Minimax and entropic proximal policy optimization

Minimax und entropisch proximal Policy-Optimierung
Master-Thesis von Yunlong Song aus Jiangxi
August 2018

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Minimax and entropic proximal policy optimization
Minimax und entropisch proximal Policy-Optimierung

Vorgelegte Master-Thesis von Yunlong Song aus Jiangxi

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Heinz Koeppl
3. Gutachten: Boris Belousov

Tag der Einreichung:

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Yunlong Song, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum / Date: Unterschrift / Signature:

_____ _____

Abstract

First-order gradient descent is to date the most commonly used optimization method for training deep neural networks, especially for networks with shared parameters, or recurrent neural networks (RNNs). Policy gradient methods provide several advantages over other reinforcement learning algorithms; for example, they can naturally handle continuous state and action spaces. In this thesis, we contribute two different policy gradient algorithms that are straightforward to implement and effective for solving challenging environments, both methods being compatible with large nonlinear function approximations and optimized using stochastic gradient descent.

First, we propose a new family of policy gradient algorithms, which we call *minimax entropic policy optimization (MMPO)*. The new method combines the trust region policy optimization and the idea of minimax training, in which stable policy improvement is achieved by formulating the KL-divergence constraint in the trust region policy optimization (TRPO) as a loss function with a ramp function transformation, and then, carrying out a minimax optimization between two stochastic gradient optimizers, one optimizing the "surrogate" objective and another maximizing the ramp-transformed KL-divergence loss function. Our experiments on several challenging continuous control tasks demonstrate that MMPO method achieves comparable performance as TRPO and proximal policy optimization (PPO), however, is much easier to implement compared to TRPO and guarantees that the KL-divergence bound to be satisfied.

Second, we investigate the use of the f-divergence as a regularization to the policy improvement, where the f-divergence is a general class of functional measuring the divergence between two probability distributions with the KL-divergence being a special case. The f-divergence can be either treated as a hard constraint or added as a soft constraint to the objective. We propose to treat the f-divergence as a soft constraint by penalizing the policy update step via a penalty term on the f-divergence between successive policy distributions. We term such an unconstrained policy optimization method as f-divergence penalized policy optimization (f-PPO). We focus on a one-parameter family of α-divergences, a special case of f-divergences, and study influences of the choice of divergence functions on policy optimization. The empirical results on a series of MuJoCo environments show that f-PPO with a proper choice of α-divergence is effective for solving challenging continuous control tasks, where α-divergences act differently on the policy entropy, and hence, on the policy improvement.

Zusammenfassung

Gradientenabstieg erster Ordnung ist heutzutage die am meisten genutzte Methode, um Neuronale Netze mit mehreren Schichten trainieren. Vor allem Netze, die ihre Parameter teilen, oder Rekurrente Neuronale Netze (RNNs), werden mit dem Gradientenabstieg erster Ordnung trainiert. Policy-Gradient-Methoden haben im Vergleich zu anderen Reinforcement Learning Algorithmen viele Vorteile. Sie können z.B. mit kontinuierlichen Zustands- und Aktionsräumen umgehen. In dieser Thesis stellen wir zwei verschiedene Policy-Gradienten-Methoden vor, die unkompliziert zu implementieren sind und, die anspruchsvolle Probleme effektiv lösen. Beide Methoden sind mit nichtlinearen Funktionsapproximationen kompatibel und sie werden mit der Methode des stochastischen Gradientenabstiegs optimiert.

Als Erstes stellen wir eine neue Famile der Policy-Gradient-Methoden vor, welche als *minimax entropisch Policy-Optimierung* (MMPO) Methode genannt wird. Die neue Methode kombiniert die Vertrauensbereich-Policy-Optimierung und die Idee des minimax Trainings. Beim minimax Training wird eine stabile Policy-Verbesserung durch das Formulieren der KL-Divergenz Nebenbedingung in TRPO als eine Kostenfunktion mit einer Rampenfunktion erreicht, was dann als eine minimax Optimierung zwischen zwei stochastischen Gradientenabstieg-Optimierern optimiert wird. Einer der Optimierer maximiert das Kostenfunktionssurrogat, während der andere Optimierer die Rampentransformierte KL-Divergenz Kostenfunktion minimiert. Unsere anspruchsvollen Experimente mit mehreren kontinuierlichen Regelproblemen zeigen, dass die MMPO Methode vergleichbare Performanz wie das TRPO und das PPO erreicht. Dennoch ist es im Vergleich zu TRPO viel einfacher zu implementieren und garantiert das Einhalten der KL-Divergenz Beschränkung.

Als Zweites, untersuchen wir die f-Divergenz als eine Regularisierung für die Policy Verbesserung, wobei die f-Divergenz eine allgemeine Klasse zur Divergenzuntersuchung zwischen zwei Wahrscheinlichkeitsdichtefunktionen ist. Die KL-Divergenz ist ein Sonderfall der f-Divergenz. Die f-Divergenz kann sowohl als harte Nebenbedingung, als auch eine

weiche Nebenbedingung behandelt werden. Wir beabsichtigen, sie als eine weiche Nebenbedingung zu nutzen, in dem wir den Policy Update zwischen Policy Distributionen mit einem Bestrafungsterm der f-Divergenz sukzessive bestrafen. Wir nennen solch eine Policy-Optimierung ohne Nebenbedingungen als die *f-Divergenz bestrafte Policy-Optimierung* (f-PPO) Methode. Wir fokussieren uns hierbei auf eine Famile mit einem Parameter der α-Divergenzen, einem Sonderfall der f-Divergenzen und untersuchen den Einfluss der Wahl der Divergenzfunktionen auf die Policy-Optimierung. Die empirischen Ergebnisse auf eine Reihe von MuJoCo Umgebungen zeigen, dass die f-PPO mit einer passenden Wahl der α-Divergenz, anspruchsvolle kontinuierliche Regelungsaufgaben effektive löst, wobei die α-Divergenz unterschiedlich auf die Policy Entropie und somit auf die Policy-Verbesserung wirkt.

Acknowledgments

I would like to first express my sincere appreciation to my thesis advisor Boris Belousov for the interesting topic, his patient introduction at the early stage, and his insightful comments and careful examination of this thesis. He consistently encouraged me to try out new ideas but steered me in the right direction whenever he thought I needed it. I would also like to acknowledge Prof. Dr. Jan Peters and Prof. Dr. Heinz Koeppl, who provided me the great opportunity to work in their excellent research groups: “Intelligent Autonomous Systems” and “Bioinspired Communication Systems”.

I must express my very great gratitude to my parents for giving me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Many thanks to Rong Zhi, Simone Parisi, Adrian Sosic and Onur Celik for helpful advice and discussions.

Contents

Figures and tables

List of Figures

List of Tables

Abbreviations, symbols and operators

List of abbreviations

Notation	Description
AC	Actor-critic
ACKTR	Actor-critic using Kronecker-Factored trust region
CNN	Convolutional neural network
DL	Deep learning
DP	Dynamic programming
f-PPO	f-divergence penalized policy optimization
GAN	Generative adversarial network
KL-divergence	Kullback-Leibler divergence
MDP	Markov decision process
MLP	Multilayer perceptron
MMPO	Minimax entropic policy optimization
MSE	Mean squared error
POMDP	Partially observable Markov decision process
PPO	Proximal policy optimization
REPS	Relative entropy policy search
RL	Reinforcement learning
RNN	Recurrent neural network
SARSA	State-action-reward-state-action
SGD	Stochastic gradient descent
TD learning	Temporal-difference learning
TRPO	Trust region policy optimization

List of symbols

Notation	Description
$\boldsymbol{\theta}$	Parameter vector of target policy
$J(\pi_{\boldsymbol{\theta}})$	Performance measure for policy $\pi_{\boldsymbol{\theta}}$
A	Advantage function
δ	TD-error
$\mathbf{F}$	Fisher information matrix
$\mathbf{H}$	Hessian matrix
D_{KL}	Kullback–Leibler divergence
D_f	f-Divergence
D_α	α-Divergence
$\pi_{\boldsymbol{\theta}}$	Parametric policy corresponding to parameter vector $\boldsymbol{\theta}$
$\pi(\mathbf{a}\|\mathbf{s})$	Probability of taking action $\mathbf{a}$ in state $\mathbf{s}$ under policy π
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
γ	Discount factor
V	Estimation of state-value function
Q	Estimation of action-value function
G	Long-term expected discounted return

List of operators

Notation	Description	Operator
$\nabla_{\boldsymbol{\theta}}$	Derivative with respect to paramterer vector $\boldsymbol{\theta}$	$\nabla_{\boldsymbol{\theta}}(\bullet)$
$\mathbb{E}$	Expectation of samples	$\mathbb{E}(\bullet)$
$\text{argmax}_{\boldsymbol{\theta}}$	Parameter vector $\boldsymbol{\theta}$ at which the function values are maximized	$\text{argmax}_{\boldsymbol{\theta}}(\bullet)$
$\text{argmin}_{\boldsymbol{\theta}}$	Parameter vector $\boldsymbol{\theta}$ at which the function values are minimized	$\text{argmin}_{\boldsymbol{\theta}}(\bullet)$
T	Transpose of a matrix	$(\bullet)^T$
log	Natural logarithm	$\log(\bullet)$
$\|\|\|$	Absolute value	$\|\bullet\|$
max	The ramp function	$\max(\bullet, 0)$
$\max_{\boldsymbol{\theta}}$	Maximize an objective with respect to the parameter vector $\boldsymbol{\theta}$	$\max_{\boldsymbol{\theta}}(\bullet)$
$\min_{\boldsymbol{\theta}}$	Minimize a loss function with respect to the parameter vector $\boldsymbol{\theta}$	$\min_{\boldsymbol{\theta}}(\bullet)$

1 Introduction

For the past decade, there has been a rapid rise in the development of artificial intelligence (AI). For example, in 2012 the Google Brain team made it possible for the computer to recognize over 22,000 object categories from a large visual database (ImageNet) that contains around 14 million natural images [2]. Facebook created a face recognition system that learns to identify human faces with 97.35% of accuracy that rivals human performance in 2014 [3]. The development of deep learning algorithms and increasing computer performance are the catalysts for the progress of AI boom. Deep learning (DL) is the study of how to generalize underlying features using deep neural networks that provides powerful nonlinear function approximation and generalization. However, the training of deep neural networks, such as supervised learning for image classifications, requires a large number of data and human labels, which is intractable in the most real-world scenario. In addition, for artificial agents to reach their full potential, enabling the agent to observe the world by generalizing the underlying features of observed objects is far from enough. Instead, learning how to behave in and interact with the environment like human does is crucial for general AI. For example, to develop robots that can be used in automotive manufacturing would potentially increase the productivity and decrease the need for human labor.

Reinforcement learning (RL) is a branch of AI that is concerned with how to make optimal sequential decisions in an environment through trial-and-error learning scheme. RL algorithms are often used to solve problems in a stochastic environment where human supervision is not possible. The performance of RL heavily dependent on the state representation (or state features) of the environment. However, in reality, the state space needed to represent the solutions becomes so large that using traditional RL methods, e.g. tabular methods, becomes infeasible. It was in the past fewer years during which RL gained more importance and interest due to recent breakthroughs in deep reinforcement learning (deep RL), where RL algorithms and deep neural networks training are combined together. The powerful function approximation properties and features representation of deep neural networks can deal efficiently with the curse of dimensionality, unlike tabular and traditional non-parametric methods. For example, in 2015, DeepMind presented a deep RL system known as a deep Q-network (DQN) [4] that combines deep neural networks and reinforcement learning at scale for the first time and is capable of achieving human-level gameplay on a diverse range of Atari 2600 games just using raw images as inputs. A year later, DeepMind's AlphaGo [5], an AI program developed using RL methods, defeats 18-time world champion Lee Sedol in a five-game Go match. Furthermore, OpenAI made a big splash in June 2018 by beating amateur human teams using an AI team of pre-trained neural networks in 5v5 match-ups at Dota 2 [6], and in July 2018 DeepMind's AI agents exceed human-level gameplay at a game of capture a flag (Quake III) [7], both matches require combined efforts and long-term planning of AI agents.

Learning how to play video games might be an interesting challenge for research purpose, but solving complex real-world problems, such as autonomous driving, is an ultimate goal of deep RL. However, despite recent breakthroughs and rising popularity in deep RL, many challenging problems need to be addressed in the quest for general AI. One of the long-standing challenges in deep RL is that the training for deep neural network policies is rather unstable and data inefficient, since the policy gradient is calculated using sampled trajectories and Monte Carlo approximation which is known to be biased and data-heavy and the training procedure requires the agent to repeatedly interact with the environment. Experience replay [8] turned out to be essential for the success of deep Q-learning, which was followed by further improvements in sample efficiency. However, finding the optimal action with respect to the Q-function is generally infeasible in continuous domains.

More advanced optimization methods that use (quasi-) second-order gradient information to update parameters along the steepest descent direction represent an alternative way of alleviating the sample efficiency problem, e.g., natural policy gradients [9]. However, the Fisher information matrix and its inverse, required by the natural policy gradient method, are expensive to compute and therefore impractical in deep RL. Trust region policy optimization (TRPO) [10] and Actor-critic using Kronecker-Factored trust region (ACKTR) [11] avoid explicitly computing the inverse Fisher matrix by using Fisher-vector products [12] or the Kronecker-factored approximated curvature (K-FAC) method [13]. However, TRPO requires a large number of samples in each batch for accurately estimating the curvature and several conjugate gradient steps for a single parameter update, and ACKTR is not straightforward to implement with Recurrent neural networks (RNNs) [14] or architectures with shared parameters.

First-order gradient methods play an important role in diverse applications of deep learning such as image recognition, natural language processing, deep RL, etc. With many advanced optimizers [15], gradient descent is by far the most

common way to optimize neural networks. For example, in deep RL, proximal policy optimization (PPO) [16] achieved state-of-the-art results on several deep RL benchmark tasks by optimizing a clipped "surrogate" objective and a mean square error loss (MSE) function of the state-value function with the stochastic gradient descent (SGD) optimizer and multiple epochs of optimization using the same sampled trajectory.

1.1 Contributions

In this thesis, we contribute two different policy gradient algorithms, the minimax entropic policy optimization (MMPO) and the f-divergence penalized policy optimization (f-PPO) that are straightforward to implement and effective for solving challenging environments with high-dimensional state and action spaces. Both methods represent the policy with function approximation methods, e.g., nonlinear function approximations, and optimize the policy with SGD. In addition, we investigate the use of f-divergences as hard constraints for policy optimization, where the constrained problem can be solved using the linear and quadratic approximation scheme that comes from TRPO [10] and yields the same approximated solution as TRPO.

1. **Minimax entropic policy optimization (MMPO)**: We propose a new family of the policy gradient method for RL, in which we combine the idea of minimax optimization [17] and trust region policy optimization [10]. The new method, which we called minimax entropic policy optimization (MMPO), tries to maximize an objective whose gradients with respect to the policy parameters indicate the direction of finding an optimal policy, and simultaneously minimize a secondary objective (a ramp-transformed KL-divergence constraint) to fulfill the KL-divergence bound. Thus, the optimization process can be viewed as a minimax game between two stochastic gradient optimizers, similar with generative adversarial training [17] [18] [19] where a minimax game is conducted between two models (a generator and a discriminator). The optimization of MMPO method only requires first-order gradient information, it is straightforward to combine with other neural network structure such as RNNs or apply advanced regularization techniques such as dropout, which has been an issue in other second-order policy optimization methods such as TRPO [10] or ACKTR [11].

2. **f-divergence constrained policy optimization**: We investigate to use a more general class of f-divergence as the similarity measure that quantifies the similarity between successive policy distributions during policy improvement, and then, ensure stable policy updates by restricting the update step inside a trust region that is determined by the f-divergence. We formulate a constrained policy optimization problem by considering the f-divergence bound as a hard constraint and solve the constrained problem approximately using the linear and quadratic approximation to the objective and constraint respectively, like the TRPO method [10] where a Kullback-Leibler (KL) divergence is used for the constraint. However, both constraints (f-divergence and KL-divergence) are generally set to be a small number and for vanishingly small policy update steps, any twice differentiable f-divergence can be replaced by the Fisher information metric that is the quadratic approximation of the KL-divergence, f-divergence constrained policy optimization yield the same update rule as the TRPO solution.

3. **f-divergence penalized policy optimization (f-PPO)**: We propose to treat the f-divergence as a soft constraint by penalizing the policy update step via a penalty term that is proportional to the f-divergence between policy distributions. We term such an unconstrained policy optimization method as the f-divergence penalized policy optimization (f-PPO) method. We focus on a one-parameter family of α-divergences, a special case of f-divergences, and study influences of the choice of divergence functions on policy optimization. The empirical results on a series of MuJoCo environments show that f-PPO with a proper choice of α-divergence is effective for solving challenging continuous control tasks, where α-divergences act differently on the policy entropy, and hence, on the policy improvement.

1.2 Outline

The thesis is organized as follows:

- In chapter 2, we introduce the general setting of reinforcement learning problems, including the Markov decision process (MDP), value functions, policy representations, etc. Afterward, we discuss important elements in deep learning, such as feedforward neural works, and advanced stochastic gradient descent optimization methods. In the end, we talk about several popular and related policy search methods, e.g., REPS [20], PPO [16].

- In chapter 3, we introduce the minimax entropic policy optimization (MMPO) method that has close connections with TRPO and the idea of minimax optimization, where the policy is represented using large nonlinear function approximations. We evaluated the MMPO method via a series of comparisons of the learning curves, the entropy performance, and the KL-divergence dynamics on several MuJoCo environments.

- In chapter 4, we provide background on the f-divergence, and its special case the α-divergence, for measuring the difference between two probability distributions. We discussed the idea of constraining policy update steps using the f-divergence and solving the constrained policy optimization problem approximately, like the TRPO [10]. Furthermore, we introduce the f-divergence penalized policy optimization (f-PPO) method and study the effects of the divergence type on policy optimization via experiments on several MuJoCo environments.
- In chapter 5, we highlight the advantages and disadvantages of our proposed methods. We discuss future work that would benefit from our proposed methods or might be interesting to the reinforcement learning community.

2 Background

In this chapter, we first study the mathematical framework of formulating classical sequential decision-making problems, Markov decision processes (MDPs), which serve as a basis for solving a wide range of optimization problems via reinforcement learning (RL) methods. Unlike dynamic programming (DP) where a perfect model of the environment is given, RL does not assume complete knowledge of the environment and requires only samples—trajectories of states, actions, and rewards by interacting with the environment. For the discrete environment where the dimension of state and action spaces are relatively small, it is efficient to solve the problem with tabular methods that use tables to represent the value functions. However, in reality, RL problems usually have high-dimensional state and action spaces that are intractable for tabular methods to find an optimal solution and can be only solved approximately using function approximations to the policy or (and) value functions. Additionally, we will discuss three main RL algorithms with function approximations: value-based methods, policy gradient methods, and actor-critic methods.

Function approximations are long-standing problems in machine learning and have been well-studied. Function approximations have two special cases: linear methods that use a linear combination of the weight vector and hand-crafted state features to approximate the function, and nonlinear methods that the approximants come from nonlinear manifolds. For many real-world problems, the dimension of state spaces become so large that the linear function approximation often leads to the curse of dimensionality, such as an exponential increase in samples for parameter updates. Deep neural networks, e.g. feedforward neural networks or convolutional neural networks (CNNs), are powerful function approximation methods that are useful for alleviating the curse of dimensionality. Therefore, we use deep neural networks as function approximators to the value functions and policies to solve challenging tasks. The second section will provide background information about deep learning, such as forward and backward propagations, advanced stochastic gradient descent algorithms. Additionally, we will also talk about the generative adversarial networks (GANs), as GANs are closely related to many RL algorithms, such as actor-critic algorithms, and more importantly, to our minimax entropic policy optimization (MMPO) method.

RL with deep neural networks involves a number of new issues that do not normally arise in conventional deep learning, such as nonstationarity, unstable training, and delayed targets (rewards). Many advanced policy gradient methods have been proposed to address these issues in deep RL, such as TRPO and PPO. At the end of this chapter, we will introduce several advanced policy gradient algorithms by which our f-divergence penalized policy optimization (f-PPO) method in Chapter 3 and minimax entropic policy optimization (MMPO) method in Chapter 4 are highly inspired.

2.1 Reinforcement learning

This section introduces the formal problem of an infinite-horizon discounted Markov decision process, which is used to formally describe an environment for reinforcement learning, and show a typical agent-environment interaction in an MDP. The formal mathematical structure of the RL problem contains several key elements, such as value functions, the policy, etc. Several challenges in RL are discussed in this chapter, including the exploration-exploitation trade-off, sample complexity, etc. Afterward, we talk about three main categories of the RL algorithm: value-based methods, policy gradient methods, and actor-critic methods, where value-based methods attempt to learn an optimal policy indirectly via value functions, policy gradient methods directly find an optimal policy, and actor-critic optimize both a policy function and a value function.

2.1.1 Markov decision process

An infinite-horizon discounted Markov decision process is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma)$, in which

- $\mathbf{s} \in \mathcal{S}$ is the state space that contains all possible states in an environment,
- $\mathbf{a} \in \mathcal{A}$ is the action space that includes all possible actions that can be executed by an agent,
- $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \in \mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability that an action $\mathbf{a}_t$ in state $\mathbf{s}_t$ at time step t will lead to next state $\mathbf{s}_{t+1}$ at next time step $t + 1$,
- $r(\mathbf{s}_t, \mathbf{a}_t) \in r$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the immediate reward by performing action $\mathbf{a}_t$ in state $\mathbf{s}_t$,

- ρ_0 is the probability distribution of the initial state $\mathbf{s}_0$ defined as $\rho_0 \colon \mathcal{S} \to [0, 1]$,
- $\gamma \in (0, 1)$ is the discount factor.

The trainsition probability of a MDP must always satisfies the Markov property $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \cdots) = \mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, where the next state $\mathbf{s}_{t+1}$ at time step t depends exclusively on current state $\mathbf{s}_t$ and current action $\mathbf{a}_t$, or the result of an action does not depend on previous actions and states.

MDPs are usually described via an agent-environment interaction procedure as Figure 2.1, where an agent repeatedly acting on the environment and the environment responses to the agent's action with new states and rewards, where new states are determined by the conditional probability $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ given $\mathbf{s}_t, \mathbf{a}_t$ and rewards are values that the agent seeks to maximize in the long run through its choice of actions. The goal of solving the MDP problems is to find an optimal strategy that maximize the long-term expected discounted return

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.1}$$

where γ is the discount factor that trades off immediate and future rewards.

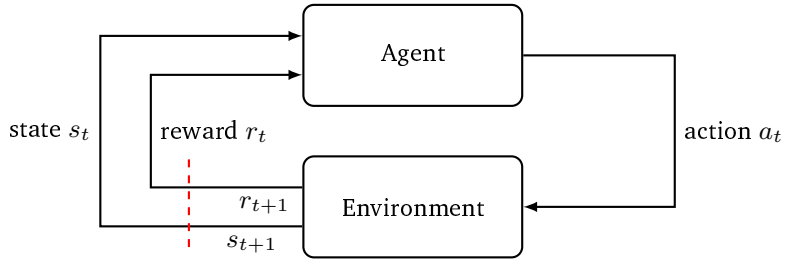

Figure 2.1.: Typical agent-environment interaction in a Markov decision process. [1]

2.1.2 Value functions and policy

Almost all reinforcement learning algorithms involve estimating *value functions*—functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform an action in a given state) [1]. In other words, the *state-value function* $V_\pi(\mathbf{s})$ or the *action-value function* $Q_\pi(\mathbf{s}, \mathbf{a})$ is an estimation of the future rewards that can be expected from the given state by following a policy π, which is defined as a mapping from states to actions (deterministic policy $\mathbf{a} = \pi(\mathbf{s})$) or a mapping from states to probabilities of selecting each possible actions (stochastic policy $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$). Standard definitions of the *state-value function* V_π, and the *action-value function* Q_π are

$$V_\pi(\mathbf{s}_t) = \mathbb{E}_\pi \left[G_t | S_t = \mathbf{s} \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = \mathbf{s} \right], \quad \forall \mathbf{s} \in \mathcal{S}$$

$$Q_\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_\pi \left[G_t | S_t = \mathbf{s}, A_t = \mathbf{a} \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = \mathbf{s}, A_t = \mathbf{a} \right] \quad \forall \mathbf{s} \in \mathcal{S}, \forall \mathbf{a} \in \mathcal{A}$$

where the action $\mathbf{a}_t$ is determined by the policy π, and G_t is the long-term expected discounted return defined in Equation (2.1).

A *policy* π is defined as the agent's behavior in the environment, for example, the control torque of a bipedal walking robot. A *deterministic policy* π is a function that defined as $\mathbf{a} = \pi(\mathbf{s})$, which is a unique mapping from state $\mathbf{s}$ to action $\mathbf{a}$. On the other hand, a *stochastic policy* is defined as a conditional probability distribution $\pi(\mathbf{a}|\mathbf{s})$, which is the likelihood of taking action $\mathbf{a}$ given state $\mathbf{s}$. The application of a policy to MDPs is done in the following way. First, an agent starts from a state $\mathbf{s}_0$ drew from the initial state distribution $\rho(\mathbf{s}_0)$. Then, a trajectory is generated $(\mathbf{s}_0, \mathbf{a}_0, r_0, \mathbf{s}_1, \mathbf{a}_1, r_1, \cdots)$ by following the policy π and the environment's transition model $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$.

In real-world scenarios, where the environment is stochastic and states are sometimes partially observable, we represent the policy using the stochastic policy. For the agent with discrete actions, the policy is a parameterized *softmax policy*:

$\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) = \exp\left(f_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a})\right)/\sum_{\mathbf{a}'\in\mathcal{A}}\exp\left(f_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}')\right)$, where $f_{\boldsymbol{\theta}}$ is a neural network with trainable weights $\boldsymbol{\theta}$, and action probability is proportional to exponentiated weight. For the agent with continuous actions, the policy is a parameterized *Gaussian policy*: $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \sim \mathcal{N}(\mathbf{a}|f_{\boldsymbol{\omega}}(\mathbf{s}), \boldsymbol{\sigma}^2)$, where the mean is the output of a deep neural network $f_{\boldsymbol{\omega}}(\mathbf{s})$ with trainable weights $\boldsymbol{\omega}$, and a separate set of trainable parameters $\boldsymbol{\sigma}^2$ specifies the standard deviation of each action, and hence, $\boldsymbol{\theta} = [\boldsymbol{\omega}, \boldsymbol{\sigma}^2]$ is the parameter vector of the policy.

2.1.3 Reinforcement learning algorithms

Several well-known model-free RL algorithms are discussed in this subsection, including temporal-difference learning (TD learning) methods, such as Q-learning and state-action-reward-state-action (SARSA), policy gradient methods, and actor-critic methods. TD-learning is a kind of *value-based* methods that optimize the value function and find an optimal policy implicitly. Policy gradient methods are *policy-based* and optimize the policy directly. *Actor-critic* methods learn both the optimal value functions and the optimal policy.

Temporal-difference learning

If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference learning [1]. TD learning is a combination of dynamic programming (DP) methods that are used to compute optimal policies given a perfect model of the environment as a MDP, and Monte Carlo (MC) methods that are ways of solving the RL problem without complete knowledge of the environment, but based on averaging sample returns. Solving RL tasks using TD learning consists of two steps: estimate the model of the environment's dynamics by finding optimal value functions (*prediction problem*), and compute an optimal policy by solving the Bellman optimality equations with the optimal value functions (*control problem*).

First, let's consider the TD learning methods for learning the optimal state-value function for a given policy. TD learning methods use experience (samples) to address the prediction problem. For simplicity purpose, we use the one-step TD learning, a special case of n-step TD learning, whose update rule of the state-value function is

$$V(\mathbf{s}_t) \leftarrow V(\mathbf{s}_t) + \alpha[r_{t+1} + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t)]$$

where $\delta_t = r_{t+1} + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t)$ is called the *TD error*, which is a measure of the difference between the current approximated value of $\mathbf{s}_t$ and the next approximation $r_{t+1} + \gamma V(\mathbf{s}_{t+1})$. TD learning updates the value function without waiting for the end of an episode like in MC, instead by using *bootstrapping* in that the value function is updated immediately after each step of an episode. However, without the complete knowledge of a model, learning the state-value function alone is not sufficient. Thus, an approximation of the action-value function $Q(\mathbf{s},\mathbf{a})$ is required to determine the optimal action could be taken from states.

Now, we discuss an *on-policy* method, which is called *state-action-reward-state-action* (SARSA), for learning an optimal action-value function, denoted as $Q^*(\mathbf{s},\mathbf{a})$. Thus, the corresponding optimal policy is derived by making the policy greedy with respect to the optimal action-value function, $\pi(\mathbf{s}) = \operatorname{argmax}_{\mathbf{a}} Q^*(\mathbf{s},\mathbf{a})$. The on-policy method means that SARSA estimates the action-value function $Q_\pi(\mathbf{s},\mathbf{a})$ for all states $\mathbf{s}$ and actions $\mathbf{a}$ from current behavior policy π. The update rule for learning the values of state-action pairs is

$$Q(\mathbf{s}_t,\mathbf{a}_t) \leftarrow Q(\mathbf{s}_t,\mathbf{a}_t) + \alpha\left[r_{t+1} + \gamma Q(\mathbf{s}_{t+1},\mathbf{a}_{t+1}) - Q(\mathbf{s}_t,\mathbf{a}_t)\right]$$

where α is the learning rate, γ is the discount factor that determines the importance of future rewards. The action-value function $Q(\mathbf{s},\mathbf{a})$ is updated after every transition from a nonterminal state s_t, where the transition $(\mathbf{s}_t,\mathbf{a}_t,r_t,\mathbf{s}_{t+1},\mathbf{a}_{t+1})$ is from one state-action pair to the next state-action pair. SARSA converges with probability 1 to an optimal policy and action-value function as long as all state–action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy [1].

Another popular model-free RL algorithm that uses TD prediction methods for finding the optimal policy of a MDP is *Q-learning*. Q-learning is an *off-policy* learning algorithm, in which a Q-learning agent improves the value of the optimal policy independently of the policy being followed. By definition, the update rule of learning an optimal action-value function is

$$Q(\mathbf{s}_t,\mathbf{a}_t) \leftarrow Q(\mathbf{s}_t,\mathbf{a}_t) + \alpha\left[r_{t+1}\gamma \max_{\mathbf{a}} Q(\mathbf{s}_{t+1},\mathbf{a}) - Q(\mathbf{s}_t,\mathbf{a}_t)\right].$$

Unlike Sarsa, where the action-value function is updated using state-action value $Q(\mathbf{s}_t,\mathbf{a}_t)$ sampled by current policy, Q-learning learns the value of the optimal policy using $\max_{\mathbf{a}} Q(\mathbf{s}_t,\mathbf{a})$ that is sampled by a second behavior policy, e.g., ϵ-greed of the action-value function.

Policy gradient methods

Both Sarsa and Q-learning are trying to estimate the model of the environment's dynamics by learning the optimal vaule function of the policy. Policy gradient methods, on the other hand, do not require the approximation of the dynamics model. Policy gradient methods are a class of model-free policy search algorithms, in which policy parameters are updated in the direction of the policy gradient that is estimated using Monte Carlo methods.

Policy gradient methods seek to directly optimize a parameterized policy based on the objective that is the performance measure of the policy

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}(\tau)}[r(\tau)] = \int \pi_{\boldsymbol{\theta}}(\tau) r(\tau) d\tau \tag{2.2}$$

where $\pi_{\boldsymbol{\theta}}$ is usually a parameterized stochastic policy distribution, e.g., Gaussian policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \sim \mathcal{N}(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta})$, and $r(\tau) = \sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t)$ is the cumulative reward of a sampled trajectory $\tau = (\mathbf{s}_1, \mathbf{a}_1, r_1, \cdots, \mathbf{s}_T, \mathbf{a}_T, r_T)$ by following the policy $\pi_{\boldsymbol{\theta}}$. The goal is to find an optimal policy $\pi^*_{\boldsymbol{\theta}}$ that maximizes the objective in Equation (2.2) via policy gradient ascent. The policy gradient with respect to the parameter vector $\boldsymbol{\theta}$ can be estimated from sampled trajectories using MC estimation

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau \\
&= \int \pi_\theta \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau \quad \text{log-trick} \\
&= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \\
&= \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a_t}|\mathbf{s_t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s_t}, \mathbf{a_t}) \right) \right] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) r(\mathbf{s}_t^i, \mathbf{a}_t^i).
\end{aligned} \tag{2.3}$$

This gradient update rule is well-knwon as the REINFORCE algorithm [21] (or the Monte Carlo policy gradient method [1]), which is one of the first policy gradient algorithm. Thus, the estimation of the gradient does not require the dynamics of the MDP, as the initial state distribution ρ_0 and state transition probabilities $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ are cancelled out in the end. However, such a Monte Carlo estimated policy gradient may suffer from a high variance problem and slow convergence.

In order to reduce the variance of the gradient estimator, the simplest solution is to subtract a baseline from the gradient,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t^i, \mathbf{a}_t^i) - b \right) \tag{2.4}$$

where the baseline b can be chosen arbitrarily [21] or optimally like in [22]. The baseline does not change the estimated gradient of the objective, but has a large impact on the variance.

Actor-critic methods

We talked about two different reinforcement learning algorithms that are classified into value-based and policy-based methods, where TD-learning are value-based methods as the objective is to find an optimal value function (critic), and policy gradient methods try to optimize the policy (actor) exclusively. Actor-critic methods combine the advantages of value-based and policy-based methods via a parameterized policy that brings the advantage of continuous actions, and a value function that allows for the actor to update with gradients that have lower variance, and thus, speeds up the learning process.

The aim of actor-critic methods is to simultaneously learn an action-value function $Q_{\boldsymbol{\omega}}(\mathbf{s}, \mathbf{a})$,or a state-value function $V_{\boldsymbol{\omega}}(\mathbf{s})$, and learn a policy $\pi_{\boldsymbol{\theta}}$, where value functions are parameterized by the parameter vector $\boldsymbol{\omega}$ and the policy is param-

eterized by the parameter vector $\boldsymbol{\theta}$. According to the *policy gradient theorem* [23], the policy gradient in Equation (2.3) can be rewrote as

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) r(\mathbf{s}_t^i, \mathbf{a}_t^i) \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) Q_{\boldsymbol{\omega}}(\mathbf{s}_t^i, \mathbf{a}_t^i)\end{aligned}$$

where $Q_{\boldsymbol{\omega}}(\mathbf{s}_t^i, \mathbf{a}_t^i)$ is the action-value function that estimates the goodness or badness of current state-action pair $(\mathbf{s}_t, \mathbf{a}_t)$. However, the Q-values estimated using Monte-Carlo method also suffer from high variance. Similarly, we subtract a baseline from the gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\Big(Q_{\boldsymbol{\omega}}(\mathbf{s}_t^i, \mathbf{a}_t^i) - b\Big)$$

where b can be an arbitrary real number. The resulted gradient has lower variance and unbiased estimation [24]. Additionally, it is also useful to subtract a state-dependent baseline, e.g., the value function $V(\mathbf{s})$, without changing the expection. This results in a new policy gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) A_{\boldsymbol{\omega}}(\mathbf{s}_t^i, \mathbf{a}_t^i)$$

where $A(\mathbf{s}_t, \mathbf{a}_t) = Q(\mathbf{s}_t, \mathbf{a}_t) - V(\mathbf{s}_t)$ is the advantage function that defined as the estimation of the goodness or badness of an action over the average performance of all actions in the current state. The advantage function is also interpreted as the estimated expectation of the TD-error $\delta(\mathbf{s}, \mathbf{a})$. In this paper, we use the generalized advantage estimation [25] scheme to get $A_\pi(\mathbf{s}_t, \mathbf{a}_t)$

$$\begin{aligned}\hat{A}_\pi^{\mathrm{GAE}(\gamma,\lambda)}(\mathbf{s}_t, \mathbf{a}_t) &= \sum_{l=0}^{\infty}(\gamma\lambda)^l \delta_{t+l}^{V_\phi}(\mathbf{s}_t, \mathbf{a}_t) \\ \delta^V(\mathbf{s}_t, \mathbf{a}_t) &= r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t') + \gamma V(\mathbf{s}_t') - V(\mathbf{s}_t)\end{aligned}$$

where $\lambda \in [0, 1], \gamma \in [0, 1]$ and V is the state-vaule function approximation. The two parameters λ and γ contribute to the bias-variance tradeoff when using an approximate value function: $\gamma \leq 1$ introduces bias into the policy gradient estimate regardless of the value function's accuracy; on the other hand, $\lambda \leq 1$ inctroduces bias only when the value function is inaccurate [25].

The update rule of policy parameters regardless of the chosen objective $J(\pi_\theta)$ is

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha\nabla_\theta J(\pi_\theta)$$

where α is the learning rate. Thus, the update steps of policy parameters are constrained using the Euclidean distance in parameter space, $\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\| \leq \alpha\|\nabla_\theta J(\pi_\theta)\|$. This upate rule is well-known as the stochastic gradient ascent.

2.2 Deep learning

We talked about reinforcement learning in previous section. The performance of RL algorithms highly depend on the generalization of the function approximator of value functions and policies, or in other words, the representation of state features plays a key role in RL. In the literature [26], investigation have shown the differences in performance when comparing different function approximation methods, such as radial basis functions (RBFs), linear methods, neural networks. Deep learning has been experiencing dramatic growth in attention and interest due to promising results in areas like computer vision, natural language processing, speech recognition. This section will focus the discussion on deep learning which is well-known for its powerful features representation of high-dimensional data and end-to-end training procedures. Additionally, we discuss the generative adversarial networks (GANs) and several advanced SGD optimization algorithms.

2.2.1 Feedforward neural networks

One of the key features of deep learning is that the computational models, called deep neural networks are composed of multiple processing layers and are able to learn representations of data with multiple levels of abstraction. The learning of a deep neural network is made by using the backpropagation method to indicate how the internal parameters should be changed. On the other hand, the prediction of the output is calculated by using forward propagation method: the data is fed to the input layer, the neurons do a linear transformation on the input by the weights and biases, the activation function transforms the linear function into a nonlinear function, the information moves from layer to layer, and finally output the result.

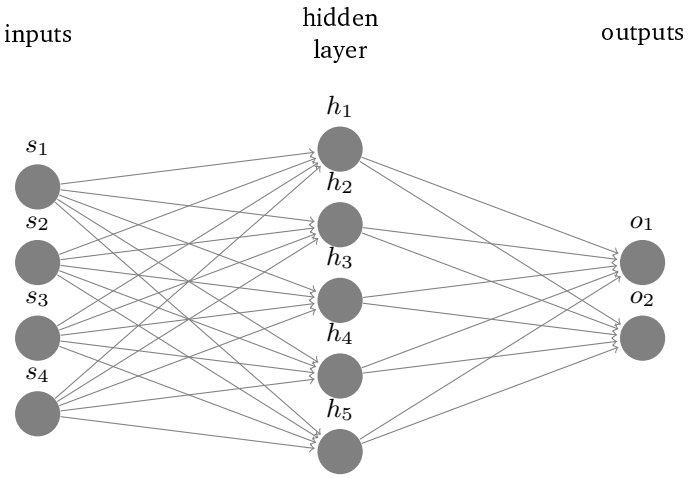

Figure 2.2.: A generic feedforward artificial neural networks (ANN). The network in the figure has three layers, an input layer with four input units, a hidden layer, and an output layer consisting of two output units. The number of circles in each layer indicates the dimensions of corresponding layers. The circles represent neurons of the network, and arrows represent the connections and data flow between the neurons of the network. The activation layer is omitted for clarity purpose.

The deep neural network is a kind of nonlinear function that usually contains a large number of parameters. Formally, the feedfoward neural netork can be expressed as a function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x})$$

where $\mathbf{x}$ is the input vector, $\boldsymbol{\theta}$ is the parameter vector of the network, and $\hat{\mathbf{y}}$ are the output values. For example, in Figure 2.2, the arrows are parameters $\boldsymbol{\theta}$ of the feedforward neural network. The goal of training a deep neural network is to find the optimal parameters by minimizing a defined loss function whose gradients with respect to the parameters are calculated with forward and backward propagations.

A good way to understand how does the forward and backward propagation work is to use the *supervised learning* example. Imagine that we build a feedforward neural network that can classify the data into different classes, e.g., image classification. We use $\mathbf{x}$ to represent the input data, $\mathbf{y}$ as true class labels, and $\hat{\mathbf{y}}$ as predicted labels. Thus, we can define a mean squared error (MSE) loss function

$$L_{\boldsymbol{\theta}} = \frac{1}{M}\sum_{i=1}^{M} \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|^2$$

for the training with a collection of dataset, $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$, where N is the size of the training dataset and M is the mini-batch size, $M \leq N$. During training, a number of randomly shuffled data $\{\cdots, (\mathbf{x}_i, \mathbf{y}_i), \cdots\}$ of size M are fed into the neural network, and then, the nework computes a batch of predictions $\hat{\mathbf{y}}$ of size M via forward propagation. The backward propagation procedure starts from the output layer by computing the gradient directly with respect to the weights in the last layer, e.g., the gradient with respect to parameters in the last layer is $\partial L_\theta / \partial \mathbf{y}$. Then, the gradients are propagated backward to the second last layer, where the gradients are computed using chain rule, e.g., the gradient with respect to parameters in the second last layer is $\partial L_\theta / \partial \mathbf{x} = (\partial L_\theta / \partial \mathbf{y})(\partial \mathbf{y} / \partial \mathbf{x})$. Following the same procedure, the gradients are propagated from the output layer all the way to input layer, through all layers in the network. After training, given an unseen data, the network is capable of predicting the class of the data via forward

propagation with high probability.

Apart from feedforward neural networks, many other networks, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and long short-term memory (LSTM) are also essential parts in deep learning. For example, CNNs have been widely used in the area where input data are high-dimensional, such as object recognition from images, and RNNs are particularly useful in sequential data, e.g., natural language processing. However, these networks will not be studied in this thesis, as they are not used.

2.2.2 Generative adversarial network

The generative adversarial network (GAN) is a framework for estimating generative models via an adversarial process, in which two models are trained simultaneously: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G [17]. The *generator* is simply a differentiable function $G_{\boldsymbol{\theta}}(\mathbf{z})$, that is used to generate samples $\hat{\mathbf{x}} = G_{\boldsymbol{\theta}}(\mathbf{z})$ from a prior input noise variables $p(\mathbf{z})$, and to capture the distribution of the traning data $p(\mathbf{x})$. The *discriminator* is second function $D_{\boldsymbol{\omega}}(\mathbf{x})$ that outputs a single scalar, which represents the probabilty that $\mathbf{x}$ came from the traning data rather than generated data $\hat{\mathbf{x}}$. A sketch of the GAN framework is presented in Figure 2.3.

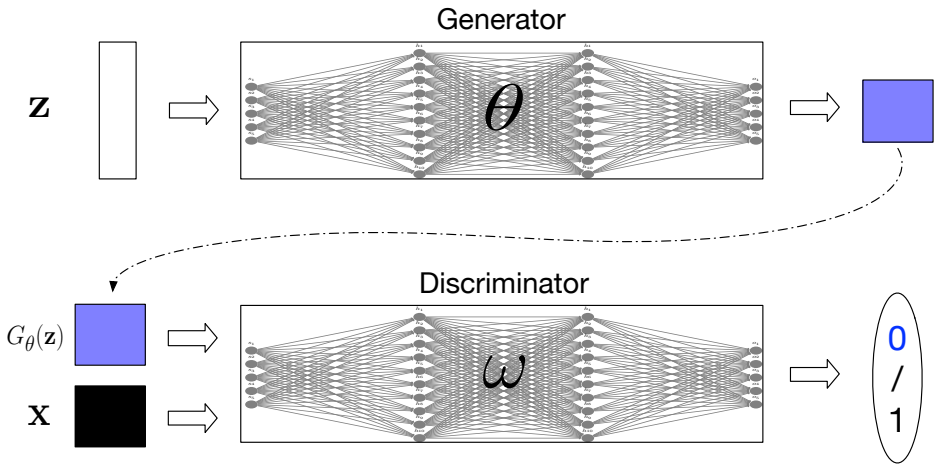

Figure 2.3.: The GAN framework consists of two adversarial models that are trained to against each other. Each model is represented by a differentiable function that has its own parameters. There are two different training scenarios. In one scenario, the generator ramdomly sample data $\mathbf{z}$ and generate fake sample $G_{\boldsymbol{\theta}}(\mathbf{z})$. Then, the discriminator takes the fake sample $G_{\boldsymbol{\theta}}(\mathbf{z})$ as input and strives to make $D_{\boldsymbol{\omega}}(G_{\boldsymbol{\theta}}(\mathbf{z}))$ approach 0. Simultaneously, the generator is trying to make $D_{\boldsymbol{\omega}}(G_{\boldsymbol{\theta}}(\mathbf{z}))$ approach 1. In the second scenario, training samples $\mathbf{x}$ are randomly sampled from the training dataset. The discriminator takes the real sample $\mathbf{x}$ as input and make $D_{\boldsymbol{\omega}}(\mathbf{x})$ approach 1.

The objective of the generator G is to minimize the maximum likelihood estimation loss function

$$\min_{\boldsymbol{\theta}} \quad J^G(\theta) = \mathbb{E}_{\mathbf{z}}\left[\log(1 - D_{\boldsymbol{\omega}}(G_{\boldsymbol{\theta}}(\mathbf{z})))\right],$$

and the objective of the discriminator D is to minimize the cross-entropy loss function

$$\min_{\boldsymbol{\omega}} \quad J^D(\boldsymbol{\omega}) = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\mathbf{x}}}\left[\log D_{\boldsymbol{\omega}}(\mathbf{x})\right] - \frac{1}{2}\mathbb{E}_{\mathbf{z}}\left[\log(1 - D_{\boldsymbol{\omega}}(G_{\boldsymbol{\theta}}(\mathbf{z})))\right].$$

Simply put, the discriminator $D_{\boldsymbol{\omega}}$ tries to optimize its parameter vector $\boldsymbol{\omega}$ such that $D_{\boldsymbol{\omega}}(\mathbf{x}) \to 1$ and $D_{\boldsymbol{\omega}}(G_{\boldsymbol{\theta}}(\mathbf{z})) \to 0$, however, the generator $G_{\boldsymbol{\theta}}$ tries to confuse $D_{\boldsymbol{\omega}}$ by optimizing the parameter vector $\boldsymbol{\theta}$ such that $D_{\boldsymbol{\omega}}(G_{\boldsymbol{\theta}}(\mathbf{z})) \to 1$.

GANs have strong connections with RL in several ways. First, both GANs and AC can be viewed as a class of multilevel optimization problems which have close parallels. Secondly, GANs can be used for imitation learning, e.g., generative adversarial imitation learning [27]. Thirdly, the idea of training two models with different loss functions simutaneously has heavily inspired us to propose the minimax entropic policy optimization method, which will be introduced in Chapter 4.

2.2.3 Stochastic gradient descent algorithms

The overview [15] discussed three variants of the first-order gradient descent method: batch gradient descent, stochastic gradient descent, mini-batch gradient descent, which differ in the amount of data used for computing the gradient of the objective function. Mini-batch gradient descent is commonly the algorithm of choice for traning neural networks because of its low-variance estimates of the gradients and the property of data efficiency. Mini-batch gradient descent, which is also called "Vanilla" mini-batch gradient descent, performs an update for every mini-batch of n traning samples

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; x^{i:i+n}, y^{i:i+n})$$

where α is the learning rate, $x^{i:i+n}$ and $y^{i:i+n}$ are sampled data and labels. However, vanilla mini-batch gradient descent has several drawbacks:

- First, tuning the learning rate α can be difficult, as small learning rate leads to slow convergence and large learning rate can overshoot the optimum.
- Secondly, a single learning rate is applied to update all parameters. Thus, it might be difficult for optimizing networks with sparse gradients.
- Thirdly, it is difficult for mini-batch gradient descent to escape from saddle points, as the gradient is close to zero in all dimensions.

Mini-batch gradient descent with momentum is a method that accelerates the learning process of the "Vanilla" method by storing the update $\mathbf{v}_t$ at each iteration, and computes the next update as a linear combination of the gradient $\nabla_{\boldsymbol{\theta}}$ and the previous update $\mathbf{v}_{t-1}$

$$\begin{aligned}\mathbf{v}_t &= \gamma \mathbf{v}_{t-1} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \mathbf{v}_t\end{aligned}$$

where γ is called the *momentum* term, usually set to 0.9 or a similar value [15]. The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation [15].

Adaptive moment estimation (Adam) is another method that computes adaptive learning rate for each parameter. Adam requires a small amount of memory space for stroing an exponentially decreasing average of past squared gradients $(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t))^2$ and an exponentially decreasing average of past gradients $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$

$$\begin{aligned}\mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1-\beta_1) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) \\ \mathbf{v}_t &= \beta_2 \mathbf{v}_{t-1} + (1-\beta_2) (\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t))^2\end{aligned}$$

where $\mathbf{m}_t$ and $\mathbf{v}_t$ are estimates of the first moment and the second moment of the gradients respectively, β_1 and β_2 are open parameters. Afterwards, bias-corrected first moment $\hat{\mathbf{m}}_t$ and second moment $\hat{\mathbf{v}}_t$ are estimated using

$$\begin{aligned}\hat{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1-\beta_1^t} \\ \hat{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1-\beta_2^t}\end{aligned}$$

Finally, the update rule of the Adam optimization algorithm is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \hat{\mathbf{m}}_t$$

where ϵ is a hyperparameter, usually set to be 10^{-8}. Adam is to date among the most popular optimization methods for deep neural network training, and hence, is the default SGD optimizer of both our the MMPO method in Chapter 3 and f-PPO method in Chapter 4.

2.3 Policy search algorithms

The benefit of policy gradient methods with a parameterized policy is that the function approximation can be used to generalize continuous states and actions. Nevertheless, the optimization with stochastic gradient descent methods is data inefficient, and the learning rate of the SGD method is difficult to choose. Several advanced methods are discussed in this section that has better data efficient and better convergence properties compared to the "Vanilla" method. For example, unlike SGD that uses the Euclidean distance, the natural policy gradient method [28] uses the Fisher information metric, which is invariant to linear transformations in the parameter space, to capture the update steps of policy parameters, and hence, converge faster. Relative entropy policy search (REPS) [20] treats the MDP problem as a constrained policy optimization problem and yields a closed-form solution, in which no learning rates are required. Trust region policy optimization (TRPO) [10] and proximal policy optimization (PPO) [16] are effective for solving MDP problems with high-dimensional states and actions, e.g., playing Atari games, by optimizing a "surrogate" objective and simultaneously restricting the policy improvement inside a trust region.

2.3.1 Natural policy gradient

The natural policy gradient [28] is a method that updates the parameters along steepest ascent directions based on the underlying structure of the parameter space. The main disadvantage of using the Euclidean metric of $\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\|$ as the measure of the closeness between the current policy and the updated policy is that the gradient is different for every parameterization $\boldsymbol{\theta}$ of the policy $\pi_{\boldsymbol{\theta}}$. Instead, the "distance" between two policy distributions can be measured using the Kullback-Leibler divergence (KL-divergence), which is a measure of how one probability distribution diverges from another distribution, where the divergence indicates that KL-divergence is not a real distance metric because KL-divergence is not symmetric. The KL-divergence can be approximated by its second-order Taylor expansion

$$\begin{aligned} D_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}_{k+1}} || \pi_{\boldsymbol{\theta}_k}) &= \int \pi_{\boldsymbol{\theta}_{k+1}}(\tau) \log \left(\frac{\pi_{\boldsymbol{\theta}_{k+1}}(\tau)}{\pi_{\boldsymbol{\theta}_k}(\tau)} \right) d\tau \\ &\approx \|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\|^T \mathbf{F}_{\boldsymbol{\theta}} \|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\| \end{aligned}$$

where $\mathbf{F}$ is known as the Fisher-information matrix defined as

$$\mathbf{F}_{\boldsymbol{\theta}} = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})^T \right]$$

The natural policy gradient uses the Fisher-information matrix as metric, thus has the following optimization problem

$$\begin{aligned} \nabla^{\mathrm{NG}}_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \delta\boldsymbol{\theta}^T \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) \\ \text{s.t.:} \quad & D_{\mathrm{KL}} \approx \delta\boldsymbol{\theta}^T \mathbf{F}_{\boldsymbol{\theta}} \delta\boldsymbol{\theta} \leq \epsilon \end{aligned}$$

where $\delta\boldsymbol{\theta} = \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k$ indicates the update steps of parameters, and $\epsilon \in [0, +\infty]$ is the KL bound used to prevent the policy from greedy updates and is usually a small positive number, e.g., $\epsilon = 0.01$. Here, $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$ is the policy gradient of any chosen objective $J(\pi_{\boldsymbol{\theta}})$, for example, the objective in Equation (2.2). The solution to this optimization problem is given as

$$\nabla^{\mathrm{NG}}_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) \propto \mathbf{F}^{-1}_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}).$$

As every parameters has the same influence under metric $\mathbf{F}_{\boldsymbol{\theta}}$, the natural gradient is invariant to linear transformation in parameter space [9].

2.3.2 Relative entropy policy search

Relative entropy policy search is a constrained optimization algorithm that bound the relative entropy to the old policy distribution during policy updates using the KL divergence, and hence, achieve a stable learning progress. The constrained optimization problem for episode-based REPS is

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & J(\pi_\theta) = \int \pi_{\boldsymbol{\theta}} R_{\boldsymbol{\theta}} d\boldsymbol{\theta} \\ \text{s.t.} \quad & \epsilon \geq \int \pi_{\boldsymbol{\theta}_{k+1}} \log \frac{\pi_{\boldsymbol{\theta}_{k+1}}}{\pi_{\boldsymbol{\theta}_k}} d\boldsymbol{\theta} \\ & 1 = \int \pi_{\boldsymbol{\theta}} d\boldsymbol{\theta} \end{aligned}$$

which can be solved efficiently by the Lagrangian multipliers method, and hence, obtain a closed-form solution for the new policy

$$\pi_{\boldsymbol{\theta}_{k+1}} \propto \pi_{\boldsymbol{\theta}_k} \exp\left(\frac{R_{\boldsymbol{\theta}}}{\eta}\right)$$

where η is the Lagrangian multiplier connected to the KL constraint ϵ. The optimal parameter η is obtained by minimizing the dual function $g(\eta)$ of the original optimization probelm

$$\min_{\eta} \quad g(\eta) = \eta\epsilon + \eta \log \int \pi_{\boldsymbol{\theta}_k} \exp\left(\frac{R_{\boldsymbol{\theta}}}{\eta}\right) d\boldsymbol{\theta}.$$

The $R_{\boldsymbol{\theta}}$ is the episode reward of following the policy $\pi_{\boldsymbol{\theta}}$. The ϵ can be chosen freely where larger values lead to bigger steps while excessively large values can destroy the policy. Its size depends on the problem as well as on the amount of available samples [20]. However, REPS is only effective for solving problems with linear function approximations of handcrafted state features, or with the tabular representation as the policy. REPS is not well suited for optimizing large nonlinear functions, e.g., deep neural networks.

2.3.3 Trust region policy optimization

Trust region policy optimization (TRPO) [10] is a constrained policy optimization algorithm that is similar to REPS and natural policy gradient, however, is more effective for optimizing large nonlinear policies such as neural networks. TRPO uses a so-called "surrogate" objective that has the benifits of importance sampling [29], and a KL constraint for the trust region update of policy parameters. Similar to REPS, TRPO is also a constrained problem defined as

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & J(\pi_{\boldsymbol{\theta}}) = \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}(\mathbf{s}_t, \mathbf{a}_t) \right] \\ \text{s.t.:} \quad & \hat{\mathbb{E}}_t \left[D_{\text{KL}}[\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t) || \pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)] \right] \leq \epsilon \end{aligned} \tag{2.5}$$

where $\hat{\mathbb{E}}_t$ indicates that the "surrogate" objective is to maximize the expectation of sampled advantages weighted by the policy ratio. Instead of solving the constrained optimization problem analytically by computing the Lagrangian primal-dual function, TRPO computes the solution approximately by using a linear approximation to the objective $J(\pi_{\boldsymbol{\theta}})$ and a quadratic approximation to KL constraint D_{KL}, and hence, has the following approximated problem

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & J(\pi_{\boldsymbol{\theta}}) \approx \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})(\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k) \\ \text{s.t.:} \quad & \hat{\mathbb{E}}\left[D_{\text{KL}}\right] \approx \delta\boldsymbol{\theta}^T \mathbf{F}_{\boldsymbol{\theta}} \delta\boldsymbol{\theta} \leq \epsilon \end{aligned}$$

which is solved by carrying out a line search in the found direction

$$\nabla_{\boldsymbol{\theta}}^{\text{TRPO}} \propto \mathbf{F}_{\boldsymbol{\theta}}^{-1} \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$$

where the $\mathbf{F}_{\boldsymbol{\theta}}^{-1}$ is the inverse Fisher information matrix and the $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$ is the gradient of the "surrogate" objective with respect to policy parameters. TRPO has demonstrated robust performance on a wide variety of tasks, such as continuous control of simulated robots, and playing Atari games, via deep neural networks as function approximations.

2.3.4 Proximal policy optimization

Proximal policy optimization is a first-order gradient method that optimizes a clipped "surrogate" objective using multiple epochs of SGD, where the clipped "surrogate" objective has the benefit of trust region policy optimization and using multiple epochs of SGD has better sample complexity. The clipped "surrogate" objective is defined as

$$\max_{\boldsymbol{\theta}} \quad J^{\text{Clip}}(\pi_{\boldsymbol{\theta}}) = \hat{\mathbb{E}}\left[\min\left(r_t(\boldsymbol{\theta})\hat{A}, \text{clip}[r_t(\boldsymbol{\theta}), 1-\beta, 1+\beta]\hat{A}\right)\right]$$

where the $r_t(\boldsymbol{\theta}) = \pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)/\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)$ is the policy ratio between the new updated policy $\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)$ and the old policy $\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)$ that was used to sample trajectories, and the $\beta \in (0, 1)$ is the clip parameter, e.g. $\beta = 0.2$. The update rule for the parameter vector is simply obtained using stochastic gradient ascent

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\theta} J^{\text{Clip}}(\pi_{\theta})$$

where α is the learning rate.

A second version of PPO is to formulate the constrained problem in Equation (2.5) as an unconstrained optimization problem by using KL divergence as a penalty (a soft constraint) instead of a hard constraint

$$\max_{\boldsymbol{\theta}} \quad J^{\text{KLPen}}(\pi_{\boldsymbol{\theta}}) = \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}(\mathbf{s}_t, \mathbf{a}_t) - \beta D_{\text{KL}}[\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)||\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)] \right] \tag{2.6}$$

where β is an adaptive hyperparameter that is adjusted according to the value of the mean KL-divergence. The update procedure can be described as

- Update policy parameters using stochastic gradient ascent,

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_\theta J^{\text{KLPen}}(\pi_\theta)$$

- Compute mean KL-divergence $d = \hat{\mathbb{E}}\left[D_{\text{KL}}[\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)||\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)]\right]$
  - If $d < \epsilon/1.5$, $\beta \leftarrow \beta/2$
  - If $d \geq \epsilon \times 1.5$, $\beta \leftarrow \beta \times 2$

where ϵ is the KL target, e.g. $\epsilon = 0.1$, α is the learning rate. Compared to TRPO, PPO is straightforward to implement and has better overall performance on a series of experiments on the MuJoCo environment [30], the Roboschool environment [16], and the Atari domain [31].

3 Minimax entropic policy optimization

3.1 Introduction

Many methods in reinforcement learning are trying to optimize a single objective. For example, *policy-based methods* [1] [21] try to optimize the policy directly by maximizing an objective and *value-based methods* [23] [32] seek to optimize a value function such that the optimal policy is found indirectly. The actor-critic (AC) [11] [33] is a class of multilevel optimization problem that optimizes both a value-objective and a policy-objective, and thus, an optimal value function and an optimal policy are found simultaneously. Similarly, generative adversarial networks (GANs) [17] can be seen as a multilevel (bilevel) optimization problem where a generator is optimized with respect to the optimum of a second model, a discriminator. GANs can also be interpreted as a minimax game between the generator and the discriminator, both players are typically represented as deep neural networks.

In this chapter, we propose a new family of policy gradient methods for RL, which combines the idea of multilevel optimization and trust region policy optimization, and is termed as minimax entropic policy optimization (MMPO). The "minimax" indicates that MMPO maximizes an objective for the policy improvement and minimizes a loss function of the ramp-transformed KL-divergence to meet the KL bound. Additionally, to reduce the variance of the estimated policy gradient, we also learn an optimal state-value function by minimizing a mean squared error loss function. With the proposed MMPO, we are able to solve a wide variety of challenging continuous control tasks from MuJoCo simulator [30], such as InvertedPendulum-v2, Walker2d-v2, and achieve state-of-the-art performance as TRPO [10] and PPO [16]. MMPO only requires first-order gradient information, and thus, is straightforward to implement and to combine with other neural network structure such as RNNs.

3.2 MMPO

We first introduce the commonly used objective, whose gradient has the same form as the "Vanilla" policy gradient in Equation (2.3), as a loss function in automatic differentiation framework and highlight the objective's relationship with the "surrogate" objective that comes from TRPO [10]. Afterward, we discuss how the KL-divergence loss function is formulated by transforming the KL constraint, a *soft inequality constraint* (inequality constraints that are recommended but not compulsory) in a constrained policy optimization problem, with a ramp function. Both the objective and the KL-divergence loss function are optimized using advanced stochastic gradient descent algorithms, which require small amounts of memory space for storing parameters that are used to adapt the learning rate.

3.2.1 Preliminaries

Let us first recall the "Vanilla" policy gradient method that tries to optimize a parameterized policy $\pi_{\boldsymbol{\theta}}$ using stochastic gradient ascent, where the gradient is estimated using sampled trajectories τ

$$\begin{aligned}
\max_{\boldsymbol{\theta}} \quad & J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau) r(\tau) d\tau && \text{Objective} \\
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta};\tau) \approx & \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t^i,\mathbf{a}_t^i)\right) && \text{Gradient} \\
& \boldsymbol{\theta} = \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta};\tau) && \text{Update rule.}
\end{aligned}$$

There are two main drawbacks in the "Vanilla" gradient method: 1) the variance of estimated policy gradients is high, 2) the sampling efficiency is poor as each sampled trajectory is used just for one gradient step.

To reduce the variance, we adopt the causality idea that policy at a later timestep t' cannot affect the reward at an earlier timestep t, where $t' > t$. Thus we can replace $\sum_{t'=\mathbf{1}}^{T} r(\mathbf{s}_{t'}^i, \mathbf{a}_{t'}^i)$ with $\sum_{t'=\mathbf{t}}^{T} r(\mathbf{s}_{t'}^i, \mathbf{a}_{t'}^i)$, which is also known as the "reward to go", or the action-value function $Q(\mathbf{s}_t, \mathbf{a}_t)$. Thus, the gradient that has lower variance can be written as

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) Q(\mathbf{s}_t, \mathbf{a}_t).$$

Furthermore, one can reduce the variance further by substracting a baseline from the gradient, a good baseline is calculated by taking the expectation $V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}\left[Q(\mathbf{s}_t, \mathbf{a}_t)\right]$. Thus, the gradient and the update rule is rewritten as

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \tau) &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\left(Q(\mathbf{s}_t, \mathbf{a}_t) - V(\mathbf{s}_t)\right) \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) A(\mathbf{s}_t, \mathbf{a}_t) \\ \boldsymbol{\theta} &= \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \tau)\end{aligned} \tag{3.1}$$

where $A(\mathbf{s}_t, \mathbf{a}_t)$ is the *advantage function* that measures how much better of an action $\mathbf{a}_t$ over the average performance of all actions in state $\mathbf{s}_t$, in which the average performance is estimated using the state-value function $V(\mathbf{s}_t)$.

The update rule in Equation (3.1) is well-known as batch gradient ascent, which computes the gradient of the objective with respect to the parameter vector $\boldsymbol{\theta}$ for the entire sampled trajectories for just one update, and thus, batch gradient ascent is very slow and is intractable for high-dimensional state and action spaces that do not fit in memory. Training a deep neural network can be demanding as the parameters update only a small step with each iteration. To perform multiple steps of updating the parameters, we estimate the policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ using mini-batch gradient

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}; \tau^{(i:i+N)}) &= \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}(\mathbf{s}_t, \mathbf{a}_t)] \\ \boldsymbol{\theta} &= \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}; \tau^{(i:i+N)})\end{aligned} \tag{3.2}$$

where the expectation $\hat{\mathbb{E}}_t[\cdots]$ indicates that the empirical average over a batch of samples of size N, commonly range between 32 and 512. Here, α is the learning rate that determines the update step of parameters. How can we define the loss function which is the essential part in most *automatic differentiation* software, e.g., TensorFlow, such that its gradient is the policy gradient $\nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}; \tau^{(i:i+N)})$? The idea is to use the weighted maximum likelihood (MLE) expression, and thus,

$$\max_{\boldsymbol{\theta}} \quad \hat{J}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t[\log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\hat{A}(\mathbf{s}_t, \mathbf{a}_t)]. \tag{3.3}$$

Alternatively, TRPO proposed to use a "surrogate" objective whose gradient with respect to the policy parameters is the same as in Equation (3.3). More importantly, the "surrogate" objective has *importance sampling* interpolation, a method for reducing the variance of the estimate of an expectation by carefully choosing a sampling distribution [34]. The "surrogate" objective is defined as

$$\begin{aligned}\max_{\boldsymbol{\theta}} \quad \hat{J}^{\text{surr}}(\boldsymbol{\theta}) &= \hat{\mathbb{E}}_{(\mathbf{s}_t, \mathbf{a}_t)\sim\pi_{\boldsymbol{\theta}_k}}\left[\hat{A}(\mathbf{s}_t, \mathbf{a}_t)\right] \\ &= \hat{\mathbb{E}}_{(\mathbf{s}_t, \mathbf{a}_t)\sim\pi_{\boldsymbol{\theta}_k}}\left[\frac{\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)}\hat{A}(\mathbf{s}_t, \mathbf{a}_t)\right]\end{aligned} \tag{3.4}$$

which has the same policy gradient as the objective of the averaged advantage weighted by log probability

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \hat{J}^{\text{surr}}(\boldsymbol{\theta}) &= \hat{\mathbb{E}}_t\left[\nabla_{\boldsymbol{\theta}}\left(\frac{\pi_{\theta_{k+1}}}{\pi_{\theta_k}}\right)\Big|_{\theta_k}\hat{A}(\mathbf{s}_t, \mathbf{a}_t)\right] \\ &= \hat{\mathbb{E}}_t\left[\left(\frac{\nabla_{\boldsymbol{\theta}}\pi_{\theta_{k+1}}|_{\theta_k}}{\pi_{\theta_k}}\right)\hat{A}(\mathbf{s}_t, \mathbf{a}_t)\right] \\ &= \hat{\mathbb{E}}_t[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)|_{\theta_k}\hat{A}(\mathbf{s}_t, \mathbf{a}_t)] \\ &= \nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}).\end{aligned}$$

Apart from using the mini-batch gradient, most supervised learning also performs multiple epochs of optimization using the same dataset. Thus, it is appealing to reuse the same sampled trajectory for updating the parameters, which however often leads to destructively large policy updates when optimizing the Equation (3.3) or Equation (3.4) directly. To achieve stable multiple epochs of optimization, PPO optimizes a KL-divergence penalized "surrogate" objective, or a clipped "surrogate" objective using SGD. Similarly, our proposed method also employs the KL-divergence as a regularization to achieve stable policy improvement.

3.2.2 The objectives

Consider the following constrained policy optimization problem from TRPO

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & J(\pi_{\boldsymbol{\theta}}) = \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}(\mathbf{a}_t|\mathbf{s}_t) \right] \\ \text{s.t.} \quad & \hat{\mathbb{E}}_t \left[D_{\text{KL}}[\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)||\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)] \right] \leq \epsilon \end{aligned} \tag{3.5}$$

where $\pi_{\boldsymbol{\theta}}$ represents a parameterized stochastic policy to be optimized and $\hat{A}(\mathbf{s}_t, \mathbf{a}_t)$ is the advantage estimate calculated from trajectories $(\mathbf{a}_0, \mathbf{s}_0, r_0, \cdots)$ that are sampled using the policy $\pi_{\boldsymbol{\theta}_k}$. The KL-divergence $\hat{\mathbb{E}}_t \left[D_{\text{KL}}[\pi_{\theta_{k+1}}(\cdot|\mathbf{s}_t)||\pi_{\theta_k}(\cdot|\mathbf{s}_t)] \right]$ measures the difference between the distribution of new updated policy $\pi_{\boldsymbol{\theta}_{k+1}}$ and the policy distribution $\pi_{\boldsymbol{\theta}_k}$ before update over the sampled states at time step t. Here, the hyperparameter $\epsilon \in \mathbb{R}$ is the maximum KL bound that usually set to be a small number, e.g., $\epsilon = 0.01$. The standard method of solving such constrained optimization problems is by optimizing the Lagrangian dual, which is known to be convex and smoothly differentiable [35], for example, by using the Broyden-Fletcher-Goldfarb-Shannon (BFGS) method [36]. For this particular probelm, even a closed-form solution for the optimal policy can be found [20, 37] without tuning the learning rate like in stochastic gradient ascent method. On the other hand, such an algorithm is difficult to optimize for problems with high dimensional state and action spaces because the dual optimization becomes prohibitively slow, and is not straightforward to implement.

Instead of solving the constrained optimization problem by introducing Lagrangian multipliers and minimizing the Lagrangian dual functions like in REPS, or by using approximations to the objective and the constraint as in TRPO, we propose to use a so-called *minimax entropic policy optimization* (MMPO) method, in which we maximize the objective directly and minimize a second loss function that is defined by mapping the relative entropy constraint with a ramp function $f(x) = \max(x, 0)$, whose graph is shaped like a ramp as illustrated in Figure 3.1. Therefore, we obtain the following optimization problem that has two objectives for updating the parameters of the policy function approximator

$$\max_{\boldsymbol{\theta}} \quad J(\pi_{\boldsymbol{\theta}}) = \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}(\mathbf{a}_t|\mathbf{s}_t) \right] \tag{3.6}$$

$$\min_{\boldsymbol{\theta}} \quad L_{\text{KL}}(\pi_{\boldsymbol{\theta}}) = \max \left(\hat{\mathbb{E}}_t \left[D_{\text{KL}}[\pi_{\theta_{k+1}}(\cdot|\mathbf{s}_t)||\pi_{\theta_k}(\cdot|\mathbf{s}_t)] \right] - \epsilon, 0 \right) \tag{3.7}$$

where maximizing the first objective $J(\pi_{\boldsymbol{\theta}})$ could improve the performance of the policy, and simultaneously minimizing the second objective L_{KL} prevents the policy update from going into an untrusted region and greedy policy improvement. Maximizing $J(\pi_{\boldsymbol{\theta}})$ without minimizing the second objective in Equation (3.7) would lead to a catastrophically greedy update of the policy and premature convergence.

The main property of the MMPO method is that the KL-divergence constraint in Equation (3.5) can be formulated as a secondary loss function using a ramp function transformation, which is $f(x) = \max(x, 0)$. By minimizing the KL loss function, the policy update step can be effectively constrained inside a trust region. There are two different cases of the KL-divergence: first, the KL-divergence violates the KL bound ϵ, second, the KL-divergence meets the KL bound. Thus,

- when $\hat{\mathbb{E}}_t \left[D_{\text{KL}}[\pi_{\theta_{k+1}}(\cdot|\mathbf{s}_t)||\pi_{\theta_k}(\cdot|\mathbf{s}_t)] \right] \leq \epsilon$, the update steps of policy parameters are not penalized as the gradient of the KL loss function is equal to 0

$$\begin{aligned} \frac{\partial}{\partial \theta} L_{\text{KL}}(\pi_\theta) &= \frac{\partial}{\partial \theta} 0 \\ &= 0, \end{aligned}$$

- when $\hat{\mathbb{E}}_t \left[D_{\text{KL}}[\pi_{\theta_{k+1}}(\cdot|\mathbf{s}_t)||\pi_{\theta_k}(\cdot|\mathbf{s}_t)] \right] > \epsilon$, the update steps of policy parameters are penalized following the gradient direction of the KL loss function

$$\begin{aligned} \frac{\partial}{\partial \theta} L_{\text{KL}}(\pi_\theta) &= \frac{\partial}{\partial \theta} \left(\hat{\mathbb{E}} \left[D_{\text{KL}}[\pi_{\theta_{k+1}}(\cdot|\mathbf{s}_t)||\pi_{\theta_k}(\cdot|\mathbf{s}_t)] \right] - \epsilon \right) \\ &= \frac{\partial}{\partial \theta} \left(\hat{\mathbb{E}} \left[D_{\text{KL}}[\pi_{\theta_{k+1}}(\cdot|\mathbf{s}_t)||\pi_{\theta_k}(\cdot|\mathbf{s}_t)] \right] \right) - 0 \\ &= \frac{\partial}{\partial \theta} \left(\hat{\mathbb{E}} \left[D_{\text{KL}}[\pi_{\theta_{k+1}}(\cdot|\mathbf{s}_t)||\pi_{\theta_k}(\cdot|\mathbf{s}_t)] \right] \right). \end{aligned}$$

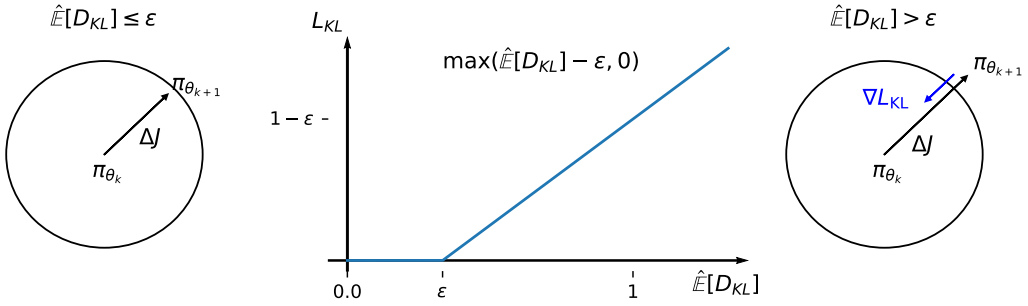

Figure 3.1.: Middle: The ramp function $f(x) = \max(x, 0)$ that is used to transform the KL-divergence constraint. **Left:** When the expection of the KL-divergence satisfies $\mathbb{E}_t[D_{\text{KL}}] \leq \epsilon$, the gradient of $L_{\text{KL}}(\pi_{\boldsymbol{\theta}})$ equals 0, hence no information loss penalty for the policy update step. **Right:** When $\mathbb{E}_t[D_{\text{KL}}] > \epsilon$, the larger the divergence $\mathbb{E}_t[D_{\text{KL}}]$ is, the steeper is the gradient $\nabla_{\boldsymbol{\theta}} L_{\text{KL}}(\pi_{\boldsymbol{\theta}})$, and hence, the greater is the information loss penalty for the policy update step.

Figure 3.1 illustrates how an averaged KL-divergence being mapped to the loss function L_{KL} using the ramp function, and how the policy updates look like when the KL-divergence is in different scenarios.

The estimated *advantage function* $\hat{A}(\mathbf{s}_t, \mathbf{a}_t)$ in Equation (3.6) is computed using the *generalized advantage estimation* (GAE) [25] scheme, where a state-value function $\hat{V}(\mathbf{s})$ is required and can be estimated using function approximation methods, e.g. non-linear function approximation. The optimization of the value function is usually treated as a supervised learning problem and can be solved efficiently using stochastic gradient descent methods by minimizing a mean-square-error (MSE) loss function. The MSE loss function for the state-value function $\hat{V}(\mathbf{s})$ is defined as

$$\min_{\boldsymbol{\phi}} \quad L(V_{\boldsymbol{\phi}}) = \mathbb{E}_t \left[\| \hat{V}_{\boldsymbol{\phi}}(\mathbf{s}_t) - \sum_{l=0}^{\infty} \gamma r_{t+l} \|^2 \right] \tag{3.8}$$

where $\sum_{l=0}^{\infty} \gamma^l r_{l+t}$ is the discounted cumulative reward from state $\mathbf{s}_t$ over all time steps in batch of trajectories and serves as the ground truth state value in state $\mathbf{s}_t$, and $\hat{V}_{\boldsymbol{\phi}}(\mathbf{s}_t)$ is the predicted state value in state $\mathbf{s}_t$. The parameter vector of the value function approximator is denoted as $\boldsymbol{\phi}$.

Entropy regularization

Exploration and exploitation, which are usually determined by the policy entropy, play an essential role in almost all RL algorithms. Maximizing the objective in Equation (3.6) repeatedly using several epochs of SGD optimization with same sampled trajectories inevitably violates the KL-bound, and thus, results in a large penalty on the policy update step. With an increasing number of policy updates, the KL divergence between newly updated policy and old policy keep increasing, hence, violates the KL bound and triggers a minimization process of the KL loss function. Minimizing the KL loss function not only keeps policy update inside a trust region but also prevents the policy entropy from decreasing. Therefore, we suggest adding an additional entropy regularization (constraint) to encourage exploitation

$$\text{subject to} \quad \hat{\mathbb{E}}_t \left[H(\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)) - H(\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)) \right] \leq \beta$$

where $\beta \in \mathbb{R}$ is commonly set to be a small positive value, e.g. 0.002, or a large negative value, e.g. $\beta = -0.002$. The hyperparameter β is a trade-off between exploration and exploitation, when $\beta > 0$, it encourages exploration, and when $\beta < 0$, it encourages exploitation. Again, the entropy constraint is a soft inequality constraint, and hence can be formulated as a loss function using the ramp function transformation as in Equation (3.7). The *entropy regularization loss function* is defined as

$$\min_{\boldsymbol{\theta}} \quad L_{\text{Ent}}(\pi_{\boldsymbol{\theta}}) = \max \left(\hat{\mathbb{E}}_t \left[H(\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)) - H(\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)) \right] - \beta, 0 \right)$$

which is optimized using SGD that is computationally efficient. For example, for a normal distribution $\mathcal{N}(\mu, \sigma)$, the entropy of the normal distribution is defined as $H = \ln(\sigma\sqrt{2\pi e})$, which only depends on the variance σ. The minimization of the entropy regularization loss function affects exclusively the variance of the policy distribution.

3.2.3 The algorithm

Now, we present the optimization algorithm (**Algorithm 1**) of our MMPO method. MMPO has four objectives to satisfy: maximize expected advantage, minimize KL-divergence loss function, minimize entropy loss function, minimize MSE loss function of the value function. We use function approximation methods, such as deep neural networks, to represent both the policy and the value function, whose parameters are updated using stochastic gradient descent/ascent methods. To improve sample efficiency, MMPO optimizes parameters of function approximators with multiple epochs of mini-batch stochastic gradient optimization using the same sampled trajectory.

The stochastic gradient optimizer is a crucial factor in our MMPO method, as all objectives are updated using the first-order gradient optimization method. Traditional SGD maintains a single learning rate for all parameters and does not adopt the learning rate during training. Many advanced optimizers have been discussed in Chapter 2, such as the Adam optimizer that computes adaptive learning rates for each parameter. Many of the advanced optimizers require a small amount of memory for storing parameters that indicate the gradient properties of the objective is optimized. Thus, we suggest using four advanced SGD optimizers for optimizing the four objectives, respectively, in order to maintain individual learning rates for the gradients of each objective.

Algorithm 1 Minimax entropic policy optimization (MMPO)

Initialize policy parameters $\boldsymbol{\theta}_0 = [\boldsymbol{\omega}, \boldsymbol{\Sigma}]$, where $\boldsymbol{\omega}$ and $\boldsymbol{\Sigma}$ are parameters of the function approximator and the variance of the policy distribution respectively
Initialize value function parameters $\boldsymbol{\phi}_0$, KL bound ϵ, entropy regularization parameter β
for iteration updates: $k = 0, 1, 2, \cdots$ **do**
  Run current policy $\pi_{\boldsymbol{\theta}_k}$ and collect a set of trajectories $\mathcal{D} = (\mathbf{a}_0, \mathbf{s}_0, r_0, \mathbf{a}_1, \mathbf{s}_1, r_1, \cdots)$ of length T
  Estimate the advantages $\hat{A}_t$
  for epoch updates $n = 0, 1, 2, \cdots, N$ **do**
    for mini-batch updates $m = 0, 1, 2, \cdots, M$ **do**
      Optimize the policy via a minimax game between two SGD optimizers

$$\boldsymbol{\theta}_{\text{new}} \leftarrow \underset{\boldsymbol{\theta}}{\text{argmax}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}(\mathbf{a}_t|\mathbf{s}_t) \right]$$

$$\boldsymbol{\theta}_{\text{new}} \leftarrow \underset{\boldsymbol{\theta}}{\text{argmin}} \quad \max \left(\hat{\mathbb{E}}_t \left[D_{\text{KL}}[\pi_{\theta_{k+1}}(\cdot|\mathbf{s}_t) || \pi_{\theta_k}(\cdot|\mathbf{s}_t)] \right] - \epsilon, 0 \right)$$

      Entropy regularization

$$\boldsymbol{\Sigma}_{\textbf{new}} \leftarrow \underset{\boldsymbol{\Sigma}}{\text{argmin}} \quad \max \left(\hat{\mathbb{E}}_t \left[H(\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)) - H(\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)) \right] - \beta, 0 \right)$$

      Optimize the value function

$$\boldsymbol{\phi}_{\text{new}} \leftarrow \underset{\boldsymbol{\phi}}{\text{argmin}} \quad \mathbb{E}_t \left[\| \hat{V}_{\boldsymbol{\phi}}(\mathbf{s}_t) - \sum_{l=0}^{\infty} \gamma r_{t+l} \|^2 \right]$$

    end for
  end for
end for

3.3 Connections with prior work

TRPO [10]: TRPO is a constrained policy optimization algorithm that is closely related to the natural policy gradient method and relative entropy policy search (REPS), all of which impose a KL-divergence constraint to ensure the policy improvement inside a trust region, however, are not straightforward to implement. The MMPO method, on the other hand, is an unconstrained policy optimization problem which can be solved efficiently using mini-batch gradient descent algorithms and is much easier to implement. At the same time, MMPO enjoys the benefits of trust region policy optimization by minimizing a secondary loss function of the KL-divergence.

PPO [16]: There are two different versions of the proximal policy optimization (PPO) algorithm, one has a "surrogate" objective being penalized by the KL-divergence, another one has a clipped "surrogate" objective, both objectives can be optimized using the SGD method. PPO achieves trust region policy optimization indirectly by either penalizing the objective using the KL-divergence or forming a pessimistic bound on the unclipped objective. Both MMPO and PPO have better sample complexity and are straightforward to implement compare to other policy search method, e.g. TRPO [10], REPS [20], ACKTR [11].

GANs [17]: There is a close relationship between GANs and MMPO, as both methods conduct a minimax game between two players. Specifically, GANs train a generator in opposition to a discriminator via optimizing two different loss functions simultaneously, and MMPO seeks to optimize the policy within a trust region through an adversarial optimization procedure of maximizing a "surrogate" objective and minimizing the KL-divergence loss function simultaneously by two SGD optimizers.

3.4 Experiments

We ran MMPO on a collection of standard simulated robotic locomotion benchmark tasks from the OpenAI MuJoCo task suite. Since MMPO is closely related to TRPO and PPO, we compared MMPO to those algorithms to answer the following two questions:

1. MMPO introduces several modifications to TRPO—most notably, it proposes a novel objective that can be optimized using the first-order information only—which makes implementation much more straightforward, but how does this affect the performance of the algorithm?

2. Both MMPO and PPO use SGD method for the optimization and have few open parameters. However, PPO optimizes a clipped "surrogate" objective directly, and MMPO optimizes multiple objectives simultaneously. Can we observe a difference in the performance of the two algorithms?

3.4.1 Policy and value function

The stochastic policy, which is defined as a conditional probability distribution $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$, is parameterized with a multilayer perceptron (MLP). For continuous control tasks, we define the stochastic policy as a Gaussian probability distribution $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mathbf{a}|f_{\boldsymbol{\omega}}(\mathbf{s}), \boldsymbol{\Sigma})$ with state-dependent mean, a function $f_{\boldsymbol{\omega}}(\mathbf{s})$ represented by a MLP, and the state-independent standard deviation that is calculated from a separate set of parameters $\boldsymbol{\Sigma}$ that specify the log standard deviation of each action. Here, the output of the MLP is the mean of the Gaussian probability distribution corresponding to the input states. Thus, the parameters of the policy is $\boldsymbol{\theta} = [\boldsymbol{\omega}, \boldsymbol{\Sigma}]$. For discrete actions, the policy can be defined by a softmax distribution with probabilities of actions propotional to exponentiated weights $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \propto \exp\left(f_{\boldsymbol{\theta}}(s)\right)$, where the function $f_{\boldsymbol{\theta}}(s)$ can be any function approximator with trainable weights $\boldsymbol{\theta}$.

The value function is also represented using a MLP, denoted as $V_{\boldsymbol{\phi}}(\mathbf{s})$, where $\boldsymbol{\phi}$ is the parameter vector of the MLP. The output of the value function is a single scale, which indicates the goodness or badness of a state $\mathbf{s}$. Table 3.1 presents architecture details of both the policy network and the value network.

	Policy network	Value network
Input layer	$N_{\mathbf{s}}$	$N_{\mathbf{s}}$
Hidden layers	FC-64 + ReLu FC-64 + ReLu	FC-64 + ReLu FC-64 + ReLu
Output layer	$N_{\mathbf{a}}$	1

Table 3.1.: Architecture of the policy network and value function network. Both networks are multilayer perceptrons (MLPs). $N_{\mathbf{s}}$ is the dimension of the state space. $N_{\mathbf{a}}$ is the dimension of the action space. "FC-64" represents the fully connected layer of 64 units. ReLu is the activation function.

3.4.2 MuJoCo environment

The MuJoCo environment [30], which stands for **Mu**lti-**Jo**int dynamics with **Co**ntact environment, contains several challenging continuous control tasks that run in a physics simulator. The MuJoCo environment is a benchmark task that

has used for testing RL algorithms by many well-known methods, such as PPO, ACKTR, etc. In this paper, we conducted experiments on 7 different 2D environments, as shown in Figure 3.2, to compare the results of our MMPO method to the results of TRPO and PPO methods. The environments have different goals and reward functions. Additionally, we provide environments' details in Table 3.2, including dimensions of the state space and the action space.

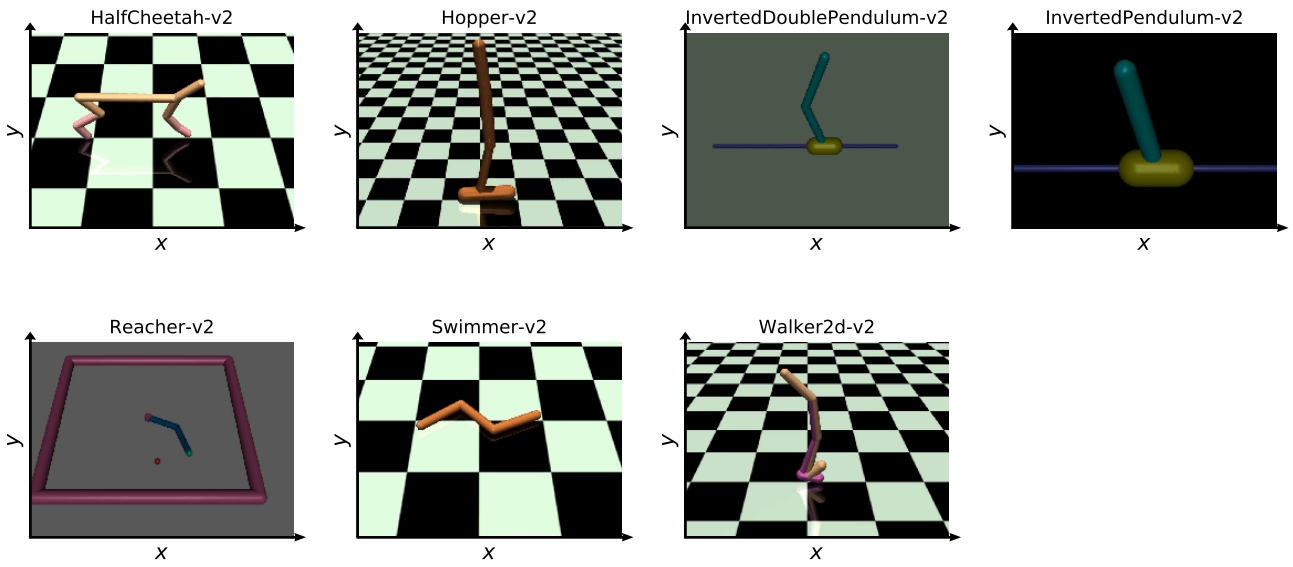

Figure 3.2.: Snapshots of 7 different 2D continuous control tasks that run in the MuJoCo physics engine with 3D scenes.

Environment ID	Dimension of state space	Dimension of action space
HalfCheetah-v2	17	6
Hopper-v2	11	3
InvertedDoublePendulum-v2	11	1
InvertedPendulum-v2	4	1
Reacher-v2	11	2
Swimmer-v2	8	2
Walker2d-v2	17	6

Table 3.2.: Details of several OpenAI MuJoCo emvironments

HalfCheetah-v2: The goal is to make a 2D cheetah robot with two legs run as fast as possible. The reward function is defined to encourage a high-speed running in the x direction, and simultaneously to punish large torques $\mathbf{u}$ that could potentially damage the robot in reality. The reward function for learning is defined as

$$r(x_t, \mathbf{u}_t) = \frac{(x_{t+1} - x_t)}{dt} - 0.1 \times \sum_{u \in \mathbf{u}_t} u^2.$$

Hopper-v2: The goal is to make a 2D robot with one leg hop as fast as possible. The reward function is defined to encourage a high-speed hopping in the x direction, and simultaneously to punish large torques $\mathbf{u}$. Additionally, a bonus reward of 1 is added to the reward function to keep the robot stay an upright posture without falling on the ground. The reward function for learning is defined as

$$r(x_t, \mathbf{u}_t) = 1 + \frac{(x_{t+1} - x_t)}{dt} - 0.001 \times \sum_{u \in \mathbf{u}_t} u^2.$$

InvertedDoublePendulum-v2: To balance a two-joint pole on a cart using the reward function in which 10 is the alive bonus reward for keeping the pole on the cart, a penalty term of punishing the position of the pole, and a second penalty term of punishing the velocity of the pole. The reward function for learning is defined as

$$r(x_t, y_t, \mathbf{u}_t) = 10 - (0.01x_t^2 + (y_t - 2)^2) - (10^{-3}v_x^2 + 5 \times 10^{-5}v_y^2).$$

InvertedPendulum-v2: To balance a one-joint pole on a cart using the reward function where 1 is the alive bonus reward for keeping the pole on the cart. No additional penalty terms are added. The reward function for learning is defined as

$$r(x_t, \mathbf{u}_t) = 1.$$

Reacher-v2: The robot is trying to reach a randomly located target using its end-effector, where the target position is $(\hat{x}, \hat{y})$, the end-effector position is (x_t, y_t), and the torques of the robot is $\mathbf{u}_t$. The reward function for learning is defined as

$$r(x_t, y_t, \mathbf{u}_t) = -\sqrt{(x_t - \hat{x}_t)^2 + (y_t - \hat{y}_t)^2} - \sum_{u \in \mathbf{u}_t} u^2.$$

Swimmer-v2: The goal is to make the robot swim as fast as possible. The reward function is defined to encourage a high-speed swimming in the x direction, and simultaneously to punish large torques $\mathbf{u}$. The reward function for learning is defined as

$$r(x_t, \mathbf{u}_t) = \frac{(x_{t+1} - x_t)}{dt} - 0.0001 \times \sum_{u \in \mathbf{u}_t} u^2.$$

Walker2d-v2: The goal is to make the robot with two legs walk as fast as possible without falling down. The reward function is defined to encourage a high-speed walking in the x direction, and simultaneously to punish large torques $\mathbf{u}$. Additionally, a bonus reward of 1 is added to the reward function to keep the robot stay an upright posture. The reward function for learning is defined as

$$r(x_t, \mathbf{u}_t) = 1 + \frac{(x_{t+1} - x_t)}{dt} - 0.001 \times \sum_{u \in \mathbf{u}_t} u^2.$$

3.4.3 Results

First, we conduct a series of experiments on several OpenAI MuJoCo environments in search of hyperparameters, most importantly, of KL-divergence bound ϵ and entropy regularization β. For each hyperparameter pair (ϵ, β), we train the policy for 6 random seeds, and with a total number of 1,000,000 timesteps for each seed. We compute an averaged normalized score by averaging the cumulative reward from running the last updated policy on the environment for 20 episodes, and then, shifting and scaling the scores for each environment such that a random policy has a score of 0 and the best result was set to 1. Apart from the KL bound ϵ and the parameter used for entropy regularization β, the other hyperparameters are given in Table A.1.

ϵ	β	score
0.01	N/A	0.904
	-0.002	0.890
	-0.004	0.849
	-0.01	0.861
0.015	N/A	**0.913**
	-0.002	0.901
	-0.005	0.898
	-0.01	0.874
0.02	N/A	0.87
	-0.003	**0.916**
	-0.004	0.866
	-0.01	0.873

Table 3.3.: Comparison of normalized scores of MMPO experiments on 7 MuJoCo environments. Averaged normalized scores for each hyperparameter setting, where each setting has experimented with 6 random seeds. Scores are based on the average performance of last updated policies over different environments and random seeds. The "N/A" indicates that entropy regularization was not used. Negative values β specify the decreasing rate of the policy entropy that should be achieved by minimizing the entropy regularization loss function.

Secondly, we compare the MMPO without using the entropy regularization with the MMPO with an entropy regularization term (MMPO+Ent), by running experiments on the 7 MuJoCo environments and showing the learning curve of the averaged cumulative reward, the entropy, and the KL-divergence. The results in Figure 3.3 indicates that the entropy decreasing rate trades off between the exploration and the exploitation, and then, influence the policy performance shown in Figure 3.3. For example, on the simple InvertedDoublePendulum-v2 environment where the exploitation is more important than the exploration, MMPO+Ent convergences faster than the MMPO method. However, in the more complicated environment of the Walker2d-v2 where the agent should keep exploring, MMPO outperforms the MMPO+Ent method in the end. Additionally, we also run the final updated policies of 6 random seeds on the 7 environments and show the averaged cumulative reward over 20 episodes, where the results are shown in Table 3.4.

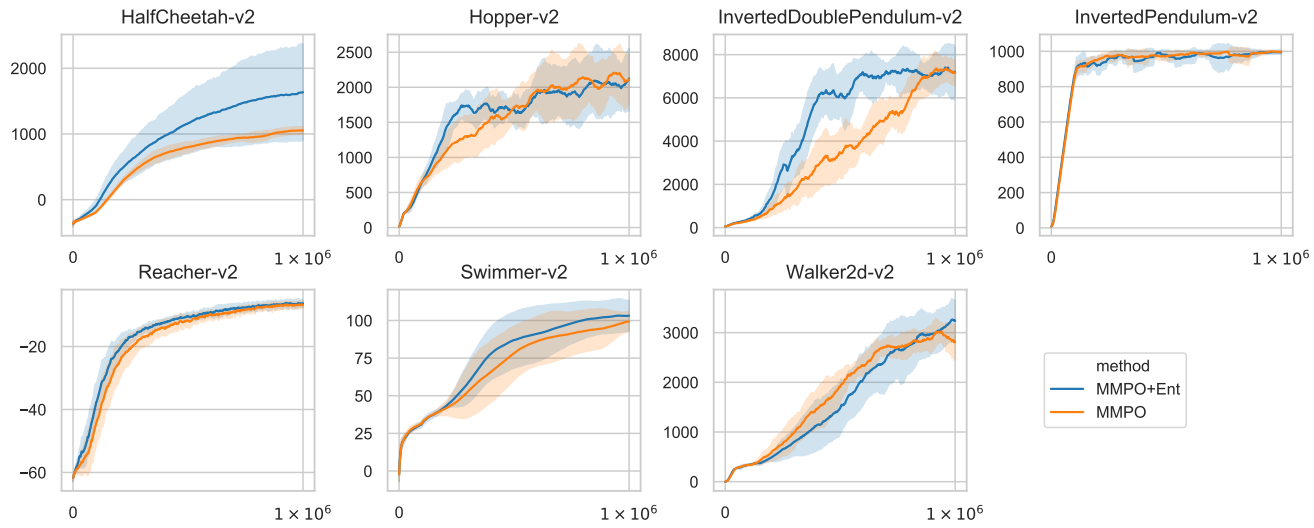

Figure 3.3.: Comparisons of policy performance on 7 MuJoCo environments between MMPO and MMPO+Ent, where MMPO+Ent represents MMPO with entropy regularization. The shaded region indicates the standard deviation over 6 random seeds. The total number of sampled timesteps is 1×10^6 for each experiment.

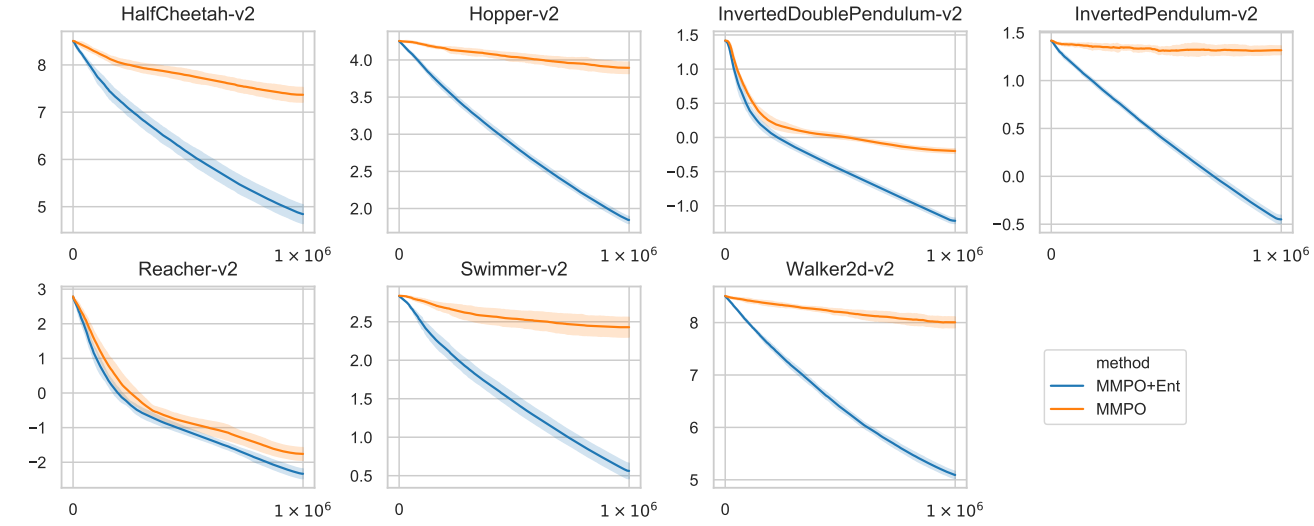

Figure 3.4.: Comparisions of policy entropy dynamics between MMPO and MMPO+Ent. The entropy regularization hyperparameter is $\beta = -0.002$, which specifies the minimum decreasing rate of the policy entropy that has to be satisfied. The shaded region indicates the standard deviation over 6 random seeds.

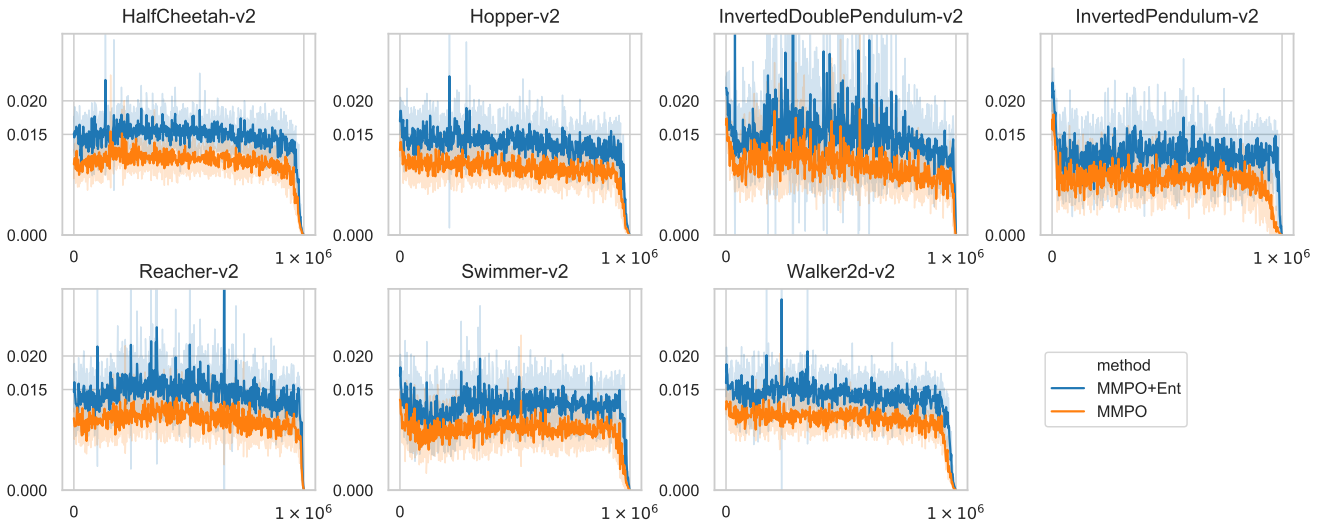

Figure 3.5.: Comparisions of the KL-divergence dynamics between MMPO and MMPO+Ent methods, where the KL-divergence bounds are $\epsilon = 0.015$ and $\epsilon = 0.02$ respectively.

Task	MMPO ($\epsilon = 0.015$)	MMPO+Ent ($\epsilon = 0.02$, $\beta = -0.003$)
InvertedPendulum-v2	**1000.00** $\pm$ 0.00	**1000.00** $\pm$ 0.00
InvertedDoublePendulum-v2	**9128.91** $\pm$ 505.38	8156.99 $\pm$ 1915.10
Reacher-v2	-8.05 $\pm$ 1.45	**-7.66** $\pm$ 1.62
Hopper-v2	**3143.46** $\pm$ 710.90	2163.49 $\pm$ 868.79
Swimmer-v2	83.7 $\pm$ 24.33	**93.7** $\pm$ 18.28
HalfCheetah-v2	1238.43 $\pm$ 64.87	**1769.00** $\pm$ 770.39
Walker2d-v2	**3441.81** $\pm$ 548.96	3130.32 $\pm$ 985.00

Table 3.4.: Comparison of the averaged culmulative reward from running the final obtained policies on 7 MuJoCo environments over 6 random seeds and 20 episodes for each seed.

Comparisons among MMPO, TRPO and PPO

Finally, we compare MMPO (with entropy regularization) to TRPO and PPO (with clipped "surrogate" objective), both of which have achieved state-of-the-art results on the MuJoCo environments. For running the TRPO and PPO experiments, we use open-source code from the OpenAI baseline [38] and the default hyperparameters. For our MMPO method, we use the hyperparameters shown in Table A.1 in Appendix A, additionally with the KL-divergence bound $\epsilon = 0.02$ and an entropy regularization of the minimum decreasing rate $\beta = -0.003$. We focus our comparisons on three different aspects: the reward, the entropy, and the KL-divergence, where the reward curve indicates how the policy is being improved, the entropy shows the information loss of the policy probability distribution and the KL-divergence shows how stable is the policy improvement.

Figure 3.6 demonstrates a series of comparisons of the learning curve on MuJoCo environments among MMPO, TRPO, and PPO. The results show that MMPO achieves comparable performance to PPO and TRPO on almost all MuJoCo environments. For example, on the easiest InvertedPendulum-v2 or the high-dimensional Walker2d-v2 environment, MMPO outperforms both TRPO and PPO methods slightly. On both HalfCheetah-v2 and Hopper-v2 environments, all methods demonstrate similar performances. On the InvertedDoublePendulum-v2 and Reacher-v2 environments, MMPO method converges slightly slower than the other two methods, however, all methods arrive at a similar point in the end. Moreover, on the Swimmer-v2, MMPO and TRPO perform much better than PPO.

Figure 3.7 shows comparisons of the entropy performance during training, where PPO and TRPO do not use any entropy regularization and do not determine the entropy decreasing rate explicitly. MMPO, however, minimizes a ramp-transformed entropy loss function, similar with the KL-divergence loss function, and is capable of control the entropy decreasing rate directly. For example, on the simple InvertedPendulum-v2 environment in which the dimension of action

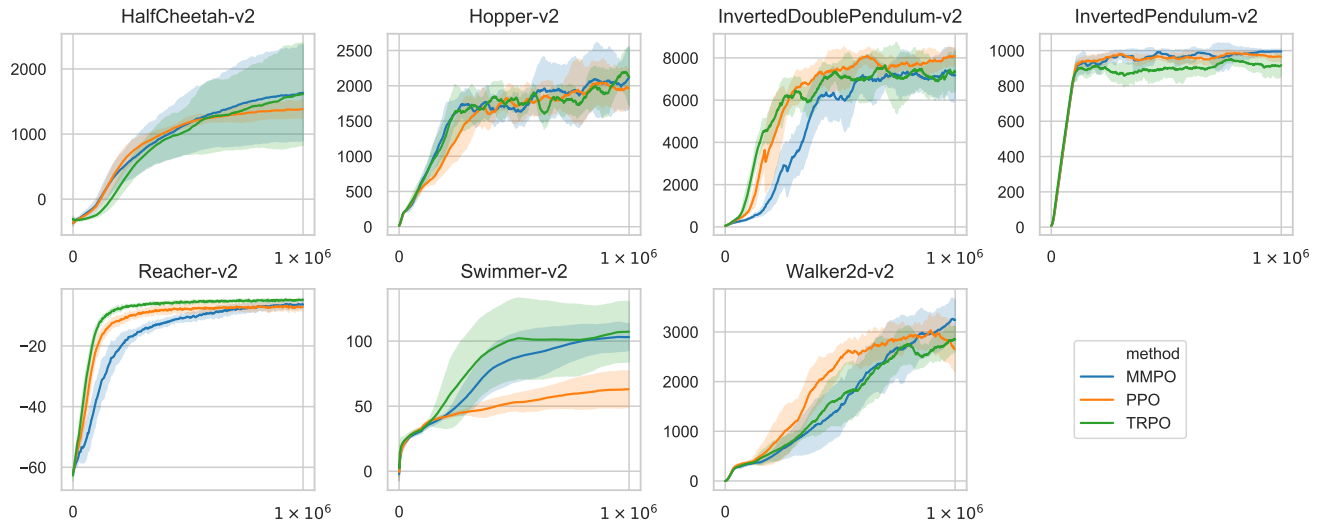

Figure 3.6.: Comparisons of averaged culmulative reward among MMPO, TRPO, and PPO methods, during learning on 7 MuJoCo environments trained for 1 million timesteps. The shaded region indicates the standard deviation over 6 random seeds.

space is only 1, the agent does not need to explore much but should focus more on exploitation. MMPO has better performance in this environment due to its lowest policy entropy in the end. However, on the more complicated Walker2d-v2 environment that has a high-dimensional action space of 6, explorations are encouraged. Hence, MMPO achieved the best final performance because of a relatively slow entropy decreasing rate.

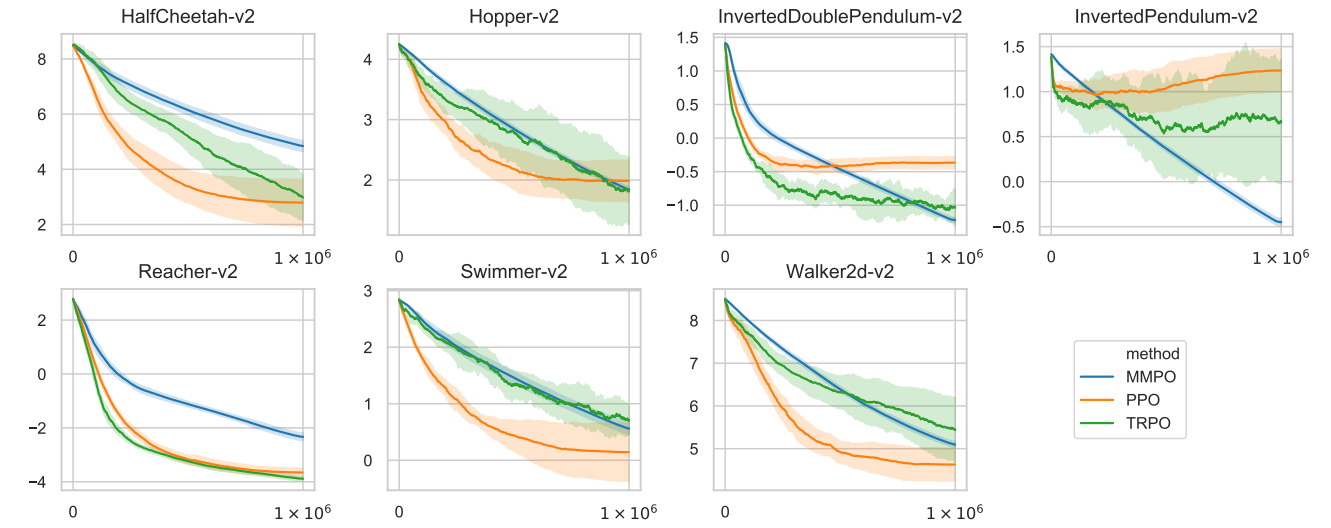

Figure 3.7.: Comparisons of the policy entropy performance among MMPO, TRPO, and PPO methods, during learning on 7 MuJoCo environments trained for 1 million timesteps. The shaded region indicates the standard deviation over 6 random seeds.

Figure 3.8 presents the change of the KL-divergence during the policy optimization, where our MMPO method in most cases restricts the KL-divergence under the KL bound $\epsilon = 0.02$ during the whole training procedure. As the MMPO method treats the KL-divergence bound as a soft constraint, it is inevitable for MMPO to occasionally violate the KL-divergence bound. The KL bound for TRPO method is 0.01, which is not always satisfied as well due to a quadratic approximation to the KL-divergence constraint. PPO can only have an indirect influence on the KL-divergence by clipping the "surrogate" objective, which can be not straightforward to implement in some cases, e.g. for policy gradient methods that do not use the "surrogate" objective, instead, an averaged log action probability weighted advantages. The decreasing curve of the KL-divergence of PPO is caused by a "linear" schedule of the clipping hyperparameter $\epsilon = 0.2 \times m$,

where m is initialized to 1 and linearly decreased to 0 through the training.

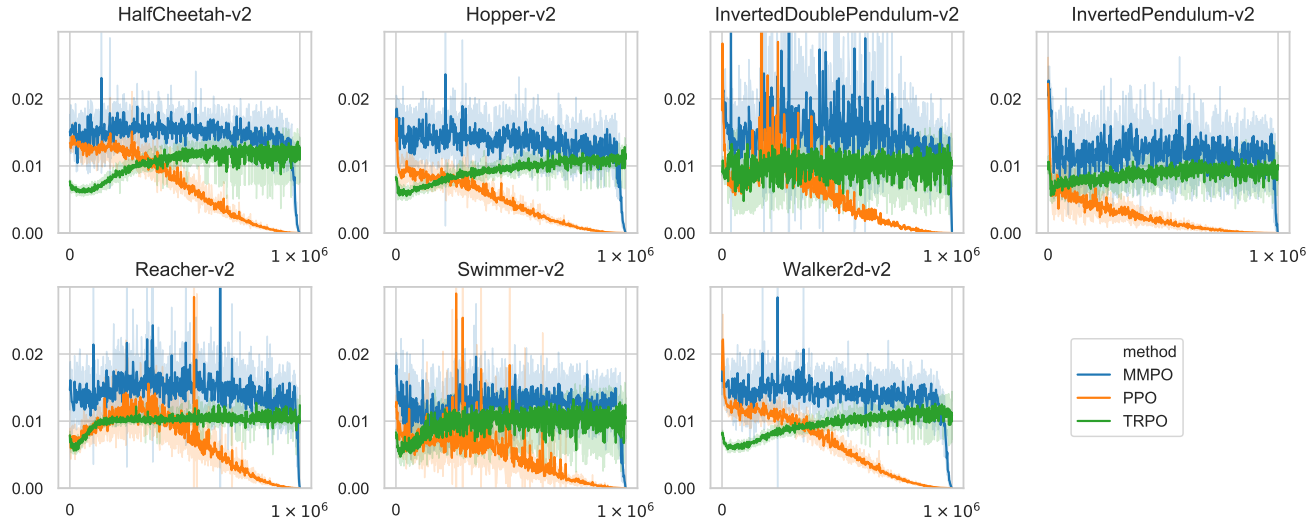

Figure 3.8.: Comparisons of the KL-divergence among MMPO, TRPO and PPO, during training on 7 MuJoCo environments trained for 1 million timesteps. The shaded region indicates the standard deviation over 6 random seeds.

3.5 Conclusion

We have proposed a practical policy gradient method for deep RL, where the policy network is optimized by simultaneously maximizing a "surrogate" objective and minimizing a ramp-transformed KL-divergence loss function, and a state value function is found by minimizing the commonly used MSE loss function. The empirical results on 7 MuJoCo environments indicate that the MMPO method is capable of solving a series of challenging continuous control tasks and achieving state-of-the-art results on MuJoCo environments like TRPO and PPO methods. However, the MMPO method is more straightforward to implement and is computationally efficient compared with the TRPO method. Additionally, the MMPO method is compatible with optimizing the objective of averaged logarithm-weighted advantages, however, PPO method is limited to maximizing the "surrogate" objective. Moreover, with the idea of transforming a soft constraint via a ramp function into an objective that can be optimized using SGD methods, it is likely that many constrained optimization problems with soft constraints can be effectively solved using first-order gradient descent methods, instead of using the Lagrangian mechanics which is intractable in most deep neural networks.

4 f-divergence penalized policy optimization

4.1 Introduction

In statistics, a diversity of functions that are used for measuring the difference between one probability distribution and the other have been proposed, e.g., the Kullback-Leibler divergence, the f-divergence, the Wasserstein distance and more. Many of these divergences are well-studied in machine learning or deep learning area. For example, the maximum likelihood estimation is equivalent to minimizing KL divergence, generative adversarial networks (GANs) [17] and f-GANs [18] estimate sample distributions by approximate minimization of the Jensen-Shannon divergence and the f-divergence respectively.

Furthermore, distance measures for probability distributions play a key role in many reinforcement learning algorithms, including REPS[20], TRPO[10], ACKTR[11], where all methods use a KL divergence constraint between successive policies during policy updates to avoid greedy policy updates. The f-divergence constrained policy improvement method [37] is known to be the first method in RL that uses the f-divergence to constrain the policy improvement. This method yields a closed-form solution like REPS, however, with the limitations of high implementation costs and being computationally expensive, and thus, both this method [37] and REPS are not suitable for solving challenging problems with high-dimensional states.

In this chapter, we first employ the approximation idea that comes from TRPO to approximately solve a f-divergence constrained trust region policy optimization problem, where the f-divergence constraint is approximated using its second-order Taylor expansion, and a linear approximation to the objective. We demonstrate that the TRPO method is a special case of the f-divergence constrained policy optimization approach. We further focus on a one-parameter family of α-divergences, a special case of the f-divergence, to study effects of the choice of divergence on policy improvement. We show that all twice differentiable f-divergences have the same second-order Talyor expansion, for example, all α-divergences are locally the same. Thus, solving the twice differentiable f-divergence constrained policy optimization approximately with linear and quadratic approximations yield the same update rule as the original KL-divergence constrained TRPO method.

Additionally, we propose a novel objective that uses the f-divergence as a penalty term to the policy gradient, the method termed as f-divergence penalized policy optimization (f-PPO). This method is similar to the PPO with KL-divergence as penalty method [16] and is effective for optimizing large nonlinear policies, e.g. feedforward neural networks. We compare the experimental results of using different α-divergences as a penalty term to the objective to study the effects of α-divergence on policy improvement.

4.2 f-PPO

This section introduces the Csiszár f-divergence [39] as a general divergence measure between two probability distributions. Furthermore, we study a one-parameter family divergence, termed as α-divergence [40] [41], that is generated by the α-function f_α with $\alpha \in \mathbb{R}$. We demonstrate that a f-divergence constrained policy optimization problem, where the f-divergence should be twice differentiable, can be solved approximately using second-order Taylor expansion of the f-divergence and yields the same update solution as the TRPO method. Additionally, the f-divergence can be treated as a penalty term, and hence, a novel f-divergence penalized policy optimization (f-PPO) method is proposed.

4.2.1 f-divergence and α-divergence

Given two distributions P and Q that possess, respectively, an absolutely continuous density function p and q with respect to a base measure dx defined on the domain $\mathcal{X}$, we define the *f-divergence*,

$$\begin{aligned} D_f(P||Q) &= \int_{\mathcal{X}} q(x) f\left(\frac{p(x)}{q(x)}\right) dx \\ &= \sum_{x_i \in \mathcal{X}} q(x_i) f\left(\frac{p(x_i)}{q(x_i)}\right) \end{aligned}$$

where the *generator function* f is a convex, lower-semicontinuous function on $(0, +\infty)$, satisfying $f(1) = 0$. Many well-known divergences, such as KL-divergence, reverse KL-divergence, Pearson χ^2, are special cases of the f-divergence, correspond to a particular choice of the generator function f.

The α-*divergence* is a one-parameter family of f-divergences generated by the α-function $f_\alpha(x)$ with $\alpha \in \mathbb{R}$. The α-function is defined as

$$f_\alpha(u) = \begin{cases} u\log(u) - (u-1) & \text{for} \quad \alpha = 1 \\ -\log(u) + (u-1) & \text{for} \quad \alpha = 0 \\ \frac{(u^\alpha - 1) - \alpha(u-1)}{\alpha(\alpha-1)} & \text{otherwise} \end{cases} \tag{4.1}$$

Choosing different α yields a large number of divergences, including the well-known KL-divergence when $\alpha = 1$. The α-divergence smoothly connects the I-divergence $D_{\mathrm{KL}}(P||Q)$ with the reverse I-divergence $D_{\mathrm{RKL}}(Q||P)$ and pass through the Hellinger distance [41]. Table 4.1 summarizes the generator functions of several well-known divergences, and its divergence function. For example, the generator function of the KL-divergence is $f(u) = u\log(u) - (u-1)$, corresponding to $\alpha = 1$.

Divergence Name	α	generator $f(u)$	$D_f(P\|\|Q)$
Squared Hellinger	$1/2$	$(\sqrt{u}-1)^2$	$\int(\sqrt{p(x)} - \sqrt{q(x)})^2 dx$
KL	1	$u\log u$	$\int p(x)\log\frac{p(x)}{q(x)}dx$
Reverse KL	0	$-\log u$	$\int q(x)\log\frac{q(x)}{p(x)}dx$
Pearson χ^2	2	$(u-1)^2$	$\int\frac{(q(x)-p(x))^2}{p(x)}dx$
Neyman χ^2	-1	$\frac{(u-1)^2}{u}$	$\int\frac{(p(x)-q(x))^2}{q(x)}dx$
Total variation	N/A	$\frac{1}{2}\|u-1\|$	$\frac{1}{2}\int\|p(x)-q(x)\|dx$

Table 4.1.: Several common f-divergences with corresponding generator functions $f(u)$.

Additionally, Figure 4.1 visualizes several generator functions of the α-divergence. The α-function f_α is convex over $(0, +\infty)$ and satisfies $f(1) = 0$, which ensures that $D_f(P||Q) = 0$ only when $P = Q$. We highlight the Hellinger distance (when $\alpha = 1/2$) with a dash line in Figure 4.1, as for every α-divergence there is a reverse divergence symmetric with respect to the $\alpha = 1/2$. For example, KL-divergence and reverse KL-divergence are symmetric regarding to $\alpha = 1/2$.

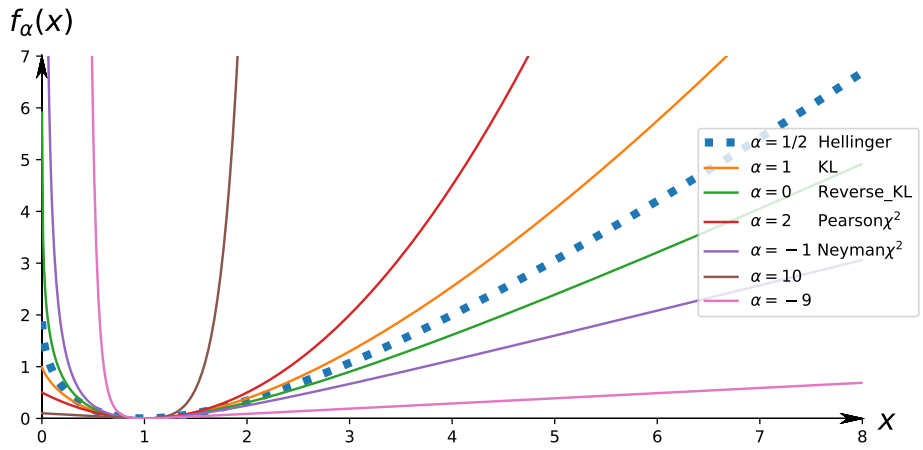

Figure 4.1.: Several α-functions for choosing different α. The x-axis represents the probability ratio of two different probability samples, e.g. $x = \frac{p}{q}$. The plots show that the α-function behaves differently for different α. For example, the function is more sensitive to large positive x when $\alpha \geq 0.5$, e.g. $\alpha = 10$, and is more sensitive to small positive x when $\alpha \leq 0.5$, e.g. $\alpha = -9$.

4.2.2 f-divergence constrained policy optimization

Consider the following constrained policy optimization problem

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & J(\pi_{\boldsymbol{\theta}}) = \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}(\mathbf{a}_t|\mathbf{s}_t) \right] \\ \text{s.t.:} \quad & \hat{\mathbb{E}} \left[D_f[\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)||\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)] \right] \leq \epsilon \end{aligned} \tag{4.2}$$

where $\hat{A}$ is the estimated advantage function, the $\epsilon \in \mathbb{R}$ is the maximum information loss measured by using averaged f-divergence, usually set to be a small number, e.g., $\epsilon = 0.01$, and $D_f[\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)||q(\cdot|\mathbf{s}_t)]$ is the averaged f-divergence between new policy distribution $\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t)$ and old policy distribution $q(\cdot|\mathbf{s}_t)$ in state $\mathbf{s}_t$, e.g., for discrete action space $\mathcal{A}$, the f-divergence is

$$D_f[\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)||\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)] = \sum_{\mathbf{a}\in\mathcal{A}} \pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}|\mathbf{s}_t) f\left(\frac{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}|\mathbf{s}_t)} \right).$$

Using the f-divergence as a constraint to the policy update has two advantages: first, it prevents policy update towards untrusted region, and secondly, it avoids greedy updates which would decrease exploration and lead to premature convergence, similar to the KL-divergence constraint.

Similar to TRPO, which has a similar constrained optimization problem in Equation (2.5), with the KL bound instead, our f-divergence constrained policy optimization problem can be also solved approximately by using a linear approximation to the objective $J(\boldsymbol{\theta})$ and a quadratic approximation to the f-divergence constraint $D_f[\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t)||q(\cdot|\mathbf{s}_t)]$,

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & J(\boldsymbol{\theta}) \approx \delta\boldsymbol{\theta}^T \nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}) \\ \text{s.t.:} \quad & \hat{\mathbb{E}} \left[D_f[\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)||\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)] \right] \approx \frac{1}{2} f''(1) \delta\boldsymbol{\theta}^T \mathbf{H}_{\boldsymbol{\theta}} \delta\boldsymbol{\theta} \leq \epsilon \end{aligned} \tag{4.3}$$

where $\delta\boldsymbol{\theta} = (\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k)$ is the update steps of policy parameters, the Hessian matrix $\mathbf{H}_{\boldsymbol{\theta}}$ is the coefficient of the quadratic term of the local Taylor expansion of the f-divergence, which can be calculated by taking the second-order partial derivatives of the f-divergence

$$\mathbf{H}_{ij} = \frac{\partial}{\partial\theta_i} \frac{\partial}{\partial\theta_j} \mathbb{E}_t \left[D_f[\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)||\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)] \right].$$

Here, the $f''(1)$ is the second-order derivative of the *generator function* $f(x)$ with respect to $x = 1$, which indicates that $f(x)$ is twice differentiable at point $x = 1$. The optimal policy parameter update is then found using conjugate gradient algorithm with line search in the direction $\boldsymbol{\theta}_{\text{new}} \propto {\mathbf{H}_{\boldsymbol{\theta}}}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

For all α-divergences, where the α-functions defined in Equation (4.1) have $f''_\alpha(1) = 1, \forall\alpha \in \mathbb{R}$, the local approximations of the divergences are all the same (see the Appendix B for the derivative). Thus, the TRPO method [10] is a special case of the f-divergence constrained policy optimization approach, both methods yield the same policy update result when optimizing the same objective, either the "surrogate" objective or the logarithm weighted objective, and using the quadratic approximation to the constraints. However, f-divergence makes the implementation more complex as there is no general closed-form expression for calculating the f-divergence between two distributions. For example, there is an expression for the KL-divergence between two univariate Gaussians, but not for the f-divergence. Thus, in reality, there is no need to solve the f-divergence, instead of the KL-divergence, constrained policy optimization using the approximation scheme that comes from TRPO. Also, it is impractical to solve the constrained problem as REPS, where Lagrangian multipliers are introduced, for large nonlinear function representations of the policy.

4.2.3 f-divergence penalized policy optimization

Similar to PPO that uses the KL-divergence as a penalty term to the "surrogate" objective in Equation (2.6), we propose to use a f-divergence as the penalty, and hence, following unconstrained policy optimization problem

$$\max_{\boldsymbol{\theta}} \quad \hat{J}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}(\mathbf{s}_t, \mathbf{a}_t) \right] - \beta \hat{\mathbb{E}}_t \left[\hat{D}_f(\pi_{\boldsymbol{\theta}_k}||\pi_{\boldsymbol{\theta}_{k+1}}) \right] \tag{4.4}$$

and β is a penalty coefficient. The estimation of the f-divergence $\hat{D}_f$ is calculated using samples, e.g. Monte Carlo estimation, and has relative high variance. Strictly speaking, the f-divergence approximation measures the difference between a subset of sampled old action probabilities $\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)$ and the corresponding new action probabilities $\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)$, unlike using a closed-form expression for measursing the difference between two policy divergences, $\pi_{\boldsymbol{\theta}_k}(\cdot|\mathbf{s}_t)$ and $\pi_{\boldsymbol{\theta}_{k+1}}(\cdot|\mathbf{s}_t)$. Nevertheless, such a "surrogate" f-divergence forms a lower bound on the performance of the policy π.

The estimated *advantage function* $\hat{A}(\mathbf{s}_t, \mathbf{a}_t)$ in Equation 4.4 is similar to the action-value function $\hat{Q}(\mathbf{s}_t, \mathbf{a}_t)$, both functions are used to measure the performance of state-action pairs $(\mathbf{s}_t, \mathbf{a}_t)$. However, the advantage function has lower variance, as it required subtracting a baseline from the action-value function,

$$\hat{A}(\mathbf{s}, \mathbf{a}) = \hat{Q}(\mathbf{s}, \mathbf{a}) - \hat{V}(\mathbf{s}) \tag{4.5}$$

where state-value function $\hat{V}(\mathbf{s})$ is the baseline and can be estimated using function approximation methods, e.g., nonlinear function approximation, which is usually treated as supervised learning problem and can be solved efficiently using stochastic gradient descent methods by minimizing a mean squared error (MSE) loss function. The MSE loss function for the state-value function is defined as

$$\min_{\boldsymbol{\phi}} \quad L(V_{\boldsymbol{\phi}}) = \mathbb{E}_t \left[\| \hat{V}_{\boldsymbol{\phi}}(\mathbf{s}_t) - \sum_{l=0}^{\infty} \gamma r_{t+l} \|^2 \right] \tag{4.6}$$

where $\sum_{l=0}^{\infty} \gamma^l r_{l+t}$ is the discounted cumulative reward from state $\mathbf{s}_t$ over all time steps in batch of trajectories and serves as the ground truth state value in state $\mathbf{s}_t$, and $\hat{V}_{\boldsymbol{\phi}}(\mathbf{s}_t)$ is the predicted state value in state $\mathbf{s}_t$. Parameter vector $\boldsymbol{\phi}$ of the value function approximator, e.g., weights of the neural network, are updated using sampled trajectories $(\mathbf{a_0}, \mathbf{s_0}, r_0, \cdots)$ and mini-batch SGD.

Therefore, we present our f-divergence penalized policy optimization (f-PPO) algortihm [2].

Algorithm 2 f-divergence penalized policy optimization (f-PPO)

Initialize policy parameters $\boldsymbol{\theta}_0$, penalty coefficient β_0, divergence function f
for Iteration updates: $k = 0, 1, 2, \cdots$ **do**
  Run current policy $\pi_{\boldsymbol{\theta}_k}$ and collect a set of trajectories $\mathcal{D} = (\mathbf{a}_0, \mathbf{s}_0, r_0, \mathbf{a}_1, \mathbf{s}_1, r_1, \cdots)$ of length T
  Estimate the advantages $\hat{A}_t$, e.g., using *generalized advantage estimation* (GAE) algorithm [25]
  for Epoch updates $n = 0, 1, 2, \cdots, N$ **do**
    for Batch updates $m = 0, 1, 2, \cdots, T/M$ **do**
      Update policy parameters using M steps of minibatch SGD for n epoch (e.g., Adam)

$$\boldsymbol{\theta}_{\text{new}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}_{k+1}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}(\mathbf{s}_t, \mathbf{a}_t) \right] - \beta \hat{\mathbb{E}}_t \left[\hat{D}_f(\pi_{\boldsymbol{\theta}_k} || \pi_{\boldsymbol{\theta}_{k+1}}) \right]$$

      Update value function parameters using M steps of minibatch SGD for n epoch (e.g., Adam)

$$\boldsymbol{\phi}_{\text{new}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \mathbb{E}_t \left[\| \hat{V}_{\boldsymbol{\phi}}(\mathbf{s}_t) - \sum_{l=0}^{\infty} \gamma r_{t+l} \|^2 \right]$$

    end for
  end for
end for

4.3 Connections with prior work

f-Divergence constrained policy improvement [37]: Our approach was highly inspired by this constrained policy optimization method that uses a general class of f-divergences in place of the KL-divergence for restricting policy updates. Similar to REPS, where the KL-divergence is used as a constraint, the f-divergence constrained policy improvement method also solves Lagrangian primal-dual functions and yield a closed-form solution for the policy update, in which, however, the optimization is computationally expensive and not straightforward to implement. However, the f-PPO

method uses the approximated f-divergence as a penalty term and solves the problem as an unconstrained policy optimization problem with stochastic gradient ascent, which makes f-PPO easier to implement and desired for solving challenging problems such as deep RL problems with high-dimensional state or action spaces.

TRPO [10]: TRPO is well-known to be a constrained policy optimization problem that is solved approximately and guarantee to give monotonic improvement by optimizing the "surrogate" objective. Combining the approximation with conjugate gradient and line search methods do not yield a closed-form solution, however, is effective for optimizing large nonlinear policies, e.g. neural networks. We show that TRPO is a special case of the f-divergence constrained policy optimization method. More importantly, when choosing a twice differentiable generator function f for the f-divergence and using the approximation scheme like TRPO, the f-divergence constrained policy optimization has the same solution as TRPO.

PPO with adaptive KL-divergence penalty [16]: This method reformulates the constrained objective in TRPO as an unconstrained objective by treating the KL-divergence constraint as a penalty term (or soft constraint) multiplied by an adaptive penalty coefficient. Both f-PPO and this method are optimized using SGD optimizer, e.g., Adam, without second-order gradient information. Differing from our method in that the f-divergence is used as the penalty and approximated using samples when dealing with continuous policy distributions, the KL-divergence for continuous probability distributions, e.g. Gaussian distribution, can be computed using closed-form expression.

PPO with clipped "surrogate" objective [16]: TRPO and f-PPO discourage greedy policy updates explicitly by conducting a particular choice of f-divergence, e.g. KL-divergence, between the updated policy and its prior as a constraint or a penalty to the objective. The PPO method with a clipped "surrogate" objective, on the other hand, achieves policy updates inside some trust regions by clipping the policy ratio within a trusted range, where the policy ratio measures how an updated action probability diverges from the previous one. The f-PPO method provides a connection between the PPO or TRPO method and the method of using f-divergence in RL as a divergence measurement of policy distributions.

4.4 Experiments

We ran f-PPO on a collection of standard simulated robotic locomotion benchmark tasks from the OpenAI MuJoCo task suite. Since the f-divergence is notably for generalizing many common divergences, especially one of its special case, the α-divergence that is further used for representing many well-known divergences, including the KL-divergence, the Squared Hellinger distance, and more, we focus on the α-divergence to study the influences of choosing different divergence functions on policy optimization. Furthermore, for every divergence in the α-divergence, there is a reverse divergence symmetric with respect to the point $\alpha = 0.5$ (corresponding to the Hellinger distance) [37], such as the KL-divergence and its reverse, the reverse KL-divergence. Thus, we compare the experimental results from a set of α-divergence pairs to answer the following two questions:

1. Does the f-divergence penalized policy optimization method of any chosen f-divergence learns an optimal policy for solving challenging environments at all?

2. How does a particular α-divergence differs from its reverse divergence on policy improvement? What about the Hellinger distance that lies in the middle point?

4.4.1 Policy and value function

We use multilayer perceptrons (MLPs) to represent both the policy network and the value function, both networks have 2 hidden layers of size 64 each, an input layer of size that equals to the dimension of the state space, and an output layer of size that has the same dimension as the action space and one respectively. More network details can be found in the previous chapter (Chapter 3).

4.4.2 MuJoCo environment

We tested our f-PPO method on 6 MuJoCo environments, including *InvertedPendulum-v2*, *InvertedDoublePendulum-v2*, *Reacher-v2*, *Hopper-v2*, *HalfCheetah-v2*, and *Walker2d-v2*, all of which have different state and action space, and different goals. Environments' details and reward functions can be found in the previous chapter (Chapter 3).

4.4.3 Empirical results

Figure 4.2 shows a series of comparisons of the learning curve on 6 different MuJoCo environments for various choices of the divergence type, where tasks range from the easiest InvertedPendulum-v2 environment to the high-dimensional HalfCheeta-v2 or Walker2d-v2 environment. In summary, all tasks can be effectively solved using the f-PPO method with a proper choice of the f-divergence function. Let us focus on the first column, where we observe that on almost all environments, the small negative value ($\alpha = -4$) tends to converge faster and has lower variance in the end, compared to the performance of the large positive value ($\alpha = 5$), in which the performance converges slower and has high variance. On the second column, where the selected values α are closer to $\alpha = 0.5$, the negative value ($\alpha = -1$) again outperforms the positive value ($\alpha = 2$) in most tasks. In general, the moderate values $\alpha \in \{0, 0.5, 1\}$ on the third column achieve the most stable and promising results.

Figure 4.3 presents comparisons of the policy entropy corresponding to the learning curve shown in Figure 4.2. The choice of the divergence function significantly affects the entropy performance regardless of tasks. For example, on all environments, large positive values $\alpha \in \{5, 2\}$ prevents the policy entropy from rapidly decreasing, or even occasionally increase the entropy on some environments, however, small negative values $\alpha \in \{-4, -1\}$ on all occasions cause a greedy update of the policy. For example, on the Walker2d-v2 environment, the α-divergence of $\alpha = 5$ causes the entropy to increase from around 8 to greater than 11, on the contrary, the divergence of a negative value $\alpha = -4$ results in a declined entropy of less than 6.

In summary, according to the empirical results on MuJoCo environments, α-divergences corresponding to negative values $\alpha < 0$ are more effective in policy improvement comparing to large positive values $\alpha > 1$. The alpha values in the range $\alpha \in [-1, 1]$ result in most stable and reliable policy improvement. For all tested divergences, the Neyman χ^2 ($\alpha = -1$) divergence outperforms all other values α on almost all environments.

4.5 Conclusion

We considered the Csiszár f-divergence [39] and its special case the α-divergence [40] [41], both of which generalize many well-known divergences, such as the KL-divergence and reverse KL-divergence. We proposed to use the f-divergence for regularizing the objective of the policy gradient method, in which the f-divergence can be either treated as a hard constraint or a penalty term to the policy gradient. First, we have shown that an f-divergence constrained policy optimization problem can be solved approximately, like the TRPO method, using a linear and quadratic approximation to the "surrogate" objective and a twice differentiable f-divergence respectively. The approximated solution for updating policy parameters is the same as the TRPO method, as all twice differentiable f-divergences have the same second-order Taylor expansion. Thus, instead of solving a constrained policy optimization problem, we proposed a novel objective, where the f-divergence is treated as a penalty term, that is optimized using a stochastic gradient ascent method. We term this unconstrained policy optimization as the f-divergence penalized policy optimization (f-PPO) method. Furthermore, we studied the effects of f-divergences, precisely α-divergences, on policy improvement via a series of experiments on MuJoCo environments. Our experiments suggest that the f-PPO method is effective for solving high-dimensional continuous control tasks by using a non-linear function approximation to the policy and the SGD for updating policy parameters. Additionally, we demonstrated that the choice of divergence type trades off between exploration and exploitation during policy update, and hence, affects the policy performance in the end. For example, the α-divergence of a negative value $\alpha < 0$ exploits greedier than its reverse divergence of a positive value $\alpha > 1$ (symmetric with respect to the point $\alpha = 0.5$), and results in better performance on tested environments in general. Moreover, moderate values $\alpha \in [-1, 1]$ achieve more stable and valid policy optimization, compared to extreme values, e.g., $\alpha = 5$.

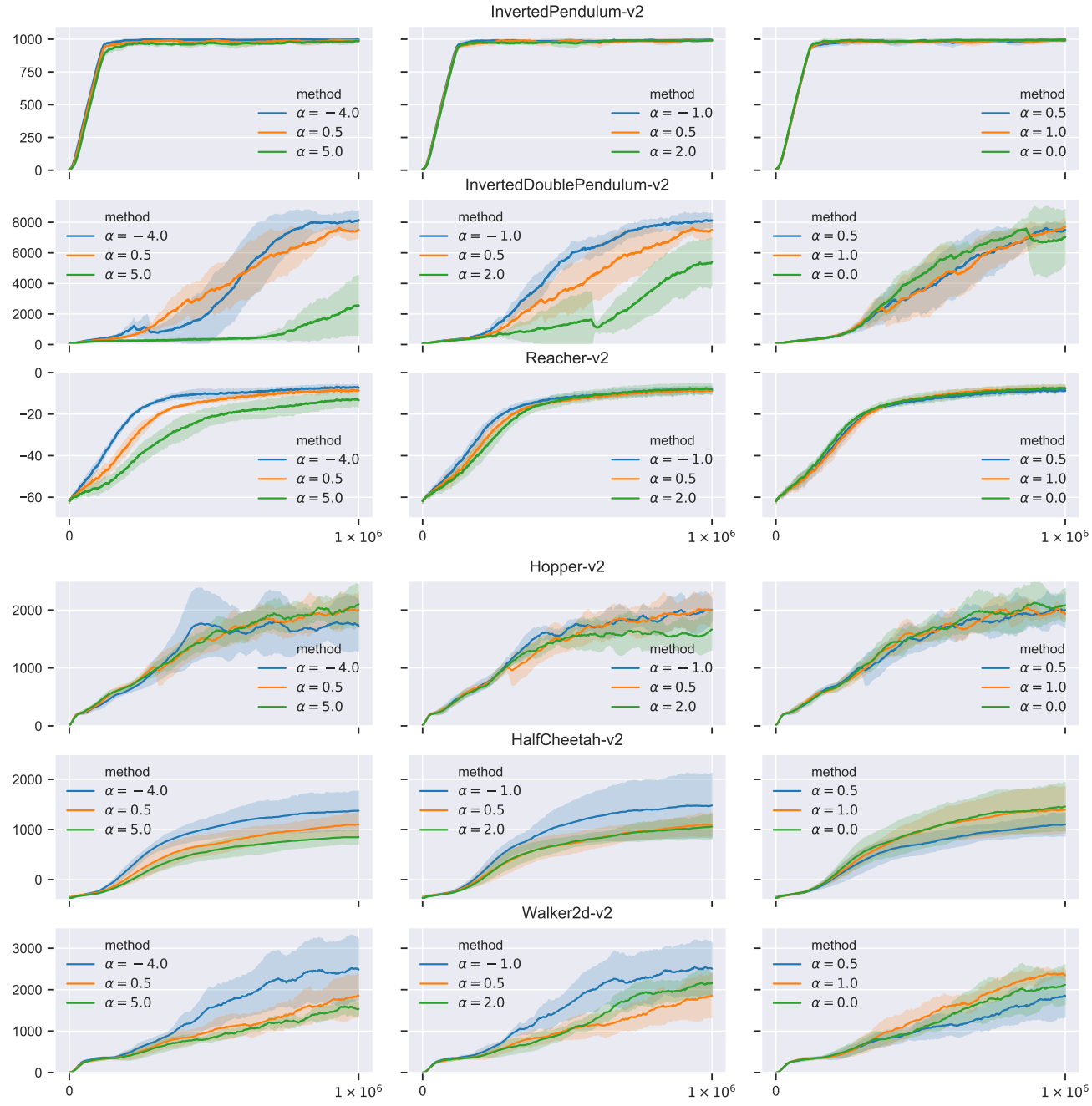

Figure 4.2.: Results of the average reward from f-PPO on 6 MuJoCo environments trained for 1×10^6 timesteps. The shaded region indicates the standard deviation over 6 random seeds. Here each row corresponds to a specific continuous control task. The difficulty of the tasks (according to the dimension of the state and action space) increases from top to bottom across the column. For each particular task, results of chosen α-divergences corresponding to different α values are divided into three subplots, from the more extreme α values on the left to the more refined values on the right. For each particular subplot, a pair of symmetric α-divergences, with respect to the point $\alpha = 0.5$, is compared to the center α-divergence ($\alpha = 0.5$, corresponding to the Hellinger distance).

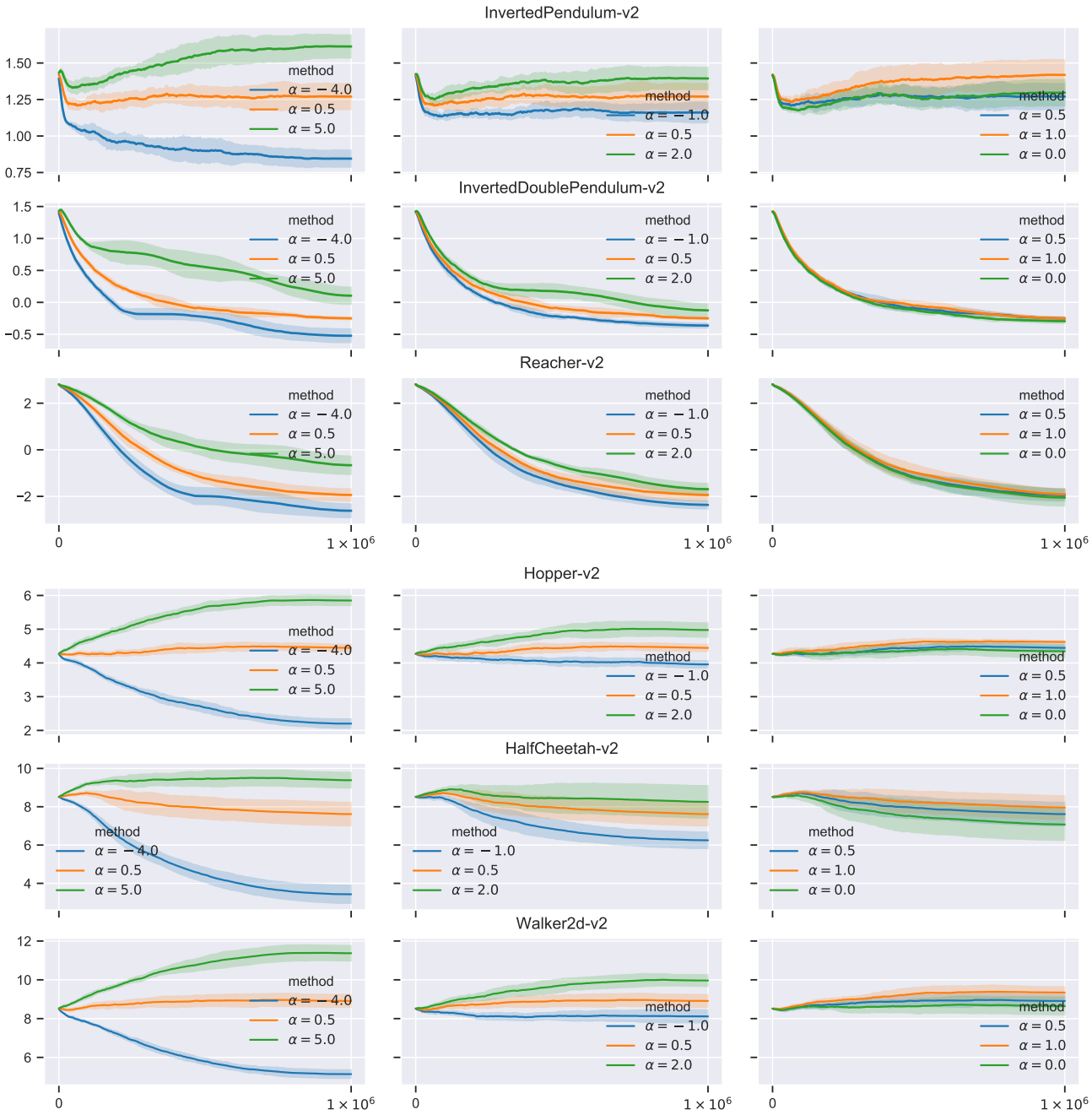

Figure 4.3.: Results of the policy entropy corresponding to the experiments above, where the x-axis shows the sampled time steps and the y-axis indicates the policy entropy. Overall, negative values $\alpha < 0$ (blue) tend to decrease the entropy faster than positive values $\alpha > 0$ (green)—in other words, large positive α's focus more on exploration in contrast to small negative α's being more on exploitation.

5 Conclusion and discussion

In this thesis, we first shown a brief summary of recent breakthroughs in deep reinforcement learning, followed by a introduction of basic concepts in reinforcement learning and deep learning. We reviewed several state-of-the-art deep RL algorithms and highlighted a number of computational problems that they present. Namely, REPS is computationally expensive, TRPO is intractable for large RNNs, and PPO is only able to optimize the "surrogate" objective.

One of our main contributions is to propose a practical policy gradient method, termed minimax entropic policy optimization (MMPO), which has several advantages compared to other deep RL methods. First, it is computationally efficient as we only need first-order gradient information. Second, it is straightforward to implement, as it does not require complicated parameter manipulations to make it work. Third, it is sample efficient because we can optimize the parameters with multiple epochs of stochastic gradient optimization using the same trajectory, which is possible due to enforcing the KL constraint explicitly and not as a by-product of clipping or approximate quadratic. As a result, MMPO shows comparable performance on several MuJoCo continuous control tasks. However, comparisons on MuJoCo environments among MMPO, PPO and TRPO methods do not show pronounced differences. Moreover, PPO has achieved great successes on not only MuJoCo environments, but also on high-dimensional continuous control problems involving 3D humanoid and the Atari domain where states are represented using images, and TRPO demonstrated strong theoretical foundations. Thus, it is difficult to conclude that MMPO is an algorithm that has better or equivalent optimization ability as PPO or TRPO methods. Applying MMPO to domains with high-dimensional state inputs, e.g., robot simulation in a 3D domain, playing video games, or real robots, and then, comparing the results to the other two methods should be done in future. In addition, the empirical results on MuJoCo environments show that MMPO decreases the entropy of the policy quite slowly and keeps exploring the environment continuously. Such properties are expected to give MMPO an advantage in partially observable Markov decision problems (POMDPs), in which agents generally have to keep exploring for a long period of time in order to prevent premature convergence.

Apart from the MMPO method, we provided new insights in achieving stable policy updates via a more general class of f-divergences. The proposed method is called f-divergence penalized policy optimization (f-PPO) and it has shown promising results in optimizing continuous control MuJoCo tasks, such as the Walker2d-v2 environment. The empirical results on MuJoCo environments demonstrated that the choice of f-divergences have a great effect on the policy entropy, and hence, on the policy improvement. For example, as shown in Figure 4.3 on the Hopper-v2 environment, the α-divergence with a value $\alpha < 0$ leads the entropy to rapid decreasing in contrast to an increasing entropy performance caused by the α-divergence of large positive value $\alpha > 1$. In the future, it would be interesting to carry out more theoretical research and investigate the reason behind such a phenomenon. Thus, it is likely that by just choosing another f-divergence, f-PPO will still yield more new policy optimization methods.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[2] Q. V. Le, "Building high-level features using large scale unsupervised learning," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8595–8598, IEEE, 2013.

[3] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708, 2014.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[6] OpenAI, "OpenAI Five," 2018.

[7] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. Garcia Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel, "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning," *ArXiv e-prints*, July 2018.

[8] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

[9] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[10] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, pp. 1889–1897, 2015.

[11] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Advances in neural information processing systems*, pp. 5285–5294, 2017.

[12] N. N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural computation*, vol. 14, no. 7, pp. 1723–1738, 2002.

[13] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *International conference on machine learning*, pp. 2408–2417, 2015.

[14] L. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, 2001.

[15] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[18] S. Nowozin, B. Cseke, and R. Tomioka, "f-gan: Training generative neural samplers using variational divergence minimization," in *Advances in Neural Information Processing Systems*, pp. 271–279, 2016.

[19] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[20] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search.," in *AAAI*, pp. 1607–1612, Atlanta, 2010.

[21] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," in *Reinforcement Learning*, pp. 5–32, Springer, 1992.

[22] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[23] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[24] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," in *Reinforcement Learning*, pp. 5–32, Springer, 1992.

[25] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[26] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade, "Towards generalization and simplicity in continuous control," in *Advances in Neural Information Processing Systems*, pp. 6553–6564, 2017.

[27] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, pp. 4565–4573, 2016.

[28] S. M. Kakade, "A natural policy gradient," in *Advances in neural information processing systems*, pp. 1531–1538, 2002.

[29] B. Arouna, "Adaptative monte carlo method, a variance reduction technique," *Monte Carlo Methods and Applications mcma*, vol. 10, no. 1, pp. 1–24, 2004.

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

[31] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *CoRR*, vol. abs/1709.06009, 2017.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[33] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[34] R. Y. Rubinstein, *Simulation and the Monte Carlo Method*. New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 1981.

[35] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[36] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[37] B. Belousov and J. Peters, "f-divergence constrained policy improvement," *arXiv preprint arXiv:1801.00056*, 2017.

[38] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Openai baselines." `https://github.com/openai/baselines`, 2017.

[39] I. Csiszár, "Eine informationstheoretische ungleichung und ihre anwendung auf beweis der ergodizitaet von markoffschen ketten," *Magyer Tud. Akad. Mat. Kutato Int. Koezl.*, vol. 8, pp. 85–108, 1964.

[40] H. Chernoff *et al.*, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *The Annals of Mathematical Statistics*, vol. 23, no. 4, pp. 493–507, 1952.

[41] A. Cichocki and S.-i. Amari, "Families of alpha-beta-and gamma-divergences: Flexible and robust measures of similarities," *Entropy*, vol. 12, no. 6, pp. 1532–1568, 2010.

[42] A. Kupcsik, M. Deisenroth, J. Peters, L. Ai Poh, V. Vadakkepat, and G. Neumann, "Model-based contextual policy search for data-efficient generalization of robot skills," conditionally accepted.

[43] K. Muelling, A. Boularias, B. Mohler, B. Schoelkopf, and J. Peters, "Learning strategies in table tennis using inverse reinforcement learning," accepted.

[44] C. Dann, G. Neumann, and J. Peters, "Policy evaluation with temporal differences: A survey and comparison," no. March, pp. 809–883, 2014.

[45] T. Meyer, J. Peters, T. Zander, B. Schoelkopf, and M. Grosse-Wentrup, "Predicting motor learning performance from electroencephalographic data," no. 1, 2014.

[46] B. Bocsi, L. Csato, and J. Peters, "Indirect robot model learning for tracking control," 2014.

[47] H. Ben Amor, A. Saxena, N. Hudson, and J. Peters, "Special issue on autonomous grasping and manipulation," 2014.

[48] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[49] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, pp. 849–856, 2009.

[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[51] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[52] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[53] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[54] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[55] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, Oct 2012.

[56] D. Pfau and O. Vinyals, "Connecting generative adversarial networks and actor-critic methods," *arXiv preprint arXiv:1610.01945*, 2016.

A Hyperparameters

In all experiments of both the MMPO (in Chapter 3) method and the f-PPO (in Chapter 4) method, the neural network structures of both the policy function and the value function are the same. Hyperparameters of running both methods on MuJoCo environments are list in Table A.1 and Table A.2 separately.

Table A.1.: Hyperparameters of running MMPO method on MuJoCo environments

Parameter	Value
Sampled timesteps per iteration (T)	2048
Adam learning rate	3×10^{-4}
Number of epochs (M) per iteration	10
Minibatch size (K)	64
Discount (γ)	0.99
GAE [25] parameter (λ)	0.95
Total sampled timesteps	1×10^{6}

Table A.2.: Hyperparameters of running f-PPO method on MuJoCo environments

Parameter	Value
Sampled timesteps per iteration (T)	2048
Adam learning rate	3×10^{-4}
Number of epochs (M) per iteration	10
Minibatch size (K)	128
Discount (γ)	0.99
GAE [25] parameter (λ)	0.95
Total sampled timesteps	1×10^{6}
Coefficient of the penalty term (β)	2.0
Tested f-divergence, or α-divergence (α)	[-4, 5, -1, 2, 0, 1, 0.5]

B All twice differentiable f-divergences are locally the same

In Chapter 4, we have discussed that a f-divergence constrained policy optimization problem, where the f-divergence constraint can be approximated using its second-order Taylor polynomial, yields a same update rule for the policy parameter vector as the TRPO solution when solving the constrained optimization problem approximately. However, not all f-divergences are twice differentiable, e.g. the *total variation distance* whose generator function is $f(x) = \frac{1}{2}|x-1|$. The α-divergence, which can be generalized using a α function f_α, is a special case of the f-divergence and is twice differentiable. Here, we show a short derivation of the second-order Taylor expansion of the α-divergence and demonstrate that its has the same form as the Fisher information metric.

$$
\begin{aligned}
D_f(q + dq||q) &= \sum_i q_i f(1 + \frac{dq_i}{q_i}) \\
&\approx \sum_i q_i \left[f(1) + f'(1)\frac{dq_i}{q_i} + \frac{1}{2} f''(1) \left(\frac{dq_i}{q_i} \right)^2 \right] \\
&= \sum_i \left[0 + 0\frac{dq_i}{q_i} + \frac{1}{2} f''(1) \left(\frac{dq_i}{q_i} \right)^2 \right] \\
&= \frac{1}{2} f''(1) \sum_i dq_i \frac{1}{q_i} dq_i \\
&= \frac{1}{2} \sum_i dq_i \frac{1}{q_i} dq_i \\
\text{where} \quad f''_\alpha(x) = x^{\alpha-2} &\Rightarrow f''(1) = 1
\end{aligned}
$$