Baconian : a unified model-based reinforcement learning library

Dong, Linsen

2021

Dong, L. (2021). Baconian : a unified model-based reinforcement learning library. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/146557>

<https://doi.org/10.32657/10356/146557>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 22:43:29 SGT

Baconian: A Unified Model-Based Reinforcement Learning Library

Dong Linsen

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Master of Engineering (M.Eng)

February 6, 2021

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

2020/08/21

.....
Date

.....
DONG LINSEN

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

...2020-08-21....
Date

.....
Prof. Wen Yonggang

Authorship Attribution Statement

This thesis contains material from 4 paper published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as:

L. Dong, G. Gao, X. Zhang, L. Chen, and Y. Wen, "Baconian: A unified opensource framework for model-based reinforcement learning," CoRR, vol. abs/1904.10762, 2019. [Online]. Available: <http://arxiv.org/abs/1904.10762>

The contributions of the co-authors are as follows:

- Prof. Wen Yonggang provided the initial project direction and edited the manuscript drafts.
- Dr. Gao Guanyu assisted the manuscript writing and polish.
- Chen Liangyu and Zhang Xinyi assisted with the experiments of the system, and implementation of the system
- I mainly designed and implemented the system introduced in the paper and prepared the manuscript.

Chapter 4 is published as:

Zhang, Huaizheng, et al. "DeepQoE: A Multimodal Learning Framework for Video Quality of Experience (QoE) Prediction." IEEE Transactions on Multimedia (2020).

Dong, Linsen, Yuanlong Li, Xin Zhou, Yonggang Wen, and Kyle Guan. "Intelligent Trainer for Dyna-Style Model-Based Deep Reinforcement Learning." IEEE Transactions on Neural Networks and Learning Systems (2020).

Gao, Guanyu, Linsen Dong, Huaizheng Zhang, Yonggang Wen, and Wenjun Zeng. "Content-Aware Personalised Rate Adaptation for Adaptive Streaming via Deep

Video Analysis." In ICC 2019-2019 IEEE International Conference on Communications (ICC), pp. 1-8. IEEE, 2019.

The contributions of the co-authors are as follows:

- Prof. Wen Yonggang provided the initial project direction and edited the manuscript drafts.
- Zhang Huaizheng provided the idea, wrote major parts of the manuscript, and also provided the parts of the experiment results.
- Dr. Gao Guanyu, Dr. Hu Han, Dr. Guan, and Dr. Zeng Wenjun assisted the manuscript writing and polish.
- I mainly implemented code and conducted parts of the experiments introduced in the paper and prepared parts of the manuscript.

2021/01/14

.....
Date

.....
DONG LINSEN

Abstract

Reinforcement Learning (RL) has become a trending research topic with great success in outperforming humans on many tasks including video games, board games, and robotics control. By leveraging Deep Learning (DL), RL algorithms can consume a large volume of data without any prior knowledge of the system dynamics. However, requiring a large amount of data also limits the applicability in many fields where data is costly to obtain. Model-based Reinforcement Learning (MBRL) is regarded as a promising way to achieve high data efficiency while maintaining comparable performance. MBRL equips a dynamic transition model to facilitate and speed up the policy searching by learning the system dynamics. But there are no satisfying open-sourced libraries for the RL community to conduct MBRL research. Therefore, to fill the gap, we propose an open-sourced, flexible, and user-friendly MBRL library, Baconian, to facilitate the research on MBRL.

In this thesis, we illustrate the library from the aspects of design principle, implementations, and the programming guide. Various benchmark results are also given. To reach high flexibility, modularized design is applied by separating the library into three components: Experiment Manager, Training Engine, and Monitor. For implementations, we provide commonly used functionalities including parameter management, TensorFlow integration etc. Moreover, we utilize Baconian to conduct RL experiments in real research topics at the case study section. First, we utilize Baconian as the framework to tune the Dyna-style MBRL hyper-parameters in an online fashion. Our proposed method reaches a similar or better performance out of all five tasks compared to three baseline methods. Second, we use Baconian to apply RL algorithms for online video bitrate selection optimization where our method outperforms the best baseline method on average bitrate metric by 7.8%.

Acknowledgments

I would like to express my sincere thanks and appreciation to my supervisor, Professor Wen Yonggang, for his guidance and patience. He has taught me how to conduct research in a logical and systematic way. I would also like to thank Dr. Li Yuanlong, Dr. Gao Guanyu, Mr. Zhang Huaizheng and Dr. Luo Yong for their efforts and guidance when we collaborated on different research problems.

Last but not least, I would like to express my appreciation to my girlfriend and my family. They support me in this journey and bring me a joyful life.

Contents

Abstract	v
Acknowledgments	vi
List of Figures	x
List of Tables	xii
List of Abbreviations	xiii
List of Publications	xiv
1 Introduction	1
1.1 Background	1
1.2 Motivation and Objectives	2
1.3 Organization of the Report	3
2 Literature Review	4
2.1 Introduction	4
2.2 Model-based Reinforcement Learning	4
2.2.1 Markov Decision Process	4
2.2.2 Policy and Value Functions	6
2.2.3 Optimality and Bellman Equation	6
2.2.4 Dynamics Model	7
2.2.5 Algorithms	8
2.2.5.1 Dyna-style Algorithms	8
2.2.5.2 Shooting Algorithms	9
2.2.5.3 Policy Search with Back-propagation through Time	9
2.3 Reinforcement Learning Library	10
2.3.1 Environment and Task Library	11

2.3.2	Algorithm Library	12
2.3.3	Distributed Training Library	13
3	Baconian: A Unified Model-based Reinforcement Learning Library	14
3.1	Features	14
3.1.1	RL Algorithms	14
3.1.2	Supported Test Environments	15
3.1.3	Experiment Utilities	15
3.1.4	Open-source Support	15
3.1.5	Fine-grained Logging and Analysis	16
3.2	System Design and Implementation	16
3.2.1	System Design	16
3.2.1.1	Experiment Manager	16
3.2.1.2	Training Engine	17
3.2.1.3	Monitor	17
3.2.2	Implementation	18
3.2.3	Control Flow Module	18
3.2.4	Stateful Modeling	18
3.2.5	Parameter Management	19
3.2.6	TensorFlow Integration	19
3.3	Usage	19
3.3.1	Building RL Experiments	20
3.3.2	Programming with Baconian	21
3.3.2.1	New Tasks/Algorithms/Dynamic Models.	21
3.3.2.2	New Training Workflow.	21
3.3.2.3	Customize Logging.	22
3.4	Benchmarking Baconian	22
3.4.1	Benchmark Tasks	22
3.4.2	Experiments Setup	23
3.4.3	Benchmarking Results	23
3.5	Conclusion	24

4	Case Study with Baconian	25
4.1	Intelligent Trainer for Dyna-Style Model-Based RL	25
4.1.1	Intelligent Trainer Algorithms	25
4.1.2	Implementations with Baconian	27
4.1.2.1	Training Process Environment	27
4.1.2.2	Customized Training Flow for Ensemble Trainer	28
4.1.3	Numerical Evaluations	28
4.2	Content-of-Interest based Adaptive Bitrate for Online Video Streaming via DRL	32
4.2.1	System Design and Workflow	32
4.2.2	DQN based Rate Adaptation with User Interests Awareness . . .	34
4.2.3	Numerical Evaluations	35
4.2.3.1	Experiment Settings	36
4.2.3.2	Performances on Bitrate Variations, Average Bitrate, and Rebuffering Time,	37
4.2.4	Implementations with Baconian	38
4.2.4.1	Video Streaming Simulator	38
4.2.4.2	Built-in Algorithms of Baconian	39
4.2.4.3	Customizing Logging for Result Analysis	39
4.3	Conclusion	39
5	Conclusion and Future Work	40
5.1	Conclusion	40
5.2	Future Work	40
	References	42

List of Figures

2.1	MDP Process	5
2.2	RL open-sourced frameworks	11
3.1	Feature list of Baconian.	15
3.2	The system design of Baconian. The system is divided into three main modularized components to minimize the coupling for flexibility and maintainability.	17
3.3	Simplified UML diagram of Baconian	18
3.4	The procedure to create an MBRL experiment in Baconian which can be referred to as a simplified sequential UML diagram. Each module is replaceable and configurable to reduce the effort of building from scratch.	20
3.5	The pendulum control tasks.	23
4.1	Illustration of “Reinforcement on Reinforcement” (RoR) framework. The inner box encapsulates a standard Dyna into a training processing environment (TPE) as the inner layer. In the outer layer we introduce an intelligent trainer, controlling the optimization of the Dyna training flow configuration in the TPE.	26
4.2	Accumulative rewards of different uni-head trainer designs for different tasks in (a)-(e). The curves show the average accumulative reward while the shaded region shows the standard deviation of the reward in 10 independent runs. The proposed uni-head trainer shows its adaptability (better than the Fixed trainer) but may fail in certain cases like Swimmer.	30
4.3	Accumulative rewards of ensemble trainer for different tasks in (a)-(e). The proposed ensemble design shows close-to-optimal or even better performance on all cases.	31

4.4	The design of the ABR Streaming method with COI based rate adaptation	33
4.5	The DQN network for interest-aware rate adaptation.	35
4.6	These results show the CDF, i.e., Empirical Cumulative Distribution Function for three major evaluation metrics. The results show that CoI tends to allocate higher bitrate for each video chunk, maintains a relatively low rebuffering time even under higher bitrate selection comparing to other methods, and control the bitrate variation to a comparative level with other methods.	38

List of Tables

3.1	Preliminary benchmark of continuous task: Pendulum-v0. DDPG(baselines) result is obtained by open-source library baselines with same hyper-parameters for comparison.	23
4.1	The average performances per session.	37

List of Abbreviations

RL	Reinforcement Learning
MBRL	Model-based Reinforcement Learning
DQN	Deep Q-Networks
DDPG	Deep Deterministic Policy Gradient
MDP	Markov Decision Process
DNN	Deep Neural Network
DL	Deep Learning
LR	Linear Regression
LWLR	Locally Weighted Linear Regression
TRPO	Trust Region Policy Optimization
SLBO	Stochastic Lower Bounds Optimization
MPC	Model Predictive Control
RS	Random Shooting
POMDP	Partially Observable Markov Decision Process
MB-MF	Model-based Initialization of Model-free Reinforcement Learning
LQR	Linear Quadratic Regulator
iLQR	iterative Linear Quadratic Regulator
GPS	Guided Policy Search
PILCO	Probabilistic Inference for Learning Control
RTS	Real-Time Strategy
PPO	Proximal Policy Optimization
TPE	Training Process Environment
RoR	Reinforcement on Reinforcement
ABR	Adaptive Bitrate
CoI	Content of Interest
MPD	Media Presentation Description
BBA	Buffer-Based approach
RBA	Rate-Based approach
CDF	Cumulative Distribution Function

List of Publications

- [1] G. Gao, L. Dong, H. Zhang, Y. Wen, and W. Zeng, *Content-aware personalised rate adaptation for adaptive streaming via deep video analysis*, 2019. DOI: 10.1109/ICC.2019.8761156. [Online]. Available: <https://doi.org/10.1109/ICC.2019.8761156>.
- [2] L. Dong, Y. Li, X. Zhou, Y. Wen, and K. Guan, *Intelligent trainer for dyna-style model-based deep reinforcement learning*, 2020. DOI: 10.1109/TNNLS.2020.3008249.
- [3] Q. Fu, L. Dong, Z. Liu, Y. Luo, Y. Wen, Y. Li, and L.-Y. Duan, *Incorporating category taxonomy in deep reinforcement learning based image hashing*, IEEE, 2019.
- [4] H. Zhang, L. Dong, G. Gao, H. Hu, Y. Wen, and K. Guan, *Deepqoe: A multimodal learning framework for video quality of experience (qoe) prediction*, 2020.
- [5] L. Dong, G. Gao, X. Zhang, L. Chen, and Y. Wen, *Baconian: A unified opensource framework for model-based reinforcement learning*, 2019. arXiv: 1904.10762. [Online]. Available: <http://arxiv.org/abs/1904.10762>.

Chapter 1

Introduction

1.1 Background

Reinforcement Learning (RL) has become an emerging technique in recent years. It demonstrates great ability in many real-world tasks including video games, go, robotics control, and autonomous driving. Most of the current RL methods that are successfully applied to real-world tasks are built upon the model-free methods and Deep Learning (DL) models. Such algorithms suffer from data inefficiency greatly especially when using high-capacity deep neural network for function approximation.

However, Model-based Reinforcement Learning (MBRL) is regarded to be sampling efficient which requires less data gathered from the environment for training. The core of MBRL is to incorporate a dynamics model built by either using domain knowledge of the real-world dynamic system or a learning-based method. The dynamics model approximates the state transition process of the original, and variously utilizing this model can help the policy optimizing.

With the rapid research progress of the RL community, many easy-to-use open-sourced RL libraries are developed. But most of them apply model-free algorithms without considering the specialty of MBRL. Therefore conducting novel MBRL research requires more tedious coding tasks as there is no satisfying open-sourced library to use. Reproducing the results by existing algorithms or comparing novel algorithms against baselines requires researchers to go through multiple codebases of open-sourced MBRL algorithms while reviewing the codebase carefully to get the correct evaluations. Many modifications are made in these MBRL algorithm codebases especially in the source code

of the environment, e.g., OpenAI Gym. These include the redefinition of the reward function, state pre-processing, and terminal signal modification. These little changes always hide behind the bulky codebase but affect the performance of algorithms greatly. Users can miss out these details easily and are unable to reach systematic comparisons.

Moreover, even if users can successfully process these details, without a unified implementation and algorithm library for MBRL, it costs users extra effort for the user to conduct novel research based on existing codebases. For example, experimenting with user-defined environments requires one to provides corresponding APIs w.r.t different codebases and comparing algorithms' performance with self-defined metrics requires the user to properly record and generate them across multiple codebases. Also, due to the diverse taxonomy of MBRL, algorithms have great differences in terms of training workflow, dynamics model type and training methodology, thus bringing more difficulties.

Therefore, to boost the progress of research on MBRL, a unified, flexible, and programmable library is in need.

1.2 Motivation and Objectives

To address the above issues and boost the research on MBRL for the RL community, we aim to provide a unified MBRL library, with various features, unified implementations, and well-written documentation. Specifically, the proposed library should achieve the following objectives:

1. To deliver a well-designed and unified implementation comprising different MBRL algorithms. This objective is the core goal of Baconian. By achieving such a goal, it provides a user-friendly way for researchers to train, evaluate, and modify different algorithms with minimal lines of code needed.
2. To offer various experiment utilities and algorithm analysis tools. With these tools, it reduces the tedious work to analyze the experiment results such as processing log files, visualizing results, comparing performance across different algorithms, etc.

3. To provide well-written user and APIs documentation to facilitate the programming and extensions beyond Baconian. Comprehensive documentation covering the usage and programming guide will help users to get started with Baconian quickly, and even contribute more features into it.
4. To demonstrate Baconian’s capability in supporting RL related research topics with demo projects. With these demo projects, users will be capable to explore more scenarios and tasks where they can apply Baconian.

1.3 Organization of the Report

The report is organized as follows. In Chapter Two, we discuss the literature review of MBRL algorithms and open-sourced RL libraries. In Chapter Three, we introduce Baconian, from three main aspects, features, system design, and usage. In Chapter Four, we introduce some applications and research outcomes built upon Baconian, which demonstrate the capability of Baconian in supporting RL related research. In the last chapter, we conclude our work and give some future work directions.

Chapter 2

Literature Review

2.1 Introduction

In this section, we will give the literature review of MBRL and related open-sourced library. We first give the literature review of Reinforcement Learning and mainly focus on the Model-based methods. Then we give an overview of the widely-used open-sourced Reinforcement Learning libraries.

2.2 Model-based Reinforcement Learning

RL aims to control an agent that can perform actions in an environment such that it maximizes the long-term reward. The actions are generated by the policy which takes the state received from the environment as the input. The reward encodes the goals defined by the environment such as successfully driving the car autonomously.

RL methods can be categorized in two types: Model-based and Model-free. Model-free methods, especially Deep Learning based methods, have achieved great success in recent years [6]–[9]. But most of these methods demand a huge number of training examples which limit the tasks to only the simulator. To address such an issue, Model-based Reinforcement Learning, by learning a model of the environment, is believed that it can lower the sample complexity while achieving a great performance of the tasks.

2.2.1 Markov Decision Process

Reinforcement Learning problem is formalized as the Markov Decision Process (MDP). It is used to modeling the sequential decision-making problem. MDP can be further

Figure 2.1: The interaction process between agent and environment[10]

generalized to Partially Observable Markov Decision Process (POMDP)¹.

In the problem of MDP, the one that performs actions is *Agent*. The system that the agent interacts with is called *environment*. Time steps are set to be discrete $t = 0, 1, \dots, T^2$. At each time step t , the agent receives the *state*, $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the state set, from the environment, and generates the *action*, $A_t \in \mathcal{A}$, where $\mathcal{A}$ is the action space, by its *policy*. At next time step $t + 1$, the environment responds to the action it receives from the agent with a new state S_{t+1} and a reward $R_t \in \mathbb{R}$. Such process is noted as state transition. A visualized MDP is shown by Figure 2.1

As the state and reward are modelled as random variables in MDP, the probability distribution over the state and reward only depends on the previous states and actions, for particular values of state s , and reward r , the probability density function is defined by Equation³ 2.1:

$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\} \quad (2.1)$$

As the interaction between agent and environment goes, a sequence of state, action, and reward will be generated, called *trajectory*: $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_T, A_T, R_{T+1}$. As T denotes the terminal time step of trajectory, T can be ∞ . Such task is continuing task, otherwise the task is episodic⁴. Agent aims to maximize the accumulative reward it receives from the trajectory, i.e., the sum over $R_1, R_2, \dots, R_{T+1}$. Such definition of goal is stated as *reward hypothesis*[10]. Formally, we define the accumulative reward starting at time step t as return, denoted by $G_t \doteq \sum_{i=t+1}^{\infty} R_i$. Further, we include the discounted

¹POMDP is beyond the scope of our discussion

²The continuous MDP is beyond the scope of discussion

³ $\doteq$ denotes the equality relation is true by definition

⁴We don't differentiate the notation of continuing and episodic tasks for the simplicity.

factor into the definition of return, $G_t \doteq \sum_{i=t+1}^{\infty} \gamma^{i-t-1} R_i$, where $\gamma \in [0, 1]$, called discount factor. Introducing of discount factor makes the sum over an infinite number of reward finite if $\gamma < 1$. We can also find as Equation 2.2 showed that successive returns are related to each other.

$$G_t \doteq R_{t+1} + \gamma G_{t+1} \quad (2.2)$$

2.2.2 Policy and Value Functions

In RL, we utilize a function to indicate the goodness of the agent in terms of the expected return it will receive. Such type of function is called value functions. Moreover, such function should be defined w.r.t. agent's policy, i.e., how the agent will react given a certain state. Hence, policy is an another import function of RL.

Formally, policy is defined as a mapping from state to action, denoted as $\pi(a|s)$. Value function of a state s under a given policy π , denoted by $v_{\pi}(s)$ is defined by Equation 2.3 which is called state value function.

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t | S_t = s] = \mathbb{E}_{\pi}\left[\sum_{i=t+1}^{\infty} \gamma^{i-t-1} R_i | S_t = s\right], \text{ for all } s \in \mathcal{S} \quad (2.3)$$

We also define that value function of a state s and a action a under a given policy π , denoted by $q_{\pi}(s, a)$ as Equation 2.4 which is called state-action value function.

$$q_{\pi}(s, a) \doteq \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a] = \mathbb{E}_{\pi}\left[\sum_{i=t+1}^{\infty} \gamma^{i-t-1} R_i | S_t = s, A_t = a\right] \quad (2.4)$$

2.2.3 Optimality and Bellman Equation

To solve an MDP task, we aim to find a policy that can achieve a high sum reward. So first the optimality of the policy should be given. We define the order over policy as $\pi > \pi'$ if and only if $v_{\pi}(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$, and the optimal policy is denoted by π_* , and the corresponding optimal state value function is denoted by v_* following Equation 2.5.

$$v_*(s) \doteq \max_{\pi} v_{\pi}(s), \text{ for all } s \in \mathcal{S} \quad (2.5)$$

Similarity, we define the optimal state-action value function as follows:

$$q_*(s, a) \doteq \max_{\pi} q_{\pi}(s, a), \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \quad (2.6)$$

Also, we can derive the relation between $v_*(s)$ and $s_*(s, a)$ easily as follows:

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \quad (2.7)$$

With the above definition, we have the following two definitions, called Bellman equation, for $v_*(s)$, and $s_*(s, a)$ respectively.

$$\begin{aligned} v_*(s) &= \max_q \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')] \end{aligned} \quad (2.8)$$

$$\begin{aligned} q_*(s, a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \quad (2.9)$$

Bellman equations express the value of a state or state-action pair as the values of other states, so if we know the optimal value of the consecutive state, we can easily calculate the optimal value for current state. These equations establish some important methods in RL.

2.2.4 Dynamics Model

MBRL uses a dynamics model denoted by $\tilde{p}(s', r | s, a)$, to approximate the environment's ground-truth dynamics $p(s', r | s, a)$ of a MDP. As in most of the cases, the reward function is considered as the prior knowledge and dynamics model is mainly used to predict the state transition of a MDP, denoted by $\tilde{p}(s' | s, a)$ ⁵. The dynamics model function, denoted as $\tilde{f}(s, a)$ can be either deterministic, i.e., $\tilde{p}$ directly predicts the state, $\tilde{p}(s' | s, a) = \tilde{f}(s, a)$, or stochastic, by generating a distribution over predicated state conditioned on the given state and action, e.g., under a Gaussian distribution setting, $\tilde{p}(s' | s, a) \sim \mathcal{N}(\mu(s, a), \Sigma(s, a))$, and $\tilde{f}(s, a) = (\mu(s, a), \Sigma(s, a))$

For deterministic models, physical-based models are widely used[11]–[13] using the physical law or domain knowledge of the state transition process. Many learning-based

⁵Typically we assume the state transition and reward are conditional independent as most cases show.

models are also applied including Linear Regression (LR)[14], Locally Weighted Linear Regression (LWLR)[15], Decision Tree[16]. Moreover, like the recent advances of Deep Learning, deep neural networks are also applied as one powerful approximator for dynamics model[17], [18].

For stochastic models, one widely used in robotics systems and related problems is Gaussian Process (GP) [19]–[21], by inferring the function p that generate the state s' . Another one is the Locally Weighted Bayesian Regression (LWBR)[22] that combines the Bayesian inference and LWLR. In [23], [24], Deep Learning based methods are used including feed-forward neural network and recurrent neural network.

2.2.5 Algorithms

What distincts the MBRL from other RL algorithms is applying a state transition model to facilitate the policy searching and optimizing. Thus it brings some variants to the algorithm design, firstly, the choice of the model is various, it can be learning-based, domain knowledge-based, the prediction results of the model can be deterministic or stochastic. Secondly, using the model to search and optimize the policy can also be various which include using as an extra sampling source for the agent, or applying gradient-based optimization to policy using the differentiable model. Such variants lead to different MBRL algorithms.

In this section, Some classical and state-of-the-art MBRL algorithms covering three major categories of MBRL will be introduced: Dyna-style algorithms, Policy Search with Back-propagation through time algorithms, and Shooting algorithms.

2.2.5.1 Dyna-style Algorithms

Dyna[25] method is proposed as a classical MBRL by treating the dynamics model as an environment to sample data from. Specifically, it first learns the dynamics model using sampled data from the environment, then trains a model-free RL algorithm using the learned dynamics model to generate the imagined data. Such two steps iterate until the training policy converges. Based on such a training process, some more advanced algorithms are proposed in recent years, categorized as Dyna-style algorithms.

In [17], the author proposes to ensemble multiple neural network-based dynamics model to overcome the model-bias problem. For policy optimizing, they use the Trust Region Policy Optimization (TRPO) [26]. When sampling from the ensemble dynamic models, the state transition process will be performed by one randomly selected dynamic model which can prevent the policy from over-fitting to any single dynamic model.

In [27], following the Dyna algorithm structure, the author designs a meta-algorithm that can provide a theoretical guarantee for the continuous improvement of policy and gives a simplified implementation, referred to as Stochastic Lower Bounds Optimization (SLBO).

2.2.5.2 Shooting Algorithms

The idea behind the Shooting algorithms of MBRL is to approximately solve the optimal control problem when system dynamics, i.e., the state transition is known or approximated by a model. Most of them follow the Model Predictive Control (MPC) [28] to re-plan the action for each time step.

One widely adopted algorithm is Random Shooting (RS) [29], [30]. In RS, the agent generates K action sequences and evaluates its performance i.e., sum reward or cost with learned dynamics model, then the one action sequence with the best performance will be selected and the agent returns the first action of it. Then the agent will re-plan after receiving the new state.

Model-based Initialization of Model-free Reinforcement Learning (MB-MF) [18] algorithm firstly uses a neural network to approximate the state transition and apply with one RS policy to generate sample trajectories as dataset $\mathcal{D}$. Then use DAgger [31] to transfer the RS policy to a model-free policy, which is TRPO, as the initialization of it, then use model-free training to further fine-tune the model-free policy.

2.2.5.3 Policy Search with Back-propagation through Time

In Dyna-style algorithms, the dynamics model is treated as the black box and only used to generate imagined data. Whereas one can explore the gradient information if the dynamic model is differentiable. By such, it is possible to derive the analytical gradients to optimize the gradients under certain assumptions. We categorize such type algorithm

as Policy Search with Back-propagation through Time since the policy is optimized in a way to maximize the sum reward across a certain time horizon.

Linear Quadratic Regulator (LQR) [32] solves the optimal control when the system dynamics are known linear dynamics system and the cost function is quadratic function while following the MPC by re-planning the action at each time step.

When the system dynamics are non-linear and the cost function is non-quadratic, iterative Linear Quadratic Regulator, iLQR [33] is proposed by approximate the true dynamics with a linear function and cost function with a quadratic function, then apply LQR to solve.

Guided Policy Search (GPS) [11], [34]–[37] as well as the variants of it, apply a Gaussian Linear time-varying system to approximate the system dynamics firstly. Then GPS uses an iLQR policy to guide the optimizing of the policy by behavior cloning using samples collected by the iLQR policy.

Probabilistic Inference for Learning Control (PILCO) [38] uses Gaussian Process to approximate the system dynamics and compute the analytical gradients of objective function w.r.t. policy parameters to update the policy. The algorithm iterates between sample collection and policy updating.

2.3 Reinforcement Learning Library

As RL becomes a trending research topic in recent years, lots of high-quality related open-source codebases are released to facilitate the research of RL. Some codebases provide the testing bed which contains rich environments and tasks to benchmark RL algorithms, others provide high-quality implementations of state-of-the-art RL algorithms. Moreover, some distributed training frameworks are also released to speed up the training of RL algorithms in a distributed manner.

In this section, we briefly introduce some widely adopted open-source codebases and how they facilitate the research of RL and reduce the efforts of researchers on implementing tedious coding and benchmarking tasks. The related libraries are also summarized at Fig. 2.2.

Type	Framework	Features
Environment & Task	OpenAI Gym	Provide standard interface and various types of tasks
	DeepMind Control Suite	Consist robotics control tasks built upon Mujoco engine which requiring license
	Pybullet	Consist robot control tasks with free simulation engine
	Rlbench	Consist Multiple types of RL tasks, e.g., model-free tasks, imitation learning tasks
	Torchcraft	Provide control and training interface for StarCraft(a RTS game)
Algorithm	OpenAI baselines	Provide high-quality and monolithic implementations of model-free methods
	Google Dopamine	Provide reusable and modular algorithm with user-friendly guide
	Facebook Horizon	Provide end-to-end platform focusing on applied RL
	Guided Poliy Search	Provide implementations for various GPS methods and control interface with real robots
	MBBL	Provide comprehensive benchmarks for many MBRL methods

Figure 2.2: RL open-sourced frameworks

2.3.1 Environment and Task Library

To conduct research over novel RL algorithms, researchers need good and rich benchmark tasks on which to compare different algorithms. Thus, some benchmark task libraries are released by the RL research community.

One widely used is Gym [39] released by OpenAI which provides various environments exposed with a unified interface. Theses tasks are classified into *Classic control and toy text, Algorithmic, Atari, Board games, 2D and 3D robots* using MuJoCo engine[40]. The environments of Gym are accompanied by a version number to record any changes of the environments which makes the results of RL algorithms over them comparable as long as the version number of the environments is aligned. Moreover, the interfaces provided by Gym are widely adopted by many other RL tasks libraries [41], [42] which make these libraries more accessible with unified interfaces.

In [43]–[46], many robotics control tasks are provided including controlling from low-level input and visual input. Some of them provide simple tasks like balancing pendulum while others assemble real-world robotics tasks like grabbing objects, opening doors.

These libraries provide a very comprehensive test-bed for research of RL over the robotics domain.

In [41], [42], [47], some video/ board games and Real-Time Strategy (RTS) games are provided. These tasks offer the test-bed for RL algorithms to compare their performance with human beings, and some RL methods are proved that can overcome the human level in games like Dota[48], Starcraft[49], etc.

These environments libraries build a comprehensive and reliable test-bed for benchmarking RL algorithms across various task domains and bring the gap of applying RL algorithms to solve real-world problems.

2.3.2 Algorithm Library

Besides the well-developed environment libraries, to facilitate the quick prototyping of RL algorithms, many RL algorithm libraries with high-quality implementations are also under active development of the RL community.

Most of the current work focuses on the model-free RL algorithms. In [50], it provides many state-of-the-art model-free algorithms high-quality implementations with some advanced exploration methods included, and support for OpenAI Gym are provided. But each algorithm is implemented in a standalone module without a unified design which brings difficult for researchers to read the code and reuse it. In [51]–[53], many modular designed RL algorithm libraries are provided to conquer the above problems while providing more or less model-free algorithms. Besides the libraries focusing on modular and flexible design so to allow users to easily make the extension, some others [54]–[57] provides the algorithms with very comprehensive benchmarking results and analysis over reproducibility, the sensitivity of RL algorithms.

Whereas for MBRL, only limited libraries and codebases are released. In [58], the GPS library provides the implementations for the variants of Guided Policy Search[11], [34]–[37] which is one widely adopted MBRL algorithms. It also contains the interfaces of simulation engine, controller, and sensor interface to perform training over some real robots. But the library mainly focuses on GPS algorithms which makes it difficult to be extended to other MBRL algorithms.

In [59], the authors provide the benchmark results of 11 MBRL algorithms across 18 environments of Gym and also identify some key research challenges of MBRL. Though the library provides the researchers the performance evaluations of different MBRL algorithms, the implementations of these algorithms are scattered across multiple codebases without unified interfaces, which incur the additional efforts for users to research MBRL based on it.

Though these high-quality projects bridge the gap between the RL algorithms design and their corresponding implementations, the RL community still lacks a well-designed and implemented MBRL library with unified interfaces. This is the issue we will address in the Baconian project.

2.3.3 Distributed Training Library

Deep Learning model usually requires large amount of computing resources, and Deep Reinforcement Learning is no exception. So to speed up the training and tweaking of these algorithms, some distributed training frameworks are proposed to make the training of these algorithms scaled to clusters or multiple nodes.

Ray[60], Horovod[61] are two widely used distributed frameworks for building and running distributed applications.

RLlib [62] is a dedicated distributed training framework specialized for RL build upon the Ray[60]. RLlib provides some primitive scalable components which enable the user to implement the RL algorithms with high performance and scalability.

Chapter 3

Baconian: A Unified Model-based Reinforcement Learning Library

In this chapter, we introduce our work, Baconian, a unified MBRL library. The project is available at <https://github.com/cap-ntu/baconian-project>. We mainly focus on the feature and system design & implementations aspect of Baconian. As we aim to provide an easy-to-use, flexible, and, extensible MBRL framework, Baconian contains some specially designed features for MBRL. We also come up with novel system architecture and implementations so to achieve these features in a modular manner.¹

3.1 Features

As one RL library, Baconian supports many RL algorithms, test environments, and experiment utilities. We summarize the main features in Fig. 3.1.

3.1.1 RL Algorithms

We implement many widely used RL algorithms. For model-based algorithms, we implement Dyna[25], ME-TRPO [17], MPC[30], and iLQR[33]. Since many model-based algorithms are built upon model-free algorithms, we also implement some popular model-free algorithms including DQN[63], DDPG[64], and PPO[65] in Baconian. These algorithms can also facilitate performance comparison across model-free and model-based algorithms.

¹The work in this chapter is derived from [5], which has been submitted for publication.

Algorithm	Dyna, GPS, ME, MPC, iLQR, DQN, DDPG, PPO
Environment	OpenAI Gym, RoboSchool, DeepMind Control Suite
Utility	Built-in Logging and Visualization, TensorFlow Integration, Parameter Management
Open-source Support	User Guide and API References, MIT License, Benchmark Results Released

Figure 3.1: Feature list of Baconian.

3.1.2 Supported Test Environments

To evaluate the performance of RL algorithms, it is a must to support a wide range of test environments. Baconian support OpenAI Gym[39], RoboSchool[66], DeepMind Control Suite[43]. These test environments cover the most essential tasks in the RL community. By providing a unified way to access these environments can reduce the effort to switch the algorithms between different testing environments.

3.1.3 Experiment Utilities

Baconian provides many utilities to reduce users' efforts on experiment set-up, hyper-parameter tuning, logging, result visualization, and algorithms diagnosis. We provide integration of TensorFlow to support neural network building, training, and managing. As the hyper-parameters play a critical role in RL experiments, we provide user-friendly parameter management utility to remove the tedious work of setting, loading, and saving these hyper-parameters.

3.1.4 Open-source Support

Baconian provide detailed user guides and API references². With user guides, researchers can hand on Baconian easily and conduct novel MBRL research upon it. With APIs ref-

²The documentation can be found at <https://baconian-public.readthedocs.io/en/latest/API.html>.

erences, researchers can program and extent the Baconian project with new features and novel algorithms. We also release some preliminary benchmark results in the codebase.

3.1.5 Fine-grained Logging and Analysis

The logging system plays an important role in reinforcement learning experiments. As it is used to record the performance of different algorithms against different tasks and perform hyper-parameter searching, algorithms analysis, and diagnosis, etc. To reduce users' effort on these tedious tasks, we provide a built-in logging and visualization module.

Meanwhile, it is a common need to customize the logging of certain values like the visited state of the agent, algorithm-specific parameters, to analyze and diagnose the algorithms. Such customized need is unable to be captured ahead which requires the users to implement by themselves. In Baconian, users can achieve such by a simple python function decorator or one line of code, and the rest of the work including logging printing, saving, and visualizing will be handed by Baconian.

3.2 System Design and Implementation

In this section, we introduce the system design and implementation of Baconian, especially, on how it handles the features mentioned above with a modular approach. Such a modular approach makes Baconian a flexible and extensible library for users to conduct any further potential modification to any built-in modules.

3.2.1 System Design

Baconian consists of three major components, namely, Experiment Manager, Training Engine, and Monitor. The system overview of Baconian is shown in Fig. 3.2.

3.2.1.1 Experiment Manager

The Experiment Manager consists of Experiment Settings, Status Collector, and Experiment Recorder. Experiments Settings manages the creating and initialization of each module. Status Collector controls the status information collected across different modules to compose a globally shared status that can be used including learning rate decay,

Figure 3.2: The system design of Baconian. The system is divided into three main modularized components to minimize the coupling for flexibility and maintainability.

exploration strategy scheduling, etc. Experiment Recorder will record the information generated from the experiment, such as loss, rewards. Such information will be handed to the Monitoring layer for rendering or saving.

3.2.1.2 Training Engine

The training Engine handles the training process of the MBRL algorithms. The novelty of the design lies in abstracting and encapsulating the training process as a Control Flow module, which controls the execution processes of the experiment based on the user’s specifications, including the agent’s sampling from the environment, policy model and dynamics model optimization, and testing. MBRL algorithms can be complicated[18], [25]. Such abstractions can decouple the tangled and complicated MBRL training processes into some independent tasks, which are further encapsulated as the sub-modules of the Control Flow module for providing flexibility.

3.2.1.3 Monitor

The monitor is responsible for monitoring and recording the experiment as it proceeds. This includes recording necessary logging, printing information/warning/error, and rendering the results.

Figure 3.3: Simplified UML diagram of Baconian

3.2.2 Implementation

Baconian is implemented following the Object-Oriented Design manner. Various design patterns are applied to decouple the complicated dependencies across different modules to enable easy extension and programming over the framework.

3.2.3 Control Flow Module

In Baconian, the control flow of MBRL is delegated to an independent module, which can improve the flexibility and extensibility of the framework. We abstract and implement two commonly used control flows, one is the model-free DRL flow including sampling, policy evaluation, and policy improvement. The other one is Dyna-like control flow [25], which is used in model-based algorithms. Furthermore, users can define control flows by inheriting the control flow module without extra changes to other modules.

3.2.4 Stateful Modeling

Most RL algorithms, similar to machine learning algorithms are split into a two-step process, i.e., iterate between the training and testing processes. Some components of

RL can behave differently under two steps, including exploration strategy, action noise, parameter perturbation, etc. Also, another need that commonly existed in RL and ML, is parameter scheduling, including learning rate, noise decaying, etc.

To implement these features in an easy and reusable way, and enable the extension for unforeseeable need when the algorithms relate to a more complicated process brought by MBRL, we equip a statuses module within every component in Baconian which may require to perform such actions, e.g., agent, environment. The status module makes the behavior of its attached component a stateful one. The component can act differently referring to different statuses while the status can be updated in a predefined transition. We further develop a global status collector module that collects and shares the status information collected from other modules during running time, such as the total number of samples obtained by an agent, total training steps. Such global status can be shared across different modules to perform operations like parameter scheduling etc.

3.2.5 Parameter Management

To facilitate hyper-parameters setting, tuning, and model storage/reload functions, we unified the parameter management in a single module which is incorporated into other modules that demand parameter management. It includes parameters saving, loading, and type checking functions, which reduce the users' efforts on these tedious works.

3.2.6 TensorFlow Integration

As the Deep Learning model and algorithms are applied to RL more and more, integration with a widely used deep learning framework is on demand. In Baconian, we integrate the TensorFlow[67] to support the deep learning techniques. Users can directly build a neural network with simple configurations instead of using low-level APIs from Tensorflow. Such encapsulation further reduces users' effort in building RL algorithms.

3.3 Usage

In this section, we introduce the usage of Baconian. First, as its primal purpose, we introduce how to build up RL experiments with fully customizable built-in modules. Then

we give two examples to show one can easily extend some novel features by programming using Baconian within only a few lines of code.

3.3.1 Building RL Experiments

Figure 3.4: The procedure to create an MBRL experiment in Baconian which can be referred to as a simplified sequential UML diagram. Each module is replaceable and configurable to reduce the effort of building from scratch.

This section presents the procedures to create MBRL experiments with the essential modules in Baconian. The procedures are shown in Fig. 3.4.

For high flexibility, most of the modules are customizable. Meanwhile, the user can directly adopt a built-in benchmark module or code if customization is unnecessary.

First, the user should create an environment and a RL algorithm module with necessary hyper-parameters configured, e.g., neural network size, learning rate. The algorithm module is usually composed of a policy module and a dynamics model module depending on different algorithms. Then, the user needs to create an agent module by passing the algorithm module, and the exploration strategy module if needed, into it.

Second, the user should create a control flow module that defines how the experiments should proceed and the stopping conditions. This includes defining how much samples

should be collected for training at each step, and what condition to indicate the completion of an experiment, etc. Some typical control flows have already been implemented in Baconian to meet the users' needs.

Finally, an experiment module should be created by passing in the agent, environment, control flow modules into it, and then launched. After that, the Baconian will handle the experiment running, monitoring, and results saving/logging, etc.

3.3.2 Programming with Baconian

With such decomposed procedures, the user can easily replace any modules following the corresponding module API to quickly prototype novel MBRL research. Besides the plenty of features included in Baconian, it is also a must to provide an easily programmable framework to conduct some novelty ideas and research outcomes for researchers. In this section, we list three typical scenarios with Baconian and show how easy it is to programming based on Baconian to quickly prototyping the research ideas with the most possible reusable components and a minimal amount of code. Moreover, we conduct well-written documentation and API references ³.

3.3.2.1 New Tasks/Algorithms/Dynamic Models.

As we implement Baconian in a unified manner, so it will bring the advantages of reusing modules as much as possible, and reduce the effort of implementing new tasks, algorithms, and dynamic models. For implementing tasks, we adopt the OpenAI Gym API with some proper modifications. For algorithms and dynamics models, we have self-designed APIs with well-written documentation so the user can understand the APIs easily.

3.3.2.2 New Training Workflow.

As in the design of Baconian, we separate the training process from the algorithms and abstract with a flow module. Such decoupling reduces the effort when the researcher needs to design a novel training workflow without re-implement the training algorithms. We give an example ⁴ to show that user can implement the early stopping feature within 20 lines code, and such feature can be directly reused by all algorithms.

³It can be found at <https://baconian-public.readthedocs.io/en/latest/>

⁴The example can be found at https://github.com/cap-ntu/baconian-project/blob/master/examples/early_stopping_flow.py

3.3.2.3 Customize Logging.

With a built-in logging module, users can easily extent customized logging processes to record any necessary information during the experiments to analyze and diagnose the algorithms. Specifically, users can achieve this in two ways, one is to using logging python decorator to capture the return value from one function that generates such logging information, e.g., decorating the predication function of agent that returns the action to analyze the action distribution. The second is to directly append one log item, and the rest process includes log processing and saving will be handled automatically. The demo code is given below:

```
// method 1: add one decorator to record
// the return value of the function for logging.
@record_return_value
def foo():
    ...
    x = 1
    return x
// method 2: manually append one log item
// declare a recorder
recorder = Recorder()
def foo():
    ...
    x = 1
    recorder.append_obj_log(x)
```

3.4 Benchmarking Baconian

To further demonstrate the usability of the framework and the quality of the implementations, we conduct some benchmarking test with widely adopted RL task[42] in this section.

3.4.1 Benchmark Tasks

We adopt the Pendulum task to verify the multiple algorithm’s performances in Baconian. In the task, the agent needs to maintain the balance of a pendulum by applying the torque to a pendulum. The reward will accumulate constantly as long as the pendulum does not fail off.

Figure 3.5: The pendulum control tasks.

Table 3.1: Preliminary benchmark of continuous task: Pendulum-v0. DDPG(baselines) result is obtained by open-source library baselines with same hyper-parameters for comparison.

Algorithms	Cumulative Reward
DDPG (baselines)	-499.89
DDPG	-417.67
PPO	-320.23
Dyna (DDPG, global MLP dynamics)	-1076.82
MPC (global MLP dynamics)	-618.72
iLQR (MLP dynamics)	-501.25

3.4.2 Experiments Setup

In all experiments, 10,000 samples are used in training and the cumulative reward is averaged over 10 times independent tests for each algorithm with different random seeds. The averaged reward over multiple runs can demonstrate a more reliable evaluation as RL algorithms are regarded as unstable and sometimes can fail to learn.

3.4.3 Benchmarking Results

The results are given in Table 3.1. As we can see from the results, we achieve similar and even better performance compared to implementations in[50] which is a widely adopted RL benchmark library that focuses on model-free method.

3.5 Conclusion

In this chapter, we introduce Baconian, by mainly focusing on the functionalities provided to the user and the system design which enables it to address the challenge to build an easy-to-use RL library. At last, we further benchmark our framework with commonly used RL tasks compared to other open-source RL libraries. The results demonstrate that Baconian provide a high-quality implementation of RL algorithms, and can be provided as a reliable toolbox for RL research.

Chapter 4

Case Study with Baconian

In this chapter, we introduce some novel research outcomes and applications built upon Baconian. With these applications, the applicability of Baconian can be well demonstrated.¹

4.1 Intelligent Trainer for Dyna-Style Model-Based RL

In [2], we propose an optimizing algorithm that aims to schedule the hyper-parameters introduced by the Dyna-style MBRL algorithm in an online manner. It requires no expert knowledge on how to properly set these hyper-parameters to achieve the best performance of the policy under training.

4.1.1 Intelligent Trainer Algorithms

The algorithm diagram is given in Figure. 4.1. We refer to the original Dyna algorithms and its training process as the inner section, Training Process Environment (TPE), where a target controller is deployed, to solve the task defined by the real environment, and using the cyber environment to sampling imagined data. Thus, such modeling is called Reinforcement on Reinforcement (RoR). By formulating the inner Dyna training process as a standard MDP, we can apply RL based policy to control the inner process. The MDP formulation of TPE is defined as follows.

- *State*: the state of TPE at time step t , $\vec{s}_t \in \mathbb{R}$ is exposed to the outer agent who can use to access the training progress. We found that using a constant (zero) to

¹The work in this chapter is derived from [1], [2], [4], which has been submitted for publication.

Figure 4.1: Illustration of “Reinforcement on Reinforcement” (RoR) framework. The inner box encapsulates a standard Dyna into a training processing environment (TPE) as the inner layer. In the outer layer we introduce an intelligent trainer, controlling the optimization of the Dyna training flow configuration in the TPE.

represent the TPE state can still work as such a simple setting allows the trainer to learn good action quickly.

- *Action*: the action at time step t , comprises three control parameters, $\vec{a}_t = [a^0, a^1, a^2]^T \in \mathbb{R}^3$ are exposed to outer agent who can utilize it to control the training process. These three actions control the sampling, training, and reset process of the target trainer under two different environments, one is the real environment, another is the cyber environment. Specifically, we use $\{0.2, 1.0\}^3$ as the default action space.
- *State Transition*: state transition represents the execution of one iteration of the Dyna algorithm, which is regarded as the state transition process of TPE. It includes sampling from the real and cyber environments, training the target controller, and training the cyber model. During such transition, some hyper-parameters are applied
- *Reward*: the reward function should reflect the performance of the target controller to guide the learning of the trainer. In the design, we use the testing reward of the target controller as the instant reward for the intelligent trainer at each TPE step.

Such a process is modeled as a typical RL environment using the MDP model. We refer to the hyper-parameter scheduling process as the outer section and we can deploy

different online scheduling policies called the intelligent trainer to maximize the performance of the inner target controller.

We firstly propose the uni-head intelligent trainer which utilizes the DQN[7], as the trainer policy to schedule the hyper-parameter settings dynamically in each step of the TPE.

The uni-head trainer, described previously, for some cases cannot adequately assess the quality of action as all actions are tested in a single streamline. In other words, the actions could be correlated and their quality could become indistinguishable. Also, for actions that generate non-negative rewards but could lead to slow convergence or locally optimal policy, To address these issues, we propose an ensemble trainer that uses a multi-head training process, that is composed of different scheduling policies including original DQN, random policy, and fixed policy. Some mechanisms are applied to boost the performance of the ensemble trainer, including physical-data memory sharing, reference sampling, order-based trainer reward function, and weight transfer.

4.1.2 Implementations with Baconian

The algorithms and experiments build upon the Baconian framework. Some import features provided by Baconian help to prototyping the algorithms easily and quickly. To build such a nested reinforcement learning process, the well-decoupled *Flow* implementations of Baconian can easily come up with a modular implementation for the algorithms.

4.1.2.1 Training Process Environment

As two RL based learning processes incorporated, the inner target controller and outer trainer are incorporated, we first encapsulate the canonical Dyna-style MBRL training process into a standard RL environment as TPE in Figure. 4.1, and formulated as a MDP. Such implementation brings a few advantages:

1. removing the unnecessary coupling of the inner training process and outer one by only exposing the necessary state, action, and reward information to the outer trainer.

2. allowing to easily replace different RL algorithms both for the inner target controller and outer trainer.
3. allowing to easily test the different designs of the TPE in terms of the state space, action space, and reward function.

4.1.2.2 Customized Training Flow for Ensemble Trainer

The introduction of the ensemble trainer and related mechanisms to boost the performance of the trainer and address the shortage brought by uni-head trainer also result in challenging implementations for it. With the design of Baconian, we can implement the algorithm more flexibly.

Most importantly is to customizing the training flow of the ensemble trainer as we described in Section. 3.2.3. since in each step for the training of ensemble trainer, there will be some extra processes including physical-data memory sharing, reference sampling, order-based trainer reward function, and weight transfer being executed. By implementing these functionalities in the training flow without changing any other modules make it fast to migrate the uni-head trainer into the ensemble trainer which reduces the chances to bring the bugs into.

4.1.3 Numerical Evaluations

We evaluate the proposed intelligent trainer and ensemble trainer for five different tasks of OpenAI gym: Pendulum (V0), Mountain Car (Continuous V0), Reacher (V1), Half Cheetah[68], and Swimmer (V1). Multiple variants of the uni-head intelligent trainer are compared with baseline algorithms. There are three baseline algorithms and four intelligent trainers. The three baseline algorithms are:

- The NoCyber trainer is a standard model-free RL training process without using synthetic data.
- The Fixed trainer follows the standard Dyna algorithm, all actions of the trainer are fixed to the mean of the action space.
- The Random trainer draw actions that follow a uniform distribution over the action space.

The four intelligent trainers are:

- DQN trainer. For exploration purposes, the epsilon-greedy method is used, with the first 10% of the trainer steps for epsilon-greedy exploration by setting the final epsilon to 0.1.
- DQN-5 actions. To test the effect of more action values in the action discretization, we adopt a larger action space as $\{0.2, 0.4, 0.6, 0.8, 1\}^3$.
- DQN-larger memory. To test the impact of the larger data memory of the trainer, we adopt the DQN memory with the size of 2000. In this case, more trainer samples are stored and relatively older historical data are used in the training of the DQN controller.
- REINFORCE. We adopt REINFORCE to replace the DQN and verify the effect of different trainer methods. REINFORCE, as a Monte-Carlo RL method requires complete episode data for training, we manually set 5 steps (manually tuned) of TPE as an episode since naturally there is no terminal signal in TPE.

Figures. 4.2 shows the results of three baseline trainers and four intelligent trainers. To summarize, our uni-head intelligent trainer can achieve better or similar performance to various baseline trainers for the tasks that can benefit from using the cyber model. For tasks that are difficult to apply any synthetic data to improve the performance, even applying synthetic data can degrade the performance, the uni-head trainer can still learn policy than close to the optimal policy. But due to the nature of the task, the trainer is unable to recover the performance of the target controller. To summarize, our uni-head intelligent trainer can achieve better or similar performance to various baseline trainers for the tasks that can benefit from using the cyber model. For tasks that are difficult to apply any synthetic data to improve the performance, even applying synthetic data can degrade the performance, the uni-head trainer can still learn policy than close to the optimal policy. But due to the nature of the task, the trainer is unable to recover the performance of the target controller.

The results, as presented in Fig. 4.3, show that the ensemble trainer achieves overall good performance even in the cases the uni-head trainer fails. For the tasks of Pendulum,

(a) Pendulum V0

(b) Mountain Car Continuous V0

(c) Reacher V1

(d) Half Cheetah

(e) Swimmer V1

Figure 4.2: Accumulative rewards of different uni-head trainer designs for different tasks in (a)-(e). The curves show the average accumulative reward while the shaded region shows the standard deviation of the reward on 10 independent runs. The proposed uni-head trainer shows its adaptability (better than the Fixed trainer) but may fail in certain cases like Swimmer.

(a) Pendulum V0

(b) Mountain Car Continuous V0

(c) Reacher V1

(d) Half Cheetah

(e) Swimmer V1

Figure 4.3: Accumulative rewards of ensemble trainer for different tasks in (a)-(e). The proposed ensemble design shows close-to-optimal or even better performance on all cases.

Mountain Car, and Reacher, the ensemble trainer perform almost as well as the DQN or Random trainer. For the tasks of Swimmer and Half Cheetah, the ensemble trainer performs better or similar to the NoCyber trainer, even though the learning process makes it learn slower in the Half Cheetah task. With the proposed ensemble trainer, we are more likely to achieve sampling cost saving in practice as it is hard to predict which kind of algorithm variant will deliver the best performance in advance.

4.2 Content-of-Interest based Adaptive Bitrate for Online Video Streaming via DRL

In [1], [4], we propose a DRL based adaptive bitrate (ABR) control policy for online video streaming that incorporating human interest in a different part of the video, referred to as Content of Interest (CoI) based rate adaptation method.

The motivation follows the fact that the human visual attention system is selective [69], and the user tends to watch more videos that are interesting to them. Allocating more bitrate budgets for the users' more interested parts of video content can achieve higher viewing experiences and reduce the information loss caused by video distortion.

But it is challenging to analyze video content from the user's perspective and incorporate the information for rate adaptation. To address these problems, we first design a deep learning-based approach for analyzing the interestingness of video content. Then, we design a DRL based method to output the video rate adaptively by incorporating video interest information. The method can learn the optimal rate adaptation policy by jointly considering buffer occupancy, bandwidth, and the interestingness of video content.

The implementations and evaluations of the proposed method and various baseline methods on the public dataset are supported by the Baconian and its various features.

4.2.1 System Design and Workflow

We illustrate the design of the Content-of-Interest based rate adaptation mechanism for ABR streaming in Figure. 4.4. The system consists of the following components.

Streaming Server: The streaming server pro-processes video files and streams video content to users. For video pre-processing, each video file will be converted into many

Figure 4.4: The design of the ABR Streaming method with COI based rate adaptation

representations with different bitrates and segmented into many equal-duration video chunks [70]. The system first analyzes the interestingness of the video content by each chunk. The Media Presentation Description (MPD) file[71] will store the available bitrate information and the interestingness value of each chunk.

Video Player: MDP file will be requested when a new session started from the player, then it will also return the available bitrates for requested video file. The video player requests the selected video chunks from the server and measures the average bandwidth for downloading each video chunk.

DQN Agent: We adopt the DQN method [9] for rate adaptation. The DQN agent will use the current buffer usage, the bandwidth, and the interestingness of the next several video chunks as the system state, and determine which bitrate should be selected for the next video chunk accordingly.

Video Interestingness Recognition: In [4], we proposed a deep learning-based model for video interestingness recognition. 3D Convolutional Networks (3D ConvNets) are used to analyze the content of the video and predict the interestingness score as 3D ConvNets are more suitable for learning spatiotemporal features [72],

4.2.2 DQN based Rate Adaptation with User Interests Awareness

The MDP formulation of the ABR problem is given as follows. A discrete-time system is applied, where the time is denoted as $t = 1, 2, 3, \dots$. The duration of each time slot may not be equal and depends on the time for downloading a video chunk. The rate adaptation problem is formulated as a Reinforcement Learning (RL) task. During the task, the agent interacts with the streaming server as the environment to improve its policy thus reach the optimal rate adaptation policy. More specifically, at each time step t the agent receives the observed system state s_t after downloading video chunk $t-1$, then takes action a_t for selecting the bitrate of current video chunk according to the current policy, and finally gets reward r_t after downloading video chunk t . These procedures will be repeated until the end of a video session.

Streaming Environment: The set of available bitrates in the streaming system for each video is denoted as B . The bandwidth during a video session is time-varying, and we denote the average bandwidth for downloading video chunk t as v_t . The interestingness of video chunk t is denoted as w_t . The selected bitrate for video chunk t is denoted as b_t .

State: The state describes the bandwidth of the streaming service, the buffer occupancy of the video player, and the interestingness of the following video chunks, etc. We denote the state at time slot t as s_t , specifically,

$$s_t = (\vec{v}_t, L_t, b_{t-1}, \vec{w}_t, \vec{u}_t), \quad (4.1)$$

where $\vec{v}_t$ is the vector consisted of the predicted average bandwidth for downloading the next k video chunks (i.e., $\vec{v}_t = (v_t, v_{t+1}, \dots, v_{t+k-1})$), L_t is the buffer occupancy before downloading video chunk t , b_{t-1} is the selected bitrate at time step $t-1$, $\vec{w}_t$ is the vector consisted of the interestingness of the following h video chunks (i.e., $\vec{w}_t = (w_t, w_{t+1}, \dots, w_{t+h-1})$), $\vec{u}_t$ is the vector consisted of the available chunk size of video chunk t . Here, the interestingness of a whole video file is known at the start of a video session, because video content will be pre-processed on the server and the interestingness information will be manifested in MPD.

Action: The control action for the agent is to decide which available bitrate to use for the next video chunk giving current system state, which can be described as

$$a_t = \pi(s_t) \rightarrow b_t, b_t \in B, \quad (4.2)$$

Figure 4.5: The DQN network for interest-aware rate adaptation.

where π is the policy for selecting bitrate.

Reward: We adopt the following utility function revised based on the QoE metrics defined in [73] for measuring the reward during a time slot,

$$r_t(s_t, a_t) = \underbrace{f(w_t)}_{\text{weight}} \underbrace{q(b_t)}_{\text{quality}} - \underbrace{\alpha R_t}_{\text{video stall}} - \underbrace{\beta |q(b_t) - q(b_{t-1})|}_{\text{quality variation}}, \quad (4.3)$$

where r_t is the reward at time step t , $f(\cdot)$ retrieve the weight of a video chunk given the interestingness of it, $q(\cdot)$ maps video bitrate to video quality, α defines the degree for the penalty of rebuffering time, R_t is the rebuffering time incurred during time slot t , and β defines the degree to penalize high-quality variations. With the reward function in Equation. 4.3, the video chunks with higher interestingness have higher weights, therefore, the agent will get more rewards if the video chunks with higher interestingness are allocated more bitrate budgets.

DQN Agent: We apply DQN [9] based agent to optimize the rate adaptation policy, and the network of DQN is given in Figure. 4.5. System states are applied as the inputs for DQN, listed in Equation. 4.1. And the outputs of the network are the action-value functions, $Q(s, a)$, which represent the quality of the state-action combinations.

We illustrate the details of the DQN based learning algorithm for rate adaptation in Algorithm 1.

4.2.3 Numerical Evaluations

In this section, we introduce the evaluation results of the proposed method with various baseline methods.

Algorithm 1 DQN for Interest-Aware Rate Adaptation

```
1: Initialize replay buffer  $D$ 
2: Initialize DQN randomly
3: for video session = 1, 2, ...,  $M$  do
4:   Initialize the video player and choose a video file
5:   Observe initial state  $s_1$ 
6:   for video chunk  $t = 1, 2, \dots, K$  do
7:     With probability  $\epsilon$  randomly select a bitrate  $a_t$ 
8:     otherwise select bitrate  $a_t = \arg \max_a Q(s_t, a; \theta)$ 
9:     Download video chunk  $t$  until completed
10:    Observe reward  $r_t$  and next state  $s_{t+1}$ 
11:    Store transition  $(s_t, a_t, r_t, s_{t+1})$  into  $D$ 
12:    Randomly sample  $N$  transitions from  $D$ 
13:    Set  $y_{t'} = r_{t'}$ , if the video session ends
14:    otherwise set  $y_{t'} = r_{t'} + \gamma \max_a Q(s_{t'+1}, a; \theta_{i-1})$ 
15:    Update the network by minimizing  $L(\theta_i) = \mathbb{E}[(y_{t'} - Q(s_{t'}, a_{t'}; \theta_i))^2]$ 
16:  end for
17: end for
```

4.2.3.1 Experiment Settings

We adopt the TVSum dataset [74] for learning the predication of video interestingness. The dataset was created by segmenting videos into two second-long video segments, and 20 users were invited to rate each segment compared to other segments from the same video. The average of the rating for each segment is used as the ground truth, and the scale is from one to five. To simulate different network conditions, we adopt the 3G/HSDPA mobile dataset [75] and the FCC broadband dataset [76] for training DQN and evaluating performance.

The performances of our method are compared to the following methods: 1) Buffer-Based approach (BBA) [77] select the bitrate as a function of the current buffer usage. 2) Rate-Based approach (RBA) chooses the maximum available bitrate less than the predicted bandwidth. 3) Robust-MPC approach [73] select the bitrate by considering buffer occupancy and bandwidth. The looking ahead horizon of Robust-MPC is three-time slots.

RB, Robust-MPC, and our proposed approach apply the harmonic mean of the average bandwidth of the last 5 video chunks to predict the bandwidth for the next video chunk.

Table 4.1: The average performances per session.

	RBA	BBA	Robust-MPC	CoI
Average Rebuffering Time (s)	0.3617	0.9439	0.7661	0.9173
Std. Dev. of Rebuffering Time (s)	0.0717	1.9731	1.4079	1.2803
Average Bitrate(kbps)	1762.3	1996.6	2014.5	2231.8
Std. Dev. of Average Bitrate(kbps)	617.1	517.7	538.5	452.4
Bitrate Variation (kbps/chunk)	76.3598	176.5488	115.5366	124.5122
Std. Dev. of Bitrate Variation (kbps/chunk)	39.5099	133.6111	74.3199	91.2549

4.2.3.2 Performances on Bitrate Variations, Average Bitrate, and Rebuffering Time,

We first evaluate the performances of different methods on rebuffering time, bitrate variation, and video quality. We run the tests over 40 video sessions, and each video session has 200 video chunks. For each video session, we randomly choose a bandwidth trace and the interestingness information of a video file. The performance of each method is illustrated in Table 4.1. From the results in Table 4.1, we can observe that the performances of our proposed CoI method on rebuffering time, average bitrate and quality variations are close to the performances of the state-of-the-art methods, including Robust-MPC, BBA, and RBA. This verifies that introducing video interestingness information for rate adaptation will not deteriorate the performances from the perspective of objective QoE metrics. Moreover, CoI reaches the highest mean average bitrate out of all the methods and the lowest standard deviation of it, for average rebuffering time, the CoI method achieves a better performance than BBA and similar performance to the Robust-MPC, for the bitrate variation, CoI method is better than the BBA and similar to the Robust-MPC.

We also give the empirical distributions of quality variations, average bitrate, and rebuffering time of different methods in Figure. 4.6. We can observe that the CoI method has the highest distributions on bitrate comparing with the rest methods. For the distributions of rebuffering time and quality variations, the CoI method both gets quite good results though not the lowest since there is a trade-off between minimizing the rebuffering time, quality variations and maximizing the video interestingness value.

Figure 4.6: These results show the CDF, i.e., Empirical Cumulative Distribution Function for three major evaluation metrics. The results show that CoI tends to allocate higher bitrate for each video chunk, maintains a relatively low rebuffering time even under higher bitrate selection comparing to other methods, and control the bitrate variation to a comparative level with other methods.

4.2.4 Implementations with Baconian

The evaluations of the proposed method and baseline method also benefit from the Baconian. It can be summarized as two-fold advantages, first, we use a customized environment with Baconian to build up a video streaming testbed, which can be usually used by multiple control algorithms. We also directly utilize many of the build-in algorithms including DQN, MPC to conduct the performance comparisons.

4.2.4.1 Video Streaming Simulator

To build the video streaming simulator, we first adopt Mahimahi[78] as the network emulator with an external network bandwidth dataset, then based on that, we apply our state, action, and reward designs and wrap the emulator into a standard Baconian environment object following MDP. It is then used as a testbed for our proposed DQN based adaptive bitrate control method, as well as other baseline methods.

With such a modular design, the environment can be easily equipped with different bitrate control methods without incurring extra coding efforts.

4.2.4.2 Built-in Algorithms of Baconian

The evaluation also benefits from the built-in algorithms, including DQN, and MPC. They are directly applied to solving the adaptive bitrate task. For the rest, we can easily implement the control policy with minimal lines of code.

4.2.4.3 Customizing Logging for Result Analysis

To analysis the performance with various metrics including the average bitrate, rebuffering time, and bitrate variations, we implement the customized logging process with minimal efforts as we following the tutorial in Section. 3.3.2.3.

4.3 Conclusion

In this chapter, we give two applications of the Baconian on real-world research topics, Dyna-style MBRL hyper-parameters auto-scheduling, and online adaptive bitrate controlling. Various features benefit the research of these problems which demonstrate the usability of the Baconian project. As two showcases, they provide examples for researchers that they can conduct similar research with the help of Baconian to speed up the progress.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this report, we present a unified, modularized, and flexible model-based reinforcement learning library. It aims to provide the researchers with an easy-to-use library for researching MBRL with minimal effort on tedious coding tasks. The library offers high-quality implementations of many state-of-art MBRL algorithms with various experiment utilities and features included. Some specially designed modules successfully address the challenges brought by the specialties of MBRL, including complicated training flow, various types of dynamic model, etc.

5.2 Future Work

Currently, Baconian provides the most essential modules as an open-sourced RL algorithm library. There are still more advanced features and functionalities to be developed and we summarize as follows.

1. To build up benchmark results for built-in MRBL algorithms on various open tasks including Open AI Gym, Mujoco, etc. Releasing the benchmark results and related well-tuned hyper-parameters can further demonstrate the reliability of the algorithms' implementations and their performance. With well-tuned parameters, researchers can easily migrate the settings to different tasks and reduce the tedious parameter tuning process.

2. To include more state-of-the-art MBRL algorithms including GPS[36], PILCO [38]. MBBRL is becoming an active research topic in the ML community recently, so including these most recent research outcomes into the library will help the users to quickly catch up with the research of MBRL
3. To integrate auto hyper-parameters tuning and searching functions. Hyper-parameters searching has always been a tedious and repeated work for ML research. Even by integrating most straight-forward parameter searching methods like grid search, Baconian will be able to save the workload of users.
4. To integrate Baconian with distributed training framework like [60]. Training with distributed system becomes a common practice to speed up the training and tuning of the ML model. By equipping the Baconian with a low-level distributed training engine, the users will be able to scale the experiments to all available computing resources thus boost the research progress greatly.

References

- [1] G. Gao, L. Dong, H. Zhang, Y. Wen, and W. Zeng, *Content-aware personalised rate adaptation for adaptive streaming via deep video analysis*, 2019. DOI: 10.1109/ICC.2019.8761156. [Online]. Available: <https://doi.org/10.1109/ICC.2019.8761156>.
- [2] L. Dong, Y. Li, X. Zhou, Y. Wen, and K. Guan, *Intelligent trainer for dynamic style model-based deep reinforcement learning*, 2020. DOI: 10.1109/TNNLS.2020.3008249.
- [3] Q. Fu, L. Dong, Z. Liu, Y. Luo, Y. Wen, Y. Li, and L.-Y. Duan, *Incorporating category taxonomy in deep reinforcement learning based image hashing*, IEEE, 2019.
- [4] H. Zhang, L. Dong, G. Gao, H. Hu, Y. Wen, and K. Guan, *Deepqoe: A multimodal learning framework for video quality of experience (qoe) prediction*, 2020.
- [5] L. Dong, G. Gao, X. Zhang, L. Chen, and Y. Wen, *Baconian: A unified opensource framework for model-based reinforcement learning*, 2019. arXiv: 1904.10762. [Online]. Available: <http://arxiv.org/abs/1904.10762>.
- [6] 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, pp. 354–359, 2017.
- [7] 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, pp. 529–533, 2015.
- [8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, p. 484, 2016.
- [9] 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.
- [10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

REFERENCES

- [11] S. Levine and V. Koltun, “Learning complex neural network policies with trajectory optimization,” in *International Conference on Machine Learning*, 2014, pp. 829–837.
- [12] F. Guenter, M. Hersch, S. Calinon, and A. Billard, “Reinforcement learning for imitating constrained reaching movements,” *Advanced Robotics*, vol. 21, no. 13, pp. 1521–1544, 2007.
- [13] P. Abbeel, M. Quigley, and A. Y. Ng, “Using inaccurate models in reinforcement learning,” in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 1–8.
- [14] T. Hester, M. Quinlan, and P. Stone, “Generalized model learning for reinforcement learning on a humanoid robot,” in *2010 IEEE International Conference on Robotics and Automation*, IEEE, 2010, pp. 2369–2374.
- [15] J. van Rooijen, I. Grondman, and R. Babuška, “Learning rate free reinforcement learning for real-time motion control using a value-gradient based policy,” *Mechatronics*, vol. 24, no. 8, pp. 966–974, 2014.
- [16] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel, “Combining model-based policy search with online model learning for control of physical humanoids,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 242–248.
- [17] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, “Model-ensemble trust-region policy optimization,” *arXiv preprint arXiv:1802.10592*, 2018.
- [18] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, “Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 7559–7566.
- [19] A. Kupcsik, M. P. Deisenroth, J. Peters, A. P. Loh, P. Vadakkepat, and G. Neumann, “Model-based contextual policy search for data-efficient generalization of robot skills,” *Artificial Intelligence*, vol. 247, pp. 415–439, 2017.
- [20] J. Strahl, T. Honkela, and P. Wagner, “A gaussian process reinforcement learning algorithm with adaptability and minimal tuning requirements,” in *International Conference on Artificial Neural Networks*, Springer, 2014, pp. 371–378.
- [21] J. Boedecker, J. T. Springenberg, J. Wülfing, and M. Riedmiller, “Approximate real-time optimal control based on sparse gaussian process models,” in *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (AD-PRL)*, IEEE, 2014, pp. 1–8.
- [22] J. G. Schneider, “Exploiting model uncertainty estimates for safe dynamic control learning,” in *Advances in neural information processing systems*, 1997, pp. 1047–1053.

REFERENCES

- [23] A. Yamaguchi and C. G. Atkeson, “Neural networks and differential dynamic programming for reinforcement learning problems,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 5434–5441.
- [24] A. S. Polydoros and L. Nalpantidis, “A reservoir computing approach for learning forward dynamics of industrial manipulators,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 612–618.
- [25] R. S. Sutton, “Dyna, an integrated architecture for learning, planning, and reacting,” *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.
- [26] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, 2015, pp. 1889–1897.
- [27] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma, “Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees,” *arXiv preprint arXiv:1807.03858*, 2018.
- [28] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer Science & Business Media, 2013.
- [29] A. V. Rao, “A survey of numerical methods for optimal control,” *Advances in the Astronautical Sciences*, vol. 135, no. 1, pp. 497–528, 2009.
- [30] A. G. Richards, “Robust constrained model predictive control,” PhD thesis, Massachusetts Institute of Technology, 2005.
- [31] S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.
- [32] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, “The explicit linear quadratic regulator for constrained systems,” *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.
- [33] Y. Tassa, T. Erez, and E. Todorov, “Synthesis and stabilization of complex behaviors through online trajectory optimization,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 4906–4913.
- [34] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, “Deep spatial autoencoders for visuomotor learning,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 512–519.
- [35] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel, “Learning deep neural network policies with continuous memory states,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 520–527.
- [36] W. H. Montgomery and S. Levine, “Guided policy search via approximate mirror descent,” in *Advances in Neural Information Processing Systems*, 2016, pp. 4008–4016.

REFERENCES

- [37] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.
- [38] M. Deisenroth and C. E. Rasmussen, “Pilco: A model-based and data-efficient approach to policy search,” in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.
- [39] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
- [40] 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*, IEEE, 2012, pp. 5026–5033.
- [41] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, “Gotta learn fast: A new benchmark for generalization in rl,” *arXiv preprint arXiv:1804.03720*, 2018.
- [42] N. Yoshida, “Gym-torcs,” *Software available at <https://github.com/ugonama-kun/gym-torcs>*, 2016.
- [43] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. Lillicrap, and M. Riedmiller, “DeepMind control suite,” DeepMind, Tech. Rep., Jan. 2018. [Online]. Available: <https://arxiv.org/abs/1801.00690>.
- [44] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei, “Surreal: Open-source reinforcement learning framework and robot manipulation benchmark,” in *Conference on Robot Learning*, 2018.
- [45] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, <http://pybullet.org>, 2016–2019.
- [46] S. James, Z. Ma, D. Rovick Arrojo, and A. J. Davison, “Rlbench: The robot learning benchmark & learning environment,” *arXiv preprint arXiv:1909.12271*, 2019.
- [47] G. Synnaeve, N. Nardelli, A. Auvolet, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier, “Torchcraft: A library for machine learning research on real-time strategy games,” *arXiv preprint arXiv:1611.00625*, 2016.
- [48] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv preprint arXiv:1912.06680*, 2019.
- [49] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, *et al.*, “Alphastar: Mastering the real-time strategy game starcraft ii,” *DeepMind blog*, p. 2, 2019.
- [50] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, *Openai baselines*, <https://github.com/openai/baselines>, 2017.

REFERENCES

- [51] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare, “Dopamine: A Research Framework for Deep Reinforcement Learning,” 2018. [Online]. Available: <http://arxiv.org/abs/1812.06110>.
- [52] S. Kolesnikov, *Accelerated dl rl*. <https://github.com/catalyst-team/catalyst>, 2018.
- [53] J. Gauci, E. Conti, Y. Liang, K. Virochsiri, Z. Chen, Y. He, Z. Kaden, V. Narayanan, and X. Ye, “Horizon: Facebook’s open source applied reinforcement learning platform,” *arXiv preprint arXiv:1811.00260*, 2018.
- [54] Y. Duan, X. Chen, R. Houthoofd, J. Schulman, and P. Abbeel, “Benchmarking deep reinforcement learning for continuous control,” in *International Conference on Machine Learning*, 2016, pp. 1329–1338.
- [55] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [56] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, *Stable baselines*, <https://github.com/hill-a/stable-baselines>, 2018.
- [57] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, “Reproducibility of benchmarked deep reinforcement learning tasks for continuous control,” *arXiv preprint arXiv:1708.04133*, 2017.
- [58] C. Finn, M. Zhang, J. Fu, X. Tan, Z. McCarthy, E. Scharff, and S. Levine, “Guided policy search code implementation,” *Software available from rll.berkeley.edu/gps*, 2016.
- [59] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, “Benchmarking model-based reinforcement learning,” *CoRR*, vol. abs/1907.02057, 2019. arXiv: 1907.02057. [Online]. Available: <http://arxiv.org/abs/1907.02057>.
- [60] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, “Ray: A distributed framework for emerging AI applications,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA: USENIX Association, Oct. 2018, pp. 561–577, ISBN: 978-1-939133-08-3. [Online]. Available: <https://www.usenix.org/conference/osdi18/presentation/moritz>.
- [61] A. Sergeev and M. Del Balso, “Horovod: Fast and easy distributed deep learning in tensorflow,” *arXiv preprint arXiv:1802.05799*, 2018.
- [62] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, “Rllib: Abstractions for distributed reinforcement learning,” *arXiv preprint arXiv:1712.09381*, 2017.

REFERENCES

- [63] I. Hosu and T. Rebedea, “Playing atari games with deep reinforcement learning and human checkpoint replay,” *CoRR*, vol. abs/1607.05077, 2016. arXiv: 1607.05077. [Online]. Available: <http://arxiv.org/abs/1607.05077>.
- [64] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. [Online]. Available: <http://arxiv.org/abs/1509.02971>.
- [65] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, vol. abs/1707.06347, 2017. arXiv: 1707.06347. [Online]. Available: <http://arxiv.org/abs/1707.06347>.
- [66] O. Klimov and J. Schulman, *Roboschool*, <https://github.com/openai/roboschool>, 2017.
- [67] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283, ISBN: 978-1-931971-33-1. [Online]. Available: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.
- [68] <https://github.com/berkeleydeeprlcourse/homework/tree/master/hw4>, Accessed: 2018-05-06.
- [69] S. M. Kosslyn and S. M. Kosslyn, *Image and brain: The resolution of the imagery debate*. MIT press, 1996.
- [70] G. Gao and Y. Wen, “Morph: A fast and scalable cloud transcoding system,” in *Proceedings of the 24th ACM international conference on Multimedia*, 2016, pp. 1160–1163.
- [71] T. Stockhammer, “Dynamic adaptive streaming over HTTP: Standards and design principles,” in *Proceedings of the 2nd annual ACM conference on Multimedia systems*, ACM, 2011.
- [72] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning spatiotemporal features with 3d convolutional networks,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.
- [73] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, “A control-theoretic approach for dynamic adaptive video streaming over HTTP,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, London, United Kingdom: ACM, 2015, pp. 325–338.

REFERENCES

- [74] Y. Song, J. Vallmitjana, A. Stent, and A. Jaimes, “Tvsum: Summarizing web videos using titles,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5179–5187.
- [75] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, “Commute path bandwidth traces from 3G networks: Analysis and applications,” in *Proceedings of the 4th ACM Multimedia Systems Conference*, ACM, 2013, pp. 114–118.
- [76] *Raw Data - Measuring Broadband America 2016*, <https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016>, 2016.
- [77] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, “A buffer-based approach to rate adaptation: Evidence from a large video streaming service,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, 2015.
- [78] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, “Mahimahi: Accurate record-and-replay for HTTP,” in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, Santa Clara, CA: USENIX Association, Jul. 2015, pp. 417–429, ISBN: 978-1-931971-225. [Online]. Available: <https://www.usenix.org/conference/atc15/technical-session/presentation/netravali>.