Second-Order Methods for Policy Search in Reinforcement Learning

A Project Report

submitted by

SHREYAS CHAUDHARI

in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.
May 2019

THESIS CERTIFICATE

This is to certify that the thesis entitled **Second-Order Methods for Policy Search in Reinforcement Learning**, submitted by **Shreyas Chaudhari** (**EE15B019**), to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelors of Technology** is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Prashanth L.A.
Research Guide
Assistant Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

I express my sincere gratitude towards my guide, Prof. Prashanth L.A. who introduced me to reinforcement learning in the elective course taught by him. His guidance at various stages of the project has helped me a lot, and his approach to problem solving has been an important learning experience for me.

I would also like to thank my co-guide, Prof. Puduru Viswanadha Reddy, for making the courses on control systems and game theory immensely interesting, which formed the stepping stones to certain concepts in reinforcement learning.

ABSTRACT

KEYWORDS: Policy Search, Second Order Methods, Newton Method, Markov Decision Processes, Function Approximation

Function approximation is essential to reinforcement learning - both in methods that approximate the value function as well as in methods that use a function approximator for the policy. In this work, the latter is considered. Numerous algorithms that use the gradient information already exist - examples being REINFORCE (Williams [1992]) and actor-critic methods (Sutton *et al.* [1999]). These fall under the class of *Policy Gradient Methods* and have supporting theoretical guarantees. We look to exploit the benefits of methods that use second order information. The Newton Method (Tibshirani [2018]) is one such iterative method, which enjoys a significantly faster convergence rate as compared to gradient based methods. Although, these methods come with numerous drawbacks too; which have been addressed in the work. We hope to exploit the desirable properties using approximations of the Newton Method, and show mildly promising results for the same. A significant insight is in the derivation of the expressions for gradient and Hessian, which may lay the path for future work in this fairly unexplored domain for reinforcement learning.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS i

ABSTRACT ii

LIST OF FIGURES v

1 Introduction 1

1.1 Reinforcement Learning 1

1.2 Policy Gradient Method 2

1.3 Newton Method 3

1.4 Problem Statement 3

2 Background 4

2.1 Markov Decision Process 4

2.1.1 Preliminaries and Notation 4

2.1.2 Policies and Value Functions 5

2.2 Gradient and Hessian Expressions 6

2.2.1 Gradient expression 6

2.2.2 Hessian expression 7

2.3 Natural Gradient Ascent 8

2.3.1 Fischer Information Matrix 9

3 Related Work 10

4 Analysis 11

4.1 Second Order Methods 11

4.1.1 Methods and Approximations 12

4.2 Properties 14

5 Experiments and Results **15**

5.1 Cartpole-v0 15

5.1.1 Set-up 15

5.1.2 Expressions for $\nabla_w J(w)$ and $\nabla_w^2 J(w)$ 16

5.2 Experiments 20

5.2.1 REINFORCE 20

5.2.2 Full Hessian Method 21

5.2.3 Approximate Newton Method 21

5.2.4 Diagonal Approximation 22

5.3 Implementational Issues 23

6 Conclusion and Future Work **24**

LIST OF FIGURES

5.1 Cartpole Set-up 15
5.2 REINFORCE algorithm 20
5.3 Newton Method ($H_1(w) + H_2(w)$) 21
5.4 Approximate Newton Method ($H_2(w)$) 22
5.5 Diagonal Approximation ($D_2(w)$) 23

CHAPTER 1

Introduction

This chapter broadly introduces the relevant domain, namely policy search methods, and motivates exploration of second-order methods for the task.

1.1 Reinforcement Learning

As aptly described in Barto and Sutton [1998] - ”Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.” In most if not all cases, the problem reduces to decision making under an incompletely-known Markov Decision Process.

The methods employed handle a delicate *exploration-exploitation* trade off. They can be broadly divided into two categories:

- Action-value methods: learn the values of actions and select actions based on their estimated action values
- Parametrized policy methods: select actions without consulting a value function

This work concentrates on parametrized policy search methods, as described below.

1.2 Policy Gradient Method

The policy gradient methods target at modeling and optimizing the policy directly. The policy is usually modeled with a parameterized function respect to w, $\pi(a|s;w)$. The policy can be parameterized in any way, as long as $\pi(a|s;w)$ is differentiable with respect to its parameter.

In gradient-based methods the update of the policy parameters take the form:

$$w_{n+1} = w_n + \alpha M(w)\nabla_w J(w)$$

and pre-conditioning matrix ($M(w)$) is taken as identity.

Two critical steps in the process are policy paramterization ($\pi(.)$) and feature generation ($\phi(s,a)$). If the action space is discrete and not too large, then a natural and common kind of parameterization is the Gibb's/softmax policy. The actions themselves might be geenrated by the hidden layers of a neural network, or may be linear in features - as in employed in the experiments for this work.

The numerous advantages of these methods are succintly described in Barto and Sutton [1998] and Furmston and Lever [2015]. The gradient takes the form:

$$\nabla_w J(w) = \sum_{i=1}^{H} \mathbb{E}_{\tau\sim p(\tau;\pi_w)}[G_i(\tau)\nabla \log \pi_w(a_i;s_i)]$$

as derived the the sections later.

1.3 Newton Method

It is an iterative method that converges to a local optima by using second-order information. The update equation takes the form:

$$w_{n+1} = w_n + \alpha M(w) \nabla_w J(w)$$

where $M(w)$ is the inverse of the Hessian matrix. Conditions on the semi-definiteness ensure the increase/decrease of the objective function with iterations.

The primary advantage that this method offers is its rate of convergence, which is significantly faster that gradient based methods. But the constraints on the concavity of the objective function and Hessian, and the computational efficiency serve as major drawbacks in using this method. The being said, the parameterization used in experimental set-ups for this work provides a simple work around for the computational cost.

1.4 Problem Statement

The project aims to include the advantages provided by second-order methods for policy search in Markov Decision Processes. This area has been vastly unexplored, and rightly so for the drawbacks that it comes with. We aim to find the best methods in the domain, and analytically and empirically survey them - in some cases against gradient based methods.

CHAPTER 2

Background

This chapter will give a more detailed background on the topics covered, and will also serve to introduce a large part of the notation used.

2.1 Markov Decision Process

The discrete time Markov Decision Process (MDP) is a very general mathematical framework with which to model optimal control problems. This framework has numerous applications in the fields of robotics, automatic control, economics and manufacturing.

2.1.1 Preliminaries and Notation

In our setup, MDPs serve to consider the optimality of an agents controlled movements through a given environment. Formally an MDP is described by the tuple S, A, T, p_1, p, π, R, where S and A are sets known respectively as the state and action space, $T \in N$ is the planning horizon and p_1, p, π, R are functions that take the following form:

$$p_1(s) : S \longrightarrow [0,1] \qquad \text{initial state distribution}$$

$$p(s'|s,a) : S^2 \times A \longrightarrow [0,1] \qquad \text{transition dynamics}$$

$$\pi(a|s) : A \times S \longrightarrow [0,1] \qquad \text{policy}$$

$$R(s,a) : S \times A \longrightarrow \mathbb{R} \qquad \text{reward function}$$

The transition probabilities follow the **Markov Property** in that they are conditionally independent of previous state and action trajectories given the previous state-action pair, i.e,

$$p(s'|s,a) = Pr(s_{t+1} = s|s_t = s, a_t = a)$$

2.1.2 Policies and Value Functions

A policy is a mapping from states to probabilities of selecting each possible action. *Return*(G_t) is simply defined as the sum of rewards. When it follows a policies, it constitutes other value functions defined below. A discounting factor γ is included in general cases (can be 1 for the undiscounted case):

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

The *value function* (state-value) of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{T} \gamma^k R_{t+k+1}|S_t = s] \text{ , for all } s \in S$$

where $\mathbb{E}_\pi[.]$ denotes the expected value of a random variable given that the agent follows policy π. Similarly, the *action-value function* id defined as the expected return starting from s, taking the action a, and thereafter following policy π:

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

2.2 Gradient and Hessian Expressions

The derivations that follow are essential to the estimators used in the algorithms that follow.

2.2.1 Gradient expression

We define the the performance measure ($J(w)$) of the policy as the value of the start state of the episode. Assuming some start state s_0:

$$J(w) = v_{\pi_w}(s_0)$$

where v_{π_w} is the true value function for π_w, the policy determined by w. Some notation for the gradient derivation: Let $\tau = s_1, a_1, \ldots, s_H, a_H$ be a trajectory sampled according to $p(\tau; \pi_w)$. Define $\tau_h = s_1, a_1, \ldots, s_h, a_h$. For the sake of derivation, consider a *non-discounted episodic case*. The derivation below follows steps that are similar to ones in [Furmston and Barber, 2012], though modified to match further derivations. We have:

$$J(w) = \sum_{h=1}^{H} \mathbb{E}_{\tau \sim p(\tau;\pi_w)}[R(s_h, a_h)] = \sum_{h=1}^{H} \mathbb{E}_{\tau_h \sim p(\tau_h;\pi_w)}[R(s_h, a_h)]$$

where the second equation follows from the Markov property. Using the "log-trick":

$$\nabla p(\tau_h; \pi_w) = p(\tau_h; \pi_w) \nabla \log p(\tau_h; \pi_w)$$

The gradient of the evaluation measure can then further be simplified:

$$\begin{aligned}
\nabla_w J(w) &= \sum_{h=1}^{H} \mathbb{E}_{\tau_h \sim p(\tau_h; \pi_w)} [R(s_h, a_h) \nabla \log p(\tau_h; \pi_w)] \\
&= \sum_{h=1}^{H} \sum_{i=1}^{h} \mathbb{E}_{\tau \sim p(\tau; \pi_w)} [R(s_h, a_h) \nabla \log p(a_i; s_i)]
\end{aligned}$$

where the second equation comes by splitting the trajectory probability. Exchanging the summations and clubbing one of them to the reward function:

$$\begin{aligned}
\nabla_w J(w) &= \sum_{i=1}^{H} \sum_{h=\tau}^{H} \mathbb{E}_{\tau \sim p(\tau; \pi_w)} [R(s_h, a_h) \nabla \log p(a_i; s_i)] \\
&= \sum_{i=1}^{H} \mathbb{E}_{\tau \sim p(\tau; \pi_w)} [(\sum_{h=i}^{H} R(s_h, a_h)) \nabla \log p(a_i; s_i)] \\
&= \sum_{i=1}^{H} \mathbb{E}_{\tau \sim p(\tau; \pi_w)} [G_i(\tau) \nabla \log p(a_i; s_i)] \\
\therefore \nabla_w J(w) &= \sum_{i=1}^{H} \mathbb{E}_{\tau \sim p(\tau; \pi_w)} [G_i(\tau) \nabla \log \pi_w(a_i; s_i)]
\end{aligned}$$

The last equation follows given that policy π_w is employed. Note that, $G_i(\tau) = \sum_{h=i}^{H} R(s_h, a_h)$.

2.2.2 Hessian expression

For ease of notation, define a shorthand form

$$\Phi(w; \tau) = \sum_{i=1}^{H} G_i(\tau) \log \pi_w(a_i|s_i)$$

Thus, the gradient expression is denoted as:

$$\begin{aligned}\nabla_w J(w) &= \mathbb{E}_{\tau\sim p(\tau;\pi_w)}[\nabla\Phi(w;\tau)] \\ &= \int_\tau p(\tau;\pi_w)\nabla\Phi(w;\tau)d\tau\end{aligned}$$

Taking the second derivative with respect to w:

$$\begin{aligned}\nabla_w^2 J(w) &= \int_\tau \nabla\Phi(w;\tau)\nabla p(\tau;\pi_w)^\top + p(\tau;\pi_w)\nabla^2\Phi(w;\tau)d\tau \\ &= \int_\tau p(\tau;\pi_w)[\nabla\Phi(w;\tau)\nabla p(\tau;\pi_w)^\top + \nabla^2\Phi(w;\tau)]d\tau \\ &= \mathbb{E}_{\tau\sim p(\tau;\pi_w)}[\nabla\Phi(w;\tau)\nabla p(\tau;\pi_w)^\top + \nabla^2\Phi(w;\tau)]\end{aligned}$$

The second derivative of $J(w)$ is called its *Hessian matrix*. Denoting the matrix by $H(w)$, the Hessian takes the form

$$H(w) = H_1(w) + H_2(w)$$

where

$$\begin{aligned}H_1(w) &= \sum_{i=1}^{H} \mathbb{E}_{\tau\sim p(\tau;\pi_w)}[G_i(\tau)\nabla\log p(\tau_h;\pi_w)\nabla^\top\log p(\tau_h;\pi_w)] \\ H_2(w) &= \sum_{i=1}^{H} \mathbb{E}_{\tau\sim p(\tau;\pi_w)}[G_i(\tau)\nabla^2\log p(\tau_h;\pi_w)]\end{aligned}$$

by expanding $\Phi(.)$ and using the definition of $p(\tau_h;\pi_w)$.

2.3 Natural Gradient Ascent

Natural gradient ascent techniques originated in the neural network and blind source sepration literature, and were introduced into the policy search literature in Kakade [2002]. Though we do no use this method, we draw a parallel to one of the terms ($H_2(w)$) of the Hessian matrix derived. In natural gradient, the Fisher Information Matrix is used as the pre-conditioning matrix.

2.3.1 Fischer Information Matrix

The Fisher Information Matrix takes the form:

$$G(w) = \mathbb{E}_{p_\gamma(\tau;\pi_w)}[\nabla^2 \log p(\tau_h; \pi_w)]$$

where $p_\gamma(\tau; \pi_w)$ is scaled probability with discount factor. Compared against $H_2(w)$:

$$H_2(w) = \sum_{i=1}^{H} \mathbb{E}_{\tau \sim p_\gamma(\tau;\pi_w)}[Q(s_i, a_i)\nabla^2 \log p(\tau_h; \pi_w)]$$

it can be seen that the difference between the two methods lies in the non-negative function w.r.t. which the expectation is taken. In the Fisher information matrix the expectation is taken w.r.t. to the geometrically weighted summation of state-action occupancy marginals of the trajectory distribution, while in $H_2(w)$ there is an additional weighting from the state-action value function. Hence, $H_2(w)$ incorporates information about the reward structure of the objective function, whereas the Fisher information matrix does not, and so it will generally contain more information about the curvature of the objective function.

CHAPTER 3

Related Work

Policy gradient based methods are quite commonly used, showing great results on various tasks. The most common ones are REINFORCE [Williams, 1992] and Actor Critic Algorithms [Konda and Tsitsiklis, 1999]. The methods use first order derivative information for policy search. Though second order methods have performance benefits, they have numerous drawback that make them impractical for use. Trust Region Policy Optimization [Schulman *et al.*, 2015] is one method that uses the Hessian information, but not in the classic way that Newton Method [Tibshirani, 2018] does. Some interesting work along the latter direction can be found in Furmston and Barber [2012], Furmston *et al.* [2016] and Furmston and Lever [2015] where Gauss-Newton and Approximate Newton methods have been proposed, some of which are also used in this work. These works draw an interesting parallel to the Natural Gradient Method [Kakade, 2002], by referring to the form of the Fisher Information Matrix [Ly *et al.*, 2017]. Thus, with minimal work and literature in this domain, this prospectively potent domain of work becomes especially exciting to explore.

CHAPTER 4

Analysis

This chapter addresses the implementational details for the experiments in Chapter 5 and the theoretical analysis from references.

4.1 Second Order Methods

A second order algorithm is any algorithm that uses any second derivative. For example, the Newton Method requires the use of the Hessian matrix w.r.t. the parameter.

In the current setting, the performance measure that is being maximized is $J(w)$ - the policy performance metric, which is paramterised by w. The expressions for the gradient and Hessian are derived in Sections 2.2.1 (gradient) and 2.2.2 (Hessian) respectively. The sections below consider a stochastic setting with a per trajectory update. In the sections that follow, a batched variant too is considered.

For the stochastic updates, the estimates for gradient and Hessian can be sampled from inside the expectation of the expression and taking S_t and A_t as random variables.

The **stochastic policy gradient** at time step i takes the form:

$$\widehat{\nabla_w} J(w) = G_i(\tau) \nabla \log \pi_w(a_i; s_i)$$

and the **stochastic policy Hessian** at time step i takes the form:

$$\widehat{\nabla_w^2} J(w) = G_i(\tau)[\nabla \log \pi(a_i|s_i; w)\nabla^\top \log \pi(a_i|s_i; w) + \nabla^2 \log \pi(a_i|s_i; w)]$$

The two terms of the Hessian - $H_1(w)$ and $H_2(w)$ will be defined accordingly.

4.1.1 Methods and Approximations

The Newton Method (described wonderfully in Tibshirani [2018]) uses second order derivative information for parameter search. One of the requirements of Newton's method is to have a *semi-definite* (positive or negative depending on maximization or minimization) Hessian.

$$x_{n+1} = x_n + (\nabla^2 f(x_n))^{-1} \nabla f(x_n)$$

Newtons method is way faster in terms of convergence than Gradient descent because of extra information from Hessian but still it is not usable in many cases because the Hessian of the objective/loss function is not semi-definite there. Another requirement is for the Hessian matrix to be invertible - which is turn requires it to have non-zero eigenvalues.

For cases as general as policy optimization, these conditions are rarely satisfied. Also, various practical constraints lead to instability, as it further elaborated in Chapter 5. Thus the properties required of the Hessian can be summarized as:

- Must be positive or negative semi-definite in the policy parameterization w.r.t. the policy parameter (positive eigen values)
- Must have non-zero eigenvalues to ensure invertibility

In general, the objective is not concave, which means that the Hessian will not be negative-definite over the entire parameter space. In such cases the Newton method can actually lower the objective and this is an undesirable aspect of the Newton method. Thus the right policy function selection plays a crucial role in employing this method.

For the experimental part of this work, we consider a widely used policy that is either log-concave or blockwise log-concave. The most commonly used one is the **softmax/Gibb's policy**:

$$\pi(a|s; w) \propto \exp w_T \phi(a, s)$$

where $\phi(a, s) \in \mathbb{R}^d$ is a feature vector. This policy is widely used in discrete systems and is log-concave in w, which can be seen from the fact that it is the sum of a linear term and a negative log-sum-exp term, both of which are concave. Since the cartpole-v0 environment considered is a discrete system, we proceed with with policy function.

Full Hessian Method

The algorithm considers the non-approximated Newton method $H_1(w) + H_2(w)$. Given the paramterization, in theory the term $H_2(w)$ should be negative semi-definite w.r.t. the parameters, but there is no such guarantee on the term $H_1(w)$. Although, this method shows a significant improvement in convergence rate, but suffers badly from instability (5.3). Thus in the form:

$$w_{n+1} = w_n + \alpha M(w) \nabla_w J(w)$$

$M(w) = -H^{-1}(w)$ and the update is done iteratively.

Approximate Newton Method

As superbly presented in Furmston and Lever [2015], $H_2(w)$ is negative semi-definite w.r.t. the parameters when a soft-max policy is used. in experiments, *we find that that is not the case.* But, because in a vast number of cases, it does hold true, we consider an optimization method with $M(w) = -H_2^{-1}(w)$.

Diagonal Approximation

What can be seen from the expression for $H_2(w)$ derived in the subsequent sections, the diagonal elements do form a matrix with negative eigenvalues. This property is highly desirable, and the performance is reflect when $M(w) = -D_2^{-1}(w)$ is taken; where $D_2(w)$ is a matrix with the diagonal entries of $H_2(w)$.

4.2 Properties

A couple of interesting properties about the terms involved are:

- That $H_2(w)$ is negative-semidefinite over the entire parameter space is a highly desirable property of a preconditioning matrix - the proof for the same follows in Furmston *et al.* [2016].
  Although, we discovered that in the practical implementation, that was not the case, leading to an underwhelming performance.
- The choice of softmax function has constant curvature with respect to the action space, i.e., $\frac{\partial^2}{\partial w^2} \log \pi(a|s;w) = \frac{\partial^2}{\partial w^2} \log \pi(a'|s;w)$ for the elements of the action space. This helps in saving computation while computing the Hessian.

CHAPTER 5

Experiments and Results

5.1 Cartpole-v0

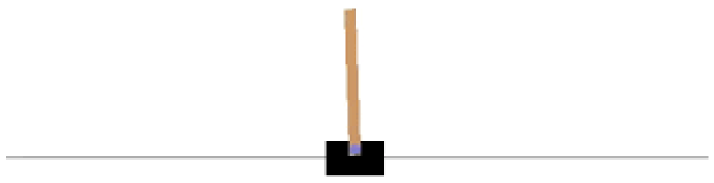

Figure 5.1: Cartpole Set-up

5.1.1 Set-up

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track [Barto *et al.*, 1983]. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity. The OpenAI-gym implementation[1] of the same is used to generate the environment. The state is a four dimensional vector ($\in \mathbb{R}^4$), of the form:

[1]https://github.com/openai/gym/wiki/CartPole-v0

Num	**Observation**
1	Cart position
2	Cart velocity
3	Pole angle
4	Pole velocity at tip

and the actions are push the cart to the left or right - constituting an action space with two elements. The reward is 1 for every step taken, including the termination step.

Note: All observations are assigned a uniform random value between +0.05 and -0.05 as the starting state, and this causes some instability in the Hessian computation, as described in the sections below.

The feature vector ($\phi(s, a)$)**:**

The feature vector is defined as a $\in \mathbb{R}^8$ vector, where the first four elements are the state vector when the action is 1, otherwise the last four are used when the action is 2. That is:

$$\phi(s, a) = \begin{cases} \begin{bmatrix} s_1 & s_2 & s_3 & s_4 & 0 & 0 & 0 & 0 \end{bmatrix} & a = a_1 \\ \begin{bmatrix} 0 & 0 & 0 & 0 & s_1 & s_2 & s_3 & s_4 \end{bmatrix} & a = a_2 \end{cases}$$

5.1.2 Expressions for $\nabla_w J(w)$ and $\nabla_w^2 J(w)$

Considering the expressions from Section 4.1,

$$\widehat{\nabla_w} J(w) = G_i(\tau) \nabla \log \pi_w(a_i; s_i)$$

$$\widehat{\nabla_w^2} J(w) = G_i(\tau)[\nabla \log \pi(a_i|s_i; w) \nabla^\top \log \pi(a_i|s_i; w) + \nabla^2 \log \pi(a_i|s_i; w)]$$

where the policy is parameterized as by the softmax function, i.e.:

$$\pi(a|s,w) = \frac{\exp w^T \phi(s,a)}{\sum_{a'} \exp w^T \phi(s,a)}$$

For the cartpole-v0 set up, there are two actions: push left and push right, denote them by a_1, a_2 respectively. The state is a four dimensional vector, denoted as $s = [s_1, s_2, s_3, s_4]$.

Introducing some notation for cleaner derivation of expressions:

$$\begin{aligned} x_1 &\doteq w^T \phi(s, a_1) \\ x_2 &\doteq w^T \phi(s, a_2) \\ \sigma_1 &\doteq softmax(x_1) \\ \sigma_2 &\doteq softmax(x_2) \end{aligned}$$

Also note that the derivative of a softmax [Bendersky, 2016] is of the form:

$$\frac{\partial \sigma_i}{\partial x_j} = \begin{cases} \sigma_i(1 - \sigma_i) & , i = j \\ -\sigma_j \sigma_i & , i \neq j \end{cases}$$

$Derivation for \nabla_w J(w):$

Assume that action picked is a_1, then derivation follows for σ_1:

$$\begin{aligned}
\nabla_w \log \sigma_1 &= \begin{bmatrix} \frac{\partial \log \sigma_1}{\partial x_1} . \frac{\partial x_1}{\partial w_1} & \dots & \frac{\partial \log \sigma_1}{\partial x_2} . \frac{\partial x_2}{\partial w_5} & \dots & \frac{\partial \log \sigma_1}{\partial x_2} . \frac{\partial x_2}{\partial w_8} \end{bmatrix} \\
&= \begin{bmatrix} \frac{(\sigma_1)(1-\sigma_1)s_1}{\sigma_1} & \dots & \frac{\sigma_1 \sigma_2 s_1}{\sigma_1} & \dots & \frac{\sigma_1 \sigma_2 s_4}{\sigma_1} \end{bmatrix} \\
&= \begin{bmatrix} (1-\sigma_1)s_1 & \dots & \sigma_2 s_1 & \dots & \sigma_2 s_4 \end{bmatrix} \\
&= \phi(s, a_1) - \sigma_1 \phi(s, a_1) - \sigma_2 \phi(s, a_2)
\end{aligned}$$

Similarly,

$$\nabla_w \log \sigma_2 = \phi(s, a_2) - \sigma_1 \phi(s, a_1) - \sigma_2 \phi(s, a_2)$$

$Derivation for \nabla_w^2 J(w):$

As mentioned in Section 4.2, the used policy parameterization has constant curvature with respect to the action space. The expression for the same is derived in the following lines. It has a *blocked matrix* structure. Consider some elements:

$$\begin{aligned}
\frac{d^2 \log \sigma_1}{dw_1^2} &= \frac{d}{dw_1} \frac{d \log \sigma_1}{dw_1} \\
&= \frac{1}{\sigma_1} \frac{d}{dw_1} \frac{d\sigma_1}{dx_1} \frac{dx_1}{dw_1} \\
&= \frac{d(1-\sigma_1)s_1}{dw_1} \\
&= -s_1 \sigma_1 (1-\sigma_1) s_1
\end{aligned}$$

$$
\begin{aligned}
\frac{d^2 \log \sigma_1}{dw_1 dw_5} &= \frac{d}{dw_5}\frac{d \log \sigma_1}{dw_1} \\
&= \frac{1}{\sigma_1}\frac{d}{dw_5}\frac{d\sigma_1}{dx_1}\frac{dx_1}{dw_1} \\
&= \frac{d(1-\sigma_1)s_1}{dw_5} \\
&= \frac{d(\sigma_1)s_1}{dx_2}\frac{dx_2}{dw_5} \\
&= s_1\sigma_1\sigma_2 s_1
\end{aligned}
$$

Thus, if the matrix $\nabla^2_w \log \sigma$ (both 1 and 2) is written as:

$$
\nabla^2_w \log \sigma_{1/2} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}
$$

then,

$$
\begin{aligned}
A_{ij} &= -s_i\sigma_1(1-\sigma_1)s_i \\
B_{ij} &= s_i\sigma_1\sigma_2 s_j \\
C_{ij} &= s_i\sigma_1\sigma_2 s_j \\
D_{ij} &= -s_i\sigma_2(1-\sigma_2)s_j
\end{aligned}
$$

This formulation has a supreme advantage as the prospective computationally heavy Hessian computations are reduced to scalar products with a matrix, and the same would hold for both the possible actions (4.2).

Also notable is that we have a symmetric matrix for $\nabla^2_w \log \sigma_{1/2}$, which is the term in $H_2(w)$. Thus it can be seen that the diagonal approximation $D_2(w)$.

5.2 Experiments

Reward plots for the four algorithms considered are shown below. Due to issues of instability in second-order methods (Section 5.3), comparative plots on the same figure have been tough to acquire.

5.2.1 REINFORCE

REINFORCE uses only first-order (gradient) information, leading to a slightly slow but guarenteed convergence. The implementational details are described previously, with the learning rate being 0.0025.

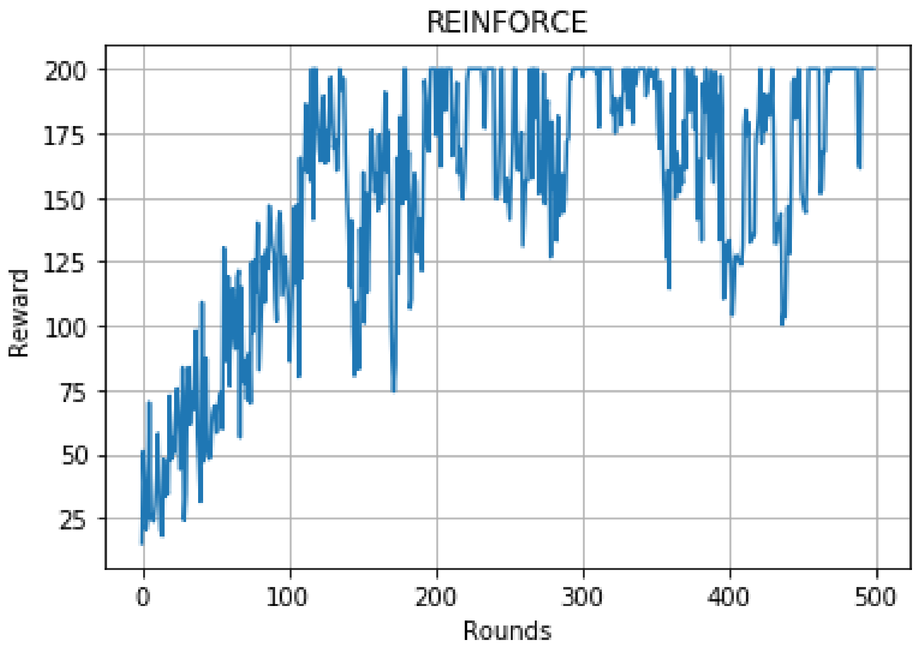

Figure 5.2: REINFORCE algorithm

5.2.2 Full Hessian Method

Attaining stability in the second order methods that follow is very tough (5.3). Thus this method converges to a good policy very fast but drops off back to 0 once it encounters nans in the code. The learning rate used is 0.0025.

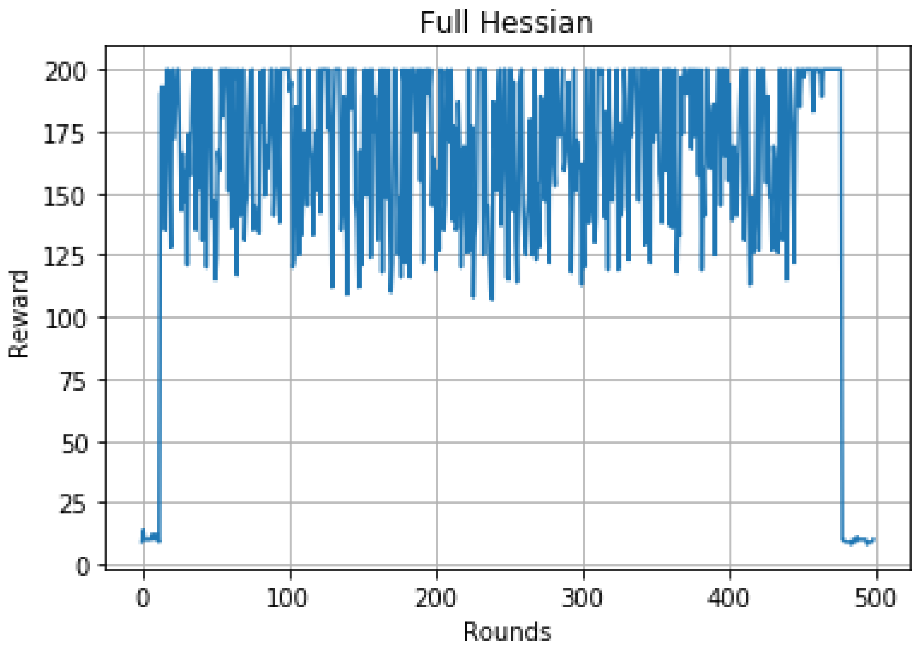

Figure 5.3: Newton Method ($H_1(w) + H_2(w)$)

5.2.3 Approximate Newton Method

Contrary to what is proposed in Furmston and Barber [2012], Furmston *et al.* [2016] and Furmston and Lever [2015], the $H_2(w)$ matrix is not semi-definite in the set-up: this has been tested empirically. Thus, lack of guarantee of ascent in the objective function and an approximation of the complete Hessian given a poor performance. The learning rate used is 0.001.

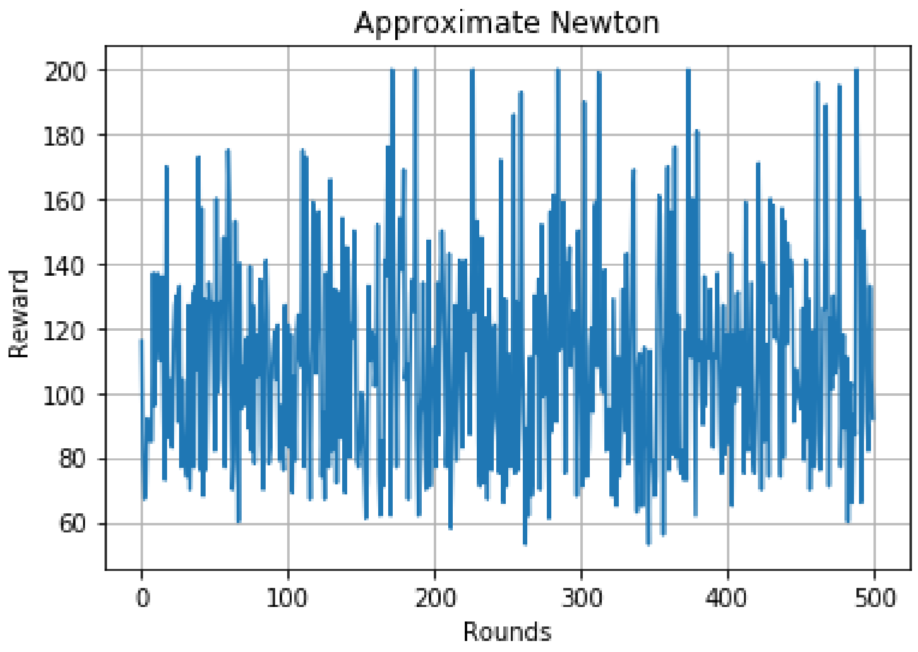

Figure 5.4: Approximate Newton Method ($H_2(w)$)

5.2.4 Diagonal Approximation

The diagonal entries of $H_2(w)$ constitute a negative semi-definite matrix, and this desirable property reflects in the performance. Note again how the performance drops once nans are encountered. he learning rate used is 0.005.

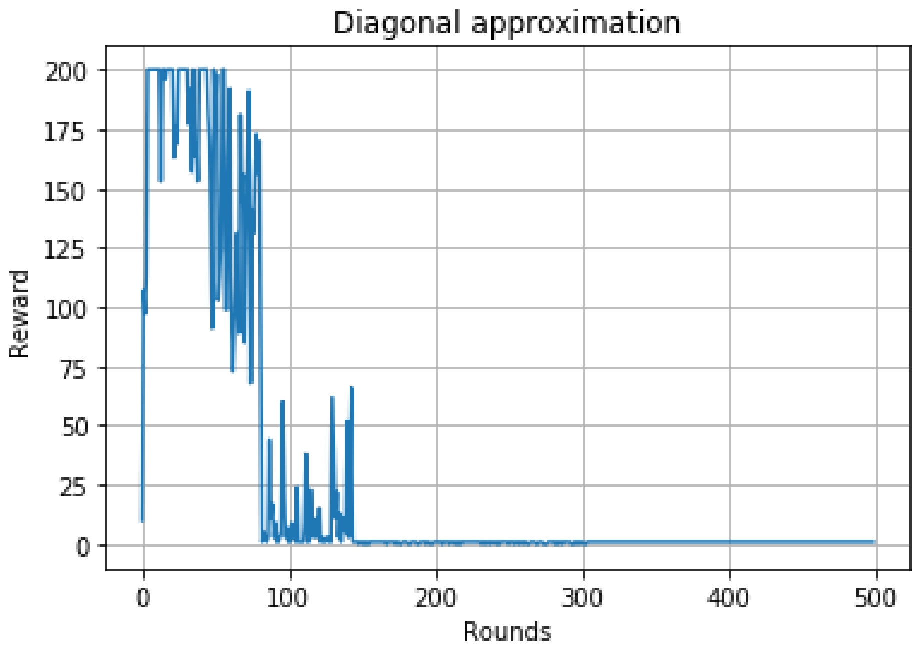

Figure 5.5: Diagonal Approximation ($D_2(w)$)

5.3 Implementational Issues

- State Scaling: As the environment returns states that are usually in the range $[0, 1]$, the computation of Hessian that has terms of the order two of states causes precision problems in python. The numbers drop very sharply with iteration, leading to singular hessian matrices whereas the matrices can actually be inverted. This leads to nans in the computation. Though checks have been set for these, they hinder with the normal iterative update rule.

- Gradient/Hessian Clipping: A commonly used method to control size of update works well to prevent running errors, but hinders the convergence of the algorithm to a local optima. As there are both the gradient and Hessian terms involved, clipping them saturates all updates thereon.

- Deterministic Policy: Quite often the policy tends towards a deterministic policy. Checking the terms in the expression for Hessian 2.2.2, it can be seen that this leads to a zero matrix. Though an update for this case may be removed by heuristic checking, due to lack of incoming updates, the algorithm remains stuck in that policy leading to termination and hence a drop in reward.

These issues make averaging across trails tough - due to multipe runs in each set of trials becoming unstable.

CHAPTER 6

Conclusion and Future Work

Second order optimization methods have immense potential advantages and improvements but numerous drawbacks. This domain has not been explored to a large extent in the field of reinforcement learning, and this work serves to mildly explore in this direction, supported by collating some existing work. The promising results could lead to numerous future directions of work like:

- This work derives a neat expression for $\nabla^2_w J(w)$, which is especially easy to work with in a stochastic update setup.
- It side-tracks a large amount of computation with the expression for Hessian considered. Even inversion becomes very simple in the case of diagonal approximation.
- Experimentally the improvement in performance is visible but not consistent and stable. Working on the stability of the problem can lead to more promising results.
- The full Hessian $(H_1 + H_2)$ is not necessarily negative semi-definite. Working on transformations/approximations to ensure the same could be a interesting area of work.
- A stochastic update rule comes with a large variance, which is already high due to second order terms. A batched version of the Newton update rule could be an exciting direction of work.
- Implementation of a quasi-Newton method using the expression for Hessian is another plausible approach to the problem.

In the knowledgeable words of Prof. Albus Dumbledore [Rowling, 1999] -

It is our choices, Harry, that show what we truly are, far more than our abilities. And we choose second-order methods.

REFERENCES

Barto, A., **R. Sutton**, and **C. Anderson**, Neuronlike adaptive elements that can solve difficult learning control problem. IEEE Transactions on Systems, Man, and Cybernetics, 1983.

Barto, A. and **R. S. Sutton**, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Bendersky, E. (2016). The softmax function and its derivative. URL `https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/`.

Furmston, T. and **D. Barber**, A unifying perspective of parametric policy search methods for markov decision processes. 25th International Conference on Neural Information Processing Systems - Volume 2, Pages 2717-2725, 2012.

Furmston, T. and **G. Lever** (2015). Approximate newton methods for policy search in markov decision processes. *arXiv*.

Furmston, T., **G. Lever**, and **D. Barber** (2016). Approximate newton methods for policy search in markov decision processes. *Journal of Machine Learning Research*, **17**, 1–51.

Kakade, S., A natural policy gradient. 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, Pages 1531-1538, 2002.

Konda, V. and **J. Tsitsiklis**, Actor-critic algorithms. NIPS Proceedings, 13, 1008-1014, 1999.

Ly, A., **M. Marsman**, **J. Verhagen**, **R. Grasman**, and **E.-J. Wagenmakers** (2017). A tutorial on fisher information. URL `https://arxiv.org/abs/1705.01064`.

Rowling, J., *Harry Potter and the Chamber of Secrets*. Scholastic, 1999.

Schulman, J., **S. Levine**, **P. Abbeel**, **M. Jordan**, and **P. Moritz**, Trust region policy optimization. International Conference on Machine Learning, 1889-1897, 2015.

Sutton, R., **D. McAllester**, **S. Singh**, and **Y. Mansour**, Policy gradient methods for reinforcement learning with function approximation. 12th International Conference on Neural Information Processing Systems, Pages 1057-1063, 1999.

Tibshirani, R. (2018). Convex optimization 10-725/36-725. URL `http://www.stat.cmu.edu/~ryantibs/convexopt-S15/lectures/14-newton.pdf`.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, **8**, 229–256.