradient
GRU	: Gated Recurrent Unit
MBPO	: Model Based Policy Optimization
DDPPO	: Directional Derivative Projection Policy Optimization
NLP	: Natural Language Processing
DT	: Decision Transformer
RNN	: Recurrent Neural Network
RGD	: Riemanninan Gradient Descent
MC	: Monte Carlo
KL	: Kullback–Leibler
TRPO	: Trust Region Policy Optimization

SYMBOLS

γ	: Discount factor
λ	: Trace delay parameter
$\mathbb{T}$, $\mathcal{D}$	: Rollout and transition buffer
n	: Rollout length
s, a, r, s'	: State, action, reward, and next state
$r(s, a, s')$, p	: Reward function and transition distribution
d^{π}, p^{π}	: Policy aware stationary state and transition distributions
$\pi(a\|s), \mu_{\theta}(s)$	: Policy distribution and deterministic policy function
$V^{\pi}(s)$	: State value function
$Q^{\pi}(s, a)$	: State-action value function
$A^{\pi}(s, a)$	: Advantage function
$G^{\pi}(s)$	: Policy return
$v_{\phi}(s)$	: State value approximation
$\delta^{\pi}(s)$	: Temporal difference

LIST OF FIGURES

LONG-HORIZON VALUE GRADIENT METHODS ON STIEFEL MANIFOLD

SUMMARY

Sequential decision-making algorithms play an essential role in building autonomous and intelligent systems. In this direction, one of the most prominent research fields is Reinforcement Learning (RL). The long-term dependencies between the actions performed by a learning agent and the rewards returned by the environment introduce a challenge in RL algorithms. One of the ways of overcoming this challenge is the introduction of the value function approximation. This allows policy optimization in RL algorithms to rely on immediate state value approximation and simplifies policy learning. In practice, however, we use value approximations that involve future rewards and values from the truncated future trajectories with a decaying weighting scheme, such as in TD(λ), to make a better trade between the bias and variance of the value estimator.

Policy Gradients (PG), a prominent approach in model-free paradigm, rely on the correlation between past actions and the future rewards within truncated trajectories to form a gradient estimator of the objective function with respect to the policy parameters. However, as the length of the truncated trajectories increases or the λ parameter approaches 1, akin to the use of Monte Carlo (MC) sampling, the variance in the gradient estimator of the PG objective increases drastically. Although the gradient estimator in PG methods has zero bias, the increase in variance leads to sample inefficiency, since the approximated value of an action may contain future rewards within the truncated trajectory that are not related to that action.

One of the alternatives to the PG algorithms that do not introduce high variance during policy optimization is the Value Gradient (VG) algorithm which utilizes the functional relation between the past actions and the future state values on a trajectory. This estimation requires a differentiable model function; hence, VG algorithms are known as model-based Reinforcement Learning (MBLR) approaches. Although it is possible to apply the VG objective on simulated sub-trajectories, as is the case for most MBLR approaches, the most effective approach is to apply the VG objective on observed trajectories via the means of reparameterization. If the observed trajectories are used, VG algorithms avoid the issue of compounding errors that occur when the model function is called iteratively with its previous prediction to simulate future trajectories. However, reparameterization does not solve the compounding error problem during the backward pass if the model being used by the VG objective is approximated by an unconstrained function.

In this thesis, we propose a model function represented by a neural network where the parameters of the affine layers lie on Stiefel manifold (St) such that the Jacobian

matrices of the affine layers are unitary matrices at every state-action pair. During the model optimization step, we employ Riemannian Gradient Descent (RGD) through parameterization with a matrix exponential function as the submersion to maintain the unitary characteristics of the parameters. We combine the affine layers with non-linearities that preserve the gradient norm so that the VG objective does not suffer from vanishing or exploding gradient issues. This approach differs from previously proposed techniques that aim to relax the vanishing or exploding gradient problem, which tends to make the optimization unstable for longer trajectories by providing a direct and mathematically grounded solution.

We built a benchmarking environment where the actions are deliberately set to contain only long-term relations with the future state values. Here, we show that the proposed model function surpasses the previously applied techniques during policy optimization with VG objective on long trajectories. We further propose the VG-PPO algorithm, which combines Proximal Policy Optimization (PPO) with the VG approach that is equipped with the proposed family of model functions. We apply VG-PPO on the MuJoCo benchmark and obtained comparable results to that of PPO even when the value approximation is performed by the MC method over long trajectories. We conclude that the VG methods are sensitive to the bias in the model approximation, yet, they provide a low variance gradient estimator for policy optimization when combined with a suitable model family. Moreover, we show that VG methods are able to capture long term dependencies if the underlying model function is not divergent such as we proposed in this work.

1. INTRODUCTION

Policy optimization in sequential decision-making problems involves decision-making over long horizons. The given decisions may have a long-lasting effect on the dynamics of the system. In Reinforcement Learning, we formalize such systems with Markov Decision Processes (MDP) and define a value function $V^{\pi}(S)$ that captures the future effect of a decision given by a policy $\pi(S)$. In general, the optimization algorithms are grouped by Generalized Policy Iteration (GPI) [1], where the value function approximation is followed by a policy update with the most recent approximate values in an alternating fashion. The value function can be estimated using immediate reward and values via the Bellman equation, which requires a value approximation or mapping, or using sampled future rewards by a Monte Carlo estimator [1]. These approaches represent the two ends of the trade between bias and variance within the value estimation. In practice, it has been observed that the combinations of these estimations, such as in TD(λ), result in higher performance since the hybrid approaches bring more control over this trade [2].

Value estimation is essential for building a policy function. Although it is possible to form a policy function by directly using the value estimations if there are finitely many decisions that can be made at a given state, the explicit representation of the policy allows finer policy optimization and makes it applicable to more than discrete action spaces. In that regard, Policy Gradient (PG) is one of the most prominent model-free approaches that explicitly form a policy function generally as a Neural Network (NN). The first algorithm of this family is REINFORCE [3], in which the value estimation is performed by a Monte Carlo estimator. The policy gradient estimator of REINFORCE is bias-free since the value function is not approximated and estimated by future reward samples using Monte Carlo. The policy optimization in PG methods relies on the correlation between the actions and future values, which may include sampled rewards from the future part of the trajectory. Due to that, the policy optimization objective of PG methods does not require a known or an approximated model function of the environment; hence PG methods are called model-free. However, the correlation-based gradient estimator of the policy optimization objective has high variance, which results in sample inefficiency.

One way of reducing the variance brought by the policy gradient estimator of the PG method is to introduce a value approximation that bootstraps to itself after n-steps or a weighted sum of multiple value estimators. The latter approach is formalized in Generalized Advantage Estimator (GAE)(λ) [4], which is similar to TD(λ) but implements it for truncated trajectories. PG methods that employ a value approximation are called Actor-Critic (AC) algorithms, where the actor stands for the policy function, and the critic stands for the approximate value function. As the truncation increases or the parameter λ decreases, which controls the amount of bootstrapping in the value estimation, the variance in the gradient estimator decreases. However, in practice, high values of λ and longer truncated trajectories yield higher performance [5]. In the truncated part of the trajectory, if the parameter λ is close to 1, the value estimate approaches the Monte Carlo value estimation, and as a result, AC algorithms suffer from high variance in the policy gradient estimator.

In this thesis, we focus on Value Gradient (VG) methods, which offer an alternative objective to that of PG methods that rely on the functional relation between past actions and future values on a trajectory instead of relying on their correlation. The functional relation is formed using an approximate or a given differentiable model function of the environment. Hence, this is a Model-Based Reinforcement Learning (MBLR) approach. Having a model function provides a low variance policy gradient estimator but may introduce bias to the estimator if the model function is approximated, which is usually the case in practice. Besides the trade-off between the bias and variance on the value estimation introduced by value approximation, VG methods introduce another bias-variance trade to the policy gradient estimator. While value approximation provides a low variance target for policy optimization, VG provides variance reduction within the policy optimization via the use of a model function or its approximation. Hence, these two approaches, value approximation and VG objective are compatible and can be used in combination to further reduce the variance of the policy gradient estimator.

There are various approaches for utilizing a model function in MBLR methods. The most noticeable one is to simulate artificial interactions between the agent and the approximate model function of an environment and use a model-free policy gradient algorithms such as Asynchronous Advantage Actor Critic (A3C) [6], PPO [5] to optimize the policy over these artificial interactions [7]–[9]. However, in practice, this approach is generally difficult to work with due to the compounding error issue. In which the approximation error of the model function at every step accumulates and grows exponentially, and resulting simulated trajectories become improbable, or the growing approximation error may cause numerical instabilities. Combining the approximate model function with the value approximation on short sequences is thus the preferred approach instead of simulating a trajectory starting from a given state until the terminal state is simulated. Another improvement, proposed by [7], in this direction, runs the approximate model on a latent space that only preserves and encodes the information related to the policy optimization. Until recently, MBLR methods that utilize the model function for simulating artificial trajectories have only been successfully applied to a small set of challenging environments such as Go [10] and Chess [11], and generally lacked the performance of the pure model-free approaches. Recently, there have been successful applications of such approaches [7,8] by combining the improvements proposed for MBLR and with powerful architectures on common RL benchmarks, namely ATARI [12] and MujoCo [13].

Alternatively, there are several approaches in MBLR that aims to improve the value approximation and policy optimization by introducing auxiliary loss functions to value and policy objectives. In these approaches, the NNs that represent value approximation and the policy function share lower layers, usually all the layers except the final ones, and the auxiliary losses contribute to the optimization of these layers in an unsupervised manner [14]. This approach provides less noisy gradients to the shared layers, but the direction in which the auxiliary objective pulls the parameters may not be optimal for policy optimization. Similar approaches exist that utilize the model function to improve the exploration aspect of the policy learning by driving the exploration so that less visited states would get high priority [15]. Overall, these approaches are most advantageous when the reward function is extremely sparse such that the first encounter of a non-zero reward requires long exploration sessions. In that case, these approaches provide better guidance to the policy than random gradients from the policy gradient estimator.

The focus of this thesis, VG methods, forms a computational graph over a truncated trajectory using the differentiable model, policy, and reward functions. Since the model and reward functions are usually unknown, we build differentiable approximations for them using NNs. On this computational graph, the objective is to maximize the value estimate at every state, which propagates the gradients backward through the truncated trajectory. Hence, unlike previously mentioned MBRL approaches, VG methods utilize the model functions to provide a policy gradient estimate that has less variance compared to that of PG. In order to form a computational graph using the approximate model function, the next state prediction of the model function must be fed as the state variable in the next iteration. Therefore, the computation graph to which the VG objective is applied is constrained to be formed over the simulated trajectories. However, the VG objective that is built on top of the simulated trajectories would suffer from the compounding error problem in the forward pass. To mitigate that issue, Heess et al. [16] proposed the Stochastic Value Gradient (SVG) algorithm, which enables VG objective to be formed over sampled truncated trajectories. The main mechanism in SVG that allows using the sampled transitions instead of simulated ones is reparameterization. SVG forms a computational graph on sampled trajectories, and since, at every state, reparameterization corrects the approximation error of the model function, it eliminates the compounding error problem in the forward pass. In addition, SVG provides a spectrum of policy gradient estimators that varies with the truncation of the sampled trajectories. At the one end of this spectrum, there is VG objective that is built on immediate value and action, which covers Deterministic Policy Gradient (DPG) [17]–[20] algorithms. DPG algorithms are unique in this spectrum as they do not require a model function; hence they are categorized as model-free approaches since the gradient of the state-action value function with respect to immediate action does not involve the model function. The other end of this spectrum is called SVG(∞), in which the gradient of a state value propagates through the entire trajectory backward in time.

Although SVG solves the compounding error problem in the forward pass, during the backward pass of the computational graph, in which the VG objective is formed over, the gradient of the state values tends to explode or diminish. There have been several proposals for the model function, such as using independent features for the state prediction or using gated architectures to mitigate this issue [16,21]. But they have failed to solve this issue, and hence, in practice, SVG algorithms are only used on short truncated trajectories. This thesis proposes a particular family of model functions that preserves the norm of the gradient during the backward pass.

We empirically show that when the aforementioned property is satisfied, VG methods are able to capture the long relations between actions and states. Hence, the proposed family of model functions enables VG methods on a variety of value estimators. The next subsection gives a more detailed description of the contributions made within this work.

1.1 CONTRIBUTIONS

There are two major contributions in this thesis. We briefly summarize them in the list below.

- A family of model functions that enables VG methods on long horizons.
- VG-PPO algorithm that implements the VG objective in Proximal Policy Optimization

The main contribution of this work aims to improve the VG algorithms by providing a family of model functions and its optimization procedure that solves the gradient exploding and vanishing issue of SVG methods. We observed that when the gradient of a state value is preserved, the policy gradient estimate of the VG objective successfully produces a low variance gradient estimation for the policy parameters. In fact, in the case where the model approximation error is significantly low, the estimated gradients are directly related to how much a past action affects the state value of which the estimation is made.

The family of model functions proposed by this thesis has the gradient norm-preserving property. We propose to use NNs as the model function, in which the parameters of the affine layers are orthonormal k-frames. Between the affine layers, we use non-linearities that also have the gradient norm-preserving property or that make a slight change in the norm. Hence, the overall architecture preserves this property. During the optimization steps, we maintain the gradient norm-preserving characteristic of the proposed model function by mapping the updated parameters onto the set of orthonormal k-frames via a proper mapping function. The set of orthonormal k-frames is represented by the Stiefel manifold (SO). Hence, we adopt Riemannian Gradient Descent (RGD) via parameterization instead of regular Gradient Descent (GD) so that the characteristic is maintained after every optimization step. We employ matrix exponential for the submersion from Euclidean space to the Stiefel manifold due to its simplicity and availability in the deep learning frameworks [22].

The second contribution is the application of the VG objective with the proposed model function to the Proximal Policy Optimization (PPO) algorithm, which we call VG-PPO. Compared to other AC algorithms, PPO relies on longer truncated trajectories where the proposed model function makes it possible to implement the VG objective. In order to compare the model functions and test the proposed algorithm, we made a benchmarking environment where the effect of an action on a future state value is delayed, and the relation between an action and a reward is one-to-one, that is a past action only affect one future reward on the trajectory. Therefore, in this environment, the correlation-based gradient estimator of PG methods performs poorly since a value estimation in a state includes mostly unrelated rewards, and gradient estimators of PG methods are unable to make a distinction, whereas gradient estimate in VG methods produces a non-zero gradient only for the related action. We also compared the previously proposed model functions with the one that is proposed in this work. In that comparisons, for longer trajectory lengths, our approach is the only one that yields optimal performance. We also compared Actor-Critic algorithms such as vanilla PPO and Advantage Actor Critic (A2C) with VG-PPO in this benchmark and show that VG-PPO is the fastest converging one to the optimal performance.

1.2 RELATED WORKS

1.2.1 Value Gradients

VG method is introduced by Fairbank [23] as a deterministic approach for policy optimization. It is asserted that the VG objective is significantly more efficient on a control task with known differentiable dynamics. Fairbank shows the similarity between PG and VG-based learning algorithms when λ is set to 1 and claims that VG methods make value approximation redundant. Moreover, the resemblance between the policy update with VG objective and backpropagation through time (BPTT) is stated, which naturally arises since the value function functionally depends on the past actions on a trajectory. Fairbank does not touch upon the model approximation or learning and hence, leaves it to the following works.

The first generally applicable VG algorithm is proposed by Hess et al. [16] that provides two different modeling of the VG objective with a differentiable model approximation. The first approach applies the VG objective to the simulated trajectories of the approximate model function, where it is stated that this approach suffers from compounding error issues. The second approach utilizes reparameterization to enable VG methods on sampled trajectories. Reparameterization corrects the model error at every step throughout the trajectory, hence preventing the compounding model error problem. They named the VG algorithm that utilizes reparameterization as Stochastic Value Gradient (SVG) since the output of the model function is represented by a Gaussian function where the model function estimates the parameters of this distribution. Depending on the length of the truncated trajectory in which the VG objective is applied, they proposed a spectrum of SVG(k) algorithms where the index k determines the length of the truncated trajectories. In their experiment, they indicated that an off-policy variant that makes use of an experience replay, SVG(1)-ER, surpasses other SVG variants, including the model-free variant known as Deep Deterministic Policy Gradient (DDPG) and a vanilla AC algorithm.

It has been noted that the gradient exploding and vanishing problem is one of the shortcomings of the SVG algorithm. To mitigate this issue, they proposed a model function where the outputs of the functions are independent of each other. Their empirical findings indicate that SVG is only suitable for extremely short trajectories; as the length of the trajectory, k, increases, the performance of SVG variants degrades. They left the investigation of eligible model function that does not lead gradients to divergence as future work.

Another application of the VG objective is proposed by Amos et al [21]. They combined model-free Soft Actor Critic (SAC) [24] algorithm with SVG and provided a model-based entropy regularized policy learning algorithm SAC-SVG(k). Similar to SVG, they build a computational graph on truncated trajectories and the length of which is denoted by k. They applied SAC-SVG to a set of MuJoCo [13] environments and compared it with SAC and several model-based approaches. They were able to surpass the performance of SAC in several of these environments. Unlike SVG, in the model learning stage, they approximated reward and termination functions besides the transition function. In terms of model function, they proposed to use Gated Recurrent Units (GRU) [25] to alleviate gradient issues. The model learning objective is inspired by the Model-Based Policy Optimization (MBPO) [9] algorithm, which argues the detrimental effects of compounding error issues and suggests utilizing short truncated trajectories for model learning. Although they were able to employ SAC-SVG for truncated trajectories up to length 10 during policy optimization, the performance starts diminishing drastically as the trajectory length increases. They have stated that the highest score, in general, is obtained when the trajectory length is kept to 2, one higher than the best SVG variant. These empirical results are consistent with our findings in policy optimization with recurrent model functions that implement GRU and Long Short Term Memory (LSTM) [26]. They have also pointed out several future directions, some of which are addressed in this thesis, including the extension of the SVG algorithm to other model-free algorithms and employing constrained MDPs.

A notable work towards mitigating the gradient divergence during policy optimization is Directional Derivative Projection Policy Optimization (DDPPO) suggested by Li et al. [27]. Similar to the SAC-SVG algorithm, they employed a model learning method inspired by MBPO. In contrast, they applied a constraint term to the model objective that aims to restrict the model Jacobian to avoid divergence in gradients. Since the optimization of such an objective would require second-order methods as the objective include Jacobian of the model function, they proposed a finite differences method for the calculation of the model Jacobian using the directional derivatives of a second model that is trained with the same experience memory. By controlling the coefficient of the constraint term in the objective function, they are able to trade between stable or accurate gradients. Additionally, they put theoretical justification for their directional derivative estimation method and showed that it converges to the true derivative of the model. They also provided extensive empirical studies on several MuJoCo control tasks. Although their results are mostly on par with the baseline model-based algorithms, they provide a novel technique for constraining the model Jacobian. Nevertheless, they obtained optimal results on short truncated trajectories, generally no longer than 3. In contrast, we use truncated trajectories in VG-PPO that extend up to 64 steps without observing any divergent behavior.

A relatively immature but promising direction for utilizing model information to improve policy optimization is proposed by Ma et al. [28]. They employed an attention-based model function that is able to capture time-dependent relations on a truncated trajectory for predicting the model dynamics. The attention mechanism is argued to provide "shortcuts"; hence the algorithm is named Model-based Temporal Shortcuts. Similar to the experiments in this thesis, they provided a test environment in which relatively long-term action-reward dependencies are present. They showed that the proposed attention-based model function is able to focus on related actions even if they are a few steps apart from the reward of interest. Unfortunately, the authors have not included benchmark scores on baseline environments yet. In our work, we implemented the proposed model function, but to the best of our understanding, we have observed that the Markovian dynamics of the environments provide trivial solutions to the model objective as a state vector contains enough information to allow for the prediction of the next one. Therefore, we did not extend this approach. Nevertheless, we believe this is a very promising direction given the success of the attention mechanism [29] in Natural Language Processing (NLP).

Besides VG methods, one of the recently developed paradigms in policy optimization, inspired by the success of the transformer architectures [29], that utilizes a transformer-based model function for an autoregressive sequence learning has been proposed by Chen et al. as Decision Transformers (DT) [30] and by Janner et al [31]. These approaches, although lacking theoretical justification, have shown promising results on some of the RL benchmarks. There is a strong resemblance between VG methods and this paradigm as they both perform policy optimization with an objective that maximizes the value estimation through backpropagation over the truncated trajectories. In contrast to previously mentioned works, DT is able to optimize the policy with trajectories of up to 50 steps. We argue that the VG mechanism can provide a theoretical basis for transformer-based sequence learning approaches, with transformers being employed as high-capacity baselines for both model and policy functions.

Another VG-like approach that propagates analytic gradients through truncated trajectories is Dreamer [7,8] proposed by Hafner et al. Contrary to previous approaches, Dreamer utilizes the model function to simulate short truncated trajectories. The emphasis in this work is put on the architecture of the model function and the latent space representation where the policy optimization is performed. Dreamer employs a complex recurrent model function that is trained with the samples drawn from a replay buffer. During policy optimization, they only use simulated trajectories and argue that the policy optimization on latent space is computationally more efficient. The policy optimization objective is a hybrid one in Dreamer. It includes both correlation-based PG objective and VG objective, which they describe as analytic gradients over the model function without mentioning the name value gradients. They provided a rich set of experiments on both MuJoCo [13] control tasks and ATARI [12] environments. In the ablation study, they compared these two objectives alongside other parameters in the proposed architecture and claimed that the VG objective does not improve upon the PG objective for most of the environments. We argue that when relatively long trajectories are used, such as the ones that they used in the training of Dreamer, the recurrent model function tends to show divergent gradient behavior. Since this is not the main point of the paper, they did not focus on exploding or vanishing gradient of the model function. Albeit the complexity of the proposed model architecture and complex training regime, they managed to surpass state-of-the-art model-free algorithms, which have not been the case for model-based algorithms until then. In this thesis, we have also observed the benefits of using a hybrid policy optimization objective, particularly during the early stages of the model training.

1.2.2 Unitary Recurrent Networks

Policy optimization that passes the gradient through sampled trajectories generates an almost identical computational graph as of Recurrent Neural Networks (RNN). This similarity has been the reason why SVG algorithms, including the one we propose in this thesis, employ model functions that are similar to that of RNNs, commonly a gated network such as LSTM or GRU. Besides these networks, in order to mitigate the vanishing or exploding gradient issue, attention mechanisms on time [29] or constrained models have been suggested. This thesis focuses on the latter approach that brings restrictions to the network parameters such that the Jacobian of the network is bounded. One of the ways of achieving this, as proposed by Arjovsky et al., is uRNN [32] where the weight matrices of the linear layers are parameterized to become unitary matrices by applying 7 consecutive norm-preserving transformations including rotation, reflection, and permutation. In addition, they proposed a parametric non-linearity called modRELU, a variation of ReLU [33], which initially preserves the norm of its input while it may slowly reduce the input norm over the course of the training. In this thesis, our ablation studies indicate modRELU yields the lowest loss during the model training stage among other norm-preserving non-linearities. In their experiments, they compared uRNN with LSTM in standard and permuted pixel MNIST [34] tasks and demonstrated that uRNN converges significantly faster than LSTM while maintaining most of the gradient norm, albeit the converged accuracy of LSTM is higher in standard pixel MNIST task.

Despite the simplicity of the parametrization approach suggested in uRNN, Wisdom et al. [35] showed that after 7 dimensions in the hidden state, the capacity of the network degrades since there are exactly 7 transformations in the parametrization.

Instead, they proposed a more rigorous parametrization that covers the entire set of unitary matrices; hence the model capacity does not degrade by the increase in the state dimension. They introduced the Stiefel manifold, a set of k-orthogonal vectors that can represent all unitary matrices with the same dimension, and suggested using Cayley transform as parametrization since the Cayley transform maps any vector in euclidean space to the Stiefel manifold. Hence, they provided an optimization procedure akin to Riemannian Gradient Descent (RGD) with Cayley Transform as retraction, which is both full capacity and guaranteed to preserve the orthonormal property of the weight matrices in the linear layers. They demonstrated the performance of the proposed approach on MNIST tasks where the proposed version of uRNN surpasses both vanilla uRNN and LSTM given enough parameters.

Besides parametrization methods that employ Cayley Transform [35]–[37], Casado [38] additionally proposed matrix exponential for parametrization as simple to implement and numerically stable alternative to Cayley Transform. Moreover, Casado introduced a generalized RGD framework and provided a PyTorch [39] implementations that include the mentioned parametrization techniques, which allows constrained optimization on manifolds [22]. This framework contains parametrization techniques for several matrix constraints including the one that is forced by the Stiefel manifold. We use this framework for the model function implementations in this thesis.

2. BACKGROUND

2.1 MARKOV DECISION PROCESS

In the field of reinforcement learning, Markov Decision Processes (MDPs) provide a mathematical framework for modeling sequential decision-making under uncertainty. MDPs are used to formalize the problem of an agent attempting to maximize a weighted sum of numerical rewards by making decisions/actions in a sequence over time.

In the rest of the thesis, we denote random variables by capital letters and samples of them or scalar values by lowercase letters. We consider a standard finite horizon discounted Markov Decision Process (MDP) with a single agent that interacts with an environment in discrete time steps until the termination state is reached. The MDP consists of a 7-tuple $\langle \mathrm{S}, \mathrm{A}, r, p, d, \rho, \gamma \rangle$, where S denotes the state space, A denotes the action space, the function $r_t = f_r(s_t, a_t, s_{t+1})$ denotes the reward function of the state, action, and next state, the density function $p(s_{t+1}, r_t | S = s_t, A = a_t)$ denotes the transition distribution, $d(.|s_t, a_t)$ denotes a conditional Bernoulli distribution that determines the end of the episode, $\rho(s)$ denotes the initial state distribution, and the scalar $\gamma \in [0, 1)$ is the discounting factor. Agent interactions are determined by the policy density function $\pi(a_t|s_t; \theta)$ parameterized by θ that returns a distribution over the action space A for every state. The finite return of a state, following a policy $\pi(.|s)$, is a random variable $G^\pi(S_t) = \sum_{i=t+1}^{T} \gamma^i r_i$ where r_i's are the sampled scalar rewards on a trajectory that ends with a terminal state, which is determined by $d(.|s_t, a_t)$, and the superscript on the return G denotes the policy under which the rewards are sampled. Additionally, we use $p^\pi(s_{t+1}|s_t)$, $p^\pi(\tau)$, and $d^\pi(s)$ to denote policy aware transition, trajectory, and stationary state distributions respectively where τ represent a sequence of states, actions, and rewards $\tau = (s_0, a_0, r_1, s_1, a_1, \dots)$. Similarly, the return of a trajectory is denoted by $G^\pi(\tau)$. The conditional expectation under the policy, termination and transition distributions describes the value $V^\pi(s_t) =$

$\mathbb{E}_{a_t\sim\pi, s_{t+1}\sim p, d_t\sim d}\big[G^\pi(s_t)\big|S = s_t]$. The objective is to maximize the expected state value over the initial state distribution $J(\theta) = \mathbb{E}_{s_0\sim\rho}\big[V^\pi(s_0)\big]$.

2.2 POLICY GRADIENTS

Policy Gradient (PG) is a framework in Reinforcement Learning (RL) that has seen significant development in recent years. It is a technique for finding an optimal policy, which is a set of rules that an agent must follow in order to maximize its reward. Unlike other RL algorithms, PG algorithms do not require an explicit model of the environment and can be used in a variety of different scenarios. The core idea behind policy gradient is to use a gradient-based optimization algorithm to update a policy in order to maximize rewards. This is done by taking the derivative of a value estimation with respect to the parameters of the policy and using gradient ascent to update the policy parameters. This process is repeated until an optimal policy is found.

2.2.1 REINFORCE

The simplest value estimation is Monte Carlo (MC) based estimation that is employed in REINFORCE [3]. The PG objective aims to maximize this value estimate with respect to policy parameters. The objective is $J(\theta) = \mathbb{E}_{\tau\sim p^\pi(\tau;\theta)}\big[G^\pi(\tau)\big]$ where θ is the parameters of the policy π. We take the derivative of the objective function $\nabla_\theta J(\theta)$ with respect to the policy parameters θ in order to apply gradient ascent as shown in Equation (2.1).

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim p^\pi(\tau;\theta)}\big[G^\pi(\tau)\big] \\
&= \nabla_\theta \int_\tau p^\pi(\tau;\theta) G^\pi(\tau) d\tau \\
&= \int_\tau p^\pi(\tau;\theta) \nabla_\theta \log p^\pi(\tau;\theta) G^\pi(\tau) d\tau \\
&= \mathbb{E}_{\tau\sim p^\pi(\tau;\theta)}\left[\nabla_\theta \log\left(\prod_{t=0}^{T} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t;\theta)\right) G^\pi(\tau)\right] \\
&= \mathbb{E}_{\tau\sim p^\pi(\tau;\theta)}\left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t;\theta)\right) G^\pi(\tau)\right] \qquad (2.1)
\end{aligned}$$

Equation (2.1) provides a Monte Carlo estimation for the policy gradients. Since the gradient estimate does not include any term related to the transition distribution, it is a model-free estimate. However, although being a bias-free estimate, the variance in this estimate is significantly large. One simple modification towards reducing the variance in the estimate is to exploit causality. It suggests that the rewards only depend on the actions taken before them.

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim p^\pi(\tau;\theta)}\Big[\Big(\sum_{t=0}^{T}\nabla_\theta \log\pi(a_t|s_t;\theta)\Big)\Big(\sum_{t=1}^{T} r_t\gamma^t\Big)\Big] \\
&= \sum_{t=0}^{T}\mathbb{E}_{\tau\sim p^\pi(\tau;\theta)}\Big[\big(\nabla_\theta \log\pi(a_t|s_t;\theta)\big)\Big(\sum_{k=t+1}^{T} r_k\gamma^{k-t-1}\Big)\Big] \\
&= \sum_{t=0}^{T}\mathbb{E}_{\tau_{0:t}\sim p^\pi(\tau;\theta)}\mathbb{E}_{\tau_{t+1:T}\sim p^\pi(\tau;\theta)}\Big[\big(\nabla_\theta \log\pi(a_t|s_t;\theta)\big)\Big(\sum_{k=t+1}^{T} r_k\gamma^{k-t-1}\Big)\Big] \\
&= \sum_{t=0}^{T}\mathbb{E}_{\tau_{0:t}\sim p^\pi(\tau;\theta)}\Big[\big(\nabla_\theta \log\pi(a_t|s_t;\theta)\big)\mathbb{E}_{\tau_{t+1:T}\sim p^\pi(\tau;\theta)}\Big[\sum_{k=t+1}^{T} r_k\gamma^{k-t-1}\Big]\Big] \\
&= \sum_{t=0}^{T}\mathbb{E}_{\tau_{0:t}\sim p^\pi(\tau;\theta)}\big[\nabla_\theta \log\pi(a_t|s_t;\theta)V^\pi(s_t)\big] \qquad (2.2)
\end{aligned}$$

The gradient estimate in Equation (2.2) describes the Policy Gradient with value function. In this equation, $\tau_{k:n}$ denotes a chunk of a trajectory starting from time step k and ending at time step n.

2.2.2 Actor Critic

The REINFORCE algorithm does not use value approximation; hence it employs MC-based value estimation. Alternatively, we can use the TD(λ) based value estimation in the PG objective in Equation (2.2). This would require value approximation if the parameter λ is not set to 1. We call this set of PG algorithms Actor-Critic (AC) algorithms. Usually, we use another neural network (NN) for the value approximation [2].

One of the value estimates preferred in AC algorithms is the advantage estimate. It is a measure of how much better or worse a given action is compared to the average action value in a state. It is denoted by $A^\pi(s_t,a_t) = Q^\pi(s_t,a_t) - V^\pi(s_t)$. The advantage estimate is equal to the expectation of TD error $\delta_t = r_t + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$ under the transition distribution.

$$\begin{aligned} A^{\pi}(s_t, a_t) &= \mathbb{E}_{s_{t+1} \sim p(.|s_t,a_t)}[\delta_t] \\ &= \mathbb{E}_{s_{t+1} \sim p(.|s_t,a_t)}[r_t + \gamma V^{\pi}(s_{t+1})] - V^{\pi}(s_t) \\ &= Q^{\pi}(s_t, a_t) - V^{\pi}(s_t) \end{aligned} \tag{2.3}$$

Asynchronous Advantage Actor Critic (A3C) [6] and its synchronous counterpart (A2C) employs advantage estimate in the Actor Critic framework. Furthermore, in order to improve the stability and the training speed of the algorithm, A3C utilizes parallel working environments that gather chunks of trajectories called rollout.

In this thesis, we use A2C both as a baseline in the comparisons and as a reference algorithm for implementing VG due to its simplicity. A2C employs n-step estimation for the advantage function $\hat{A}^{\pi}_n(s_t) = V^{\pi}(s_{t+n}) - \gamma^n V^{\pi}(s_{t+n}) + \sum_{i=t+1}^{n} \gamma^{i-t-1} r_i$. Similarly, the target for the state-value approximation is n-step and is calculated by $\hat{V}(s_t) = \hat{A}^{\pi}_n(s_t) + V^{\pi}(s_t)$ where $\hat{V}^{\pi}(s)$ denotes the target estimation.

Algorithm 1 Advantage Actor Critic (A2C) Algorithm

```
Initialize actor parameters $\theta$ and critic parameters $\phi$
Initialize K environments $\{E_1, \dots, E_k\}$ in parallel
Initialize total time step $T$ and step size $n$
for $t \in [0, \text{floor}(T/K)]$ ; $t \leftarrow t + n$ do
    for each environment $E_i$ do
        Sample $n$-step rollout $\tau^i_{t:t+n}$ from environment $E_i$ with $\pi(a|s;\theta)$
        Initialize value target estimate $\hat{V}^{\pi}_{\phi}(s_t) = v_{\phi}(s_{t+n})$
        for each $s_k, a_k, r_k, s_{k+1} \in \text{reversed}(\tau^i_{t:t+n})$ do
            $\hat{V}^{\pi}_{\phi}(s_t) \leftarrow \gamma \hat{V}^{\pi}_{\phi}(s_t) + r_k$
            $\hat{A}^{\pi}_{\phi}(s_k) \leftarrow \hat{V}^{\pi}_{\phi}(s_t) - v_{\phi}(s_k)$
            Calculate $\hat{\nabla}_{\theta}$ and $\hat{\nabla}_{\phi}$ as in Equations (2.5) and (2.4)
            Store the calculated gradients $\nabla_{\theta}$ and $\nabla_{\phi}$
        end for
    end for
    Update the parameters $\theta$ and $\phi$ using the mean of the stored gradients
end for
```

A2C implements two neural networks, one for the actor-network $\mu(s;\theta)$ that parameterize the policy distribution, which is denoted by $\pi_{\theta}(a|s)$, and the other one is for the critic network $v_\phi(s)$ that approximates the state-value function. The target value that replaces its state values by the approximation of the critic network is denoted by $\hat{V}^\pi_\phi(s)$.

$$\hat{\nabla}_\phi = \mathbb{E}_{s\sim d^\pi(s)}\Big[(v_\phi(s) - \hat{V}^\pi_\phi(s))\nabla_\phi v_\phi(s)\Big] \tag{2.4}$$

$$\hat{\nabla}_\theta = \mathbb{E}_{s\sim d^\pi(s), a\sim\pi_\theta(a|s)}\Big[\nabla_\theta \log \pi_\theta(a|s)\hat{A}^\pi_\phi(s)\Big] \tag{2.5}$$

Equations, (2.4) and (2.5) give gradient estimates based on the Policy Gradient theorem to update the critic and actor networks respectively.

Algorithm 1 shows the pseudo-code for the A2C algorithm. Increasing the number of parallel environments improves the gradient estimates at Equations (2.4) and (2.5) by providing more gradient samples to the Monte Carlo sampling estimation. The practice of employing parallel working environments has become a common practice for on-policy algorithms due to its impact on performance. In the implementations of this thesis, we take advantage of synchronously running parallel environment workers.

2.2.3 Generalized Advantage Estimation

In Actor-Critic algorithms, we can use a variety of value estimations both in the value approximation objective as the value target or in the policy optimization. Generalized Advantage Estimation (GAE) [4] combines Monte Carlo sampling and temporal difference (TD) learning to estimate an advantage function akin to TD(λ). It combines the TD error terms δ_t for n-steps by an exponentially decaying weighting scheme. This allows it to have a control over the bias and variance in the advantage estimates and trade between them to possibly provide more stable and accurate learning.

The advantage estimate of GAE is denoted by GAE$^{\lambda,\gamma}(s)$. In this subsection, for the clarity of the notation, we omit the policy π from the value estimators. GAE returns a weighted sum of all the k-step advantage estimates $\hat{A}_k(s)$.

$$\hat{A}_k(s_t) = \gamma^k V(s_{k+t}) + \sum_{i=t+1}^{k} \gamma^{i-t-1} r_i - V(s_t)$$

$$
\begin{aligned}
&= \gamma^k V(s_{k+t}) + \sum_{i=t+2}^{k} \gamma^{i-t-1} r_i - V(s_{t+1}) + \gamma V(s_{t+1}) + r_{t+1} - V(s_t) \\
&= V(s_{k+t}) + \sum_{i=t+2}^{k} r_i - V(s_{t+1}) + \delta_t \\
&= \sum_{i=0}^{k-1} \gamma^i \delta_{t+i} \qquad (2.6)
\end{aligned}
$$

The k-step advantage can be written as the sum of TD errors at every step up to k as shown in Equation (2.6). Hence, we can form a GAE estimate using TD errors.

$$
\begin{aligned}
\mathrm{GAE}^{\lambda,\gamma}(s_t) &= (1-\lambda) \sum_{k=1}^{\infty} \lambda^{k-1} \hat{A}_k(s_t) \\
&= (1-\lambda) \sum_{k=1}^{\infty} \lambda^{k-1} \left(\sum_{i=0}^{k-1} \gamma^i \delta_{t+i} \right) \\
&= (1-\lambda) \sum_{k=0}^{\infty} \lambda^k \gamma^k \delta_{t+k} (1 + \lambda + \lambda^2 + \cdots + \lambda^{n-k-1}) \\
&= (1-\lambda) \sum_{k=0}^{\infty} \lambda^k \gamma^k \delta_{t+k} \frac{1}{1-\lambda} \\
&= \sum_{k=0}^{\infty} \lambda^k \gamma^k \delta_{t+k} \qquad (2.7)
\end{aligned}
$$

GAE has been shown to improve the performance of Actor-Critic algorithms in a variety of tasks, including continuous control and Atari [12] game playing. It has also been used in combination with other methods, such as trust region optimization [4,40], to further improve the performance of Actor-Critic algorithms. Therefore, we use GAE as the advantage estimation in our implementations, as it is compatible with the VG objective and provides a strong baseline.

2.3 PROXIMAL POLICY OPTIMIZATION

On-policy and model-free RL algorithms tend to require a large number of samples due to the instabilities in the learning and high variance in the policy gradient estimator. One of the key drawbacks of on-policy algorithms such as A2C is that the experience used in the parameter updates can not be reused for another update since the samples become an off-policy experience. However, the optimization steps are relatively small, so the change in the policy parameters only slightly makes the samples used in the optimization step off-policy experience.

The Trust Region Policy Optimization (TRPO) algorithm, the predecessor of PPO, proposes a constrained optimization framework such that the updated policy parameters stay close to the policy parameters used to gather the experience. This way, TRPO aims to reuse the same experience for multiple policy updates before the final policy is off enough from the policy that gathers the experience. TRPO makes use of a powerful equation [41] that enables expressing the expected value of a policy in terms of another one using the advantage function.

$$\begin{aligned}
\mathbb{E}_{\tau\sim p^{\hat{\pi}}(\tau)}\Big[\sum_{k=0}^{\infty}\gamma^k A^{\pi}(s_k,a_k)\Big] &= \mathbb{E}_{\tau\sim p^{\hat{\pi}}(\tau)}\Big[\sum_{k=0}^{\infty}\gamma^k\big(\gamma V(s_{k+1})+r_{k+1}-V(s_k)\big)\Big]\\
&= \mathbb{E}_{\tau\sim p^{\hat{\pi}}(\tau)}\Big[-V(s_0)+\sum_{k=0}^{\infty}\gamma^k r_{k+1}\Big]\\
&= -\mathbb{E}_{\tau\sim p^{\hat{\pi}}(\tau)}\big[V(s_0)\big]+\mathbb{E}_{\tau\sim p^{\hat{\pi}}(\tau)}\Big[\sum_{k=0}^{\infty}\gamma^k r_{k+1}\Big]\\
&= -J(\theta)+J(\hat{\theta}) \qquad (2.8)
\end{aligned}$$

The expression in Equation (2.8) formulates a way for calculating the expected value $J(\hat{\theta})$ of policy $\hat{\pi}$ using the expected value of another policy π while using the trajectories of $\hat{\pi}$. Assume the parameter $\hat{\theta}$ is a result after an update performed on the parameter θ. In order to take advantage of Equation (2.8) for another policy update on the parameters $\hat{\theta}$ using the previous experience, the trajectories under the expectation need to be distributed by π instead of $\hat{\pi}$. TRPO relaxes Equation (2.8) by providing an approximation so that the following policy updates are allowed to use previous experience.

$$\begin{aligned}
J(\hat{\theta}) &= \mathbb{E}_{\tau\sim p^{\hat{\pi}}(\tau)}\Big[\sum_{k=0}^{\infty}\gamma^k A^{\pi}(s_k,a_k)\Big]+J(\theta)\\
&= \mathbb{E}_{s\sim d^{\hat{\pi}}(s)}\Big[\mathbb{E}_{a\sim\hat{\pi}(a|s)}\big[A^{\pi}(s,a)\big]\Big]+J(\theta)
\end{aligned}$$

$$\approx \mathbb{E}_{s\sim d^{\pi}(s)}\Bigg[\mathbb{E}_{a\sim\hat{\pi}(a|s)}\big[A^{\pi}(s,a)\big]\Bigg] + J(\theta) \tag{2.9}$$

The expression in Equation (2.9) is a first-order approximation to the one in Equation (2.8) as shown by [41] Kakade et al. Hence, the approximation holds as long as the two policies, $\hat{\pi}$ and π, are close. Following this, TRPO formalizes a constraint optimization where the objective is to maximize the expected value of a policy using the samples of its previous version. The Kullback–Leibler (KL) divergence term in the constraint prevents significant changes between the two policies.

In order to fully utilize the samples from an earlier policy π, TRPO employs an importance sampling term that enables replacing $\hat{\pi}$ by π under the expectation. Combining these, TRPO proposes a policy optimization problem given below.

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \mathbb{E}_{s\sim d^{\pi}(s), a\sim\pi(a|s)}\left[\frac{\hat{\pi}(a|s;\theta)}{\pi(a|s)}\hat{A}^{\pi}(s)\right] \\ \text{subject to} \quad & \mathbb{E}_{s\sim d^{\pi}(s)}\Big[D_{KL}\Big(\hat{\pi}(.|s)\Big\|\pi(.|s;\theta)\Big)\Big] \leq \epsilon \end{aligned} \tag{2.10}$$

The optimization objective in Expression (2.10) forces the parameter update to stay in the on-policy region after optimization steps. The ϵ parameter determines the closeness of the two policies. Relaxing the constraint allows aggressive policy updates that maximize the advantage estimate $\hat{A}^{\pi}$ while tightening it allows for accurate approximation in the value estimation. Although the above optimization problem in Expression (2.10) provides a rigorous and analytical policy optimization that reuses past experience, in practice, the non-linear constraint optimization problem is replaced by an unconstrained problem where the constraint term is introduced to the objective with the coefficient β, i.e., $\mathbb{E}_{s\sim d^{\pi}(s), a\sim\pi(a|s)}\Big[\frac{\hat{\pi}(a|s;\theta)}{\pi(a|s)}\hat{A}^{\pi}(s) - \beta\Big[D_{KL}\Big(\hat{\pi}(.|s)\Big\|\pi(.|s;\theta)\Big)\Big]\Big]$. Instead, PPO suggests an alternative surrogate objective that is simple to implement and provides easily tuneable clipping parameter ϵ shown in Equation (2.11).

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_{s\sim d^{\pi}(s), a\sim\pi(a|s)}\Big[\min\Big(r_{\theta}(s,a)\hat{A}^{\pi}(s), \text{clip}(r_{\theta}(s,a), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi}(s)\Big)\Big] \tag{2.11}$$

The clip objective in Equation (2.11) eliminates the gradients that are more off than allowed by the ratio $r_\theta(s,a) = \frac{\hat{\pi}(s,a;\theta)}{\pi(s,a)}$, which also stands as the importance sampling term in the policy optimization objective. Here ϵ determines the range of the policy distribution ratio in which gradients are not cut off by the minimum and the clip operations during the backward pass. Overall, PPO promises policy optimization with reusable experience, hence improving the sample efficiency and stability.

Algorithm 2 Proximal Policy Optimization (PPO) Algorithm

```
Initialize actor parameters $\theta$ and critic parameters $\phi$
Initialize K environments $\{E_1, \dots, E_k\}$ in parallel
Initialize total time step $T$, number of epochs $E$, and step size $n$
for $t \in [0, \text{floor}(T/K)]; t \leftarrow t + n$ do
    Initialize rollout store $\mathbb{T}$ and advantage estimate store $\mathbb{A}$
    for each environment $E_i$ do
        Sample and store $n$-step rollout $\mathbb{T} \leftarrow \mathbb{T} \cup \tau^i_{t:t+n} \sim p^{\hat{\pi}}(\tau)$
        Calculate and store advantage estimates $\mathbb{A} \leftarrow \hat{A}^\pi$
    end for
    for each epoch $\in [1, E]$ do
        Shuffle $\mathbb{T}$
        for each batch of $(s_k, a_k, r_{k+1}, s_{k+1}) \in \mathbb{T}$ do
            Calculate PG estimate using $\mathbb{A}$ and $\mathbb{T}$ according to (2.11)
            Calculate $\frac{\partial}{\partial \phi}$ of the value loss with the batch
            Update the parameters $\theta$ and $\phi$
        end for
    end for
end for
```

In PPO, similar to A2C there are multiple parallel environment workers and actor and critic networks. The objective of the critic is the same, although the advantage estimate that is used in the objective may differ. The key point of PPO is the multi-epoch usage of the same experience, which includes state-action transitions, rewards, and advantage estimates of the experience collecting policy. If the number of epochs is set to 1, PPO becomes equivalent to A2C since there will be no clipping applied and the importance sampling ratio r_θ is equal to 1. In practice, the number of epochs is assigned a value between 5 to 10, depending on the task. In terms of performance in common RL benchmarks and simplicity, PPO stands as one of the most prominent model-free approaches. Hence, we implement the VG objective to combine the sample reusability feature of PPO with the low variance policy gradient estimate of VGs. The challenge in combining PPO with VG objective is that PPO prefers long rollout lengths, and VG objective fails in long trajectories. The model function proposed in this thesis makes it possible to implement the VG objective by enabling VG estimates on long horizons. The details of that algorithm are discussed in the next chapter.

2.4 VALUE GRADIENTS

The policy gradient estimate in PG methods relies on the correlation between actions and rewards. This estimate, although being model-free, as shown in Equation (2.1) tends to have large variance. Although value approximation provides a trade between variance and bias in the gradient estimate by n step bootstrapping, in practice, AC algorithms favor large values of n; hence the variance of the PG estimate remains large. The alternative gradient estimate to that of PG methods is the VG estimate, which relies on the functional relation instead of likelihood or correlation. VG methods link the rewards and value estimates to the past actions along a truncated trajectory, i.e., n-step rollouts. In order to achieve this, VG methods combine differentiable model and reward functions to form a computational graph (CG) over a truncated trajectory. This functional relation allows PG to produce a low variance gradient estimate.

Similar to PG, the objective in VG is the maximization of expected value estimates with respect to policy parameters. Unlike the PG approach, VG utilizes reparameterization to form the CG on sampled truncated trajectories as suggested in SVG [16].

2.4.1 Reparameterization

Derivative operation through samples of a parametric distribution requires additional steps. One of the approaches proposed by Williams [3] is log-trick, which is used in PG methods shown in equation (2.12).

$$\nabla_\theta \mathbb{E}_{p(y|x;\theta)}\Big[f(y)\big|X=x\Big] = \mathbb{E}_{p(y|x;\theta)}\Big[\nabla_\theta \log p(y|x;\theta) f(y)\big|X=x\Big] \tag{2.12}$$

Alternatively, reparameterization separates deterministic parameters from the stochastic parts of a distribution that does not depend on the parameters. Equation (2.13) shows its application on the derivative of an expectation.

$$\nabla_\theta \mathbb{E}_{p(y|x;\theta)}\Big[f(y)\big|X=x\Big] = \mathbb{E}_{\rho(\epsilon)}\Big[\nabla_\theta f\big(h(x,\epsilon)\big)\big|X=x\Big] \tag{2.13}$$

Here, the samples of the distribution $p(y|x;\theta)$ are written in terms of a deterministic function $y = h(x,\epsilon)$, which takes the input x and parameter-independent sample ϵ. This allows the derivative operation to easily pass over the expectation.

The choice of the deterministic function h depends on the distribution. In this thesis, we reparameterize parametric Normal distributions $s \sim \mathcal{N}(\mu, \sigma^2)$ by the parameters μ and σ as $s = (\mu + \epsilon)\sigma$, where $\epsilon \sim \mathcal{N}(0,1)$ [42]. We use straight-through [43] reparameterization for the categorical distribution. In the following chapters, we present a comparison of the two approaches that we discussed in this subsection for simple gradient estimation.

2.4.2 Stochastic Value Gradient

Value gradient objectives utilize the differentiable model and reward functions to provide policy gradient estimates on a truncated trajectory. This mandates the use of deterministic functions in which the computational graph is built over the outputs of the model function. If the model is not exact but approximated, then iteratively calling the deterministic model function may cause the approximation errors to compound over time, which leads to divergence and numerical instabilities.

Instead of a deterministic approach, SVG utilizes reparameterization to build a computation graph on the sampled trajectories, where at each step the approximate model is given sampled states instead of the previous prediction, which does not cause a compounding error problem.

$$\begin{aligned}
\nabla_\theta V(s_t) &= \nabla_\theta \mathbb{E}_{p(s_{t+1}|s_t,a_t)\pi(a_t|s_t)}\big[r_{t+1} + \gamma V(s_{t+1})\big] \\
&= \nabla_\theta \mathbb{E}_{p(\epsilon_s,\epsilon_a)}\big[r(s_t, f(s_t,\epsilon_s,\mu(s_t,\epsilon_a)), \mu(s_t,\epsilon_a)) + \gamma V(f(s_t,\epsilon_s,\mu(s,\epsilon_a))\big] \\
&= \mathbb{E}_{p(\epsilon_s,\epsilon_a)}\Big[\nabla_\theta r(s_t,\tilde{s}_{t+1},\tilde{a}_t) + \nabla_\theta \gamma V(\tilde{s}_{t+1})\Big] \\
&= \mathbb{E}_{p(\epsilon_s,\epsilon_a)}\Big[\mathbb{J}^\mu_\theta(\tilde{a}_t)\Big(\nabla^{r_t}_{\tilde{a}_t} + \mathbb{J}^f_a(\tilde{s}_{t+1})\big(\nabla^{r_t}_{\tilde{s}_{t+1}} + \gamma\nabla^{V_{t+1}}_{s_{t+1}}\big)\Big)\Big]
\end{aligned} \tag{2.14}$$

Equation (2.14) provides the gradient of the 1-step value estimation. The reparametrization is applied by the model function $f(s_t, a_t, \epsilon_s)$ and to the deterministic policy function $\mu(s_t, \epsilon_a)$. The parameter-independent parts within the transition and the policy distributions that describe the noise are denoted by ϵ_s and ϵ_a respectively. Hence, the sample next state s_{t+1} and the sample action a_t are rewritten by the model and deterministic policy functions as $\tilde{s}_{t+1} = f(s_t, a_t, \epsilon_s)$ and $\tilde{a}_t = \mu(s_t, \epsilon_a)$. For the clarity of the notation, we replace functions f and μ with $\tilde{s}$ and $\tilde{a}$. The term $\mathbb{J}^f_x$(y) denotes the Jacobian of the function f at y with respect to x, similarly, ∇^g_x denotes the gradient of g w.r.t x where g is a scalar function.

In policy optimization, the sampled transition contains sample actions, rewards, and states. Hence, the noise terms ϵ_s and ϵ_a need to be inferred using the sample transition. Following the Bayesian approach provided in SVG [16], we modify the value gradient term as in Equation (2.15).

$$
\begin{aligned}
&\mathbb{E}_{p(\epsilon_s,\epsilon_a)}\mathbb{E}_{p(\tilde{s}_{t+1},\tilde{a}_t|\epsilon_s,\epsilon_a,s_t)}\Big[r(s_t, \tilde{s}_{t+1}, \tilde{a}_t) + \gamma V(\tilde{s}_{t+1})\Big] = \\
&\mathbb{E}_{p(s_{t+1},a_t|s_t)}\mathbb{E}_{p(\epsilon_s,\epsilon_a|s_{t+1},a_t,s_t)}\Big[r(s_t, \tilde{s}_{t+1}, \tilde{a}_t) + \gamma V(\tilde{s}_{t+1})\Big]
\end{aligned} \tag{2.15}
$$

Instead of sampling the noise values ϵ_s and ϵ_a from a noise distribution, we use the inferred noise values in the deterministic functions f and r.

In policy optimization with multi-step value estimation, the gradient of the value functions propagates backwardly through the computational graph formed over the sampled transition. The propagating gradient is denoted by $\nabla^{V_t}_{s_t}$ and Equation (2.16) shows its calculation while omitting the distributions under the expectation for clarity.

$$
\nabla^{V_t}_{s_t} = \Big(\mathbb{J}^{\mu}_s(\tilde{s}_t)\mathbb{J}^f_a(\tilde{s}_t, \tilde{a}_t) + \mathbb{J}^f_s(\tilde{s}_t, \tilde{a}_t)\Big)\Big(\nabla^r_{s_{t+1}} + \gamma\nabla^{V_{t+1}}_{\tilde{s}_{t+1}}\Big) + \Big(\nabla^r_{s_t} + \mathbb{J}^{\mu}_s(\tilde{s}_t)\nabla^r_{\tilde{a}_t}\Big) \tag{2.16}
$$

SVG combines these to form SVG(∞) and SVG(H), where H denotes the length of trajectory in which the value is estimated. SVG(∞) employs MC value estimation over finite trajectories while SVG(H) truncates the value estimation with the approximate value at the H'th step.

The approximate model function is trained with the negative log likelihood loss function by the samples from a replay buffer that keeps the transitions collected during the training. The value approximation, critic, is employed for SVG variants other than SVG(∞), which replaces the true state value in Equations (2.16) and (2.14). Although it is possible to formulate SVG as an off-policy algorithm [16,21], we focus on on-policy SVG algorithms in this thesis.

Following is a pseudo-code for SVG(H) algorithm. We initialize a critic function denoted by v_ϕ, an approximate model function f_ξ, and the deterministic policy function μ_θ that parameterizes the policy distribution π, which is used in experience collection. In a loop, we sample n-step rollouts and store them in a replay buffer. Next, we update the model function f_ξ with random samples from the buffer. Finally, we update the policy and the critic functions with the latest n-step rollouts.

Algorithm 3 SVG(H) Algorithm

```
Initialize $\mu_\theta$ and $f_\xi$ functions
Initialize Critic $v_\phi$
Initialize a replay buffer $\mathbb{D}$
Initialize K environments $\{E_1, \dots, E_k\}$ in parallel
Initialize total time step $T$ and step size $n$
for $t \in [0, \text{floor}(T/K)]; t \leftarrow t + n$ do
    for each environment $E_i$ do
        Sample and store $n$-step rollout $\mathbb{D} \leftarrow \mathbb{D} \cup \tau^i_{t:t+n} \sim p^{\hat{\pi}}(\tau)$
    end for
    Update the model function $f_\xi$ with samples from $\mathbb{D}$
    Initialize policy gradient accumulator $\nabla_\theta \leftarrow 0$
    Initialize value target $\hat{V} \leftarrow v_\xi(s_{t+n+1})$
    Initialize $\nabla^{V_{t+n+1}}_{s_{t+n+1}} \leftarrow \nabla_s V(s_{t+n+1})$
    for each $(s_k, a_k, r_{k+1}, s_{k+1}) \in \text{reversed}(\tau_{t:t+n})$ do
        $\hat{V} \leftarrow r_{k+1} + \gamma \hat{V}$
        Infer $\epsilon_{s_k}$ and $\epsilon_{a_k}$
        Calculate $\nabla^{V_k}_{s_k}$ according to Equation (2.16)
        Calculate $\nabla^{V_k}_\theta \leftarrow \nabla_\theta V(s_k)$ according to Equation (2.14)
        Accumulate policy gradient $\nabla_\theta \leftarrow \nabla^{V_k}_\theta + \gamma \nabla_\theta$
    end for
    Update the value function $v_\xi$ with the target value $\hat{V}$
    Update policy function $\mu_\theta$ with accumulated gradients $\nabla_\theta$
end for
```

The SVG algorithm provides a sample-based application of VG objective for varying trajectory lengths in the value estimation.

3. METHOD

Value Gradient (VG) algorithms utilize the model function to connect actions, rewards, and state values in a single computational graph, similar to that of Recurrent Neural Networks (RNN). This graph enables VG methods to perform direct value maximization given that the rewards and the values within the value estimate are in the graph. The model function guides the gradient of the value of interest through this graph backward in time. If the spectral radius of the Jacobian of the reparameterized policy aware transition function $g_{\theta,\phi}(s_t, \epsilon_s, \epsilon_a) \rightarrow \tilde{s}_{t+1}$ with respect to s is lower than 1 for all states, then the gradients passing through the computational graph vanish. Conversely, if the spectral radius is larger than 1 for all states, the gradients explode. In practice, when the transition function is modeled as a fully connected neural network, we observe divergence or converging to 0 issues, although the Jacobian matrices of the function g have mixed spectral radiuses.

3.1 PARAMETERIZATION

The policy-aware transition function g is composed with itself multiple times to reparameterize a sample trajectory $g^n(s_t, \varepsilon) \rightarrow s_{t+n}$. The function composition by itself is represented by $g^n(.)$ where n denotes the number of function g in the composition. For the clarity of the notation, we denote the set of state and action noise parameters on a trajectory by $\varepsilon = (\epsilon_{s_1}, \dots, \epsilon_{a_n})$. The derivative operator transforms function composition to the multiplication of Jacobian matrices. Therefore, Jacobian of the composite function $g^n(.)$ is the product of Jacobian matrices of function g at each state on the trajectory $\mathbb{J}_s^{g^n}(s_t, \varepsilon) = \prod_{k=1}^{n} \mathbb{J}_s^{g}(s_{t+k}, \epsilon_{s_{t+k}}, \epsilon_{a_{t+k}})$.

If the Jacobian matrices belong to a set of bounded and non-zero matrices and they are closed under multiplication, the gradient passing through the finite composition of $g^n(.)$ does not diverge nor vanish. One such set of matrices is unitary matrices, which are composed of k orthonormal vectors of size k. A model function composed of linear layers and non-linearities in between, such as the neural networks that we use in Policy Gradient (PG) algorithms, may have a unitary Jacobian matrix if the weight matrices are unitary and the non-linearity preserves the norm of the gradient passing through, e.g absolute value function. However, the model aware transition function g is composed of two functions, μ and f, that is $\mathbb{J}_s^g(s, \epsilon_s, \epsilon_a) = \mathbb{J}_s^f(s, \tilde{a}, \epsilon_s) + \mathbb{J}_s^\mu(s, \epsilon_a)\mathbb{J}_{\tilde{a}}^f(s, \tilde{a}, \epsilon_s)$.

Although the Jacobian of the policy-aware transition function with respect to input state $\mathbb{J}_s^g$ is a square matrix, the Jacobian matrices of the policy function $\mathbb{J}_s^\mu$ and the transition function $\mathbb{J}_{\tilde{a}}^f$ are not necessarily square matrices, if $|A| \neq |S|$. Hence we extend the set of unitary matrices by including a set of orthonormal k vectors of size larger than k. Stiefel manifold is the set of such matrices denoted by $St(n, k)$, such that $A \in St(n, k) : A^*A = I_k$ where n is the dimension of the orthonormal vectors and $k \leq n$ is the number of orthonormal vectors in A.

In order to represent the Jacobian matrix of a function on $St(n, k)$, the function must be an injective mapping. . In the case where $|A| < |S|$, the Jacobian matrix of the policy function can not be in the manifold $\mathbb{J}_s^\mu \notin St(n, k)$, even though $\mathbb{J}_{\tilde{a}}^f \in St(n, k)$. Hence, in our implementation we ignore the contribution of $\mathbb{J}_s^\mu \mathbb{J}_{\tilde{a}}^f$ to $\mathbb{J}_s^g$. Subjectivity constraint allows model function f to map states into a latent space with higher dimensions as this is often the case for Neural Networks (NN). Hence, the increase in dimension is allowed within the layers of the model function.

$$\begin{aligned} \max \quad & \mathbb{E}_{s_t, a_t, s_{t+1} \in \mathfrak{D}} \Big[\log p\big(s_{t+1}; f(s_t, a_t, (\xi_i, \ldots, \xi_L))\big) \Big] \\ \text{subject to} \quad & \xi_i \subset St(n_i, k_i) \quad \forall i \in (1, \ldots, L) \end{aligned} \tag{3.1}$$

The optimization problem in expression (3.1) describes the optimization of the model function with L layers. The state, action, and next state are sampled from a replay buffer $\mathfrak{D}$. The set of ξ_i's are the weight matrices of the linear layers in the model function f. The objective is to maximize the log-likelihood of the transition distribution p, usually, a Gaussian distribution, parameterized by f for the next state samples. The constraint only allows weights that are in $St(n, k)$ for the layers of the model function. However, this optimization problem is difficult to solve with high dimensional non-linear functions due to the constraints.

3.1.1 RIEMANNIAN GRADIENT DESCENT

The challenge in Gradient Descent (GD) on a manifold $\mathcal{M}$ is that the vectors in the tangent space of the manifold $\nabla \in T_p\mathcal{M}$ may lead outside of the manifold. In order to map the points in the neighboring of a point $p \in \mathcal{M}$ along the direction in a vector of tangent space ∇ back to manifold, we describe a mapping $\phi : T_p\mathcal{M} \to \mathcal{M}$, where the first order approximation of ϕ is called retraction. Moving straight on the manifold according to ϕ makes a geodesic on the manifold $\gamma_{p,\nabla}(t)$ where $t \in [0, 1]$. A geodesic stays on the manifold while following the direction of ∇ starting at point p and it satisfies $\gamma_{p,\nabla}(0) = p$. Instead of following the gradient ∇, Riemannian Gradient Descent (RGD) follows the geodesic at point $p \in \mathcal{M}$. There exists a unique geodesic for any point in $\mathcal{M}$. Hence, the RGD update rule is $\xi_{t+1} = \phi(-\nabla_\xi \mathcal{L}_f(s_t, a_t, \xi))$ where $\mathcal{L}_f$ is the loss function. In order to use RGD in this objective the mapping ϕ must be a diffeomorphism.

Instead of mapping the function gradient back to the manifold, we may re formalize the constraint optimization problem with parametrization of the matrices from Euclidean space. The new parameter matrix $\xi \in \mathbb{R}^n$ is put on the manifold by submersion $\Phi : \mathbb{R}^n \to \mathcal{M}$ and given to the objective function. Hence, the new optimization objective is unconstrained in Euclidean space.

$$\begin{array}{llcll} \max & f(\xi) & \to & \max & (f \circ \Phi)(\xi) \\ \text{s.t} & \xi \in \mathcal{M} & & \text{s.t} & \xi \in \mathbb{R}^n \end{array} \tag{3.2}$$

The original Riemannian optimization is equivalent to the one that we obtain after parametrization. The parameterization, as shown by Casado [38], does not change the optimization problem by adding additional saddle or locally optimal points. For the sake of simplicity, we explain the parametrization technique employed in this thesis with special orthogonal group $SO(n)$, which is a subset of the Stiefel manifold where the matrices are unitary. The parametrization that we will describe can be easily extended for the Stiefel manifold.

We use matrix exponential function $\exp(A) = I + \frac{A}{1!} + \frac{A^2}{2!} + \dots$ as the submersion from skew matrices to orthogonal matrices. Here, the $\exp(A)$ maps skew square matrices, which satisfies $A^T = -A$, to $SO(n)$.

$$\begin{aligned} A^T + A &= 0 \\ \exp(A^T + A) &= \exp(0) \\ \exp(A)^T \exp(A) &= I \end{aligned} \tag{3.3}$$

In Equation (3.3), the matrix exponential of a skew matrix A satisfies the orthogonality property of $SO(n)$. We applied a trivial parametrization to map matrices from Euclidean space to the space of skew matrices. Let $X \in \mathbb{R}^{n\times n}$, skew$(X) = X - X^T = A$ would satisfies the skewness property. Combining the mappings $\exp(A)$ and skew(X), we obtain submersion $\Phi(X) : \mathbb{R}^{n\times n} \to SO(n) = (\exp \circ \text{skew})(X)$ The new optimization problem is given below.

$$\max \quad \mathbb{E}_{s_t,a_t,s_{t+1}\in\mathfrak{D}}\Big[\log p\big(s_{t+1}; f(s_t, a_t, (\Phi(\xi_i), \dots, \Phi(\xi_L)))\big)\Big] \tag{3.4}$$

Both $\exp(A)$ and skew(X) are differentiable functions, but $\exp(A)$ is computationally involved. Therefore, during the policy optimization phase, we map Euclidean parameters ξ to $St(n, k)$ once and reuse them throughout the trajectory on which the VG objective is built.

3.1.2 Non-Linearity

The unitary weight parametrization allows an affine layer to preserve the norm of the gradient passing through it. Let, $g(x; W, b) = Wx + b$ be an affine layer. A function g satisfies the gradient norm preserving property if $\|\mathbb{J}_x^g(x)\nabla\|_2 = \|\nabla\|_2$. The explicit form of the inner product $\nabla^T \mathbb{J}_x^g(x)^T \mathbb{J}_x^g(x)\nabla = \nabla^T\nabla$ indicates when $\mathbb{J}_x^g \in St(n, k)$ the affine layer g satisfies the norm preserving property.

Similarly, for non-linearities, the above-mentioned condition must be met in order to obtain a model function that does not lead to gradient divergence or vanishing. One of the element-wise operations that satisfy the condition is the absolute value function $h(x) : \mathbb{R} \to \mathbb{R}^{+}$. The Jacobian of the element-wise function h is a diagonal matrix where the diagonal entries are either 1 or -1. Hence, the square of the Jacobian matrix is the Identity matrix in the set $SO(n)$. However, in practice, the absolute value function leads to poor performance, and we did not use it in our experiments.

An alternative parametric non-linearity, modReLU, is proposed by Arjovsky et al [32] that slightly reduces the gradient norm but does not hinder the model performance.

$$\sigma(x; b) = \begin{cases} (|x| + b)\frac{x}{|x|} & \text{if } |x| + b \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

Equation (3.5) describes modReLU with a learnable parameter b. In our experiments, we use modReLU in the model function extensively.

3.2 VALUE GRADIENT ON STIEFEL MANIFOLD

In the previous section, we introduced a model function f that does not lead to exploding or vanishing gradient issues. We combine this model function with the VG algorithm on long trajectories and show that it does not allow gradient divergence nor vanishing.

The value estimate contains intermediate rewards on the trajectory and the state value of the final value approximation. At every step, these terms additively contribute to the backward propagating gradient $\nabla_{s_t}^{V_t}$. The Jacobian matrix of the policy-aware transition function $\mathbb{J}_s^g$ is the only multiplicative term in the per-step evaluation of $\nabla_{s_t}^{V_t}$. Since $\mathbb{J}_s^g$ is able to preserve the gradient norm when the weight matrices of the affine layer in the model function lie on Stiefel manifold, the propagating gradient $\nabla_{s_t}^{V_t}$ does not exponentially shrink nor explode.

We ignored the path on the computational graph of the policy-aware model function g that passes through policy function $\mu(s_t)$ to state s_t due to the injectivity requirements of the affine layers whose weights lie on $St(n, k)$. Although this leads to an approximation of the original gradient estimate, which include the contribution of the policy function to state value gradient $\nabla_{s_t}^{V_t}$, we observed that this term tends to vanish if the policy function is several layers deep. Moreover, removing the path pulls out the constraints on the policy function and allows any neural networks that could be employed in Policy Gradient algorithms.

Generalized Advantage Estimate: We use GAE as our value estimation in the proposed VG algorithms. GAE provides a control for the impact of two variance reduction methods with a parameter $\lambda \in [0, 1]$. Small values of λ would weigh more on value approximation, while values that are close to 1 weigh VG objective more than value approximation. The second parameter in GAE is the truncated trajectory length n, which determines the length of the computational graph of the VG objective.

Combining the proposed model function and the value estimation, we describe the propagating gradient $\nabla_{s_t}^{GAE_t}$ and the policy gradient ∇_{θ}^{t} in the policy optimization in Equation (3.6) and (3.7) respectively.

$$\nabla_{s_t}^{GAE_t} = \mathbb{J}_s^f(\tilde{s}_t, \tilde{a}_t)\Big(\nabla_{\tilde{s}_{t+1}}^{\delta_t} + \gamma \nabla_{\tilde{s}_{t+1}}^{GAE_{t+1}}\Big) + \nabla_{\tilde{s}_t}^{\delta_t} \tag{3.6}$$

$$\nabla_{\theta}^{t} = \mathbb{J}_{\theta}^{\mu}(s_{t+1})\Big(\nabla_{\tilde{a}_t}^{\delta_t} + \mathbb{J}_{\tilde{a}}^f(\tilde{s}_t, \tilde{a}_t)\big(\nabla_{\tilde{s}_{t+1}}^{\delta_t} + \gamma \nabla_{\tilde{s}_{t+1}}^{GAE_{t+1}}\big)\Big) \tag{3.7}$$

Latent Space: It is possible to build a model function on a latent space where the encoder can be any network architecture. The propagating gradient $\nabla_{s_t}^{GAE_t}$ is not affected by the encoder network as long as the state input s of the model $f(s, a, \epsilon_s)$ lies on the latent space induced by the encoder network. This allows us to use complex encoders depending on the task. However, if the reward function depends on the environment state, which is the case if we do not train an approximate reward function that depends on the latent state, we need to make sure that the encoder is injective and its inverse is differentiable.

Reparametrization: We apply reparameterization to policy and transition distributions. Depending on the action space of the environment we either use Gaussian-based [42] $\tilde{a} = \mu_a + \sigma_a \epsilon_a$ or straight-through [43] $\tilde{a} = \mu_a + \text{sg}(a - \mu_a)$ reparametrization in the policy distribution where sg is a stop gradient function with zero Jacobian $\mathbb{J}_x^{\text{sg}}(x) = 0$. The same approach also holds for the transition distribution. However, in the model function, we need to make sure that reparameterization does not change the norm of the gradient. The Jacobian matrix of the straight-through reparameterization is the identity matrix; hence it does not affect the gradient norm. In Gaussian based reparametrization, if the Gaussian parameters μ_s and σ_s both depend on s, we would obtain a Jacobian matrix $\mathbb{J}_s^f = \mathbb{J}_s^{\mu_s} + \epsilon_s \mathbb{J}_s^{\sigma_s}$ that may change the gradient norm. Instead, we model σ_s as a parametric function that does not depend on s; hence σ_s does not contribute to the Jacobian matrix of f. The use of state-independent variance has been suggested for policy distribution in a previous work [40]. We observed that the choice of state-independent variance does not affect the performance of the model function.

3.3 VALUE GRADIENT PROXIMAL POLICY OPTIMIZATION

In this section, we describe the implementation of VG to Proximal Policy Optimization (PPO), which is one of the most prominent model-free on-policy algorithms due to its effective usage of on-policy experience. In order to take advantage of the experience and reuse it multiple times, PPO relies on long truncated trajectories. Once the agents collect the truncated trajectories the trajectory data is used in policy and value optimizations in multiple epochs, akin to supervised learning. It is often the case that the advantage estimate in PPO is built over long trajectories in which the parameter λ that determines the contribution of the future TD errors δ to the value estimation, is chosen as close to 1. Therefore, a similar value estimate with the VG objective requires a computational graph over long trajectories. Due to gradient divergence issues, applying the VG objective with an unconstrained model function is implausible.

VG-PPO aims to address the difficulty that arises from the usage of very long trajectories by providing a non-divergent and norm-preserving model function as described in the previous sections. The combined algorithm benefits from the low variance gradient estimator of VG methods and efficient on-policy data usage of PPO.

Modifications: Once the trajectory data is collected, PPO calculates the advantage of the states in the trajectories. In our implementation, we employed GAE instead of n-step value estimation. The policy optimization step uniformly samples the advantage estimate $\hat{A}_t$ and transition tuple $(s_t, a_t, s_{t+1}, r_t, d_t)$. However, in order to form a computation graph, the VG objective expects a truncated trajectory. Therefore, we instead sample consecutive trajectory chunks of size $h : h \leq n$ during the policy optimization. However, in PPO, the advantage estimate $\hat{A}^n$ is precalculated with the values of the data collecting policy, which is the same policy before the parameter updates. Since the VG objective requires a computational graph over the sampled trajectories at every update step, the advantage estimate must be recalculated by the value function of the data-collecting policy. Hence, we employ another value function V^{data} that is used to recalculate the advantage estimate $\hat{A}^\pi$ before every policy optimization step. The value function V^{data} is reassigned with the updated parameters before the policy optimization.

Algorithm 4 VG-PPO Algorithm

```
Initialize actor parameters $\theta$ and critic parameters $\phi$
Initialize model parameters $\xi$ and $V_{\text{data}}$ parameters $\zeta$
Initialize K environments $\{E_1, \dots, E_k\}$ in parallel
Initialize total time step $T$, number of epochs $E$
Initialize rollout size $n$ and chunk size $h$
Initialize replay buffer $\mathbb{D}$ for the model function
for $t \in [0, \text{floor}(T/K)]; t \leftarrow t + n$ do
    Initialize rollout store $\mathbb{T}$
    for each environment $E_i$ do
        Sample and store $n$-step rollout $\mathbb{T} \leftarrow \mathbb{T} \cup \tau^i_{t:t+n} \sim p^{\hat{\pi}}(\tau)$
    end for
    Store experience in $\mathbb{D} \leftarrow \mathbb{D} \cup \mathbb{T}$
    Update the model function $f_\xi$ with samples from $\mathbb{D}$
    Update $V^{\text{data}} \leftarrow V^\pi_\phi$
    for each epoch $\in [1, E]$ do
        for each chunk $\tau_{k:k+h} \in \mathbb{T}$ do
            Reparameterize trajectory chunk $\hat{\tau}_{k:k+h} \leftarrow \text{reparam}(\mu_\theta, f_\xi, \tau_{k:k+h})$
            Calculate Advantage estimate $\hat{A}^\pi_\theta$ using $V^{\text{data}}$ and $\hat{\tau}_{k:k+h}$
            Calculate VG estimate according to (3.9)
            Calculate $\frac{\partial}{\partial \phi}$ of the value loss with the batch
            Update the parameters $\theta$ and $\phi$
        end for
    end for
end for
```

The derivative of the PPO objective in Equation (2.11) is a piecewise function that describes when the gradient is set to zero and the update is rejected.

$$\nabla_\theta \mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_{s\sim d^\pi(s), a\sim\pi(a|s)} \begin{cases} \nabla_\theta \log \hat{\pi}(a|s;\theta) r_\theta \hat{A}^\pi & \text{if } \hat{A}^\pi > 0 \text{ and } r_\theta < 1+\epsilon) \\ \nabla_\theta \log \hat{\pi}(a|s;\theta) r_\theta \hat{A}^\pi & \text{if } \hat{A}^\pi < 0 \text{ and } r_\theta > 1-\epsilon) \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

The only term in the PPO objective that depends on θ is contained in the numerator of the ratio $r(s, a)$. Hence, the non-zero gradient is obtained by $\nabla_\theta \frac{\hat{\pi}(a|s;\theta)}{\pi(a|s)} = \nabla_\theta \log \hat{\pi}(a|s;\theta) r_\theta$. However, in the VG objective, the dependency to θ is built within the advantage estimate. In this regard, in order to achieve the same clipping behavior, we propose another objective function in Equation (3.9).

$$\mathcal{L}^{\text{CLIP}}_{\text{VG}}(\theta) = \mathbb{E}_{s\sim d^\pi(s), a\sim\pi(a|s)} = \Big[\min\big(r\hat{A}^\pi_\theta, \text{clip}(r\hat{A}^\pi_\theta, (1-\epsilon)\hat{A}^\pi, (1+\epsilon)\hat{A}^\pi)\big)\Big] \tag{3.9}$$

We simplify the notation for clarity in Equation (3.9) by removing function inputs and denoting dependency to the parameter θ by writing it as a subscript. The objective function $\mathcal{L}^{\text{CLIP}}_{\text{VG}}$ includes differentiable advantage estimate $\hat{A}^\pi_\theta$ of the data generating policy π. Since in the original objective function, the parameter θ depends only on the ratio r, clipping it would reject the gradient. Hence, the objective function $\mathcal{L}^{\text{CLIP}}_{\text{VG}}$ puts the advantage estimate inside the clip operation so that clip operation may reject the gradients as the advantage estimate $\hat{A}^\pi$ in the boundary does not depend on the parameter θ.

Algorithm 4 provides the pseudocode for the VG-PPO algorithm. Here chunks are the truncated trajectories that contain consecutive transitions from the rollout data. Chunks allow applying the VG objective on manageable-size trajectories. The computational graph is built on a trajectory chunk; hence we do not propagate the value gradients more than h. The proposed model function in this thesis proves to be crucial for implementing the VG objective to PPO. We show the comparisons of VG-PPO, VG with the proposed model function, VG with the previously proposed model function, and PG algorithms in the next chapter.

4. EXPERIMENTS

In this work we aim to show and propose a solution for gradient divergence issues in VG-based algorithms on long trajectories. Therefore, we developed an environment in which the rewards are not just delayed, but their relations between the actions are one-to-one; hence reward-action dependency is not trivial. We call this environment Delayed Reward Lookup (DRL). The second benchmark environment that we use in this thesis is a MuJoCo [13] control task Walker-2d, where the task is to control a robotic frame that achieves higher results as it traverses along a direction and is penalized if it falls. Moreover, we show a comparison of the correlation base policy gradient estimate with the reparameterization-based one. We show that when the distribution is given, reparameterization enjoys significantly lower variance in the gradient estimator without introducing a bias. In order to further observe the efficiency of VG-based policy gradient estimate, we build a maze-like discrete environment where at each discrete state the agent makes two decisions, one for navigation that consists of four possible choices and a key decision that determines the reward when it reaches the goal state. Due to the discrete dynamics of the environment, we were able to employ Markov Matrices, which do not cause gradient divergence since the set of Markov matrices is closed under multiplication. We used both an exact model function and an approximation in the form of a Markov matrix. We show that both the approximated and the exact model functions are able to specify the key decision that determines the reward.

4.1 DISCRETE STATE SPACE

In the discrete state space, Markov transition matrices are the natural choice to represent model functions. Similarly, for the policy function, if the action space is also discrete, we may represent it with a Markov matrix. We obtain a parametric and learnable Markov Matrix by applying the row-wise or column-wise softmax function

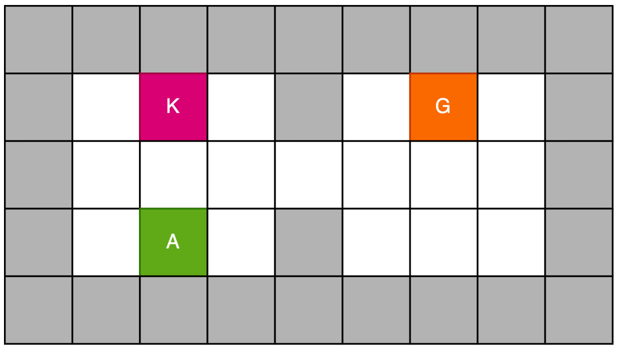


Figure 4.1 : Decision Maze environment. The cells K denotes the key state, G denotes the goal state and A denotes the location of the action.

to any real-valued matrix. The Jacobian matrix of a state value with respect to policy parameters is a product of multiple Jacobian $\mathbb{J}_s^f$ matrices as shown in the recursive Equation (3.6). Since the Jacobian matrix of the model function and the policy function are Markov matrices, due to the linearity, the product is also a Markov matrix. Hence Markov matrices provide non-divergent model and policy functions.

In this experiment, we build a discrete grid-world environment. The state space is a one-hot vector that has one 1 on the index that denotes the position of this vector represents, and the remaining indices, except the last one, are all set to 0. The state vector contains a scalar at the final index that codes the decision given in the key state. The action space is a multi-discrete space where there are two categorical actions that an agent in this environment needs to provide. The first categorical action space is related to the navigation actions left, right, up, and down. The second categorical action is a simple on-and-off action. The possible values for the intermediate reward in the environment are -1, 1, and 0. The agent is only given a non-zero reward at the goal state denoted by the letter G. In the goal state, the reward is determined by the memory value of the state. The key state, represented by K, denotes the position in which the memory cell can be modified. The action in the key states directly determines the memory value, and once it is set, the memory value does not change. Hence, the action taken in the key state determines the reward at the goal state while still satisfying the Markov properties. We name this environment Decision Maze, as shown in Figure 4.1 due to the maze-like structure and a critical decision that must be made by the agent.

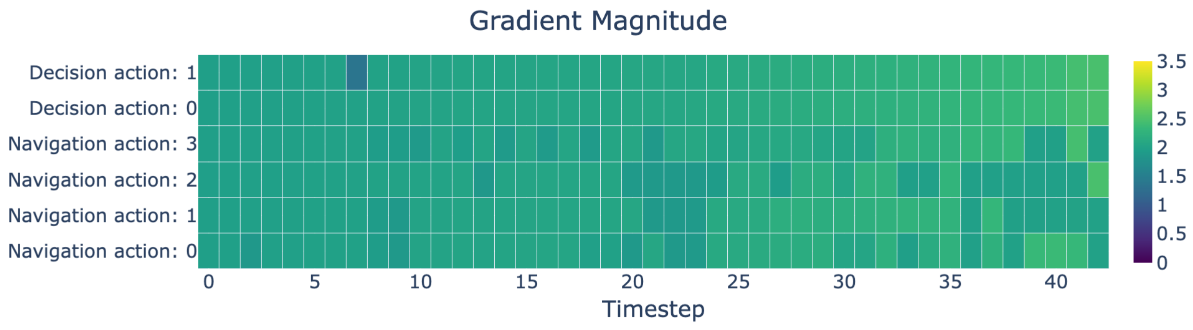


Figure 4.2 : Gradient norm of the reward function at the final state with respect to actions in a randomly sampled 43 step trajectory. Each box denotes a gradient magnitude for each action categories in the trajectory.

In our tests, we trained a Markov matrix-based model function with the data collected by a random agent. We sampled trajectories using a random policy from the environment. We show that VG objective is able to distinguish critical actions from unrelated ones. In Figure 4.2, we take the derivative of the last reward with respect to each action category throughout the sampled trajectory. The derivative with respect to decision actions is only different at one point where the key state is visited; hence VG objective is able to capture the relationship between the critical action and the final reward. In this sampled trajectory, the key state is visited at the same state where the gradient magnitude is different from the other decision action gradients.

This observation provides intuition for how the VG objective provides a low variance gradient estimate. However, for actions that are not critical or equally valuable, basic assumptions like equally distributing the reward, such as in the PG objective, may suffice. Hence, we observed that some of the environments may not have non-trivial reward-action relationships. Therefore, VG methods may not improve the variance of the gradient estimate compared to PG-based estimates if the latter is already able to capture the relations between actions and rewards. One of the best examples of this environment is grid-world or maze environments.

In order to observe the sample requirements of VG and PG-based gradient estimates in the discrete framework with Markov matrices, we conducted another experiment. We first simplified the environment so that the trajectory length is the same in order to make a better comparison by removing the navigation aspect of the environment.

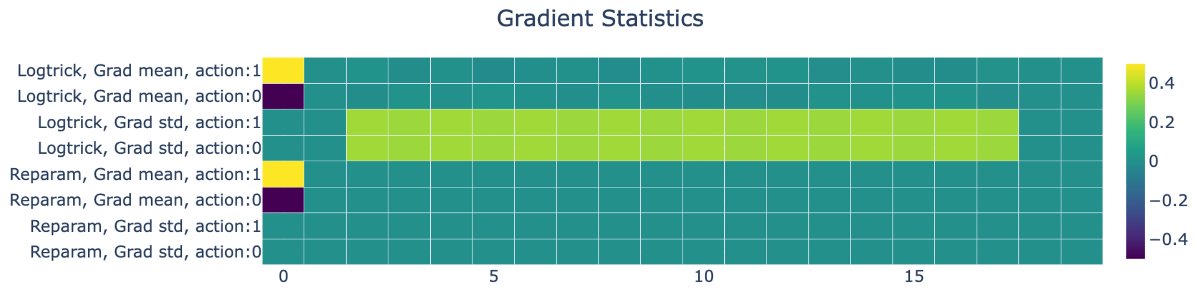


Figure 4.3 : Comparison between log-trick and reparameterization in terms of variance and bias. The trajectory length is 10, and there are 20 different discrete states. The mean and variance for each action in each state is given in the Figure.

Hence, there is only decision action on a fixed-length trajectory. The state vector contains the memory of the action performed on the first state, hence the reward function satisfies the Markov property. We experimented with both an exact Markov matrix of the transition distribution and with a learned one. Although the results shown in 4.3 are calculated with the exact matrix, the learned one performs very similarly but only slightly reduces the gradient norm through the starting state.

The statistics in Figure 4.3 is calculated using 1000 sample trajectories. The results suggest both approaches have the same mean, while log-trick, which is the mechanism used in the PG objective, introduces variance on every action gradient. The reasoning behind this is that the PG objective relies on correlation and assumes that all rewards that are observed after an action are related to that action while only one of them is related in this environment.

In the discrete case, Markov matrices provide an appropriate set of model functions that do not lead policy gradients to diverge or vanish. However, that is not generally applicable since only a small subset of environments, which are mostly toy problems, have discrete state and action spaces. Nevertheless, these experiments clearly show the promise of VG approaches.

4.2 DELAYED REWARD LOOKUP

We extend our empirical studies to continuous environments. We crafted a delayed reward environment where the state has a memory that keeps track of the first k

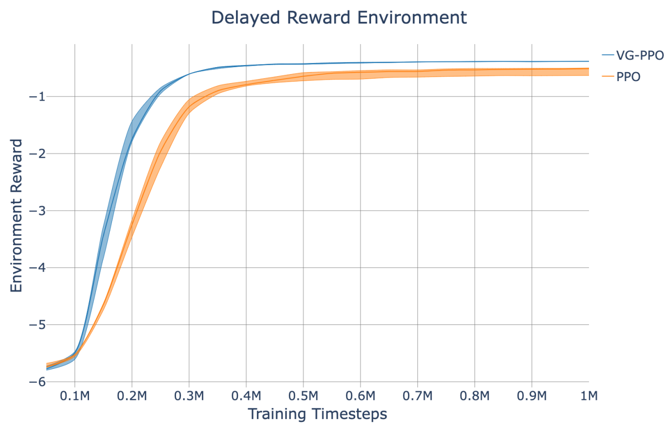
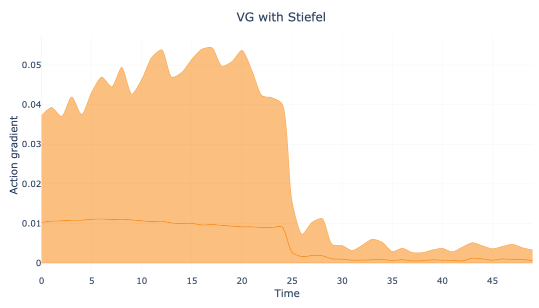


Figure 4.4 : Comparison between PPO and VG-PPO. Both algorithms use the same hyperparameters. The rollout size is set to trajectory length which is 50. The delay is set to 25. VG-PPO enjoys faster and slightly better convergence. The shaded region represents the second and the third quarters of 5 different seeds.

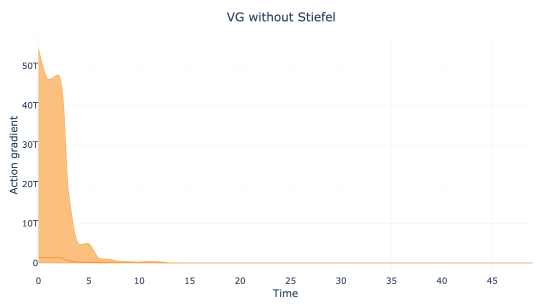
continuous actions made during the trajectory. The intermediate rewards are zero except for the last k rewards, where the reward function compares a fixed value assigned for the state of which it evaluates the intermediate reward with the index of the state vector that corresponds to the memory of one of the past actions.

Similar to the previously mentioned environment, Delayed Reward Lookup (DRL) environment has a fixed trajectory length. During our comparison, we change the trajectory length to show the effect of long trajectories in policy optimization.

The state space has the $k + n$ dimensions where k denotes the memory size and n denotes the trajectory length. Action space is continuous and bounded between -1 and 1. The objective of the environment is to learn the correct decisions that must be given at the first k steps so that the last k rewards are matched with their corresponding values. Therefore, after the first k steps, the actions are not related to the objective of the environment. We denote the reward function at the t'th state with $r_t(s_t)$ and action at time t with a_t. The reward function is defined as $r_t(s_t) = \|s_t(t - n + k) - \alpha_t\|^2$ where α_t is a fixed value for the t'th reward function in which the objective is to match that value with the corresponding index of the state vector s_t. Let $0 \leq m \leq k + n$ be any step in a trajectory, the $n + k - m$'th index of a state is only affected by the action a_m.



((a)) VG-PPO with Stiefel manifold.



((b)) SVG with unconstraint model.

Figure 4.5 : Comparison of the model functions with and without Stiefel manifold constraint. The plots show the statistics of the gradient magnitude of all actions over the length of the trajectory.

In DRL, we observed that without a constrained model function and when the trajectory length is 50, the gradient estimate of the VG objective diverges at some step in the trajectory. We compared VG-PPO and PPO algorithms and show that even with the model learning, VG-PPO converges faster and to a better overall score as shown in Figure 4.4.

We also compared VG-PPO with the SVG algorithm. We show that in long trajectories, the action gradients in the SVG algorithm tend to diverge in Figure 4.5(b). Unlike SVG, VG-PPO on the Stiefel manifold not only solves the divergence issue but has an interpretable action gradient distribution, as shown in Figure 4.5(a), such that the gradient magnitude for the first half of the actions is significantly larger, which relates to the reward mechanism of the DRL environment.

In these experiments, we show that the model function on the Stiefel manifold enables VG algorithms on long trajectories. When the model being used by the VG algorithm is unconstrained such as in [16,21], the model function causes divergent behavior in the gradient estimate that hinders the performance of VG algorithms on long trajectories.

4.3 ROBOTIC CONTROL TASK

We benchmark the proposed VG-PPO algorithm in the Walker-2d environment, one of the MuJoCo control tasks. This environment poses a challenge for model learning as it does for policy optimization due to the complex physical dynamics of the environment.

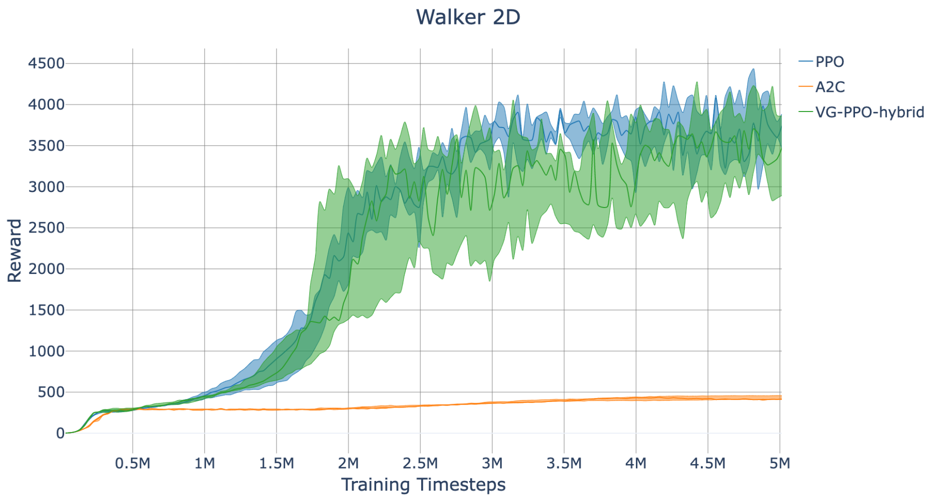


Figure 4.6 : Walker2d environment. Figure contains the comparisons of PPO, A2C, and VG-PPO-hybrid. The rollout length is set to 64 for all algorithms and remaining hyperparameters are set to the same values whenever possible. The shaded region represents the second and the third quarters of 5 different seeds.

We compared VG-PPO, a hybrid version where the loss function contains VG and PG terms, with PPO, A2C, and SVG(64). Due to the gradient issues in the SVG algorithm on long trajectories such as the one the we used in this experiment, SVG(64) could not learn the task. Hence, we did not put SVG(64) into Figure 4.6, which shows a comparison of VG-PPO with model-free algorithms.

In this environment, we observed that model approximation error hinders the performance of the VG-based gradient estimator. The complex dynamics in the walker-2d environment puts a challenge for model learning. We observed that the VG objective is quite sensitive to the model accuracy, and at the beginning of the training relying solely on the VG objective may harm the course of the training. Hence, we employed a hybrid loss in the VG-PPO-hybrid algorithm. The objective function includes the weighted sum of PG and VG terms. During the beginning of the training, where the accuracy of the model function is low, we rely solely on the PG objective. The weight of the VG term increases linearly with each training step of the agent. We observed that VG-PPO-hybrid does not lead to catastrophic gradients due to the low accuracy of the model function at the beginning of the training. However, the suggested algorithm does not outperform its model-free counterpart, yet makes it possible to use VG objective on long trajectories where the SVG algorithm completely fails. We argue that being unable to reach a sufficiently low model approximation error indicates high bias in the VG objective hence PG objective can be an alternative in that case.

5. CONCLUSIONS

In this thesis, we introduce a family of model functions with a proper learning scheme that enables Value Gradient (VG) algorithms on long truncated trajectories. Longer trajectories allow VG objective to capture long-term dependencies with significantly less variance than Policy Gradient (PG) based gradient estimates. In the discrete state space experiments, we show that VG-based gradient estimate is able to capture the long-term reward-action relations on a single randomly sampled trajectory if the model is given or it is well approximated. Furthermore, we show that compared to PG based gradient estimate, VG provides almost zero variance gradient estimate.

The focus of this thesis is on building a non-divergent model function such that it preserves the gradient norm passing through it on a trajectory. We show that if the neural network that represents the approximate model function contains linear layers that have unitary weight matrices and non-linearities that are able to preserve the gradient norm, the model function does not lead to exploding or vanishing gradients during the policy optimization phase. In order to maintain the characteristic of the proposed family of neural networks, we employ Riemannian Gradient Descent (RGD). We model the optimization problem as a constraint problem where the constraints are forcing weight matrices to lie on the Stiefel manifold (St). We argued that parameterization of the weight matrices with matrix exponential function provides a solution for the proposed optimization problem.

In addition, we implement the VG objective on model-free Proximal Policy Optimization (PPO) algorithm that enjoys sample efficiency via a sample reuse mechanism. The combined algorithm is called VG-PPO. The proposed model function is a critical addition in VG-PPO since PPO typically relies on long truncated trajectories and Monte Carlo (MC) sampling-like value estimations. We show the efficiency of VG-PPO in a continuous task where the reward-action dependencies are long and not trivial. We observed that a model function on the Stiefel manifold reflects the learned dynamics of the environment onto the policy gradient estimate besides preventing divergence.

We further test VG-PPO in a challenging robotic simulation called Walker-2d. However, due to the model approximation errors, the training process of VG-PPO becomes unstable. Therefore, we introduced the VG-PPO-hybrid algorithm that includes both VG and PG terms in the policy optimization objective. We observed an improvement, but VG-PPO-hybrid lacks behind the performance of its model-free counterpart, albeit being able to utilize value estimate on 64-step truncated trajectories. We observed that the model approximation plays a critical role in the VG objective since the VG-based gradient estimate is sensitive to the approximation errors in the model function. We argued that higher capacity models would make it possible to run VG-PPO on environments with complex dynamics. Our observations suggest a hybrid approach may better generalize the gradient estimates in policy optimization. When the model approximation is not sufficiently low, the algorithm may rely more on the PG term and if the approximation error is low, the algorithm may increase the weight of the VG term in the objective.

5.1 FUTURE DIRECTIONS

In this thesis, we show a promising approach for designing model functions that enables VG algorithms on long trajectories. Yet, there are several possible improvements or modifications that can be made for the proposed algorithms. We state some of them that we find valuable.

Reparameterized Distribution: For the reparameterization of the model function, we assumed the transition distribution is a Gaussian distribution with independent axes. Instead, a flow function that transforms a trivial distribution to match the target distribution may represent the model function. This would allow fitting complex transition distributions while still providing differentiable samples. However, the flow function must be designed such that it does not lead to exploding or vanishing gradient problem. In the design of such flow functions, Stiefel manifold may provide the matrices for the transformations in the flow function.

Attention Mechanism: In this work, we proposed a model function similar to recurrent neural networks in which it is recursively called on a trajectory. Instead, an alternative approach can be employed for capturing long-term dependencies between actions and rewards. Attention in time is a promising direction for policy optimization, and VG may provide a mathematical baseline for such approaches. An unsupervised attention-based sequence learning function may learn to attend to related actions or rewards during the unsupervised training. This attention map can be utilized in policy optimization, similar to the VG approach, to improve the gradient estimate.

Multi Task Learning: We observed in our empirical studies, a well-approximated model function enables VG objective to provide low bias and variance gradient estimates. However, training a model function and performing policy optimization simultaneously may cause a delay in the model approximation performance. Furthermore, model learning is independent from tasks and can be seen as an unsupervised problem given diverse enough interaction data. Hence a learned model can be shared among several related tasks where a value approximation can not be shared due to its direct relation with the task and the related reward. Hence, VG offers a scalable approach for multi-task learning compared to model-free PG algorithms. We argue that multi-task learning can benefit from VG approaches more than single-task learning.

REFERENCES

[1] **Sutton, R.S. and Barto, A.G.** (2018). *Reinforcement Learning: An Introduction*, The MIT Press, second edition, `http://incompleteideas.net/book/the-book-2nd.html`.

[2] **Tesauro, G.** (1994). TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play., *Neural Comput.*, *6*(2), 215–219, `http://dblp.uni-trier.de/db/journals/neco/neco6.html#Tesauro94`.

[3] **Williams, R.J.** (1992). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning, *Mach. Learn.*, *8*(3–4), 229–256, `https://doi.org/10.1007/BF00992696`.

[4] **Schulman, J.**, **Moritz, P.**, **Levine, S.**, **Jordan, M.I. and Abbeel, P.** (2016). High-Dimensional Continuous Control Using Generalized Advantage Estimation, ***Y. Bengio and Y. LeCun**, editors, 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, `http://arxiv.org/abs/1506.02438`.

[5] **Schulman, J.**, **Wolski, F.**, **Dhariwal, P.**, **Radford, A. and Klimov, O.** (2017). Proximal Policy Optimization Algorithms., *CoRR*, *abs/1707.06347*, `http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17`.

[6] **Mnih, V.**, **Badia, A.P.**, **Mirza, M.**, **Graves, A.**, **Lillicrap, T.**, **Harley, T.**, **Silver, D. and Kavukcuoglu, K.** (2016). Asynchronous Methods for Deep Reinforcement Learning, ***M.F. Balcan and K.Q. Weinberger**, editors, Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, PMLR, New York, New York, USA, pp.1928–1937, `https://proceedings.mlr.press/v48/mniha16.html`.

[7] **Hafner, D.**, **Lillicrap, T.P.**, **Ba, J. and Norouzi, M.** (2020). Dream to Control: Learning Behaviors by Latent Imagination, *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, `https://openreview.net/forum?id=S1lOTC4tDS`.

[8] **Hafner, D.**, **Lillicrap, T.P.**, **Norouzi, M. and Ba, J.** (2021). Mastering Atari with Discrete World Models, *9th International Conference on*

Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, `https://openreview.net/forum?id=0oabwyZbOu`.

[9] **Janner, M., Fu, J., Zhang, M. and Levine, S.** (2019). When to Trust Your Model: Model-Based Policy Optimization, ***H. Wallach**, **H. Larochelle**, **A. Beygelzimer**, **F. d'Alché-Buc**, **E. Fox and R. Garnett**, editors, Advances in Neural Information Processing Systems*, volume 32, Curran Associates, Inc., `https://proceedings.neurips.cc/paper/2019/file/5faf461eff3099671ad63c6f3f094f7f-Paper.pdf`.

[10] **Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D.** (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search, *Nature*, *529*(7587), 484–489.

[11] **McGrath, T., Kapishnikov, A., Tomasev, N., Pearce, A., Hassabis, D., Kim, B., Paquet, U. and Kramnik, V.** (2021). Acquisition of Chess Knowledge in AlphaZero, *CoRR*, *abs/2111.09259*, `https://arxiv.org/abs/2111.09259`, `2111.09259`.

[12] **Bellemare, M.G., Naddaf, Y., Veness, J. and Bowling, M.** (2012). The Arcade Learning Environment: An Evaluation Platform for General Agents, *Journal of Artificial Intelligence Research*, *Vol. 47*, 253–279, `http://arxiv.org/abs/1207.4708`, cite arxiv:1207.4708.

[13] **Todorov, E., Erez, T. and Tassa, Y.** (2012). MuJoCo: A physics engine for model-based control, *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.5026–5033.

[14] **Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R.H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Sepassi, R., Tucker, G. and Michalewski, H.** (2019). Model-Based Reinforcement Learning for Atari, *CoRR*, *abs/1903.00374*, `http://arxiv.org/abs/1903.00374`, `1903.00374`.

[15] **Pathak, D., Agrawal, P., Efros, A.A. and Darrell, T.** (2017). Curiosity-driven Exploration by Self-supervised Prediction, *ICML*.

[16] **Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T. and Tassa, Y.** (2015). Learning Continuous Control Policies by Stochastic Value Gradients, ***C. Cortes**, **N. Lawrence**, **D. Lee**, **M. Sugiyama and R. Garnett**, editors, Advances in Neural Information Processing Systems*, volume 28, Curran Associates, Inc., `https://proceedings.neurips.cc/paper/2015/file/148510031349642de5ca0c544f31b2ef-Paper.pdf`.

[17] **Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D.** (2016). Continuous control with deep reinforcement learning, ***Y. Bengio and Y. LeCun**, editors, 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, `http://arxiv.org/abs/1509.02971`.

[18] **Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. and Riedmiller, M.** (2014). Deterministic Policy Gradient Algorithms, ***E.P. Xing and T. Jebara**, editors, Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, PMLR, Bejing, China, pp.387–395, `https://proceedings.mlr.press/v32/silver14.html`.

[19] **Barth-Maron, G., Hoffman, M.W., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N. and Lillicrap, T.P.** (2018). Distributed Distributional Deterministic Policy Gradients, *CoRR*, *abs/1804.08617*, `http://arxiv.org/abs/1804.08617`, `1804.08617`.

[20] **Fujimoto, S., van Hoof, H. and Meger, D.** (2018). Addressing Function Approximation Error in Actor-Critic Methods, *CoRR*, *abs/1802.09477*, `http://arxiv.org/abs/1802.09477`, `1802.09477`.

[21] **Amos, B., Stanton, S., Yarats, D. and Wilson, A.G.** (2021). On the Model-Based Stochastic Value Gradient for Continuous Reinforcement Learning learning, ***A. Jadbabaie, J. Lygeros, G.J. Pappas, P. A.nbsp;Parrilo, B. Recht, C.J. Tomlin and M.N. Zeilinger**, editors, Proceedings of the 3rd Conference on Learning for Dynamics and Control*, volume144 of *Proceedings of Machine Learning Research*, PMLR, pp.6–20, `https://proceedings.mlr.press/v144/amos21a.html`.

[22] **Lezcano-Casado, M.** (2019). Trivializations for gradient-based optimization on manifolds, *Advances in Neural Information Processing Systems, NeurIPS*, pp.9154–9164.

[23] **Fairbank, M. and Alonso, E.** (2012). Value-gradient learning, *The 2012 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

[24] **Haarnoja, T., Zhou, A., Abbeel, P. and Levine, S.** (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, ***J. Dy and A. Krause**, editors, Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, PMLR, pp.1861–1870, `https://proceedings.mlr.press/v80/haarnoja18b.html`.

[25] **Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H. and Bengio, Y.** (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation, *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*.

[26] **Hochreiter, S. and Schmidhuber, J.** (1997). Long short-term memory, *Neural computation*, *9*(8), 1735–1780.

[27] **Li, C.**, **Wang, Y.**, **Chen, W.**, **Liu, Y.**, **Ma, Z.M. and Liu, T.Y.** (2022). Gradient Information Matters in Policy Optimization by Back-propagating through Model, *International Conference on Learning Representations*, `https://openreview.net/forum?id=rzvOQrnclO0`.

[28] **Ma, M.**, **D'Oro, P.**, **Bengio, Y. and Bacon, P.L.** (2021). Long-Term Credit Assignment via Model-based Temporal Shortcuts, *Deep RL Workshop NeurIPS 2021*, `https://openreview.net/forum?id=doy35IAGewq`.

[29] **Vaswani, A.**, **Shazeer, N.**, **Parmar, N.**, **Uszkoreit, J.**, **Jones, L.**, **Gomez, A.N.**, **Kaiser, Ł. and Polosukhin, I.** (2017). Attention is all you need, *Advances in Neural Information Processing Systems*, pp.5998–6008.

[30] **Zheng, Q.**, **Zhang, A. and Grover, A.** (2022). Online Decision Transformer, ***K. Chaudhuri**, **S. Jegelka**, **L. Song**, **C. Szepesvari**, **G. Niu and S. Sabato**, editors, Proceedings of the 39th International Conference on Machine Learning*, volume162 of *Proceedings of Machine Learning Research*, PMLR, pp.27042–27059, `https://proceedings.mlr.press/v162/zheng22c.html`.

[31] **Janner, M.**, **Li, Q. and Levine, S.** (2021). Offline Reinforcement Learning as One Big Sequence Modeling Problem, ***M. Ranzato**, **A. Beygelzimer**, **Y. Dauphin**, **P. Liang and J.W. Vaughan**, editors, Advances in Neural Information Processing Systems*, volume 34, Curran Associates, Inc., pp.1273–1286, `https://proceedings.neurips.cc/paper/2021/file/099fe6b0b444c23836c4a5d07346082b-Paper.pdf`.

[32] **Arjovsky, M.**, **Shah, A. and Bengio, Y.** (2016). Unitary evolution recurrent neural networks, *International conference on machine learning*, PMLR, pp.1120–1128.

[33] **Agarap, A.F.** (2018). *Deep Learning using Rectified Linear Units (ReLU)*, `http://arxiv.org/abs/1803.08375`, cite arxiv:1803.08375Comment: 7 pages, 11 figures, 9 tables.

[34] **Deng, L.** (2012). The mnist database of handwritten digit images for machine learning research, *IEEE Signal Processing Magazine*, *29*(6), 141–142.

[35] **Wisdom, S.**, **Powers, T.**, **Hershey, J.R.**, **Roux, J.L. and Atlas, L.** (2016). Full-Capacity Unitary Recurrent Neural Networks, *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, Curran Associates Inc., Red Hook, NY, USA, p.4887–4895.

[36] **Maduranga, K.D.G.**, **Helfrich, K.E. and Ye, Q.** (2019). Complex Unitary Recurrent Neural Networks Using Scaled Cayley Transform, *Proceedings*

of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI'19/IAAI'19/EAAI'19, AAAI Press, `https://doi.org/10.1609/aaai.v33i01.33014528`.

[37] **Helfrich, K.**, **Whimott, D. and Ye, Q.** (2018). Orthogonal recurrent neural networks with scaled Cayley transform, ***J. Dy and A. Krause**, editors, 35th International Conference on Machine Learning, ICML 2018*, 35th International Conference on Machine Learning, ICML 2018, pp.3133–3143, funding Information: This research was supported in part by NSF Grants DMS-1317424 and DMS-1620082. Publisher Copyright: © 2018 by authors.All right reserved.; null ; Conference date: 10-07-2018 Through 15-07-2018.

[38] **Lezcano-Casado, M.** (2019). Trivializations for Gradient-Based Optimization on Manifolds, Curran Associates Inc., Red Hook, NY, USA.

[39] **Paszke, A.**, **Gross, S.**, **Massa, F.**, **Lerer, A.**, **Bradbury, J.**, **Chanan, G.**, **Killeen, T.**, **Lin, Z.**, **Gimelshein, N.**, **Antiga, L.**, **Desmaison, A.**, **Kopf, A.**, **Yang, E.**, **DeVito, Z.**, **Raison, M.**, **Tejani, A.**, **Chilamkurthy, S.**, **Steiner, B.**, **Fang, L.**, **Bai, J. and Chintala, S.**, (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library, Advances in Neural Information Processing Systems 32, Curran Associates, Inc., pp.8024–8035, `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-lea` `pdf`.

[40] **Schulman, J.**, **Levine, S.**, **Abbeel, P.**, **Jordan, M. and Moritz, P.** (2015). Trust Region Policy Optimization, ***F. Bach and D. Blei**, editors, Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, PMLR, Lille, France, pp.1889–1897, `https://proceedings.mlr.press/v37/schulman15.html`.

[41] **Kakade, S. and Langford, J.** (2002). Approximately Optimal Approximate Reinforcement Learning, *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p.267–274.

[42] **Kingma, D.P. and Welling, M.** (2014). Auto-Encoding Variational Bayes, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, `http://arxiv.org/abs/1312.6114v10`.

[43] **Bengio, Y.**, **Léonard, N. and Courville, A.C.** (2013). Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation, *CoRR*, *abs/1308.3432*, `http://arxiv.org/abs/1308.3432`, `1308.3432`.