REINFORCEMENT LEARNING METHODS FOR BIPEDAL ROBOT WALKING IN PARTIALLY OBSERVED ENVIRONMENT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

UĞURCAN ÖZALP

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
SCIENTIFIC COMPUTING

JUNE 2021

Approval of the thesis:

REINFORCEMENT LEARNING METHODS FOR BIPEDAL ROBOT WALKING IN PARTIALLY OBSERVED ENVIRONMENT

submitted by **UĞURCAN ÖZALP** in partial fulfillment of the requirements for the degree of **Master of Science in Scientific Computing Department, Middle East Technical University** by,

Prof. Dr. Bülent Karasözen
Director, Graduate School of **Applied Mathematics** ______________

Prof. Dr. Ömür Uğur
Head of Department, **Scientific Computing** ______________

Prof. Dr. Ömür Uğur
Supervisor, **Scientific Computing, METU** ______________

Examining Committee Members:

Prof. Dr. I am the Chair
Computer Engineering Department, METU ______________

Prof. Dr. I am the Supervisor
Computer Engineering Department, METU ______________

Assoc. Prof. Dr. I may be Co-Supervisor
Computer Engineering Department, METU ______________

Assoc. Prof. Dr. A Member with High Title
Computer Engineering Department, METU ______________

Assist. Prof. Dr. A Member with Low Title
Computer Engineering Department, Hacettepe University ______________

Date: ______________

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: UĞURCAN ÖZALP

Signature :

ABSTRACT

REINFORCEMENT LEARNING METHODS FOR BIPEDAL ROBOT WALKING IN PARTIALLY OBSERVED ENVIRONMENT

ÖZALP, UĞURCAN
M.S., Department of Scientific Computing
Supervisor : Prof. Dr. Ömür Uğur

June 2021, 33 pages

Deep Reinforcement Learning (DRL) methods on mechanical control are successful on many environments and used instead of traditional optimal and adaptive control methods on some complex cases. However, DRL algorithms does still have challenges. One is control on partially observable environments. When an agent is not informed well about the environment, it must recover information from past observations. In this thesis, DRL control of Bipedal Walker (OpenAI GYM) environment is studied by DRL by continious actor-critic algorithm Twin Delayed Deep Deterministic Policy Gradient (TD3). Since environment is partially observable, several neural architectures are implemented First one is feed-forward neural network under the observable environment assumption, while second and third ones are Long Short Term Memory (LSTM) and Transformer using last 16 time step observation as input to recover hidden state, because environment is assumed to be partially observable.

Keywords: deep reinforcement learning, partial observability, robot control, actor-critic methods, etc.

ÖZ

PEKİŞTİRMELİ ÖĞRENME YÖNTEMLERİYLE KISMİ GÖZLENEBİLİR ORTAMDA ÇİFT BACAKLI ROBOTUN YÜRÜTÜLMESİ

ÖZALP, UĞURCAN
Yüksek Lisans, Bilimsel Hesaplama Bölümü
Tez Yöneticisi : Prof. Dr. Ömür Uğur

Haziran 2021, 33 sayfa

google translate şimdilik -> Mekanik kontrol üzerine Derin Takviyeli Öğrenme (DRL) yöntemleri birçok ortamda başarılıdır ve bazı karmaşık durumlarda geleneksel optimal ve uyarlanabilir kontrol yöntemleri yerine kullanılır. Ancak, DRL algoritmalarının hala zorlukları vardır. Birincisi, kısmen gözlemlenebilir ortamlarda kontroldür. Bir temsilci çevre hakkında yeterince bilgilendirilmediğinde, geçmiş gözlemlerden bilgileri kurtarmalıdır. Bu tezde, Bipedal Walker (OpenAI GYM) ortamının DRL kontrolü, sürekli aktör-eleştirmen algoritması Twin Delayed Deep Determinstic Policy Gradient (TD3) ile DRL tarafından incelenmiştir. Çevre kısmen gözlemlenebilir olduğundan, birkaç sinir mimarisi uygulanmaktadır Birincisi, gözlemlenebilir ortam varsayımı altında ileri beslemeli sinir ağı iken, ikincisi ve üçüncüsü, kurtarmak için girdi olarak son 16 zaman adımı gözlemini kullanan Transformer ve Uzun Kısa Süreli Bellek (LSTM) gizli durum, çünkü çevrenin kısmen gözlemlenebilir olduğu varsayılır.

Anahtar Kelimeler: pekiştirmeli derin öğrenme, kısmi gözlemlenebilirlik, robot kontrolü, aktör-eleştirmen metodları, vs.

ACKNOWLEDGMENTS

I would like to express my very great appreciation to my thesis supervisor Assoc. Prof. Dr. Ömür Uğur for his patient guidance, enthusiastic encouragement and valuable advices during the development and preparation of this thesis. His willingness to give his time and to share his experiences has brightened my path.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABBRV	Abbreviation

CHAPTER 1

INTRODUCTION

Artificial intelligence (AI) is the ability of a computer program or a machine to think and learn like natural intelligence performed by humans and animals. One way is to create an intellgent agent is using some methods to detect patterns on data and use it to make predictions on unseen data. This approach is called Machine Learning.

Humans and animals exhibit several different behaviours in terms of interaction with environment, such as utterance, movement. Their behavior is based on past experience, the situation they are in and their objective. Like humans and animals, an intelligent agent is expected to take action according to based on its perception for some objective. A major challenge to machine learning is creating agents that will act more natural and humanlike. As a subfield of Machine Learning, Reinforcement Learning allows an agent to learn how to control itself (act) in different situations. It models environment to give reward or punishment to agent according to environmental state and agent actions, and focuses on learning to predict what actions will lead to highest reward (or lowest punishment) for the future using past experience.

Traditional RL algorithms need feature engineering from observation. For complex problems, the way to extract features is ambiguous or observations are not enough to create a good model. As a newer technique, deep neural networks (DNNs) allows to extract high level features from data which has huge state-space (like visual observation) and missing observation. Along with recent developments in DNNs, Deep Reinforcement Learning (DRL) algorithms allows an agent to interact with environment in more complex way. The problem with deep learning is selection of a correct neural network.

Since its discovery, robots have been crucial devices for the human race, whether smart or not. Intelligent humanoid and animaloid robots have been in continuous development since early 1980s. This type of robots has legs unlike driving robots. Since most of world terrain is unpaved, this type of robots are good alternative to driving robots. Locomotion is major task for them. Stable bipedal (2-leg) walking is one of the most challenging problem among the control problems. It is hard to create accurate model due to high order of dynamics, friction and discontinuities. Even so, design of walking controller using traditional methods is difficult due to same reasons. For bipedal walking, Deep Reinforcement Learning (DRL) approach is an easier choice.

In this thesis, Bipedal Locomotion Deep Reinforcement Learning (DRL) by is investigated through *Bipedal-Walker-v3* [1] and *Bipedal-Walker-Hardcore-v3* [2] environment of open source GYM library [4]. Several neural architectures are used and results are compared.

1.1 Problem Statement: Bipedal Walker Robot Control

1.1.1 OpenAI Gym and Bipedal-Walker Environment

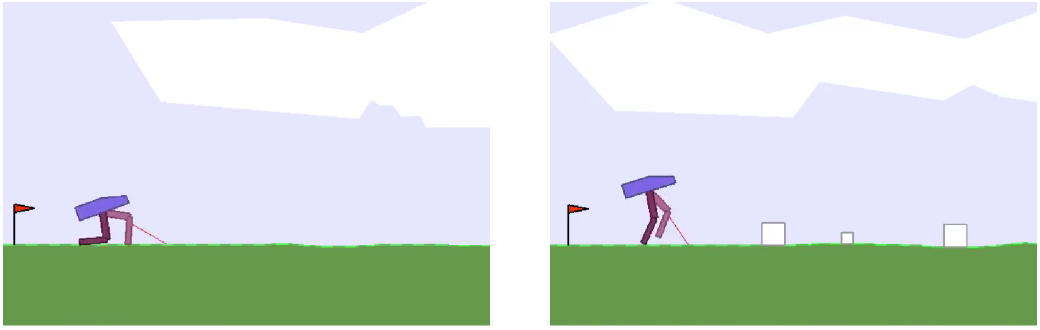
OpenAI Gym [4] is opensource framework, containing many environments to service reinforcement learning algorithms.

Bipedal-Walker environments [1] [2] is part of Gym environment library. One of them is classical version where the terrain is relatively smooth, while other one is hardcore version which contains ladders, stumps and pitfalls in terrain.

For both settings, the task is to move forward the robot as much as possible. It has continious action and observation space.

Locomotion of the Bipedal Walker is difficult control problem due to following reasons.

- **Nonlinearity**: The dynamics is nonlinear, unstable and multimodal. Dynamical behavior of robot changes for different situations like ground contact, single leg



(a) Bipedal-Walker-v3 Snapshot (b) Bipedal-Walker-Hardcore-v3 Snapshot

Figure 1.1: Bipedal Walkers Snapshots

contact and double leg concact.

- **Uncertainity**: The terrain where robot walks may vary. Designing a controller for all types of terrain is difficult.s

- **Partially Observability**: The robot observes ahead of it with lidar measurements and cannot observe behind.

These difficulties make hard to implement analytical methods for control task. RL approach is better to tackle first 2 one. For partial observability problem, more elegant solution is required. This is done by creating a belief state from past observations to inform agent. Agent uses its belief state and observations to choose how to act. However, relying on observations is also possible.

1.1.2 Deep Learning Library: PyTorch

PyTorch is an open source library developed byFacebook's AI Research lab (FAIR) [16]. It is based on Torch library [5] and has Python and C++ interface. It is an automatic differentation library with accelerated math operations backed by graphical processing units (GPUs). This is what a deep learning library requires. And the clean pythonic syntax made it most famous deep learning tool among researches.

1.2 Proposed Methods and Contribution

Partially observable environments are always a hardwork for reinforcement learning algorithms. In this work, the walker environment assumed as fully observable environment at first. A feed-forward neural network architecture is proposed to control the robot. Then, the environment is assumed to be partially observable. In order to recover latent states, Long Short Term Memory (LSTM) and Transformer neural networks are proposed.

LSTMs are used in many deep learning applications which includes sequential data. It is a variant of Recurrent Neural Networks (RNN). It is a good candidate for RL algorithms which solves partially observable environments.

Transformers are developed to handle sequential data as RNN models does. However, it processes the whole sequence at same time, while RNNs process the sequence in order. It replaced RNNs in Natural Language Processing (NLP) tasks over RNNs due to its major improvements. However, this is not the case for Reinforcement Learning, yet.

1.3 Related Work

Reinforcement Learning methods are used in many mechanical control tasks such as autonomus driving [14] [23] [21] [28] and autonomus flight [10] [3] [22].

Rastogi [17] used Deep Deterministic Policy Gradient (DDPG) algorithm to walk their physical bipedal walker robot along with simulation environment. They concluded that DDPG is infeasible to control walker robot since it requires long time for convergence. Kumar et al. [11] also used DDPG to perform robot walking in 2D simulation environment. Their agent converged in approximately 25k episodes. Song et al. [24] pointed out the partial observability problem of bipedal walker, using Recurrent Deep Deterministic Policy Gradient (RDPG) [9] algorithm and acquired better results than original Deep Deterministic Policy Gradient (DDPG) algorithm.

Fris [7] used Twin Delayed DDPG (TD3) using Long Short Term Memory (LSTM) for their quadrocopter landing task. Fu et al. [8] used vanilla RNN with attention mechanism using TD3 for car driving task, but not explicit Transformer. They reported that their method outperformed 7 baselines. Upadhyay et al. [26] used all of feed forward network, LSTM an original Transformer architectures for balancing pole on a cart from Cartpole environment of Gym, and Transformer yield worst results among three architectures.

1.4 Outline of the Thesis

Here, outline the thesis structure.

CHAPTER 2

REINFORCEMENT LEARNING

Machine Learning is ability of computer program which allows adaptation to new situations through experience, without explicitly programmed [13].

Supervised Learning: It is task of learning a function $f\colon X \mapsto Y$ that maps an input to an output based on N example input-output pairs (x_i, y_i) such that $f(x_i) \approx y_i, \forall i \in 1, 2, ..., N$ [20].

Input x can be thought as state of an agent. That makes y correct action at state x. For supervised learning, both x and y should be available, where the correct action are provided by a friendly supervisor [].

Unsupervised Learning: Unsupervised learning is discovering structure on input examples without any label on it. Based on N example input pairs (x_i), it is discovering function $f\colon X \mapsto Y$ $f(x_i) = y_i, \forall i \in 1, 2, ..., N$, where y_i is disovered output.

Again, input x can be thought as state of an agent. However, correct action is not available and there is no given hint in this case. It can learn relations among states but it does not know what to do since there is no target or utility [20].

Reinforcement Learning: Reinforcement Learning one of three main machine learning paradigm along with Supervised and Unsupervised Learning. It is closest kind of learning demonstrated by humans and animals since it is grounded by biological learning systems. It is based on maximizing cumulative reward over time to make agent learn how to act in an environment [25]. Each action of agent is either rewarded or punished according to reward criteria. Therefore, reward function is mathematical representation of what to teach agent. The agent explores environment by taking various actions in different states to get experience, based on trial-and-error. Then it exploits experiences to get highest reward from environment considering instant and future rewards over time.

Formally, Reinforcement Learning is learning a policy function $\pi\colon S \mapsto A$ which maps inputs (states) $s \in S$ to outputs (actions) $a \in A$. Learning is done by maximizing value function $V^{\pi}(s)$ (cumulative reward) for all possible states, which depends on policy π.

Reinforcement Learning is different than supervised learning because correct actions are not provided. Meanwhile, it is also different than unsupervised learning because the agent is forced to learn a behaviour. The agent is evaluated at each time step without supervision.

2.1 Reinforcement Learning and Optimal Control

Optimal control is a field of mathematical optimization, concerned by finding control policy of a dynamical system (environment) for given objective. For example, objective might be total reveune for a company as system, minimal fuel burn for a car as system, or total production for a factory.

Reinforcement Learning is kind of naive subfield of Optimal Control. However, RL algorithms find policy (controller) by error minimization of objective from experience, while Optimal Control methods are concerned exact analytical optimal solutions based on dynamic model of environment and agent.

Optimal Control methods are efficient and robust when mathematical model of environment is available, accurate enough and solvable for optimal controller. However, some real world problems usually do not exhibit all of these conditions. Reinforcement Learning is an easier way to derive a control policy.

2.2 Challenges

The Reinforcement Learning Environment poses a variety of obstacles that we need to address and potentially make trade-offs among them [6] [25].

Exploration Explotation Dilemma: A RL agent is supposed to maximize rewards (knowledge exploration) by observing the environment (exploration of environment). This gives rise to the exploration-exploitation dilemma that is the inevitable trade-off between them. Exploration is taking a range of acts to benefit about the consequences. Typically results in low immediate rewards and high rewards for the future. Explotation is taking action that has been learned. Typically results in high immediate rewards and low rewards in the future.

Generalization and Curse of Dimensionality: A RL agent should also be able to generalize experiences to act on unseen situation before. This issue arises when state space and action space is high dimensional since experiencing all possibilities is impractical. This is solved by introducing function approximator. Deep Reinforcement Learning uses neural network as function approximator.

Delayed Consequences: A RL agent should be aware reason of reward or punishment. Once it gets reward or punishment, it should be able to discriminate whether reward is caused by instant actions or past actions.

Partial Observability: Partial observability is absence of all required observation to infer instant state. For instance, a driver does not know engine temperature or rotational speed of gears. Although driver is able to drive in that case, s/he would not be able to drive well on traffic in absence of rear view mirror or side mirror. In real world, most of systems are partially observable. This problem is usually tackled by incorprating observation history from agents memory in acting.

Safety of Agent: Mechanical agents can kill or degrade themselves and their surroundings. This safety problem is important on both exploration stage and full operation. Simulation of environment is a good way to train agent with safety but causes incomplete learning due to inaccuracy compared to real environment.

2.3 Sequential Decision Making

In discrete time setting, the agent takes action a_t, then observes observation o_t and obtains reward r_t at time t. History is set of past actions observations and rewards, $h_t = \{a_0, o_0, r_0, ...a_t, o_t, r_t\}$. State s_t is function of the history, $s_t = f(h_t)$, which represents situtaion of environment as much as possible.

2.4 Markov Decision Process

Markov Decision Process (MDP) is a sequential decision making process with Markov property. It is represented as a tuple (S, A, P, R, γ). Markov property means that the conditional probability distribution of the future state depends only on the instant state and action instead of the entire past, so it is memoryless.

State Space S: A set of all possible configurations of system.

Action Space A: A set of all possible actions of agent.

Model P: Model is mathematical representation of how environment evolves through time, including transition probabilities $P(s'|s, a)$ where $s' \in S$ is next state, $s \in S$ is instant state and $a \in A$ is taken action.

Reward R: A set of rewards coming from environment. At each state transition $s_t \mapsto s_{t+1}$, a reward r_t is given to agent. Rewards can be either deterministic or stochastic.

2.4.1 Reward Function

Reward function R is expected value of reward at given state s and taken action a. Therefore, it is defined as function of state and action, $R\colon S \times A \mapsto R$.

$$R(s, a) = \mathbb{E}[r_t|s_t = s, a_t = a] \; \forall t = 0, 1, ... \tag{2.1}$$

2.4.2 Policy

Policy is a mapping $\pi\colon S \mapsto A$ which maps states to actions.

2.4.3 Optimization Goals

Return and Discount Factor: Discount factor $\gamma \in [0, 1)$ is measure of importance of rewards in the future for value function. This value is used in for return calculation. At time t, G_t is return which is cumulative sum of future rewards, scaled by γ.

$$G_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i = r_t + +\gamma G_{t+1} \tag{2.2}$$

Since return depends on future rewards, it also depends on policy of agent since policy affects future rewards.

State Value Function: State Value Function V^π is expected return when policy π is followed in future.

$$V^\pi(s) = \mathbb{E}[G_t | s_t = s] \; \forall t = 0, 1, ... \tag{2.3}$$

State-Action Value Function: State-Action Value Function Q^π is expected return when policy π is followed in future, but any action taken at instant step.

$$Q^\pi(s, a) = \mathbb{E}[G_t | s_t = s, a_t = a] \; \forall t = 0, 1, ... \tag{2.4}$$

2.5 Model Free Reinforcement Learning

Model Free Reinforcement Learning is suitable if environment model is not available but agent can experience environment by consequences of its actions.

2.5.1 Q Learning

2.5.1.1 Deep Q Learning

2.5.1.2 Double Deep Q Learning

2.5.2 Policy Gradient

2.5.2.1 Deterministic Policy Gradient

2.5.2.2 Deep Deterministic Policy Gradient

2.5.2.3 Twin Delayed Deep Deterministic Policy Gradient

CHAPTER 3

NEURAL NETWORKS AND DEEP LEARNING

This chapter of thesis is a brief summary of useful machine and deep learning algortihms used in the work, for ones who does not have enough knowledge about the topic.

3.1 Neural Networks

Despite of tons of variants, a neural network is defined as parametrized function approximator which is inspried by biological neurons. The first models of neural network developed by a neurophysiologist Warren McCulloch and a mathematician Walter Pitts in 1943 [12]. However, the idea of neural network known today arised after development of a simple binary classifier called perceptron invented by Rosenblast et al. [18]. It is a learning framework inspired by human brain. Although there are many types of neural networks, they are all based on linear transformations and nonlinear activations.

Neural networks can approximate any nonlinear function if designed as complex as required. Parameters are updated by backpropagation algorithm to minimize loss between its outputs and desired outputs.

3.2 Backpropagaion

Some optimization theory

3.3 Neural Network Types

3.3.1 Perceptron

Perceptron is a binary classifier model. In order to allocate input x into a class, feature vector $\phi(x) \in \mathbb{R}^{1\times d_k}$ is generated by a fixed nonlinear function. Then, a linear model is generated with linear transformation weights $W \in \mathbb{R}^{d_k\times 1}$ in the following form 3.1.

$$y = f(\phi(x)W) \tag{3.1}$$

where f is called activation function. For perceptron, it is defined as step function 3.2 while other functions like sigmoid, tanh can also be defined.

$$f(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

A learning algorithm of a perceptron aims determining the parameter vector W. It is best motivated by error minimization of data samples once a cost function is constructed.

3.3.2 Feed Forward Neural Networks (Multilayer Perceptron)

Structure of perceptron make a way for feed forward neural layers. Unlike stated below, a neural layer might output multiple values (say $o \in \mathbb{R}^{1\times d_o}$) as vector from input (say $x\mathbb{R}^{1\times d_x}$). Such a setting forces parameter $W \in \mathbb{R}^{d_x\times d_o}$ to be a matrix. Moreover, activation function is not necessariliy step function. It can be any nonlinear function like sigmoid, tanh, relu etc. Feed Forward Neural Networks are generalization of perceptron algorithm to approximate any function f^*. Neural layers are stacked to construct deep feed forward neural network. It defines a nonlinear mapping $y = f(x; \theta)$ between input x and output y, parametrized by parameters $\theta = \{W\}_n, n = 1, \ldots, N$..

Assuming input signal is x (output of previous layer), activation value of the layer (h) is evaluated as by linear transformation followed by nonlinear activation f 3.3 applied elementwise.

$$net = xW + b \text{ and } h = f(net) \tag{3.3}$$

3.3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [19] are type of neural network to process sequential data. It is specialized for data having sequential topology. It is used commonly used for sequence based applications.

Sequential data can be inferred by Recurrent Neural Networks. In Feed Forward Layers, output only depends on its input, while Recurrent Layer output is dependent on both input at time t and its output in previous time step $t - 1$.

RNN can be thought as multiple copies of same network which passes message to its successor through time. A RNN layer is similar to MLP layer 3.3, except input is concatenation of output feedback and input itself 3.4.

Given input sequence $x \in \mathbb{R}^{T \times d_x}$, output sequence $h \in \mathbb{R}^{T \times d_h}$ is evaluated recursively. Initial output h_0 can be either parametrized or assigned as zeros vector. Again, nonlinear activation f 3.3 applied elementwise.

$$net_t = h_{t-1}\tilde{W} + x_t W + b \text{ and } h_t = f(net_t) \tag{3.4}$$

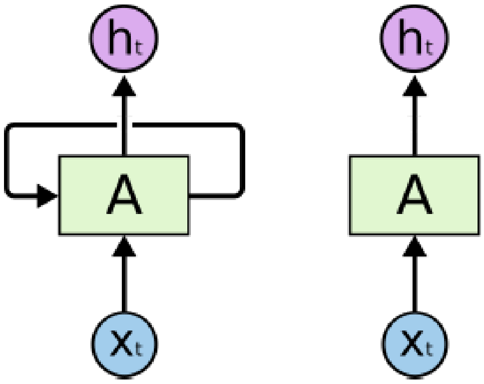


Figure 3.1: Recurrent Layer (left) and Feed Forward Layer (right) illustration.

3.3.3.1 Long Term Dependence Problem of Vanilla RNNs

Conventional RNNs have problem with vanishing/exploding gradient problem. This causes long term dependence problem. For example, given word sequence, bold words in the following sentences are hard to predict with Vanilla RNN.

The clouds are in the **sky**.

I grew up in France... I speak fluent **French**.

In order to overcome this problem new architectures are developed such as Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU).

3.3.3.2 Long Short Term Memory

LSTM is a special type of RNN. It is explicitly designed to allow learning long-term dependencies. A single LSTM cell has 4 neural layer while vanilla RNN layer has only one neural layer. In addition to hidden state h_t , there is another state called cell state C_t. Information flow is controlled by 3 gates.

Forget Gate: Forget gate controls past memory. According to input, past memory is either kept or forgotten. Sigmoid function (σ) is used as activation function, applied elementwise.

$$f_t = \sigma([h_{t-1}; x_t]W_f + b_f) \tag{3.5}$$

Input Gate: Input gate controls contribution from input to cell state (memory). Hy-

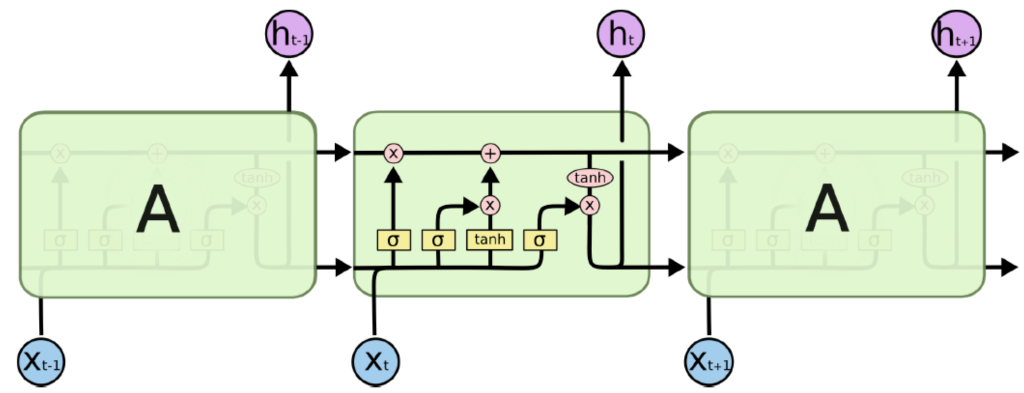


Figure 3.2: LSTM Cell.

perbolic tangent layer creates new candidate of cell state from input.

$$i_t = \sigma([h_{t-1}; x_t]W_i + b_i) \tag{3.6}$$

$$\hat{C}_t = \tanh([h_{t-1}; x_t]W_C + b_C) \tag{3.7}$$

Cell State Update: Once what are to be forget and added are decided, cell state is updated.

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t \tag{3.8}$$

Output Gate: Sigmoid layer decides what part of new cell state to be output. Cell state is filtered by tanh to push values to be in $(-1, 1)$.

$$o_t = \sigma([h_{t-1}; x_t]W_o + b_o) \tag{3.9}$$

$$h_t = o_t \odot \tanh(C_t) \tag{3.10}$$

3.3.4 Attention Mechanism

As stated earlier, recurrent neural networks are prone to forget long term dependencies. LSTM and GRU are invented to overcome this problem. Although they reduced this problem, they cannot attend specific parts of the input. For example, for sentiment analysis, specific keywords are important to determine sentiment of a sentence. However, last state of encoded input is not able to remember that words. Therefore, people came with the idea of weighted avearing all states through time where weights depends on both input and output.

Assume that input sequence $X \in \mathbb{R}^{T \times d_X}$ is encoded to $H \in \mathbb{R}^{T \times d_H}$. The context vector is calculated using weight vector $\alpha(t)$.

Calculation of weight vector depends on the task. For each time step, a score function is calculated between hidden state $H \in \mathbb{R}^{T \times d_H}$ and query $\boldsymbol{q}$ (which may be many things depending on task). Also, score function is also depends on choice. Then, attention score is $\boldsymbol{\alpha} \in \mathbb{R}^T$ is calculated using arbitrary function f depending on choice.

$$
\begin{aligned}
\boldsymbol{\alpha} =& f(\boldsymbol{q}, H) \\
\text{Attention}(q, H) =& \sum_{\tau=0}^{T} \alpha_\tau h_\tau
\end{aligned}
\tag{3.11}
$$

3.3.4.1 Transformer

The Transformer was proposed in the paper Attention is All You Need [27] . Unlike recurrent networks, this architecture is solely builded on attention layers.

A transformer layer consists of feed-forward and attention layers, which makes the mechanism special. Like RNNs, it can be used as both encoder and decoder. While encoder layers attend to itself, decoder layers attends both itself and encoded input.

Attention Layer: An attention layer is a mapping from 3 vectors called query $Q \in \mathbb{R}^{T \times d_k}$, key $K \in \mathbb{R}^{T \times d_k}$ and value $V \in \mathbb{R}^{T \times d_v}$ to output, where T is time length, d_k and d_v are embedding dimensions. Output is weighted sum of values V while weights are evaluated by compatibility metric of query Q and key K. In vanilla transformer, compatibility of query and key is evaluated by dot product, normalizing by $sqrt(d_k)$. For a query, dot product with all keys are evaluated, then softmax function is applied to get weights of values. This approach is called Scaled Dot-product Attention.

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{Q^T K}{\sqrt{d_k}})V \tag{3.12}$$

Multi-Head Attention: Instead of performing single attention; keys, queries and values are linearly projected from d_m dimensional vector space to h different spaces using projection matrices. Then, attention is done h times, and results are then concatenated and linearly projected to final values of the layer.

Projection matrices are model parameters, $W_i^Q \in \mathbb{R}^{d_m \times d_k}$, $W_i^K \in \mathbb{R}^{d_m \times d_k}$, $W_i^V \in \mathbb{R}^{d_m \times d_v}$ for $i = 1, ..., h$. Also output matrix is used to project multiple values into single one, $W^O \in \mathbb{R}^{hd_v \times d_m}$.

$$\begin{aligned} \mathrm{MHA}(Q, K, V) =& \mathrm{Concat}(a_1, a_2, ...a_h)W^O \\ a_i =& \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{3.13}$$

Feed Forward Layer: Both encoder and decoder contains feed forward layer, containing two linear transformations with ReLU activation.

$$\mathrm{FFN}(x) = \mathrm{ReLU}(xW_1 + b)W_2 + b_2 \tag{3.14}$$

Layer Normalization: Layer normalization is a layer to overcome xyz []. Given an input $x \in \mathbb{R}^K$, mean and variance statistics are evaluated along the dimensions (3.15).

$$\begin{aligned} \mu =& \frac{1}{K} \sum_{n=1}^{K} x_k \\ \sigma^2 =& \frac{1}{K} \sum_{n=1}^{K} (x_k - \mu)^2 \end{aligned} \tag{3.15}$$

Then, the input is first scaled to have zero mean and unity variance along dimensions. The term ϵ is added to prevent division by zero. Optionally, the result is scaled by elementwise multiplication by $\gamma \in \mathbb{R}^K$ and addition by $\beta \in \mathbb{R}^K$ where these are learnable parameters.

$$\mathrm{LN}(x) = \frac{x-\mu}{\sigma+\epsilon} * \gamma + \beta \tag{3.16}$$

Encoder Layer: Encoder Layer starts with a residual self attention layer. Self attention means that query, key and value are same vectors. Then it is followed by feed forward neural layer, Both sublayers are employed with resudial connection with layer normalization, i.e summation of layer input and output is passed through layer normalization.

$$\begin{aligned} att =& \mathrm{LN}(x + \mathrm{MHA}(x, x, x)) \\ y =& \mathrm{LN}(att + \mathrm{FFN}(att)) \end{aligned} \tag{3.17}$$

Decoder Layer: Similar to encoder layer, decoder layer has also self-attention and feed forward layers. In addition, there is another attention layer which is over encoder outputs. Same as encoder, all sublayers have resudial connection with layer normalization. Let's call encoded sequence $e \in \mathbb{R}^{T \times d_m}$ and decoded sequence $d \in \mathbb{R}^{T \times d_m}$ (masked). Assume that the sequence decoded up to tth point in sequence. Then, d_{t+1} is calculated as follows.

$$\begin{aligned} att =& \mathrm{LN}(d_{1:t} + \mathrm{MHA}(d_{1:t}, d_{1:t}, d_{1:t})) \\ dec =& \mathrm{LN}(att + \mathrm{MHA}(att, e, e)) \\ d_{t+1} =& \mathrm{LN}(dec + \mathrm{FFN}(dec)) \end{aligned} \tag{3.18}$$

Positional encoding: Since there are no recurrent or convolutional architecture in the model, sequential information needs to be embedded. Positional encodings has same dimension, so that input embeddings can be added to at the beginning of encoder or decoder stacks. For position pos, $2i$ or $2i+1$th dimension has following values

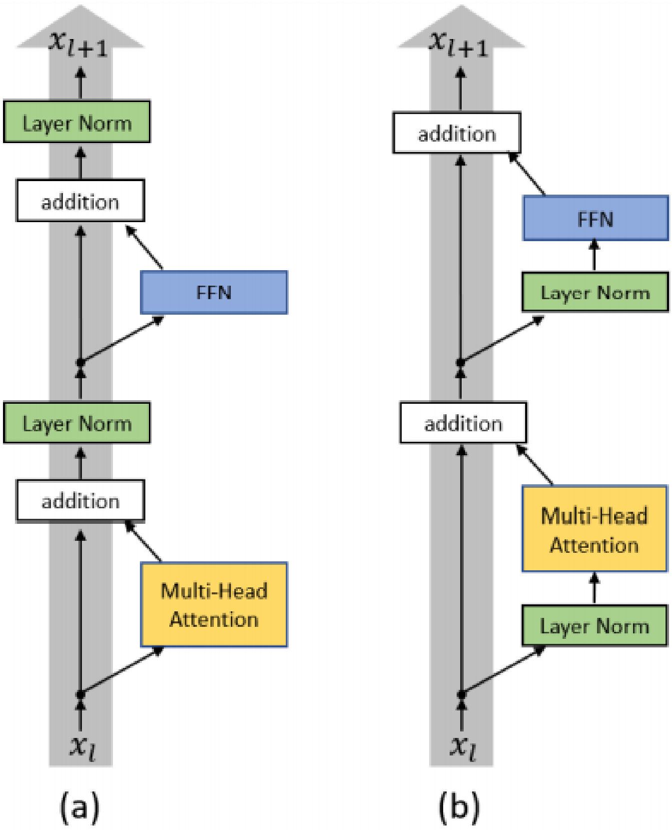


Figure 3.3: (a) Post-LN Transformer layer, (b) Pre-LN Transformer layer.

$(i \in \mathbb{N})$, as proposed in the original paper.

$$
\begin{aligned}
\mathrm{PE}_{pos,2i} &= \sin(pos/10000^{2i/d_m}) \\
\mathrm{PE}_{pos,2i+1} &= \cos(pos/10000^{2i/d_m})
\end{aligned}
\tag{3.19}
$$

3.3.4.2 Pre-Layer Normalized Transformer

Original transformer architecture includes layer normalization operations after attention and feed-forward layers. It is unstable since gradients of output layers are high, so Pre-Layer Normalized transformer is proposed by [29]. Moreover, Parisotto et al. Xiong et al. [15]. proposed gated transformer which also includes layer normalizations before attention and feedforward layer. They also stated that although gated architecture improves many RL tasks drastically, non-gated pre-layer normalized transformer are good enough.

Encoder equations are as follows.

$$\begin{aligned} att =& x + \mathrm{MHA}(\mathrm{LN}(x), \mathrm{LN}(x), \mathrm{LN}(x)) \\ y =& att + \mathrm{FFN}(\mathrm{LN}(att)) \end{aligned} \tag{3.20}$$

Decoder equations are as follows.

$$\begin{aligned} att =& d_{1:t} + \mathrm{MHA}(\mathrm{LN}(d_{1:t}), \mathrm{LN}(d_{1:t}), \mathrm{LN}(d_{1:t})) \\ dec =& att + \mathrm{MHA}(\mathrm{LN}(att), \mathrm{LN}(att), \mathrm{LN}(att)) \\ d_{t+1} =& dec + \mathrm{FFN}(\mathrm{LN}(dec)) \end{aligned} \tag{3.21}$$

CHAPTER 4

BIPEDAL WALKING BY TWIN DELAYED DEEP DETERMINISTIC POLICY GRADIENTS

4.1 Details of the Environment

Observation Space: Hull angle, hull angular velocity, translational velocity on two dimension, joint positions, joint angular speeds, leg ground concats and 10 lidar rangefinder measurements. Details are summarized at Table 4.1

Num	Observation	Max	Max
0	Hull Angle	0	2π
1	Hull Angular Vel	$-\infty$	$+\infty$
2	Vel x	-1	$+1$
3	Vel y	-1	$+1$
4	Hip 1 Joint Angle	$-\infty$	$+\infty$
5	Hip 1 Joint Speed	$-\infty$	$+\infty$
6	Knee 1 Joint Angle	$-\infty$	$+\infty$
7	Knee 1 Joint Speed	$-\infty$	$+\infty$
8	Leg 1 Ground Contact Flag	0	1
9	Hip 2 Joint Angle	$-\infty$	$+\infty$
10	Hip 2 Joint Speed	$-\infty$	$+\infty$
11	Knee 2 Joint Angle	$-\infty$	$+\infty$
12	Knee 1 Joint Speed	$-\infty$	$+\infty$
13	Leg 1 Ground Contact Flag	0	1
14-23	Lidar measures	$-\infty$	$+\infty$

Table 4.1: Observation Space of Bipedal Walker

Action Space: Torque provided to knee and pelvis joints of both legs. Details are presented in Table 4.2.

Num	Observation	Max	Max
0	Hip 1 Torque	-1	$+1$
1	Hip 2 Torque	-1	$+1$
2	Knee 1 Torque	-1	$+1$
3	Knee 2 Torque	-1	$+1$

Table 4.2: Action Space of Bipedal Walker

Rewarding: Directly proportional to distance traveled forward, +300 points given if agent reaches end of path. -100 point if agent falls, and small amount of negative reward proportional to applied motor torque (preventing applying unnecessary torque).

4.1.1 Partial Observability

In DRL, partial observability is handled by 2 ways in literature [6]. First is incorporating fixed number of last observations while second way is updating hidden belief state using recurrent neural network at each time step.

Our approach is using fixed number of past states into LSTM, BiLSTM and Transformer based networks.

4.2 RL Method and hyperparameters

TD3 algorithm is used for learning task. For all networks following hyperparameters are used.

4.3 Proposed Neural Networks

For all networks, varing backbones used to encode state information from observations for both actor and critic networks. As backbones, following networks are proposed.

- Feed Forward Network with residual connection (hidden dim: 128, feed forward dim: 384)

- LSTM (hidden dim: 128, number of layer: 1)
- Bidirectional LSTM (hidden dim: 96, number of layer: 1)
- Transformer (hidden dim: 128, feed forward dim: 256, number of layer: 1)

Actions are passed through feed forward layer with GELU activation and summed up by state encoding. Then, this summation is again passed through feed forward layer with GELU activation. Lastly, a linear layer is used for critic estimation in critic network and feed forward layer with tanh activation for action estimation in actor network.

4.3.1 Feed Forward Network

4.3.2 Long Short Term Memory

4.3.3 Transformer (Pre-layer Normalized)

4.4 Results

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

5.2 Future Work

REFERENCES

[1] BipedalWalker-v2, January 2021.

[2] BipedalWalkerHardcore-v2, January 2021.

[3] P. Abbeel, A. Coates, M. Quigley, and A. Ng, An Application of Reinforcement Learning to Aerobatic Helicopter Flight, in *NIPS*, 2006.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, OpenAI Gym, arXiv:1606.01540 [cs], June 2016, arXiv: 1606.01540.

[5] R. Collobert, K. Kavukcuoglu, and C. Farabet, Torch7: A Matlab-like Environment for Machine Learning, 2011.

[6] G. Dulac-Arnold, D. Mankowitz, and T. Hester, Challenges of Real-World Reinforcement Learning, arXiv:1904.12901 [cs, stat], April 2019, arXiv: 1904.12901.

[7] R. Fris, The Landing of a Quadcopter on Inclined Surfaces using Reinforcement Learning, 2020.

[8] X. Fu, F. Gao, and J. Wu, When Do Drivers Concentrate? Attention-based Driver Behavior Modeling With Deep Reinforcement Learning, arXiv:2002.11385 [cs], June 2020, arXiv: 2002.11385.

[9] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, Memory-based control with recurrent neural networks, arXiv:1512.04455 [cs], December 2015, arXiv: 1512.04455.

[10] K. Kopşa and A. T. Kutay, *Reinforcement learning control for autorotation of a simple point-mass helicopter model*, METU, Ankara, 2018.

[11] A. Kumar, N. Paul, and S. N. Omkar, Bipedal Walking Robot using Deep Deterministic Policy Gradient, arXiv:1807.05924 [cs], July 2018, arXiv: 1807.05924.

[12] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, The bulletin of mathematical biophysics, 5(4), pp. 115–133, December 1943, ISSN 1522-9602.

[13] T. M. Mitchell, *Machine Learning*, McGraw-Hill Education, New York, 1st edition edition, March 1997, ISBN 9780070428072.

[14] X. Pan, Y. You, Z. Wang, and C. Lu, Virtual to Real Reinforcement Learning for Autonomous Driving, arXiv:1704.03952 [cs], September 2017, arXiv: 1704.03952.

[15] E. Parisotto, H. F. Song, J. W. Rae, R. Pascanu, C. Gulcehre, S. M. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, M. M. Botvinick, N. Heess, and R. Hadsell, Stabilizing Transformers for Reinforcement Learning, arXiv:1910.06764 [cs, stat], October 2019, arXiv: 1910.06764.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, Advances in Neural Information Processing Systems, 32, pp. 8026–8037, 2019.

[17] D. Rastogi, Deep Reinforcement Learning for Bipedal Robots, 2017.

[18] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain., Psychological Review, 65, pp. 386–408, 1958.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning representations by back-propagating errors, Nature, 323(6088), pp. 533–536, October 1986, ISSN 1476-4687.

[20] S. J. Russell and P. Norvig, *Artificial Intelligence A Modern Approach*, Prentice Hall, 3 edition, 1995, ISBN 9780136042594.

[21] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, Deep Reinforcement Learning framework for Autonomous Driving, Electronic Imaging, 2017(19), pp. 70–76, January 2017.

[22] S. R. B. d. Santos, S. N. Givigi, and C. L. N. Júnior, An experimental validation of reinforcement learning applied to the position control of UAVs, in *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2796–2802, October 2012, iSSN: 1062-922X.

[23] S. Shalev-Shwartz, S. Shammah, and A. Shashua, Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving, arXiv:1610.03295 [cs, stat], October 2016, arXiv: 1610.03295.

[24] D. R. Song, C. Yang, C. McGreavy, and Z. Li, Recurrent Deterministic Policy Gradient Method for Bipedal Locomotion on Rough Terrain Challenge, in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 311–318, November 2018.

[25] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, A Bradford Book, February 1998.

[26] U. Upadhyay, N. Shah, S. Ravikanti, and M. Medhe, Transformer Based Reinforcement Learning For Games, arXiv:1912.03918 [cs], December 2019, arXiv: 1912.03918.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, Attention is All You Need, 2017.

[28] S. Wang, D. Jia, and X. Weng, Deep Reinforcement Learning for Autonomous Driving, arXiv:1811.11329 [cs], May 2019, arXiv: 1811.11329.

[29] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T.-Y. Liu, On Layer Normalization in the Transformer Architecture, arXiv:2002.04745 [cs, stat], June 2020, arXiv: 2002.04745.

APPENDIX A

PROOF OF SOME THEOREM

This is appendix text.

```
%%% Here is the US Census data from 1900 to 2000
%%% Copied from:
%%% https://www.mathworks.com/help/matlab/examples ...
%%%              /predicting-the-us-population.html
%%%

%%% Don’t use too long lines

% Time interval
t = (1900:10:2000)’;

% Population
p = [75.995 91.972 105.711 123.203 131.669 ...
   150.697 179.323 203.212 226.505 249.633 281.422]’;

% Plot
plot(t,p,’bo’);
axis([1900 2020 0 400]);
title(’Population of the U.S. 1900-2000’);
ylabel(’Millions’);

n = length(t);
s = (t-1950)/50;
A = zeros(n);
A(:,end) = 1;
for j = n-1:-1:1
   A(:,j) = s .* A(:,j+1);
end
```

Listing A.1: The `lintest` function in a floating “listing” environment.