Optimal Learning

Under-Graduate project report submitted in partail fulfillment of the requirements for the degree of Bachelor of Technology

by

Ritesh Kumar

Advisor: Prof. Vipul Arora

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

April 2019

Abstract

Name of the student: **Ritesh Kumar** Roll No: **160575**

Project title: **Optimal Learning** Department: **Electrical Engineering**

Project supervisor: **Prof. Vipul Arora**

Month and year of Project Report submission: **April 2019**

Optimal Learning is a current emerging field in Machine Learning which is finding a very wide application in various areas. As with any other field of machine learning, there are also multiple approaches to approach optimal learning. We do a through study of recent state of the art papers in this field and learn about topics related to it like Reinforcement Learning, Glorot's initialization and Bayesian Optimization. Lastly we propose an augmentation to Bayesian Optimization which leads to faster convergence towards the function minimum. We also present our experiments with various algorithms and techniques related to above topics.

Code for this project can be found here: https://github.com/ritesh99rakesh/UGP

Acknowledgements

I would like to extend my sincerest gratitude to my advisors Prof. Vipul Arora, for his constant support throughout the semester, believing in me and giving me a chance to work under his able guidance. I would also like to thank my co-mentor Mr. Tharun Reddy for his help in the project and his never ceasing motivation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are witnessing phenomenal advances in computer vision (CV), natural language understanding (NLU), and artificial intelligence (AI) tasks ranging from image classification, object recognition, document classification, to complex ones such as answering questions via comprehension understanding, and even answering questions about images and videos. A key aspect of intelligence is the capability of doing many different things. Current machine learning systems excel at mastering a single skill, such as Go, Jeopardy. But, when we instead ask an AI system to do a variety of seemingly simple problems, it will struggle. A champion Jeopardy program cannot hold a conversation. In contrast, a human can act and adapt to a wide variety of new, unseen situations. The move from hand-designed features to learned features in machine learning has been widely successful. In spite of this, optimization algorithms are still designed manually. The problem we deal with in this project is as follows

- To get a deep understanding of Reinforcement learning and how it is used in different state-of-the-art algorithms of learning to learn. Also, using the knowledge related to reinforcement learning and Bayesian Optimization, my task was to improve upon current algorithm of sir which uses HJB equation.

For this task, I did a detailed study of basic and intermediate level Reinforcement learning algorithms. I also studied two state of the art papers on learning to learn which use reinforcement learning. I also read two papers published by Prof. Vipul Arora under the topic *Optimal Learning*, and presented a report on my understanding of the paper. Then taking inspiration from *Learning to learn without gradient descent by gradient descent*,

we decided to learn more about Bayesian Optimization. I read three papers for BO and implemented the code for it using *Expected Improvement* acquisition function in python. Presented a report on them. Finally, I proposed a novel method to use Bayesian Optimization and sir algorithm for faster convergence, and got some interesting results. I am currently converting sir Optimal learning Matlab code to Python using Tensor variables. It will greatly help to generalize the algorithm and save us from writing the derivative when we try a new technique.

Chapter 2

Motivation and Objectives

A good deal of work has gone in designing machine learning algorithms that are able to solve variety of specific problems. For example, we have machine learning algorithms that can be used to solve finance to biological to astronomy problems. But one of the hindrance to all these state of the art algorithms is that they are very specific and that one algorithm that performs excellent in one field will suffer in some other field. Also, with today's rising data and complexity of our models, we are faced with the problem of faster convergence. All these problems have a common solution in Optimal Learning as it the field of machine learning which studies of algorithms that can be applied in various fields and also searches for quicker convergence to global optimum.
My objective for this project are:

- To study latest state of the art papers in Optimal Learning and present a report on method and techiques used in them for better learning.
- Throughly study and present a brief report on Reinforcement learning and how to use it in optimal learning.
- Implement reinforcement learning code in python gym and study effects of change in various parameters.
- Do a through study involved in faster convergence of neural networks like Glorot's initilaization.
- Learn about Bayesian Optimization and study its positives and negatives related to function approximation.

- Implement Bayesian optimization and learn more about various acquisition functions.
- Look for improvement in Bayesian optimization which will help us in faster convergence to function minimum
- Implement the proposed algorithm and try it on various functions.

Chapter 3

Theory

3.1 Reinforcement Learning

3.1.1 Introduction

In this section is an introduction to Reinforcement Learning. In reinforcement learning (RL) we consider an **agent** interacting with an **environment**. The agent selects **actions** and, based on the selected action, observes **states** and **rewards** from the environment. The RL problem is to design an agent that selects **optimal actions**. In most ways, optimality is defined here as maximizing **expected long-term reward**. An algorithm for finding an optimal agent is an RL algorithm, to the RL problem[13].

3.1.2 Reinforcement Learning vs Supervised Learning

Reinforcement Learning is different from *supervised learning*, the kind of learning studied in the field of machine learning. Supervised learning is learning from a training dataset of labeled examples provided by a external supervisor[13]. Each example is a description of a situation together with the label - of the correct action the system should take to that situation[13]. This is an important class of learning, but alone it is not adequate for learning from interaction. In interactive problems it is often not possible to obtain examples of desired behavior that are both representative and correct of all the situations in which the agent has to act.

3.1.3 Reinforcement Learning vs Unsupervised Learning

Reinforcement Learning is also different from *unsupervised learning*, which is about finding hidden structure in collections of unlabeled dataset. One of the main differences between Reinforcement Learning and unsupervised learning is that we have got rewards in RL which act as the feedback for the actions taken by the agent. There is no provision of feedback in case of unsupervised learning. Although one might be tempted to think of RL as a kind of unsupervised learning because it does not rely on examples of correct behavior, reinforcement learning is trying to maximize a reward signal instead of trying to find structure hidden[13].

3.1.4 Exploration-Exploitation Trade-off

Main challenge that arise in reinforcement learning, is the trade-off between exploration and exploitation[13]. To obtain a lot of reward, a RL agent must choose actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to *exploit* what it has already experienced in order to obtain reward, but it also has to *explore* in order to make better action selections in the future[13]. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best[exp].

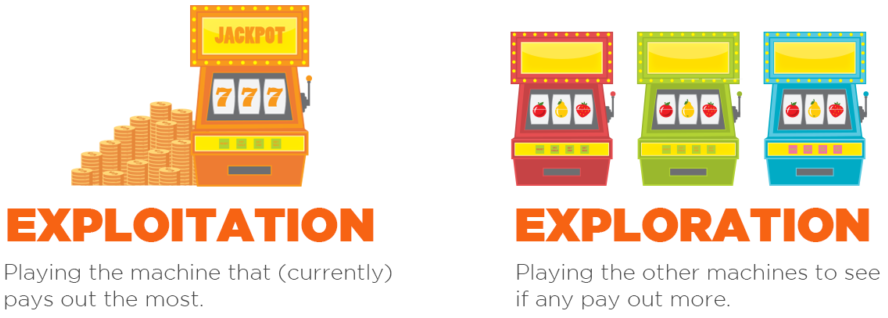

FIGURE 3.1: Exploration-Exploitation Trade-off[exp]

3.2 Markov Decision Processes

Typically reinforcement learning problems are formulated using concepts from **Markov Decision Processes** (MDP). To understand MDP we must know Markov Property.

3.2.1 Markov Property

The future is independent of the past given the present: Once the current state in known, the history of information encountered so far may be thrown away, and that state is a sufficient statistic that gives us the same characterization of the future as if we have all the history[Rai].

3.2.2 Markov Decision Process

A Markov decision process is a 4-tuple (S, A, P_a, R_a), where

1. S is a finite set of states,
2. A is a finite set of actions (alternatively, A_s is the finite set of actions available from state s),
3. $P_a(s, s') = p(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t + 1$,
4. $R_a(s, s')$ is the immediate reward(or expected immediate reward) received after transitioning from state s to state s', due to action a

The core task of Markov Decision Process (MDP) is to find a 'policy'(π) for the decision maker. The goal is to choose a policy π that will maximize some cumulative function of the random rewards[Rai], typically the expected discounted sum over a potentially infinite horizon:

$$G_t = \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}), \text{ where } a_t \text{ is given by the policy and } \gamma \text{ is discount factor}$$

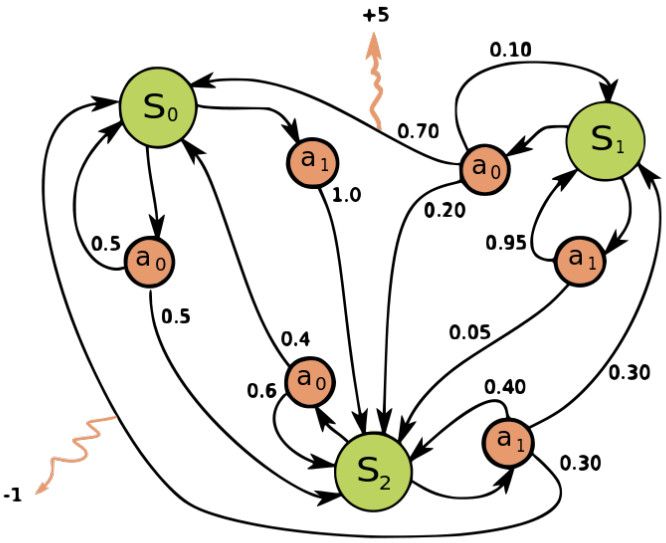

FIGURE 3.2: Markov Decision Process[mar]

3.2.3 Bellman Equation

Value Function: For any policy π, we can define the Value Function (for more information see [Rai])

$$V^{\pi}(s) = E\Big[\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) | s_0 = s, \pi\Big]$$

$V^{\pi}(s)$ is the expected payoff starting in state s and following policy π.
Bellman Equation:Gives a recursive definition of the above Value Function

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{\pi(s)}(s, s') \times V^{\pi}(s')$$

where $R(a)$ is the reward the agent receives when it lands at state s (also see [kra].

3.2.4 Solving Markov Decision Process

The solution for an MDP is a policy which describes the best action for each state in the MDP, known as the optimal policy[13].
General Strategy:

- *Dynamic Programming*

  1. Given a fixed policy(π) estimate the value of each state
  2. Given a fixed value function, improve the policy(π)

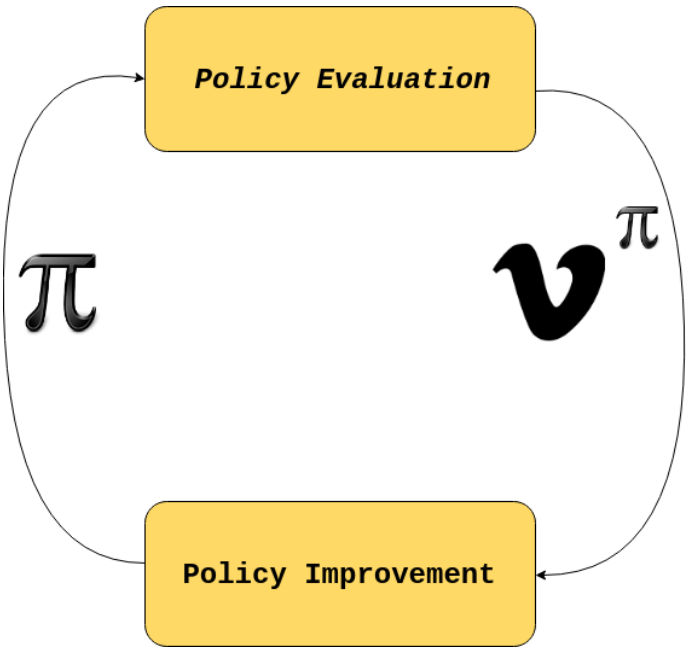

FIGURE 3.3: Dynamic Programming Solution of MDP

Methods[13]

- ***Policy Iteration Algorithm***

  1. Initialization
     - $V(s) \in \mathbf{R}$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$
  2. Policy Evaluation

     Loop
     - $\Delta \longleftarrow 0$
     - Loop for each $s \in S$:
       * $v \longleftarrow V(s)$
       * $V(s) \longleftarrow \sum_{s' \in S} P_{\pi(s)}(s, s') \times V(s')$
       * $\Delta \longleftarrow \max(\Delta, \|v - V(s)\|)$
     - until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   - $policy - stable \longleftarrow true$
   - For each $s \in S$:
     - $old - action \longleftarrow \pi(s)$
     - $\pi(s) \longleftarrow argmax_a \sum_{s' \in S} P_{\pi(s)}(s, s') \big[R_{\pi(s)}(s, s') + \gamma V(s')\big]$
   - If $policy - stable$, then stop and return $V \sim v_*$ and $\pi \sim \pi_*$; else go to 2

- ***Value Iteration Algorithm***

  Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

1. Initialize $V(s)$, for all $s \in S$, arbitrarily except that $V(terminal) = 0$
2. Policy Evaluation and Improvement

   Loop
   - $\Delta \longleftarrow 0$
   - Loop for each $s \in S$:
     - $v \longleftarrow V(s)$
     - $V(s) \longleftarrow \max_{\pi(s)} \sum_{s' \in S} P_{\pi(s)}(s, s') \big[R_{\pi(s)}(s, s') + \gamma V(s')\big]$
     - $\Delta \longleftarrow \max(\Delta, \|v - V(s)\|)$
   - until $\Delta < \theta$

   Output a deterministic policy, $\pi \sim \pi_*$, such that

$$\pi(s) = argmax_a \sum_{s' \in S} P_{\pi(s)}(s, s') \big[R_{\pi(s)}(s, s') + \gamma V(s')\big]$$

Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.

3.3 Recurrent Neural Networks

Any type sequential data cannot be modeled using simple feed-forward neural networks. Such networks cannot take into account the structure associated with sequential data. Figure 3.4 shows an example of how a combination of neural networks has been used to model sequential data. Seeing the figure it seems very intuitive that there has to some sort of a dependency between hidden layers of various networks. In RNNs the hidden state of

each step depends upon hidden state of previous time step (see figure 3.5). Each hidden state is typically defined as

$$\mathbf{h}_t = \mathrm{f}(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1}) \tag{3.1}$$

Where $\mathbf{U}$ is a $K \times K$ transition matrix and f is some non linear function (eg. tanh). Here $\mathbf{h}_{t-1}$ acts as a memory. Now each neural network can have many hidden layers and can be 3.6 generalized to figure ADD FIG. Where A is a chunk of neural network, an input $\mathbf{x}_t$ and output the value $\mathbf{h}_t$

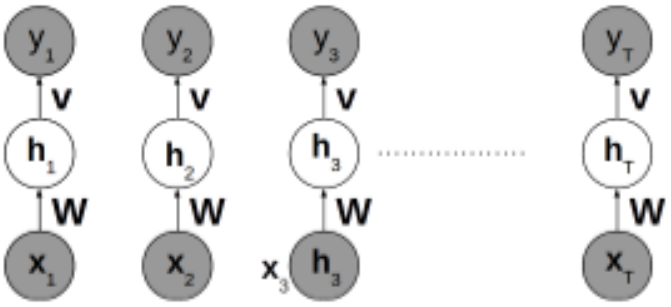

FIGURE 3.4: Modeling sequential data using Neural Networks

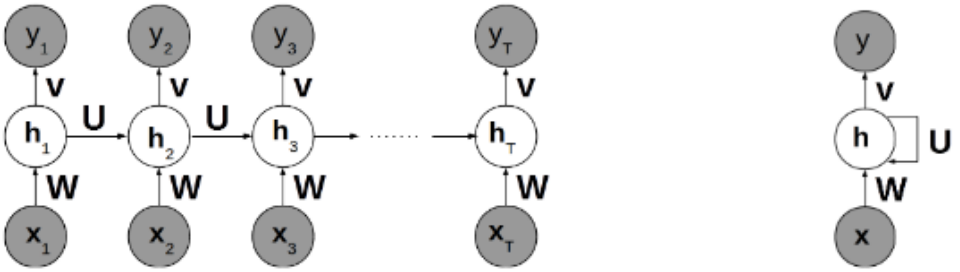

FIGURE 3.5: vanilla RNN

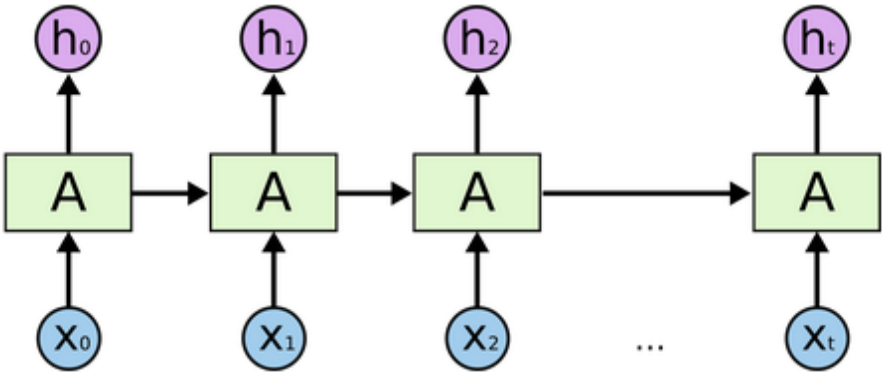

FIGURE 3.6: Modeling sequential data using Neural Networks

3.3.1 Long-Term Dependencies: A Problem

RNN seems very appealing for the reason that it might be able to connect previous information to the present time step. Eg. knowledge about previous conversation is required to be a part of dialog. But in practice as gap between relevant information and the time step where is it need increases RNNs are unable to learn to connect and retain information. Training RNNs using gradient based optimization techniques leads to the problem of exploding/vanishing gradients where RNNs fail to learn information

3.3.2 Long Short Term Memory Networks

LSTM (Long Short Term Memory) is a very special type of RNN that is well known for handling problems related to vanishing/exploding gradients while training. It consists of a memory cell that encodes knowledge of all the inputs seen till the current time step. The behaviour of cells is controlled by gates due to which LSTMs attain the ability to remove/add information to the cell state. The gates control whether to forget current cell value, whether to read cell's input, whether to output new cell value The gate are named as follows: 1. forget gate (f) 2. input gate (i) 3. output gate (o). The update equations for LSTMs are

$$
\begin{aligned}
\mathbf{i_t} &= \sigma(\mathbf{W_{ix}x_t} + \mathbf{W_{im}m_{t-1}} + \mathbf{b_i}) \\
\mathbf{f_t} &= \sigma(\mathbf{W_{fx}x_t} + \mathbf{W_{fm}m_{t-1}} + \mathbf{b_f}) \\
\mathbf{o_t} &= \sigma(\mathbf{W_{ox}x_t} + \mathbf{W_{om}m_{t-1}} + \mathbf{b_o}) \\
\mathbf{c_t} &= \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \tanh(\mathbf{W_{cx}x_t} + \mathbf{W_{cm}m_{t-1}} + \mathbf{b_c}) \\
\mathbf{m_t} &= \tanh(\mathbf{c_t}) \odot \mathbf{o_t} \\
\mathbf{p_t} &= \text{Softmax}(\mathbf{m_t})
\end{aligned}
$$

where all the W matrices are trainable parameters and are shared across the LSTM cells. The $\odot$ is element-wise dot product b/w two vectors. The non-linearities used are Sigmoid ($\sigma(.)$) and hyperbolic tangent (tanh(.)). The last equation gives the probabilty distribution $p(S_t|S_1 \dots S_{n-1})$. In our case S_n can correspond to a video embedding or word embedding and $S_1, \dots, S_{n-1}$ is a sequence of such embeddings used learn probabilty distribution of next embedding given previous embeddings.

3.4 Glorot's Initialization

3.4.1 Random Initialization

The simplest way to initialize weights and biases is to set them to small uniform random values. **This works well for NNs with single hidden layer**. But this simple approach does not work sufficient well for deep NNs(especially those that use ReLu activation).

3.4.2 Glorot's Initialization

One common initialization scheme for deep NNs. Two ways (for more information see [10]):

- In this we initialize each weight with $N\left(0.0, \frac{2.0}{(fan-in+fan-out)}\right)$. Here fan-in is number of input nodes and fan-out is number of hidden notes.
- Can choose weights from a uniform distribution $U\left[\frac{-\sqrt{6}}{\sqrt{fan-in+fan-out}}, \frac{\sqrt{6}}{\sqrt{fan-in+fan-out}}\right]$

3.5 Bayesian Optimization

3.5.1 Introduction

1. Consider hyperparameter tuning for a ML model
2. Find hyperparameters that minimizes the test error
3. Can think of this as minimizing the test error function w.r.t hyperparameters
    - we do not know the function(black box access)
    - therefore cannot use the gradient, Hessian
4. Each **evaluation is expensive** in the sense that the number of evaluations that may be performed is limited, typically to a few hundred. This limitation typically arises because each evaluation takes a substantial amount of time (typically hours), but may also occur because each evaluation bears a monetary cost (e.g., from purchasing cloud computing power, or buying laboratory materials), or an opportunity cost (e.g., if evaluating f requires asking a human subject questions who will tolerate only a limited number)

3.5.2 Formulation

Suppose we have a function $f : \mathcal{X} \longrightarrow R$ that we wish to minimize on some domain $X \subseteq \mathcal{X}$[7]. That is, we wish to find

$$x^* = argmin_{x \in X} f(x)$$

A common approach to optimization problems is to make some assumptions about f. *Bayesian Optimization* procees by maintaining a probabilistic belief about f and designing an *acquisition function* to determine where to evaluate the function next.
Bayesian optimization is particularly well-suited to global optimization problems where f is an expensive black-box function. In brief

- BO use past queries and function estimate + uncertainty to decide where to query next.
- Next query pt. is chosen using an acquisition fn. $a(x)$.

For this BO requires 2 ingridents

- **Regression Model**: to learn the fn. given the current set of query-value pairs $\{x_n, f(x_n)\}_{n=1}^N$. Although not strictly required, BO almost always reasons about f by choosing an appropriate Gaussian process prior:

  $$p(f) = \mathcal{GP}(f; \mu, K)$$

  Given observations $\mathcal{D}$, we can condition our distribution on $\mathcal{D}$ as usual:

  $$p(f|\mathcal{D}) = \mathcal{GP}\left(f; \mu_{f|\mathcal{D}}, K_{f|\mathcal{D}}\right)$$

- **Acquisition fn**: $a(x)$ tells the utility of any future pt. x. The acquisition function is typically an inexpensive function that can be evaluated at a given point that is commensurate with how desirable evaluating f at x is expected to be for the minimization problem[9].

3.5.3 Acquisition Functions

Some acquisition functions(also see [kra])

1. **Probability Improvement**: Suppose

$$f' = \min f$$

is the minimal value of f observed so far. PI evaluates f at the point most likely to improve upon this value. This corresponds to the following utility function associated with evaluating f at a given point x:

$$u(x) = \left\{ \begin{array}{ll} 0, & \text{for } f(x) > f' \\ 1, & \text{for } f(x) \leq f' \end{array} \right\}$$

That is, we receive a unit reward if $f(x)$ turns out to be less that f', and no reward otherwise. The PI acquisition function is then the expected utility as a function of x

$$\begin{aligned} a_{PI}(x) &= E\left[u(x)|x, \mathcal{D}\right] \\ &= \int_{-\infty}^{f'} \mathcal{N}(f; \mu(x), K(x,x))df \\ &= \Phi(f'; \mu(x), K(x,x)) \end{aligned}$$

The point with the highest probability of improvement is selected.

2. **Expected Improvement**: The loss function associated with probability of improvement is somewhat odd: we get a reward for improving upon the current minimum independent of the size of the improvement! This can sometimes lead to odd behavior, and in practice can get stuck in local optima and under explore globally.
We define a new acquisition fn.

$$u(x) = \left\{ \begin{array}{ll} 0, & \text{for } f(x) > f' \\ f' - f(x), & \text{for } f(x) \leq f' \end{array} \right\}$$

The EI acquisition function is then the expected utility as a function of x

$$\begin{aligned} a_{EI}(x) &= E\left[u(x)|x, \mathcal{D}\right] \\ &= \int_{-\infty}^{f'} (f' - f)\mathcal{N}(f; \mu(x), K(x,x))df \\ &= (f' - \mu(x))\Phi(f'; \mu(x), K(x,x)) + K(x,x)\mathcal{N}(f; \mu(x), K(x,x)) \end{aligned}$$

The point with the highest expected improvement is selected.

Significance:
The expected improvement has two components. The first can be increased by reducing the mean function $\mu(x)$. The second can be increased by increasing the variance $K(x,x)$. These two terms can be interpreted as explicitly encoding a trade-off between exploitation (evaluating at points with low mean) and exploration (evaluating at points with high uncertainty).

3. **Lower Confidence Bound(LCB)**: takes into account exploitation vs exploration. Assuming the posterior predictive for a new pt. $x = \mathcal{N}(\mu(x), K(x,x))$ then LCB is

$$a_{LCB}(x) = -(\mu(x) - \alpha\sqrt{K(x,x)})$$

where α is parameter of trade-off b/w mean and variance.
Theoretic result: Under certain conditions, the iterative application of this acquisition fn. will converge to global optima of f.

3.5.4 μ and σ are functions of x

We construct the mean vector by evaluating a *mean function* μ_0 at each x_n. We construct the covariance matrix by evaluating a *co-variance function* or *kernel* Σ_0 at each pair of points x_i, x_j. The kernel is chosen so that points x_i, x_j that are closer in the input space have a large positive correlation, encoding the belief that they should have more similar function values than points that are far apart[7].
The resulting prior distribution on $[f(x_1), ..., f(x_n)]$ is

$$\begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_m) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_m) \end{bmatrix} \mid \begin{bmatrix} \mu_0(x_1) \\ \mu_0(x_2) \\ \vdots \\ \mu_0(x_m) \end{bmatrix}, \begin{bmatrix} \Sigma_0(x_1,x_1) \dots \Sigma_0(x_1,x_N) \\ \Sigma_0(x_2,x_1) \dots \Sigma_0(x_2,x_N) \\ \vdots \\ \Sigma_0(x_N,x_1) \dots \Sigma_0(x_N,x_N) \end{bmatrix} \right)$$

$$\mathbf{f}_N \sim \mathcal{N}(\mathbf{f}_N | \mu_N, \mathbf{\Sigma}_N)$$

Suppose we observe $\mathbf{f}_N$ without noise for some N and we wish to infer the value of $f(x)$ at some new point x. To do so, we have the prior over $[\mathbf{f}_N, f(x)]$ is given by above equation.

We can compute the conditional posterior of $f(x)$ using Bayes' rule and we get

$$
\begin{aligned}
f(x)|\mathbf{f}_N &\sim \mathcal{N}\left(f(x)|\mu(x), \sigma^2(x)\right) \\
\mu(x) &= \mathbf{\Sigma}_* \mathbf{\Sigma}_N^{-1}(\mathbf{f}_N - \mu_0) + \mu_0(x) \\
\sigma^2(x) &= \Sigma_0(x,x) - \mathbf{\Sigma}_* \mathbf{\Sigma}_N^{-1} \mathbf{\Sigma}_*^\top \\
\mathbf{\Sigma}_* &= \Big[\Sigma_0(x,x_1)\ \Sigma_0(x,x_2) \dots \Sigma_0(x,x_N)\Big]
\end{aligned}
$$

Chapter 4

Literature Survey

4.1 Learning to Learn by Gradient Descent by Gradient Descent

4.1.1 Problem Statement

How the design of an optimization algorithm can be cast as a learning problem, allowing the algorithm to learn to exploit structure in the problems of interest in an automatic way?

4.1.2 Mathematical Formulation

We have to optimize the objective function $f(\theta)$ defined over some domain $\theta \in \Theta$. We define an optimizer g, specified by its own parameters ϕ. This results in the update of the optimizee f of the form (see also [5])

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t, \phi))$$

The update rule g is modeled using RNN.
The final optimizee parameters are defined as a function of ϕ and f, ie $\theta^*(f, \phi)$
Now given a distribution of function f we can write the expected loss as

$$L(\phi) = E_f\Big[f\big(\theta^*(f, \phi)\big)\Big]$$

Consider a RNN m, parameterized by ϕ, whose state is denoted by h_t and it outputs the update steps g_t. While the objective function above depends only on the final parameter value, for training the optimizer it is convenient to use an objective function that depends on the entire trajectory of optimization(for some horizon T),

$$
\begin{aligned}
L(\phi) &= E_f\left[\sum_{t=1}^{T} w_t f(\theta_t)\right] \text{ where} \\
\theta_{t+1} &= \theta_t + g_t \\
\begin{bmatrix} g_t & h_{t+1} \end{bmatrix} &= m(\nabla_t, h_t, \phi) \text{ and} \\
\nabla_t &= \nabla_\theta f(\theta_t)
\end{aligned}
$$

We can minimize $L(\phi)$ using gradient descent on ϕ.

4.1.3 ML Approach Used in the Paper

- ***Casting the problem of transfer learning as generalization***

  In ordinary statistical learning we have a particular function of interest, which is

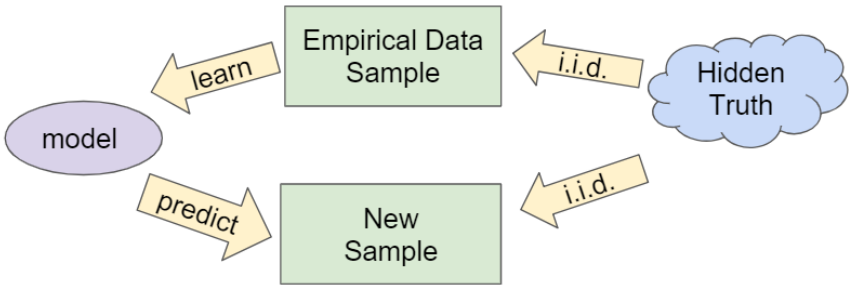

FIGURE 4.1: Generalization[5]

constrained through a data set. In choosing a model we specify our inductive biases as to how the function should behave at points not observed yet. In this respect, **generalization** refers to the capacity to make predictions about the behaviour of the function at novel points. In the problem the examples are themselves problem instances, which means generalization corresponds to the ability to transfer knowledge between different problems. This reuse of problem structure is called as **transfer learning**.

By taking the meta-learning perspective, we can cast the problem of **transfer learning** as one the **generalization**.

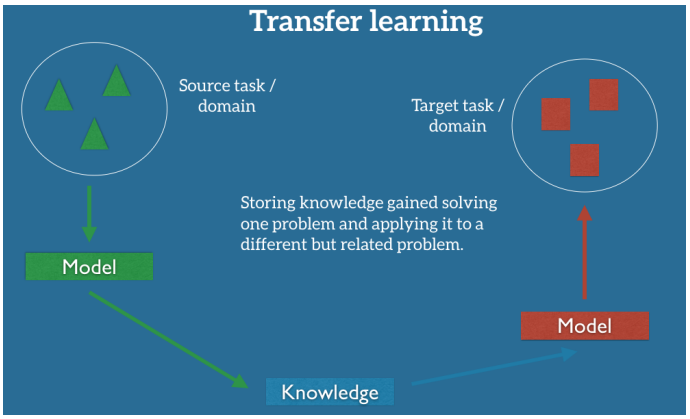

FIGURE 4.2: Transfer Learning[tra]

- ***Coordinatewise LSTM***

  Directly applying RNN will lead to the optimization of at least tens of thousands

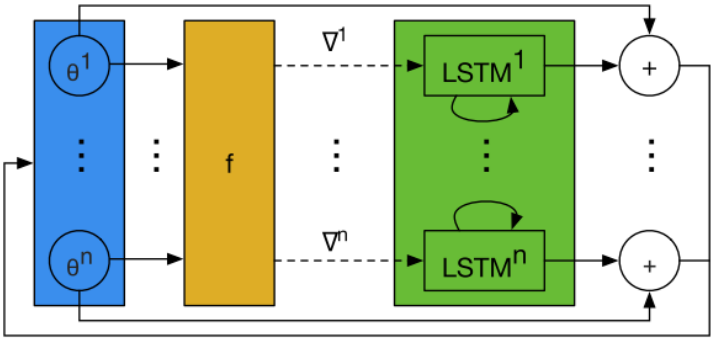

FIGURE 4.3: Coordinatewise LSTM[5]

of parameters. This is not feasible as it would require huge hidden state and an enormous number of parameters. To avoid this difficulty we will use an optimizer m which operates coordinatewise on the parameters of the objective function. This network architecture allows us to use a very small network that only looks at a single coordinate to define the optimizer and share optimizer parameters across different parameters of the optimizee (see more at [5]).

Upside:
It has the effect of making the optimizer invariant to the order of parameters in the network, since the same update rule is used independently on each coordinate. The update rule for each coordinate is implemented using *LSTM*, using the now-standard forget gate architecture.

4.2 Learning to Learn without Gradient Descent by Gradient Descent

4.2.1 Problem Statement

Learning to learn can be used to learn both models and algorithms. In this paper, the goal of meta-learning is to produce an algorithm for global black-box optimization. How to find a global minimizer of an black-box loss function f?

4.2.2 Mathematical Formulation

Consider a black-box loss function f and X as its domain. The function f is not available to the learner in simple closed form at the test time, but can evaluated at query point x in the domain[8]. In other words, we can only observe the function f through unbiased noisy point-wise observations y, ie

$$f(x) = E\big[y|f(x)\big]$$

The optimization algorithm gives the following loop:

1. Given the current state of knowledge h_t propose a query point x_t
2. Observe the response y_t
3. Update internal statistics to produce h_{t+1}

Using a RNN parameterized by θ with combined update and query rule

$$h_t, x_t = RNN_\theta(h_{t-1}, x_{t-1}, y_{t-1})$$
$$y_t = p(y|x_t)$$

Loss Function:
We have 2 choices for loss function:

$$L_{final}(\theta) = E_{f,y_{1:T-1}}\big[f(x_T)\big], \text{ for some time horizon T}$$

or

$$L_{sum}(\theta) = E_{f,y_{1:T-1}}\left[\sum_{t=1}^{T} f(x_t)\right]$$

But we will prefer $L_{sum}(\theta)$ because the amount of information conveyed by L_{final} is temporally very sparse. By instead utilizing a sum of losses to train the optimizer we are able to provide information from every step along this trajectory. Note that similar type of loss function is considered for the previous paper - *Learning to Learn by Gradient Descent by Gradient Descent*

Exploration/Exploitation:
We can encourage exploration in the space of optimizers by encoding an exploratory force directly in to the loss function:

$$L_{EI}(\theta) = -E_{f,y_{1:T-1}}\left[\sum_{t=1}^{T} EI(x_t|y_{1:t-1})\right]$$

where $EI(.)$ is the expected posterior improvement of querying x_t given observations up to time t.

4.2.3 ML Approach Used in the Paper

- ***Meta-learning phase***
  In the meta-learning phase, we use a large number of differentiable functions gen-

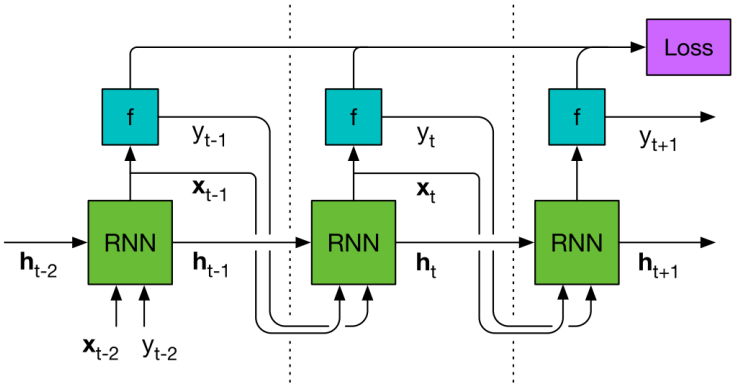

FIGURE 4.4: RNN in the paper[8]

erated with a GP to train RNN optimizers by gradient descent. We consider two types of RNN[8]:

  - long-short-term memory networks (LSTMs)
  - differentiable neural computers (DNCs)

The RNN optimizer learns to use its memory to store information about previous queries and function evaluations, and learns to access its memory to make decisions about which parts of the domain to explore or exploit next. That is, by unrolling the RNN, we generate new candidates for the search process (also see [8]).

4.3 Optimal Convergence Rate in FFNNs using HJB Equation

4.3.1 Problem Statement

To use control theoretic approach (HJB equation) for both batch and instantaneous updates of weights in feed-forward neural networks. How to use HJB(Hamilton-Jacobi-Bellman) equation to generate an optimal weight update law?

4.3.2 Mathematical Formulation

The output $\mathbf{y}_p \in \mathbf{R}^{N_o}$ for a given input pattern $\mathbf{x}_p \in \mathbf{R}^{N_i}$ can be written as (also see [12]),

$$\mathbf{y}_p = f(\mathbf{w}, \mathbf{x}_p)$$

Here $\mathbf{w} \in \mathbf{R}^{N_w}$ is the vector of weight parameters involved in the FFNN to be trained, with total N_x weight parameters. The derivative of $\mathbf{y}$ w.r.t time is

$$\begin{aligned}\frac{\partial \mathbf{y_p}}{\partial t} &= \frac{\partial f(\mathbf{w}, x_p)}{\partial \mathbf{w}} \frac{\partial \mathbf{w}}{\partial t} \\ &= \mathbf{J_p} \frac{\partial \mathbf{w}}{\partial t}\end{aligned}$$

where $\mathbf{J_p} = \frac{\partial f(\mathbf{w}, x_p)}{\partial \mathbf{w}}$ is the Jacobian matrix. The desired output is given by ${\mathbf{y_p}}^d = f(\mathbf{w}, x_p)$ and its derivative w.r.t time is

$$\begin{aligned}\frac{\partial {\mathbf{y_p}}^d}{\partial t} &= \mathbf{J_p} \frac{\partial \mathbf{w}}{\partial t} \\ &= 0\end{aligned}$$

The estimation error is $\mathbf{e_p} = \mathbf{y_p}^d - \mathbf{y_p}$ and its derivative is given by

$$\begin{aligned}\frac{\partial \mathbf{e_p}}{\partial t} &= \frac{\partial \mathbf{y_p}^d}{\partial t} - \frac{\partial \mathbf{y_p}}{\partial t} \\ &= -\mathbf{J_p}\frac{\partial \mathbf{w}}{\partial t}\end{aligned}$$

Define $\frac{\partial \mathbf{w}}{\partial t} = \mathbf{u}$. In batch mode, all the patterns are learnt simultaneously. Hence,

$$\frac{\partial \mathbf{e}}{\partial t} = -\mathbf{J}\mathbf{u}$$

The cost function is deined over time interval $(t, T]$ as

$$V(\mathbf{e}(t)) = \int_t^T L(\mathbf{e}(\tau), \mathbf{u}(\tau))d\tau$$

where $L(\mathbf{e}, \mathbf{u}) = \frac{1}{2}(\mathbf{e}^\top \mathbf{e} + \mathbf{u}\top\mathbf{R}\mathbf{u})$. We have to find optimal update law $\mathbf{u}(t)$, hence we get the HJB equation

$$\min_{\mathbf{u}} \Big\{ \frac{dV*}{d\mathbf{e}} \frac{\partial \mathbf{e}(t)}{\partial t} + L(\mathbf{e}(t), \mathbf{u}(t)) \Big\} = 0$$

4.3.3 ML Approach Used in the Paper

- **Global Minimization**

  The paper[12] uses Lyapunov function to impose further condition on the solution. However, it turns out that these conditions compel the error $e(t)$ to reduce at each step. This behaves like a greedy search scheme, ending up in the local minimum. On the contrary, the convergence to global optimum may sometimes require $e(t)$ to increase also.

Suggestion:
We can use exploration/exploitation trade-off idea of Reinforcement Learning here. If we encourage exploration in the initial period of training slowly shift the importance towards exploitation we can have better chances of convergence to global minimum.

4.4 HJB-Equation-Based Optimal Learning Scheme for NNs With Applications in Brain–Computer Interface

4.4.1 ML Approach Used in the Paper

Algorithm	Update law, $\dot{\hat{\mathbf{w}}} = \mathbf{u}$
BP	$\mathbf{u} = \eta \mathbf{J}^\top \mathbf{e}$
LF	$\mathbf{u} = \mu \frac{\|\mathbf{e}\|}{\|\mathbf{J}^\top \mathbf{e}\|} \mathbf{J}^\top \mathbf{e}$
LM	$\mathbf{u} = \left[\mathbf{J}^\top \mathbf{J} + \mu \mathbf{I}\right]^{-1} \mathbf{J}^\top \mathbf{e}$
HJB	$\mathbf{u} = \frac{1}{r} \mathbf{J}^\top \mathbf{C} \mathbf{e}$
single output online HJB	$\mathbf{u} = \frac{1}{(r \mathbf{J} \mathbf{J}^\top)^{1/2}} \mathbf{J}^\top \mathbf{e}$
EVDHJB	$\mathbf{u} = \mathbf{R}^{-1} \mathbf{J}^\top \mathbf{C} \mathbf{e}$
HJB with time optimality	$\mathbf{u} = \frac{\sqrt{2P(\mathbf{e})}}{\|\mathbf{J}^\top \mathbf{e}\|} \mathbf{R}^{-1/2} \mathbf{J}^\top \mathbf{e}$

TABLE 4.1: Weight Update Laws of Various Algorithms

- The HJB formulation from previous paper is extended by deriving a closed from solution without using eigenvalue decomposition(EVD), interested reader is directed to [6]. This significantly reduces the number of computations, which is otherwise needed for EVD in each iteration.
- Another advantage of this algorithm is that it is easily integrable with state-of-the-art adaptation schemes like adding a momentum term, AdaGrad, etc..
- The update rule:

$$\mathbf{u}^*(t) = \frac{\sqrt{\mathbf{e}^\top(t)\mathbf{e}(t)}}{|\mathbf{J}^\top \mathbf{e}(t)|} \mathbf{R}^{-1/2} \mathbf{J}^\top \mathbf{e}(t)$$

- In the general case, $\mathbf{e}^\top \mathbf{Q} \mathbf{e}$ can be replaced by any other cost function say $P(\mathbf{e}(t))$, then we get

$$\mathbf{u}^*(t) = \frac{\sqrt{2P(\mathbf{e}(t))}}{|\mathbf{J}^\top \mathbf{e}(t)|} \mathbf{R}^{-1/2} \mathbf{J}^\top \mathbf{e}(t)$$

Chapter 5

Experiments

This chapter provides results and observations I obtained when implementing various algorithms

5.1 Reinforcement Learning

5.1.1 Games

Two games were tried from gym library(python)

1. ***Taxi***
   Description: There are four designated locations in the grid world indicated by R(ed), B(lue), G(reen), and Y(ellow). When the episode starts, the taxi starts off at a random square and the passenger is at a random location. The taxi drive to the passenger's location, pick up the passenger, drive to the passenger's destination (another one of the four specified locations), and then drop off the passenger. Once the passenger is dropped off, the episode ends.
   Observations: There are 500 discrete states since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is the taxi), and 4 destination locations.
   Actions: There are 6 discrete deterministic actions:

   - 0: move south
   - 1: move north

- 2: move east
- 3: move west
- 4: pickup passenger
- 5: dropoff passenger

Rewards: There is a reward of -1 for each action and an additional reward of +20 for delivering the passenger. There is a reward of -10 for executing actions "pickup" and "dropoff" illegally.

2. ***Frozen Lake***
   Description: The game consist of 4×4 grid where each tile can be frozen(safe to stand) and hole(not safe to stand). So if you land on a hole you lose. Your task is to reach the defined destination without stepping on holes. **Actions**: There are 4 discrete non-deterministic actions:

   - 0: move left
   - 1: move down
   - 2: move right
   - 3: move up

   The actions are non-deterministic because the surface is slippery and there is a possibility that the action you intended will take you some other direction.

5.1.2 Comparing Different Parameters

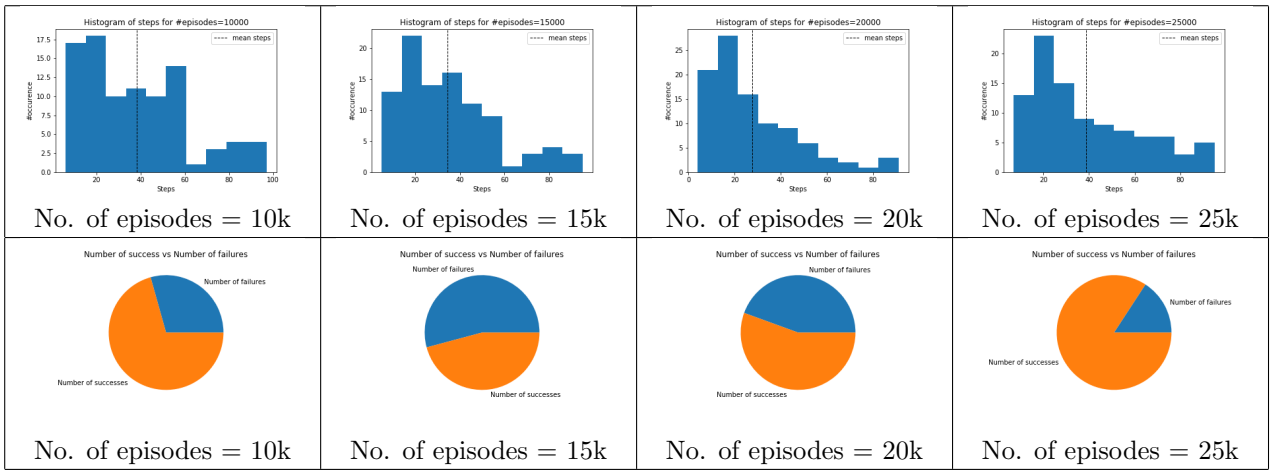

FIGURE 5.1: Number of episodes dependence of game Taxi

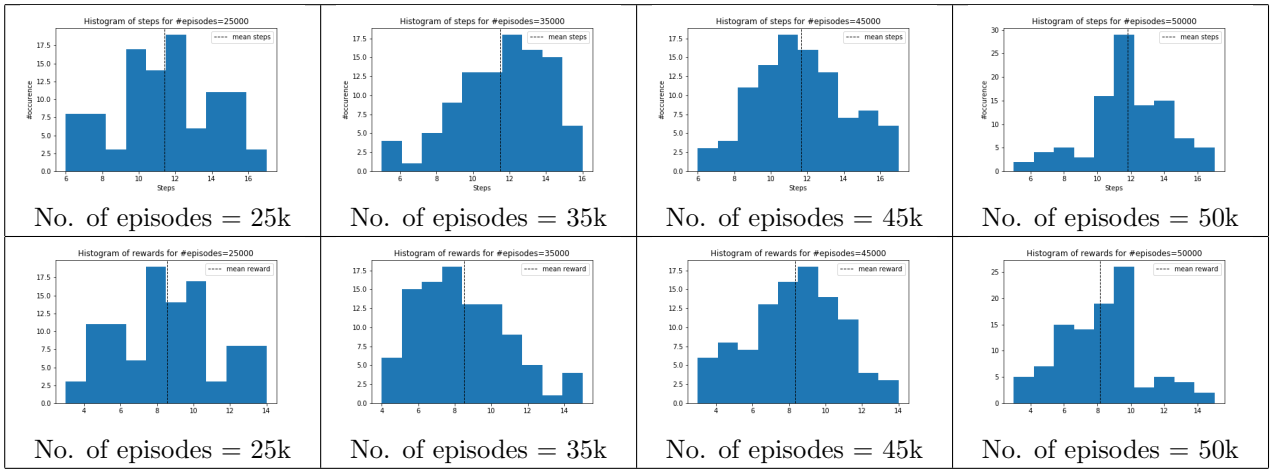

FIGURE 5.2: Number of episodes dependence of game Frozen Lake

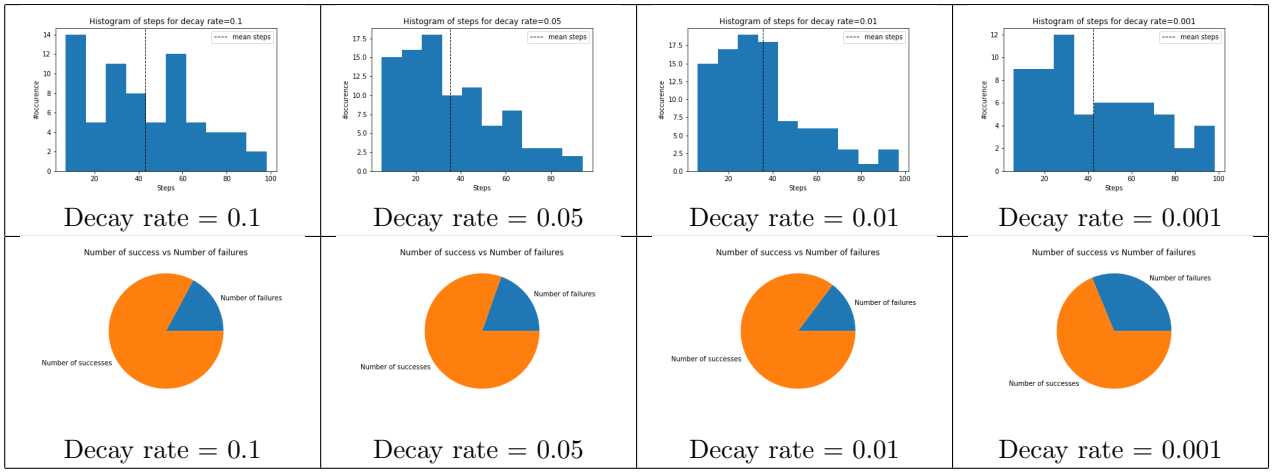

FIGURE 5.3: Delay Rate dependence of game Taxi

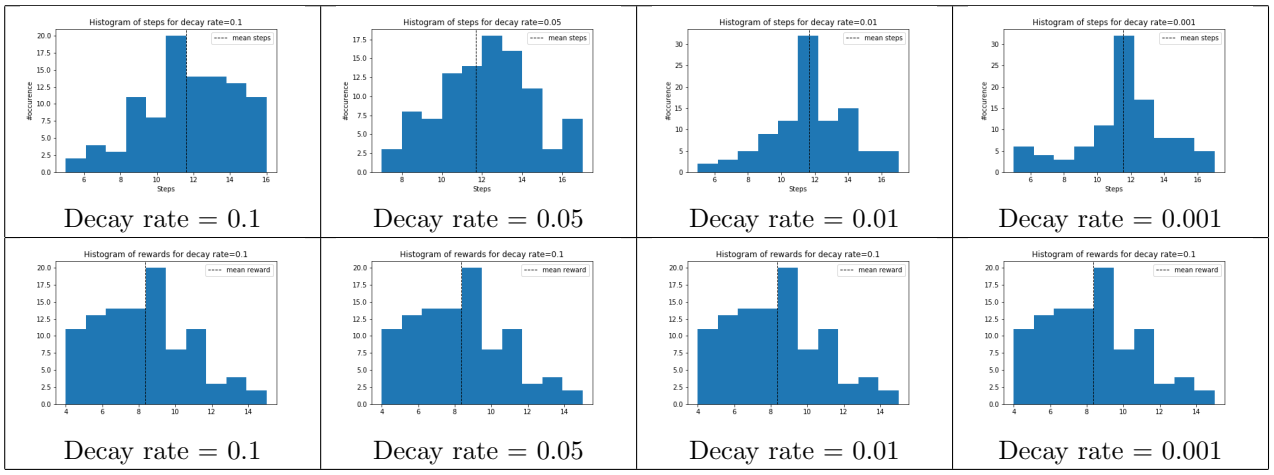

FIGURE 5.4: Delay Rate dependence of game Frozen Lake

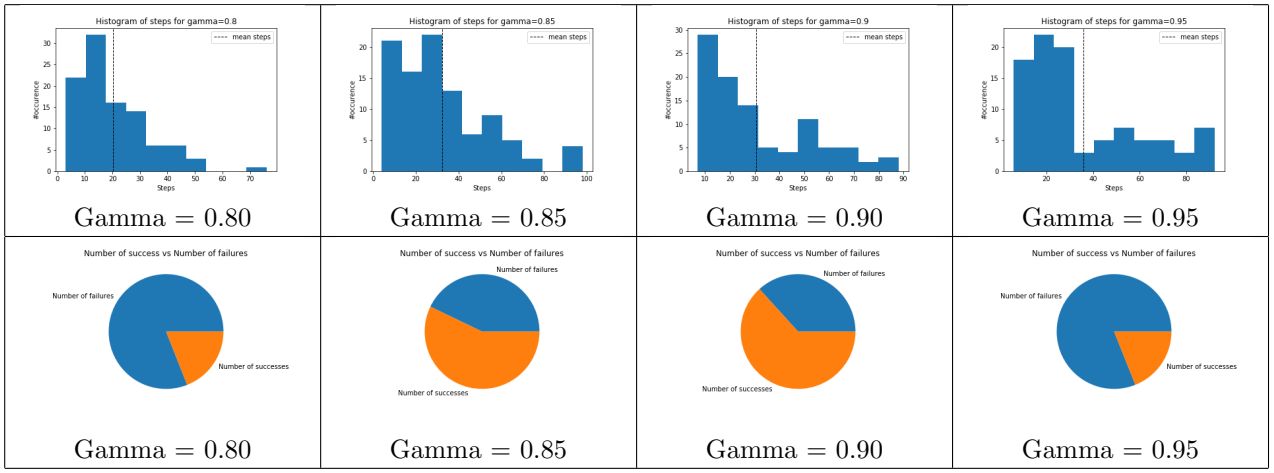

FIGURE 5.5: Gamma of dependence game Taxi

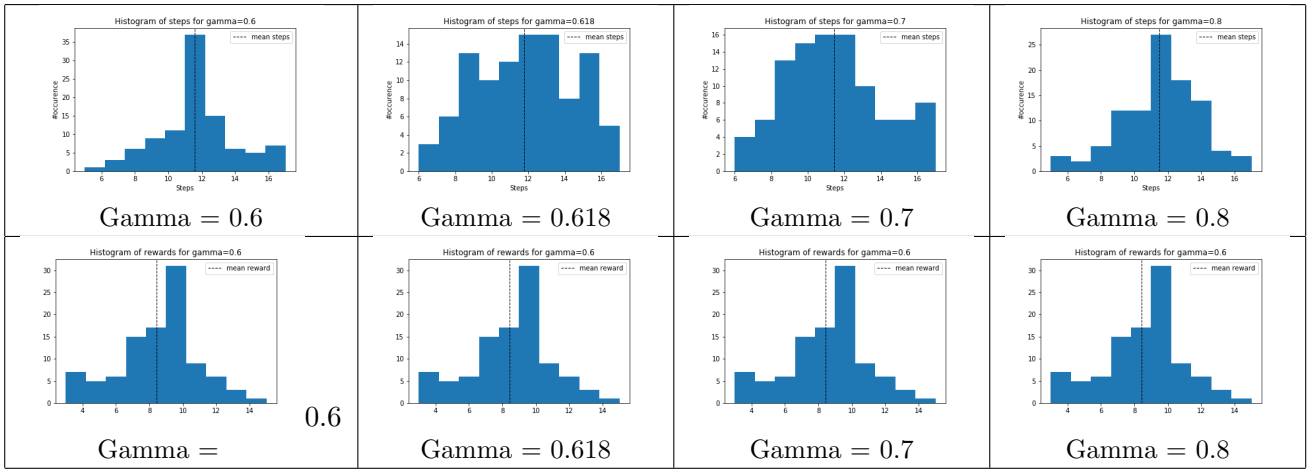

FIGURE 5.6: Gamma dependence of game Frozen Lake

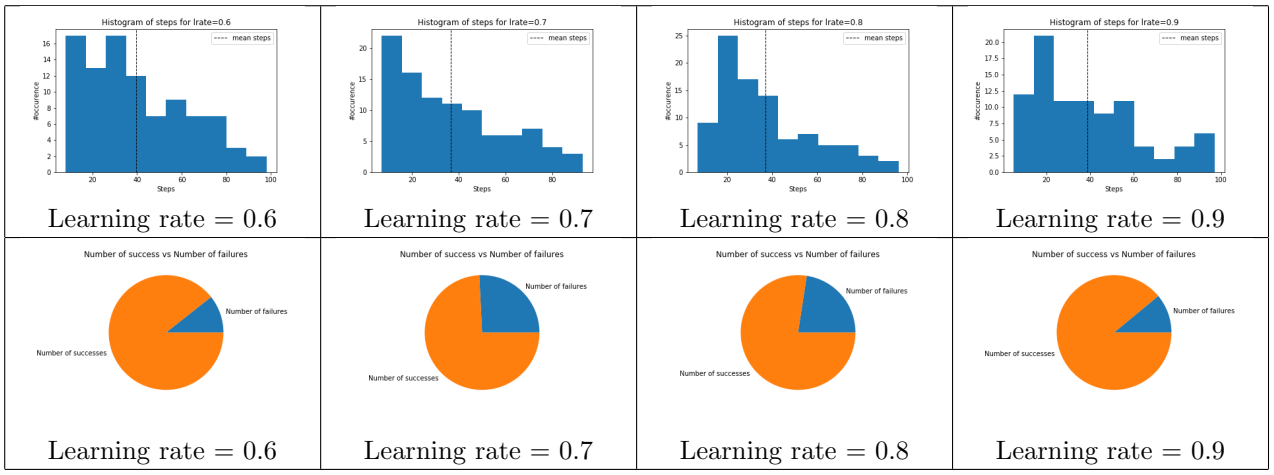

FIGURE 5.7: Learning rate dependence of game Taxi

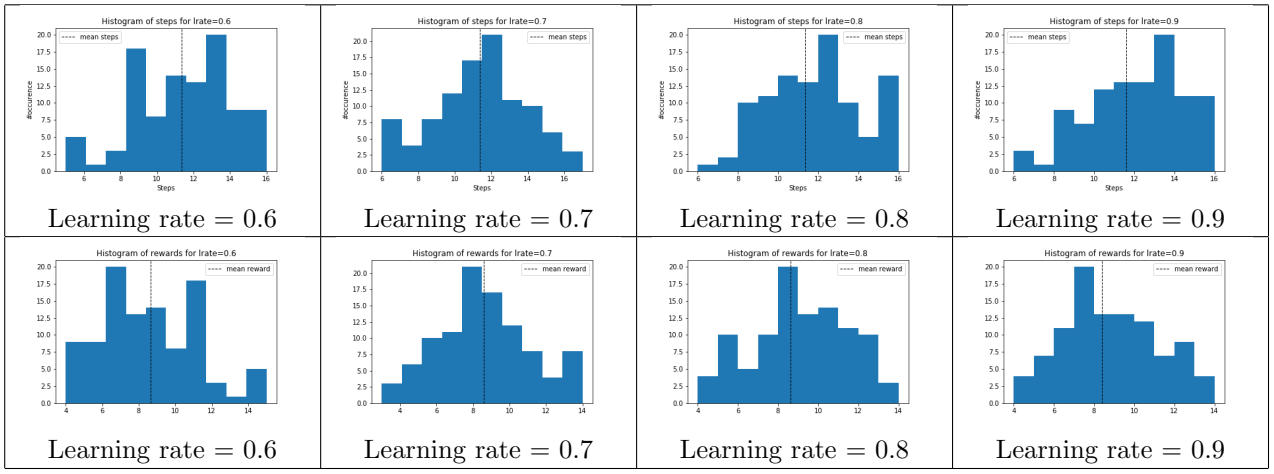

FIGURE 5.8: Learning rate dependence of game Frozen Lake

5.2 Bayesian Optimization

My implementation of Bayesian optimization using Expected Improvement is shown in Figure: 5.9

5.3 Proposed Improvement

In Bayesian optimization our aim is to find the function minima. As explained earlier we do this by using Expected Improvement acquisition function. But when basic EI acquisition function is applied we are not using the gradient information available with us. This slows down the process of finding the function minima.

Proposal: When applying EI acquisition function we choose the next point x by maximizing the utility function defined earlier. We can further improve our estimate of x by applying few iterations of any gradient method, like Gradient Descent, HJB equation. So, we augment Bayesian optimization algorithm to refine our new x by applying gradient descent. The proposed algorithm can be summarized below:

1. Using the acquisition function obtain new x, call it x_{new}
2. Improve the estimate of x by applying gradient methods like Gradient Descent, HJB based optimization

$$x_{new} \longleftarrow \text{ImproveX}(x_{new})$$

3. Using the improved estimate of x_{nex}, evaluate the function f
4. Go to 1 if not converged

where ImproveX(x) is

1. Apply Gradient Descent/HJB based optimization on x
2. return x

One of applications of the above algorithm is in optimal convergence. If we consider function f as our loss function which we wish to minimize and x as the weights of the function which have to be tuned in order to minimum f, then we can apply above algorithm for faster convergence compared to Bayesian optimization. Only constraint here is that, here we have assumed that we can calculate gradient of this function f. This constraint can be removed by using methods like LSTM and this is what I plan to explore in future.

One example

Consider the function f (refer Figure: 5.10)

$$f(x) = \sin(3x) + x^2 - 0.7x$$

in the range $x \in [-1.0, 2.0]$ which we want to estimate using Bayesian Optimization. Below I show two different iterations where in first we run plain Bayesian Optimization algorithm, and in second we run the above proposed algorithm. Clearly second method (Figure: 5.12) is able to converge to function minima at faster rate compared to first algorithm (Figure: 5.11).

Optimization.png Optimization.png

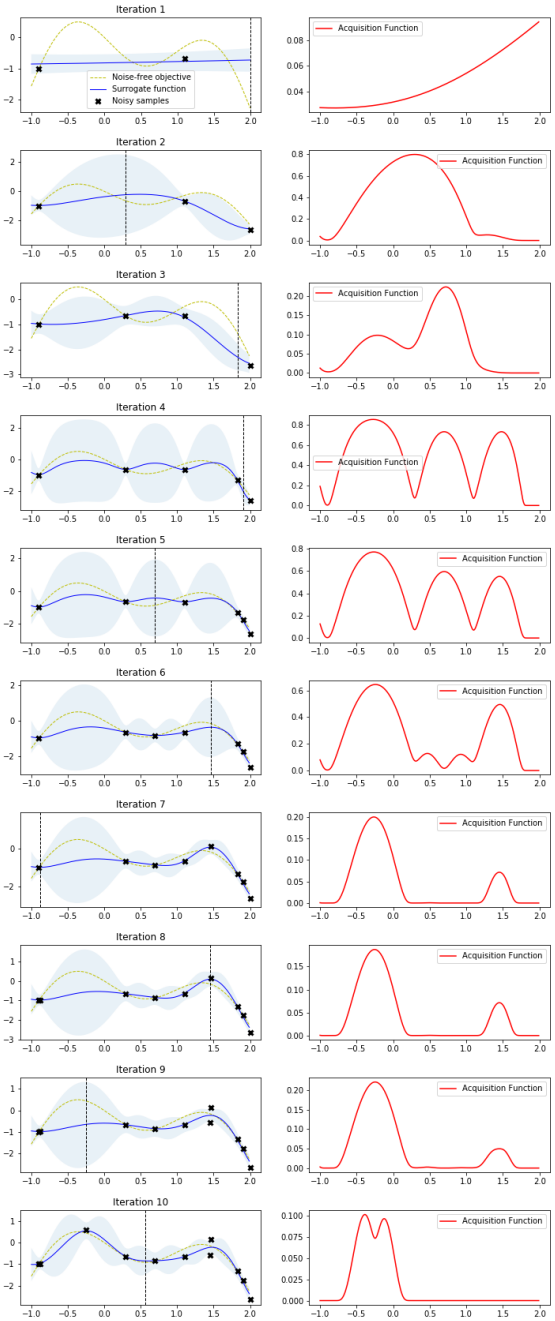

FIGURE 5.9: Bayesian Optimization

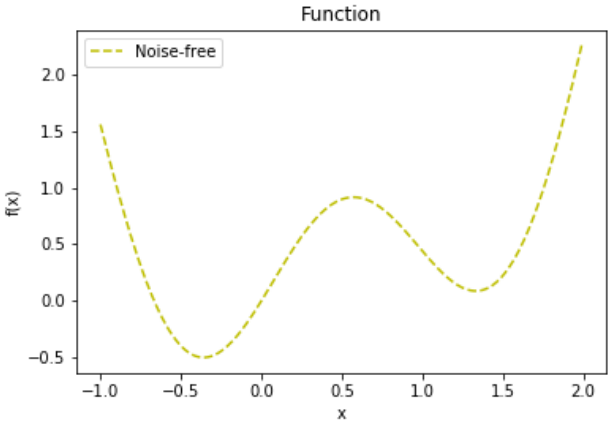

FIGURE 5.10: Function we wish to estimate

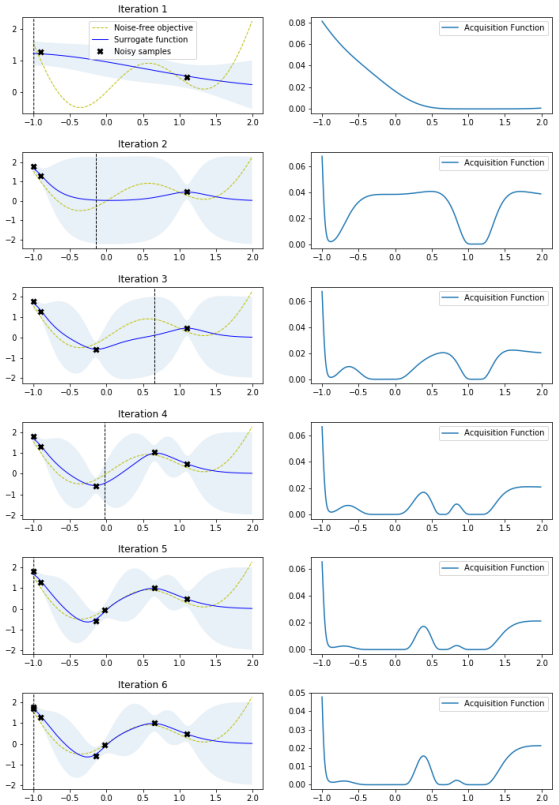

FIGURE 5.11: Basic Bayesian Optimization Algorithm

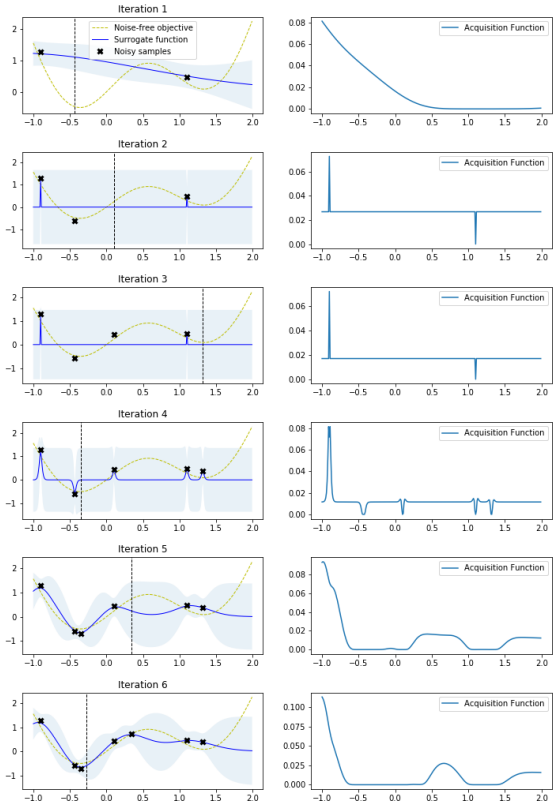

FIGURE 5.12: Proposed Bayesian Optimization Algorithm

Chapter 6

Conclusions and Future Work

6.1 Conclusion

1. To summarize, I did a through study of Optimal Learning theory and read latest state of the art papers in the field.
2. I also did a through study of Reinforcement learning and Markov Decision Process and implemented it for some popular games.
3. I read about Bayesian Optimization and presented a report on role of acquisition function in Bayesian Optimization.
4. Proposed an algorithm for faster minima of loss function using Bayesian Optimization and implemented its code.

6.2 Future Work

1. I am presently converting HJB paper ([12] and [6]) Matlab code to Python using tensor variables and will complete it soon.
2. Apply HJB paper to different loss functions using the above implementation and present a detailed report on it.
3. Explore more methods for better global convergence of HJB paper ([12] and [6]).

Bibliography

[exp] Exploration exploitation figure. https://jeongyoonlee.com/2017/08/09/transition-from-exploration-to-exploitation/. Accessed 11 March 2019.

[kra] Krasser, m. (2019). bayesian optimization - martin krasser's blog. [online]. http://krasserm.github.io/2018/03/21/bayesian-optimization/. Accessed 11 Mar. 2019.

[mar] Markov decision process figure. https://en.wikipedia.org/wiki/Markov_decision_process#/media/File:Markov_Decision_Process.svg. Accessed 20 April 2019.

[tra] Transfer learning figure. https://medium.com/kansas-city-machine-learning-artificial-intelligen/an-introduction-to-transfer-learning-in-machine-learning-7efd104b6026. Accessed 20 April 2019.

[5] Andrychowicz, M., Denil, M., Gómez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 3981–3989. Curran Associates, Inc.

[6] Arora, V., Behera, L., and Yadav, A. P. (2015). Optimal convergence rate in feed forward neural networks using hjb equation. *arXiv preprint arXiv:1504.07278*.

[7] Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

[8] Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., Botvinick, M., and de Freitas, N. (2017). Learning to learn without gradient descent by gradient descent. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 748–756. JMLR. org.

[9] Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

[10] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

[Rai] Rai, P. P. Reinforcement learning. Technical report, Indian Institute of Technology Kanpur.

[12] Reddy, T. K., Arora, V., and Behera, L. (2018). Hjb-equation-based optimal learning scheme for neural networks with applications in brain–computer interface. *IEEE Transactions on Emerging Topics in Computational Intelligence*, (99):1–12.

[13] Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.