Dissertation

Report on

Online Learning for Noisy Polynomial

By

Deepanshu Chauhan

21566005

Under the Guidance of

Prof. Manu Gupta

Department of Management Studies (DoMS) and Joint Faculty at Mehta Family School of Data Sciences and Artificial Intelligence

IIT Roorkee

Mehta Family School of Data Sciences and Artificial Intelligence

Indian Institute of Technology Roorkee

June 2023

CANDIDATE'S DECLARATION

I hereby declare that the work carried out in this seminar entitled **" Online Learning for Noisy Polynomial "** is presented in the partial fulfilment of the requirements for the award of degree of "Master in Technology" in Data Science, submitted to the Mehta Family School of Data Science and Data Science, Indian Institute of Technology Roorkee, under the guidance of **Prof. Manu Gupta**, Department of Management Studies (DoMS) and Joint Faculty at Mehta Family School of Data Science and Artificial Intelligence, IIT Roorkee. The matter embodied in this dissertation has not been submitted for the award of any other degree.

Prof. Manu Gupta
Assistant Professor
Department of Management Studies (DoMS)
and Joint Faculty at Mehta Family School of Data Science
and Artificial Intelligence
Indian Institute of Technology, Roorkee

Deepanshu Chauhan
21566005

Date: June 2, 2023

Acknowledgement

I would like to express my genuine gratitude and thanks to my respected supervisor **Prof. Manu Gupta**, Assistant professor, Department of Management Studies (DoMS) and Joint Faculty at Mehta Family School of Data Science and Artificial Intelligence, Indian Institute of Technology Roorkee, for his valuable guidance and consistent encouragement throughout the work. I also extend my heartfelt thanks to my family and friends for their good wishes.

Deepanshu Chauhan
(**21566005**)

Abstract

We study deep contextual bandits, a class of contextual bandits where each context-arm pair is associated with a feature vector having an unknown reward-generating function. We compared already available algorithms like LinUCB, Neural UCB, Neural LinUCB, and Neural Linear. Implemented Neural LinUCB algorithm on the real-world dataset, which has finite action, and modified the algorithm for infinite action cases to optimize an unknown polynomial function. Our algorithm does not use Deep Neural Network to optimize the unknown reward function, as in the case of Neural LinUCB; instead, it relies on the reinforcement learning algorithm only. The objective of algorithm is to learn the maxima of the reward generating function which is a polynomial but degree of polynomial is unknown to the agent. The proposed algorithm is able to learn reward generating function in most of the cases.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

This section will explore Reinforcement learning and provide a detailed explanation of contextual bandits and different algorithms.

Reinforcement learning is a part of Machine Learning in which an agent, which is basically a driver tries to solve the problem by selecting the best action among available actions, the agent learn the behavior through trial and error by interacting with an environment that is dynamic in nature. In other words, the agent tries to map different situations through actions to maximize the reward. Reinforcement learning is different from other forms of machine learning types i.e., supervised learning and unsupervised learning. In supervised learning, the agent (supervised learning model) learns how to act on labelled data provided by some external supervisor and learns the hidden pattern in the data to predict answers for unseen data. On the other hand, reinforcement learning is a sequential problem. It's generally impossible for an agent to face such examples representing all possible situations. The agent must learn from their experience by interacting with the dynamic environment. On other hand, unsupervised learning's primary goal is finding the hidden pattern in unlabelled data, which looks similar to reinforcement learning. Still, the purpose of reinforcement learning is to maximize reward, unlike unsupervised learning concern only about the hidden structures in the data.

Moreover, the difference that distinguishes reinforcement learning problems from other learning types is finding the optimal trade-off between exploration and exploitation. The agent should be able to exploit the knowledge it has about the environment to maximize its reward. Also, it should explore enough to choose the best action in the future. The main task is to perform these two operations simultaneously. If the agent does not explore enough then it may choose sub-optimal action every time or the agent explores most of the time then it may able to exploit the agent current knowledge and loose reward for exploring. So, the trade-off between exploration and exploitation will be such that the agent explores enough so that it will be able to choose optimal action as soon as possible.

Some examples of reinforcement learning could be:

- Playing game: To win any game, the player must learn the best move to make in every situation to beat the opponent
- Drive cars: To drive a car, the driver must sense his environment and make necessary decisions based on the current situation
- Preparing breakfast: Any person who wants to prepare breakfast faster must learn the sequence of moves after observing the actual situation and objects' location in the kitchen. Moreover, he must know the best time for warming milk or toasting bread based on his preferences

All above situations involve interactions between an active decision-making agent and its dynamic environment, within which the agent tries to achieve a goal despite having uncertainties about its environment. The agent's current action can affect the future state of the environment and their effects cannot be fully predicted, hence there must be frequent monitoring of environment by agent to react on any uncertain situations. The knowledge of the environment's state is also useful to improve performances of algorithm over time.

Reinforcement learning can be used in the situation where we can not map every possible situation, like Large Language Models (LLMs) which are hot research topics these use Reinforcement Learning Algorithms for training their model. Reinforcement learning have caught the eye of machine learning researchers in the last few years and have found interactions with other engineering and scientific disciplines as well as applications in psychology and neuroscience.

1.2 Multi-Armed Bandit Algorithm

An important challenge of reinforcement learning is to find a way of balancing exploration of unknown action's value and exploration of current knowledge. In fact, it evaluates the actions that the algorithm tries rather than learning from information about the correct actions.

In the multi-armed bandit problem, expected reward is associated to each of the k possible actions. If A_t is action selected at time step t and its reward is R_t, then value of an arbitrary action is

$$Q_*(a) = E[R_t|A_t = a] \tag{1.1}$$

If the agent knows all action's value, then it would always choose the arm with highest value and would solve the problem. However, these values are uncertain so, it's impossible for the agent to know the exact action value. The agent knows only the expected value (estimated) value.

If the agent knows the value estimate for all the actions at any time step t and choose the action having maximum estimated value, then this action is called greedy action. Whenever the agent chooses a greedy action then it is said to exploit its current knowledge, otherwise, when the agent chooses an action in non-greedy manner, it is said to explore the environment in order to get good estimations. Exploitation is a way to maximize the immediate expected reward, while exploration allows the agent to get the better estimation of every action for future exploitation.

This is the well-known conflict between exploration and exploitation and its solution is different in any specific case and it depends on the precise values of the estimates, uncertainties and the number of remaining steps. Balancing exploration and exploitation can be done in many possible ways that are simple or more sophisticated. The following sub-section will review some famous bandit algorithms.

1.2.1 Epsilon Greedy Algorithm

The simplest action selection rule is to select at each time step, either selecting an arm with current highest estimated action value. This is a greedy action selection method

$$A_t = \underset{a}{\text{argmax}} Q_t(\text{a}) \tag{1.2}$$

It chooses the greedy action most of the time, but occasionally with a small probability ϵ, to select random actions from all the actions, independent of the current action value estimate. This rule is called ϵ-greedy action selection method. With this method, it's possible to exploit current knowledge most of the time and to explore other non-greedy actions the rest of the time, in order to improve the knowledge for the subsequent exploitation step.

This method is based on the average of the observed rewards are efficient in a stationary environment, if the environment is dynamically changing over time, i.e., non-stationary. Then in that case, it becomes important to give more weight to recent rewards more than the past ones, the past observation do not reflect actual state as well as recent observations. This could be done by using exponential decay of weights.

Algorithm 1 ϵ-Greedy Algorithm for Multi-Armed Bandit

```
1: Initialize for $a = 1$ to $k$:
2:     $Q(a) \leftarrow 0$
3:     $N(a) \leftarrow 0$
4: repeat
5:         $A \leftarrow \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ \arg\max_a Q(a) & \text{with probability } 1-\epsilon \end{cases}$
6:         $R \leftarrow \text{bandit}(A)$
7:         $N(A) \leftarrow N(A) + 1$
8:         $Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$
9:     for time step $t$
```

1.2.2 Upper Confidence Bound (UCB) Action Selection

Improvement for greedy methods is to select among the non-greedy actions, the action with maximum potential for being optimal, observing uncertainties in the estimates and how the estimates are close to being maximal. This can be done by selecting actions as:

$$A_t = \underset{a}{\text{argmax}} \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{1.3}$$

where $N_t(a)$ is the number of times the action a is chosen prior to time t and c is a parameter controlling the degree of exploration. This positive constant determines the trade-off between exploration and exploitation.

Its idea is to consider, in maximization, a measure of uncertainty in the estimate of an action value, captured by the square-root term. UCB algorithm performs better than epsilon-greedy algorithm, but it is more difficult to generalize it to a general setting, beyond the multi-armed bandit problem.

1.2.3 Thompson Sampling

Thompson sampling approaches the problem of multi-arm bandit model in slightly different manner. A probability model is built from the reward obtained, then action is chosen after sampling from obtained rewards. Generally, beta distribution works well, beta distribution takes only two parameters which are number of successes of arm at time t and number of failures of arm at time t. Then sample data from this belief distribution for each arm and then finally choose arm greedily with respect to sample.

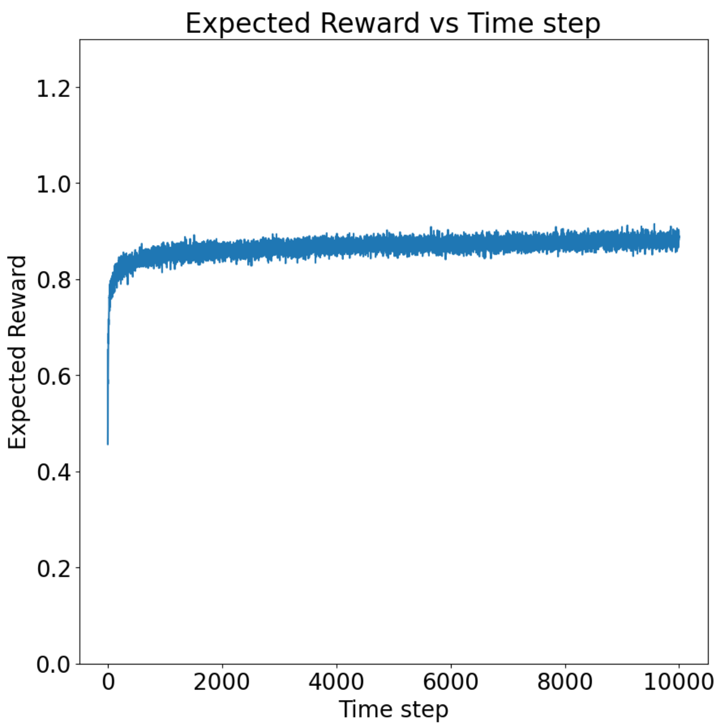


Figure 1.1: Expected reward vs time

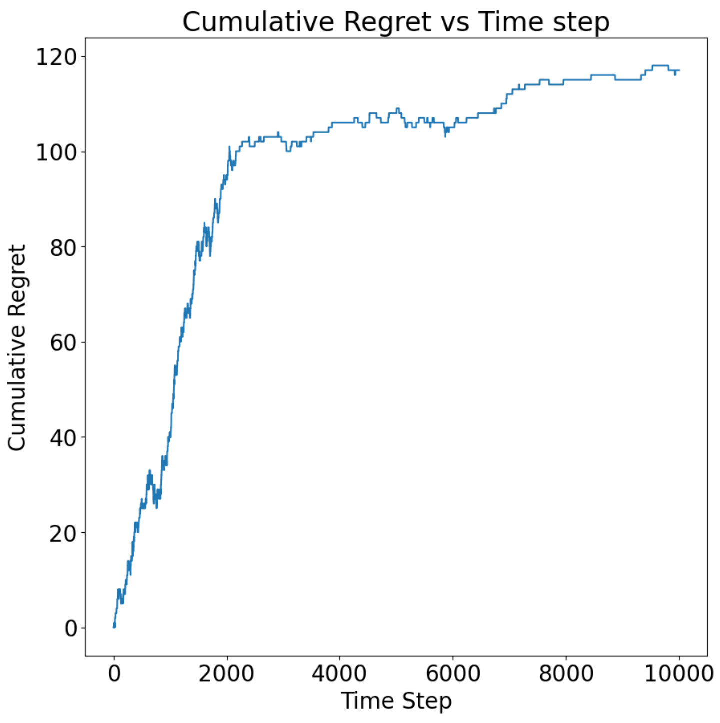


Figure 1.2: Cumulative Regret vs time

1.3 Contextual Bandits

The contextual bandit is a variant of the extensively studied multi-armed bandit problem which has been widely studied both in theory and practice (Abbasi-Yadkori et al., 2011; Chu et al., 2011). Contextual bandits as well as non-contextual bandits involve making a sequence of decisions on which the agent must choose action from an action space A. After choosing an action, a stochastic reward r is revealed for the chosen arm only. In a contextual bandit, before making each decision the bandit is shown some context $x \in X$. Asymptotically optimal approaches to trading-off between exploration and exploitation for the non-contextual muti-armed bandit can be derived. But deriving such solutions in the contextual case has resulted in simplifying assumptions such as assuming a linear relationship between context and the reward. These simplifying assumptions are often unsatisfactory.

At each time step the agent is presented with some context $x \in X$ from the environment. The agent chooses an action $a \in A$ from a set of possible actions $\{a_1, a_2, ..., a_m\}$. Every time the agent takes an action it receives a real-valued reward r, however it does not observe a reward for non-taken actions. The goal of the agent is to maximize the cumulative reward over time.

The agent builds a model for $P(r \mid a, x)$ from observed (x,a,r) triplets. But the agent only observes triplets involving actions a it has taken, so the agent drives its own data generation process. Thus, it is necessary for the agent to trade-off exploration and exploitation. If the agent learns early on to associate higher expected rewards with a particular action, it may not choose the arm with highest expected reward in new, unseen context. Also, the agent cannot take actions randomly as it will forgo higher expected reward for known best actions in particular contexts.

1.3.1 Deep Contextual Bandits

In deep contextual bandits a neural network is used to model $P(r \mid a, x : w)$, where w are network weights. At each time step a context x is presented with a context x and set of possible actions $\{a_1, a_2, ..., a_m\}$. Unroll the m actions into (x, a_i) pairs which are passed to the network for inference. Choose the action a, whose (x_i, a_i) pair has the highest expected reward.

Initially, when the amount of data is small, the uncertainty on the network weighs is high. As the model observes more (x,a,r) triplets, the uncertainty on the weights decreases and the network explores less and less, converging to an almost purely exploitative

Algorithm 2 Deep Contextual Multi$_{a}rmedBandits$

```
1: $nextRetrain \leftarrow N$ :
2: while True do
3:     for $i \leftarrow 1$ to $nextRetrain$ do
4:         receive context $x$ and possible actions $\{a_1, a_2, \ldots, a_m\}$
5:         compute corresponding weights $w_d$
6:         for $a \in \{a_1, a_2, \ldots, a_m\}$ do
7:             Compute predicted reward $r = f(x, a; w_d)$
8:         Choose action $a$ with the highest predicted reward
```

strategy.

1.4 Dynamic pricing as multi-arm bandit problem

One of the possible applications of multi-armed bandit model is the dynamic pricing which generally arises in the online market because of the uncertain environment. The objective for the companies and retailers is to adjust to these uncertain environments and change the price of products and services accordingly in order to maximize the revenue. With the boom of internet usage in the market, it has enabled many industries to start their business online and with increase of competitor in the e-commerce and retail market, the right price to offer the customer became a headache for them. The company wants to increase their revenue and static pricing, which is generally used. It does not work especially in the e-commerce because static pricing does not simulate the demand/price curve fig 1.3, nor does it allow a firm to respond to change in price of the competitor.

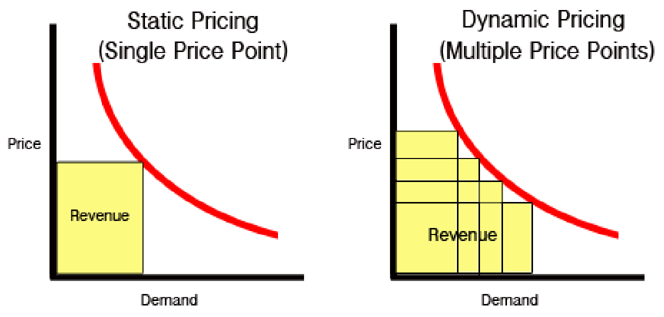


Figure 1.3: Static vs dynamic Pricing (link)

There can be two approaches for dynamic pricing, one is passive learning and other is active learning. In passive learning, the historical data is used to learn the underlying parameters like demand, inventory, price etc, using this historical data a model is built to estimate the parameters and obtained optimal price is used to get the next day optimal price. This process is repeated for next few days and the above steps are repeated. The problem with these approaches is that such methods try to optimize for short term without even trying to learn the demand. Whereas in active learning, there is no fixed policy, and the agent must learn the optimal policy, it views dynamic pricing as an optimization problem under uncertainty along with learning the price elasticity of demand. In general, elasticity is a measure of sensitivity of variable to change in sensitivity of another variable. Here, price elasticity measures the degree to which a consumer varies their demand according to the changes of price.

In this multi-arm bandit model for dynamic pricing, the problem is to maximize the rewards which is revenue generated from each basket of the product, arms here are the products or basket of product. The arms are correlated to each other with non-stationary reward function.

In today's world, the dynamic pricing is not only used by e-commerce giants but also used by small retailers, but dynamic pricing algorithm differs for small retailers as compared to the big e-commerce companies. Retailers cannot change the price of the item so easily because it can cause lower customer satisfaction and lead to the loss of loyal customer. There are many obstacles for retailers to apply the dynamic pricing, one is unavailability of the large volume of sales data which is very hard to get for every product because each product does not have high sales volume. The learning happens at attribute/features level not at product level in dynamic pricing for retailers, this helps the retailer to choose prices of subsequent products that share similar features.

The algorithm used in the dynamic pricing can also be used in the decision making when the environment is uncertain, and the goal is to maximize the objective function. There are many applications of type of algorithm in recommender system, advertisement on websites, etc. One of the main concerns in broad problem is the length of the learning phase, there must be enough exploration so that the agent can gather enough information to make correct decision. Therefore, there must be a trade-off between exploration and exploitation. The passive learning approach only focuses on the exploitation which results in loss of revenue in the long run. There are other approaches for multi-arm bandit model like The Upper Confidence (UCB) in which reward for each arm a confidence bounds are constructed and the arm with highest upper confidence bound is chosen, it is very hard to implement for complex problems like decision making under uncertainty. Another algorithm is Thompson Sampling (TS) which uses Bayes rule for updating the model, the arm with equal probability as probability of optimal arm is chosen. Thompson Sampling and Upper Confidence Bound algorithm are consistent algorithm but Thompson Sampling is easier to implement as compared to the UCB algorithm for complex problem because Thompson sampling can deal with a wide range of data models that go beyond just looking at individual rewards.

The report is organized as follows. In next, we describe the previous works which took place in the field of dynamic pricing then we move to the working example of our algorithm and its comparison with other already available algorithm and the conclude along with outline of the future work.

Chapter 2

Literature Review

2.1 General

Before the advancement in the field of Artificial Intelligence, particularly in the area of Artificial Neural Network (ANN), the contextual bandits had been used and, in some cases, the contextual bandits are still being used. Existing solutions can model either linearly, which enables uncertainty driven exploration, or non-linearly, by using epsilon-greedy exploration policies. The problem arises with contextual bandits that it was not able to model complex function. ANN is the powerful technique and can model almost all unknown function easily by increasing the number of hidden layers.

(Collier and Llorens, 2018), introduces Deep Contextual Multi-Armed Bandits, which is a integration of Reinforcement learning techniques with ANN, they tackle the dilemma of exploration-exploitation trade-off through Thompson sampling from the posterior over the weights of a Bayesian Neural Network. In order to adjust the level of exploration automatically as more data is made available to the model, the dropout rate is learned rather than considered a hyper-parameter. In the model, they used a neural network to model $P(r|a, x; w)$, where w are the network weights and train he network using Concrete Dropout and do not disable dropout at inference time. At each time step, a context x and a set of possible actions $\{a_1, a_2, \ldots, a_m\}$. Sample dropout noise d from a uniform distribution, $U(0, 1)$, which is used to sample from the posterior over the weights w. We unroll the m actions into $(x, a_i,)$ pairs which are passed to the network for inference. Choose the action a whose (x, a_i) pair has the highest expected reward. Theoretically for Thompson sampling we should update the model after each observation. In practice, real time online learning is unsuitable for industrial settings, so retrain the model on an exponential scale in the number of data points.

The above algorithm is used for Mushroom Task, the data-set contains 8,124 mushrooms classified as poisonous or edible, with up to 22 features for each mushroom. At each time step, the agent must decide whether to eat it or not. The agent gets reward of 1 for eating an edible mushroom and a reward of 0 for eating a poisonous mushroom, and with probability p a reward of 1 for not eating a mushroom and p is set to 0.5.

As expected, the epsilon-greedy agent accumulates regret linearly, due to taking random actions 5% of the time. Both fixed rate dropout and Concrete dropout agents rapidly learn to "solve" the task, but concrete dropout not only has a much lower final cumulative regret than the fixed rate dropout agent (less than half) but it also learns to stop exploring earlier. The discontinuities in the graph are due to fact that it was retrain on an exponential scale.

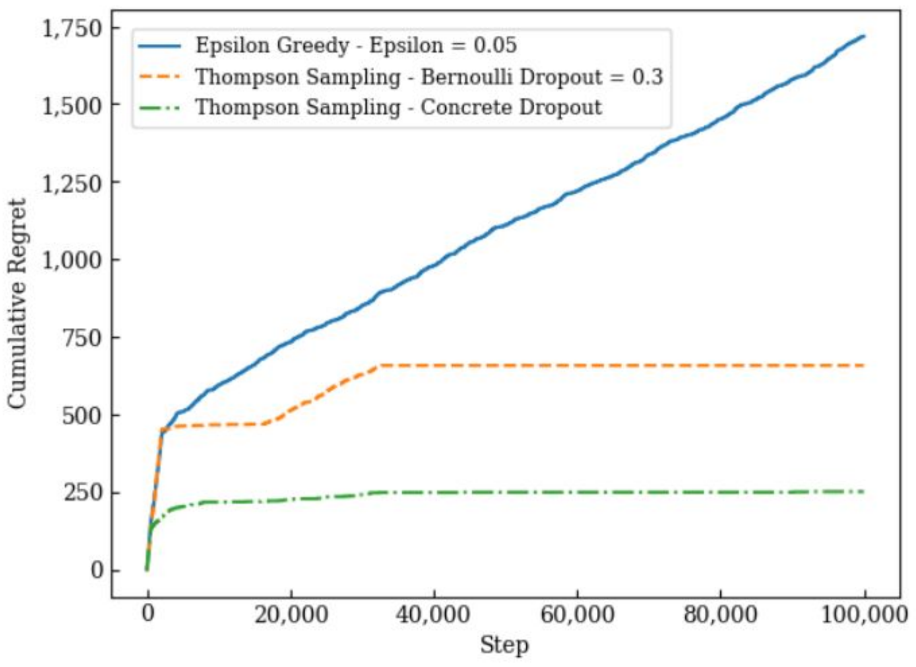


Figure 2.1: Mushroom Task - comparison between epsilon greedy, Thompson sampling with Bernoulli dropout

2.2 Contextual Bandits for News Recommendation

Personalized web services strive for to adapt their services (advertisements, new, articles, etc) to individual users by making use of both content and user information. There are mainly two challenges for personalised news recommendation. First, web service is featured with dynamically changing pools of content, rendering traditional collaborative filtering methods inapplicable. Second, the scale of most web services of practical interest calls for solutions that are both fast in learning and computation.

(Li et al., 2010) model personalized recommendation of news articles for Yahoo! as a contextual bandit problem, a principled way to learn algorithm sequentially selects articles to serve users based on contextual information about the users and articles to serve users based on contextual information about the users and articles, while simultaneously adapting its article-selection strategy based on user-click feedback to maximize total user clicks.

Often, both users and content are represented by sets of features. User features may include historical activities at an aggregated level as well as declared demographic information. Content features may contain descriptive information and categories. In scenario, exploration and exploitation must be deployed at an individual since the views of different users on the same content can vary significantly. Since there may be a very large number of possible choices or actions available, it becomes critical to recognize commonalities between the content items and to transfer that knowledge across the content pool.

2.2.1 Formulation as Multi-Armed Bandit

Algorithm 3 LinUCB with disjoint linear models

```
1:  Inputs: $\alpha \in \mathbb{R}^+$
2:  for $t = 1, 2, 3, \dots, T$ do
3:      Observe features of all arms $a \in A_t$: $x_{t,a} \in \mathbb{R}^d$
4:      for $a \in A_t$ do
5:          if $a$ is new then
6:              $A_a \leftarrow I_d$ (d-dimensional identity matrix)
7:              $b_a \leftarrow \mathbf{0}_{d\times 1}$ (d-dimensional zero vector)
            end if
8:          $\hat{\theta}a \leftarrow A_a^{-1} b_a$
9:          $p_{t,a} \leftarrow \hat{\theta}_a^T x_{t,a} + \alpha\sqrt{x_{t,a}^T A_a^{-1} x_{t,a}}$
        end for
10:     Choose arm $a_t = \arg\max_{a \in A_t, p_{t,a}}$ with ties broken arbitrarily, and observe a real-
        valued payoff $r_t$
11:     $A_{a_t} \leftarrow A_{a_t} + x_{t,a_t} x_{t,a_t}^T$
12:     $b_{a_t} \leftarrow b_{a_t} + r_t x_{t,a_t}$
    end for
```

The problem of personalized news article recommendation can be naturally modelled as a multi-armed bandit problem with context information. A contextual-bandit algorithm A proceeds in discrete trials $t = 1, 2, 3, \dots$. In t trial:

1. The algorithm observes the current user u_t and a set A of arms or actions together

with their feature vectors $x_{t,a}$ for $a \in A_t$. The vector $x_{t,a}$ summarizes information of both the user u_t and arm a, and will be referred to as the context.

2. Based on observed payoffs in previous trials, A chooses an arm $a_t \in A_t$ receives payoff r_{t,a_t} whose expectation depends on both the user u_t and the arm a_t
3. The algorithm then improves its arm-selection strategy with the new observation, $(x_{t,a_t}, a_t, r_{t,a_t})$. It is important emphasize here that no feedback is observed for unchosen arms $a \neq a_t$

2.3 Neural UCB Algorithm

Neural networks are used to model the unknown reward function. (Zhou et al., 2020) proposes a algorithm called as Neural UCB which leverages the representation power deep neural networks and uses a neural network based random feature mapping to construct an upper confidence bound (UCB) of reward for efficient exploration. They prove that under standard assumptions neural UCB achieves $\mathcal{O}(\sqrt{T})$ regret where T is number of rounds. It is the first neural network based contextual bandit algorithm weather near optimal regret guarantee.

Most studied model in the literature is linear contextual bandit, which assumes that the expected reward at each wrong is linear in the vector. While linear reward assumption open fails to hold in practice, which motivates the study of non-linear or non-parametric contextual bandits.

In order to overcome the above shortcomings, deep neural networks (DNNs) have been introduced to learn the underlying reward function in contextual bandit problem, thanks to their strong representation power. We call these approaches collectively as neural contextual bandit algorithms. Given the fact that DNNs enable the agent to make use of non-linear models with less domain knowledge, existing work (Riquelme et al., 2018) study neural-linear bandits. That is, they use all but the last layers of DNN as a feature map, which transforms contexts from the raw input space to one low dimension space, usually with better representation and less frequent updates. Then they learn a linear exploration policy on top of the last hidden layer off the DNN with more frequent updates. These attempts have achieved great empirical success, but no regret guarantees are provided.

2.3.1 Algorithm

The key idea of Neural UCB is to use a neural network $f(x;\theta)$ to predict the reward of context x, and upper confidence bounds computed from the network to guide exploration.

Initialisation

It initializes the network by randomly generating each entry of θ from an appropriate Gaussian distribution: for $1 \leq l \leq L-1$, W_l is set to be $\begin{pmatrix} \mathrm{W} & 0 \\ 0 & \mathrm{W} \end{pmatrix}$ (here l is current neural layer), where each entry of W is generated independently from $N(0, 4/m)$: W_l is set to $(w^T, -w^T)$, where each entry of w is generated independently from N(0,2/m).

Learning

At round t, Neural UCB algorithm observes the contexts from all the actions, $\{x_{t,a}\}_{a=1}^{k}$. First, it computes an upper confidence bound $U_{t,a}$ for each action a, based on $x_{t,a}$, θ_{t-1} (the current neural network parameter) and a positive scaling factor γ_{t-1}. It then chooses action a_t with the largest $U_{t,a}$ and receives the corresponding reward r_{t,a_t}. At the end of round t, NeuralUCB update θ_t by applying weight updating algorithm to (approximately) minimize $L(\theta)$ using gradient descent, and updates γ_t. They choose gradient descent for the simplicity of analysis, although the training method can be replaced by stochastic gradient descent with a more involved analysis.

Algorithm 4 Neural UCB

```
1:  **Input:** Number of rounds $T$, regularization parameter $\lambda$, exploration parameter $\nu$,
    confidence parameter $\delta$, norm parameter $S$, step size $\eta$, number of gradient descent
    steps $J$, network width $m$, network depth $L$.
2:  **Initialization:** Randomly initialize $\theta_0$ as described in the text.
3:  Initialize $Z_0 = \lambda I$.
4:  **for** $t = 1$ to $T$ **do**
5:      Observe $\{x_{t,a}\}_{a=1}^{k}$
6:      **for** $a = 1$ to $K$ **do**
7:          Compute $U_{t,a} = f(x_{t,a};\theta_{t-1}) + \gamma_{t-1}\sqrt{g(x_{t,a};\theta_{t-1})Z_{t-1}^{-1}g(x_{t,a};\theta_{t-1})/m}$
8:          Let $a_t = \arg\max_{a\in[K]} U_{t,a}$
        **end for**
9:      Play $a_t$ and observe reward $r_{t,a_t}$
10:     Compute $Z_t = Z_{t-1} + g(x_{t,a_t};\theta_{t-1})g(x_{t,a_t};\theta_{t-1})^\top/m$
11:     Let $\theta_t = \mathrm{TrainNN}(\lambda, \eta, J, m, \{x_i, a_i\}_{i=1}^{t}, \{r_i, a_i\}_{i=1}^{t}, \theta_0)$
12:     Compute $\gamma_t = \sqrt{1 + C_1 m^{-1/6}\sqrt{\log m}L^4 t^{7/6}\lambda^{-7/6}}$
        $.(\nu\sqrt{\log\frac{\det Z_t}{\det \lambda I} + C_2 m^{-1/6}\sqrt{\log m}L^4 t^{5/3}\lambda^{-1/6} - 2\log\delta} + \sqrt{\lambda}S)\ (\lambda + C_3 tL)[(1 -$
        $\eta m\lambda)^{J/2}\sqrt{t/\lambda} + m^{-1/6}\sqrt{\log m}L^{7/2}t^{5/3}\lambda^{-5/3}(1 + \sqrt{t/\lambda})]$.
```

Algorithm 5 Gradient Descent

```
1: Input: Regularization parameter $\lambda$, step size $\eta$, number of gradient descent steps $U$,
   network width $m$, contexts $\{x_{i,a_i}\}_{i=1}^t$, rewards $\{r_{i,a_i}\}_{i=1}^t$, initial parameter $\theta^{(0)}$
2: Define $L(\theta) = \sum_{i=1}^t \left(f(x_i, a_i; \theta) - r_{i,a_i}\right)^2/2 + m\lambda\|\theta - \theta^{(0)}\|_2^2/2$
3: for $j = 0$ to $J - 1$ do
4:     $\theta^{(j+1)} = \theta^{(j)} - \eta\nabla L(\theta^{(j)})$
   end for
5: Return $\theta^{(J)}$
```

2.4 Neural LinUCB Algorithm

They propose a novel learning algorithm that transforms the raw feature vector using the last hidden layer of a deep ReLU (Rectified Linear Activation Unit) neural network (deep representation learning), and uses an upper confidence bound (UCB) approach to explore in the last linear layer (shallow exploration). They prove that under standard assumptions, our proposed algorithm achieves $O(\sqrt{T})$ finite time regret, where T is the learning time horizon. When compared with existing neural contextual bandit algorithms, this approach is computationally much more efficient since it only needs to explore in the last layer of the deep neural network.

Neural LinUCB Algorithm main contributions are as follows:

1. They propose a contextual bandit algorithm, Neural-LinUCB, for solving a general class of contextual bandit problems without knowing the specific reward generating function. The proposed algorithm learns a deep representation to transform the raw feature vectors and performs UCB-type exploration in the last layer of the neural network, which we refer to as deep representation and shallow exploration. Compared with LinUCB and neural bandits such as NeuralUCB and NeuralTS, this algorithm enjoys the best of two worlds: strong expressiveness due to the deep representation and computational efficiency due to the shallow exploration.

2. Despite the usage of a DNN as the feature mapping, they prove a $O(\sqrt{T})$ regret for the proposed Neural-LinUCB algorithm, which matches the regret bound of linear contextual bandits. This is the first work that theoretically shows the convergence of bandit algorithms under the scheme of deep representation and shallow exploration. It is notable that a similar scheme called Neural-Linear was proposed by (Riquelme et al., 2018) for Thompson sampling algorithms, and they empirically showed that decoupling representation learning, and uncertainty estimation improves the performance. This work confirms this observation from a theoretical perspective.

3. They also conduct experiments on contextual bandit problems based on real-world data-sets, demonstrating a better performance and computational efficiency of Neural-LinUCB over LinUCB and NeuralUCB, which well aligns with their theory

2.4.1 Neural Tangent Kernel

There is a line of related work to this work on the recent advance in the optimization and generalization analysis of deep neural networks. (Jacot et al., 2018) first introduced the neural tangent kernel (NTK) to characterize the training dynamics of network outputs in the infinite width limit. From the notion of NTK, a fruitful line of research emerged and showed that loss functions of deep neural networks trained by (stochastic) gradient descent can converge to the global minimum. The generalization bounds for over-parameterized deep neural networks are also established in (Arora et al., 2019), (Allen-Zhu et al., 2019), (Cao and Gu, 2019). Recently, the NTK based analysis is also extended to the study of sequential decision problems including bandits, and reinforcement learning algorithms.

2.4.2 Deep Representation and Shallow Exploration

The linear parametric form in linear contextual bandits might produce biased estimates of the reward due to the lack of representation power. In contrast, it is well known that deep neural networks are powerful enough to approximate an arbitrary function. Therefore, a natural extension of linear contextual bandits is to use a deep neural network to approximate the reward generating function $r(.)$. Nonetheless, DNNs usually have a prohibitively large dimension for weight parameters, which makes the exploration in neural networks based UCB algorithm inefficient.

In this work (Xu et al., 2020), they study a neural contextual bandit algorithm, where the hidden layers of a deep neural network are used to represent the features and the exploration is only performed in the last layer of the neural network. In particular, they assume that the reward generating function r$(\cdot)$ can be expressed as the inner product between a deep represented feature vector and an exploration weight parameter, namely $r(.) = \langle \theta^*, \psi(\cdot) \rangle$, where $\theta^* \in \mathbb{R}^d$ is some weight parameter and $\psi(\cdot)$ is unknown feature mapping. This decoupling of the representation and the exploration will achieve the best of both worlds: efficient exploration in shallow (linear) models and high expressive power of deep models. To learn the unknown feature mapping, they propose to use a neural network to approximate it. In what follows, they describe a neural contextual bandit algorithm that uses the output of the last hidden layer of a neural network to transform the raw feature vectors (deep representation) and performs UCB-type exploration in the last layer of the neural network (shallow exploration). Since the exploration is performed only in the last linear layer, they call this procedure Neural-LinUCB.

Specifically, in round t, the agent receives an action set with raw features $X_t = \{x_{t,1}, x_{t,2}, \ldots, x_{t,k}\}$. Then the agent chooses an arm a_t that maximizes the following upper confidence bound:

$$a_t = argmax_{k\in[K]} \left\{ \langle \phi(x_{t,k}; w_{t-1}), \theta_{t-1} \rangle + \alpha_t \|\phi(x_{t,k}; w_{t-1})\|_{A_{t-1}^{-1}} \right\} \tag{2.1}$$

where θ_{t-1} is a point estimate of the unknown weight in the last layer $\phi(x, w)$ is defined as w_{t-1} is an estimate of all the weight parameters in the hidden layers of the neural network, $\alpha_t > 0$ is the algorithmic parameter controlling the exploration, and A_t is a matrix defined based on historical transformed features:

$$A_t = \lambda I + \sum_{i=1}^{T} \phi(x_{i,a_i}; w_{i-1}) \phi(x_{i,a_i}; w_{i-1})^\top \tag{2.2}$$

and $\lambda > 0$. After pulling arm a_t, the agent will observe a noisy reward $\hat{r}_t := \hat{r}(x_{t,a_t})$ defined as:

$$\hat{r}(x_{t,a_t}) = r(x_{t,a_t}) + \xi_t, \tag{2.3}$$

where ξ_t is an independent ν-sub Gaussian random noise for some $\nu > 0$ and $r(.)$ unknown reward function.

Upon receiving the reward $\hat{r}_t$, the agent updates its estimate θ_t of the output layer weight by using the same $\mathcal{L}^2$-regularized least-squares estimate in linear contextual bandits (Zhou et al., 2020). In particular, we have

$$\theta_t = A_t^{-1} b_t \tag{2.4}$$

Our goal is to minimize the following empirical loss function:

$$L_q(w) = \sum_{i=1}^{qH} (\theta_i^\top \phi(x_{i,a_i}; w) - \hat{r}_i)^2 \tag{2.5}$$

2.4.3 Comparison with LinUCB and Neural UCB

The high-level idea of decoupling the representation and exploration in this algorithm is also like that of the Neural-Linear algorithm, which trains a deep neural network to learn a representation of the raw feature vectors, and then uses a Bayesian linear regression to estimate the uncertainty in the bandit problem. However, these two algorithms are significantly different since Neural-Linear is a Thompson sampling-based algorithm that uses posterior sampling to estimate the weight parameter θ^* via Bayesian linear regression, whereas Neural-LinUCB adopts upper confidence bound based techniques to estimate the weight θ^*. Nevertheless, both algorithms share the same idea of deep representation and shallow exploration, and they view this Neural-LinUCB algorithm as one instantiation of the Neural-Linear scheme.

amsmath algorithm algpseudocode

Algorithm 6 Deep Representation and Shallow Exploration (Neural-LinUCB)

```
1: **Input:** regularization parameter $\lambda > 0$, number of total steps $T$, episode length $H$,
   exploration parameters $\{\alpha_t > 0\}_{t\in[T]}$
2: **Initialization:** $A_0 = \lambda I$, $b_0 = 0$; entries of $\theta_0$ follow $N(0, 1/d)$, and $w^{(0)}$ is initialized
   as described; $q = 1$; $w_0 = w^{(0)}$
3: **for** $t = 1, \ldots, T$ **do**
4:     receive feature vectors $\{x_{t,1}, \ldots, x_{t,K}\}$
5:     choose arm $a_t = \arg\max_{k\in[K]} \theta_{t-1}^\top \phi(x_{t,k}; w_{t-1}) + \alpha_t \|\phi(x_{t,k}; w_{t-1})\|_{A_{t-1}^{-1}}$, and obtain
   reward $\hat{r}_t$
6:     update $A_t$ and $b_t$ as follows:
7:         $A_t = A_{t-1} + \phi(x_{t,a_t}; w_{t-1})\phi(x_{t,a_t}; w_{t-1})^\top$,
8:         $b_t = b_{t-1} + \hat{r}_t\phi(x_{t,a_t}; w_{t-1})$,
9:     update $\theta_t = A_t^{-1} b_t$
10:    **if** $\mathrm{mod}(t, H) = 0$ **then**
11:        $w_t \leftarrow$ output of Algorithm 7
12:        $q = q + 1$
13:    **else**
14:        $w_t = w_{t-1}$
15: **Output:** $w_T$
```

Algorithm 7 Update Weight Parameters with Gradient Descent

```
1: **Input:** initial point $w_q^{(0)}, = w^{(0)}$, maximum iteration number $n$, step size $\eta_q$, and loss
   function.
2: **for** $s = 1, \ldots, n$ **do**
3:     $w_q^{(s)} = w_q^{(s-1)} - \eta_q \nabla_w L_q(w_q^{(s-1)})$
4: **end for**
5: **Output:** $w_q^{(n)}$
```

2.4.4 Experimental Comparison

Data-set: They evaluate the performances of all algorithms on bandit problems created from real-world data. Specifically, following the experimental setting in (Zhou et al., 2020)., they use data-sets (Shuttle) Statlog, Magic and Covertype from UCI machine learning repository, and the MINST data-set from (LeCun et al., 1998). Each instance represents a feature vector $x \in \mathbb{R}^d$, that is associated with one of the K arms, and dimension d is the number of attributes in each instance.

	Statlog	Magic	Covertype	MNIST
Number of attributes	9	11	54	784
Number of arms	7	2	7	10
Number of instances	58,000	19,020	581,012	60,000

Table 2.1: Specifications of data-sets from the UCI machine learning repository

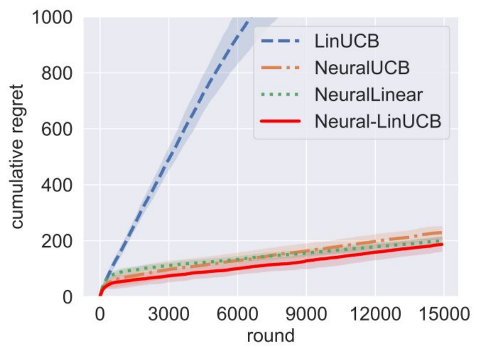
2.5 Barycentric Spanner

Barycentric spanner term is usually used in computational geometry to describe a geometric structure that can approximates the pairwise distances between set of points on a metric scale. This method most commonly used where the exact pairwise distances between all points are expensive or impractical to compute.

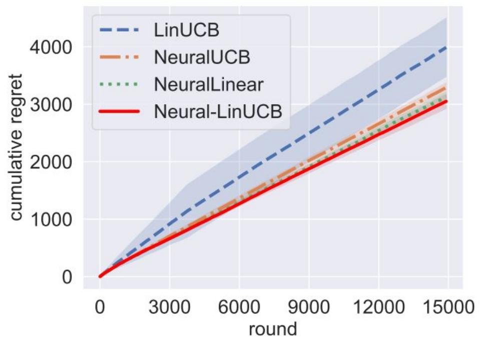
In this method, a subset of points, called as landmarks, is selected from given set of points. The spanner is constructed in such a way that any pair of points in the original set, their distance in the spanner is not much larger than the actual distance in the metric space. The construction of a barycentric spanner involves dividing the metric space into several regions, with each regions assigned to a particular landmark. The goal is to assign each point to its closest landmark, minimizing the distortion of distances.

It plays a crucial role contextual bandits problems by providing a efficient way for exploration-exploitation split. By constructing a barycentric spanner over context space, we can easily approximate the distances between the contexts. By doing this, we can identify similar or related contexts, which can guide the exploration and exploitation strategy of the bandit problem.

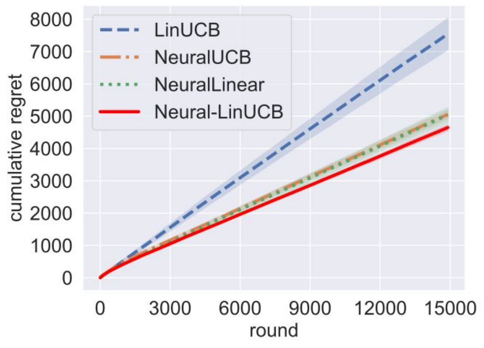
Using barycentric spanners in contextual bandit algorithms reduces the dimensionality of the context space by replacing the original context space with a smaller set of landmarks. This reduction in the dimensionality simplifies the exploration process and makes more computationally efficient.



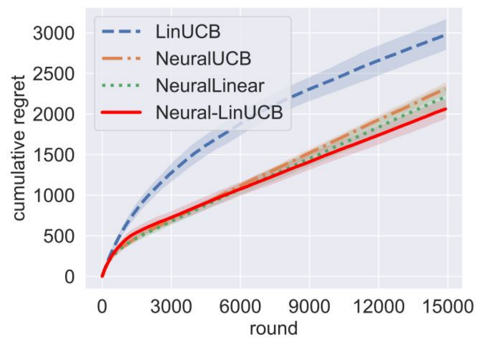
(a) Statlog



(b) Magic



(c) Covertype



(d) MNIST

Figure 2.2: The cumulative regrets of LinUCB, NeuralUCB, Neural-Linear and Neural-LinUCB over 15, 000 rounds. Experiments are averaged over 10 repetitions (Xu et al., 2020)

Chapter 3

Our Algorithm

This section includes the working procedure of our algorithm and other baseline algorithms, such as Iterated Least Square (ILS) and Constrained Iterated Least Square (CILS), for comparison on different sets of examples, which will be discussed in the next chapter.

3.1 Baseline Algorithms

There are two algorithms that we consider as baseline algorithms: (i) Iterated Least Square (ILS) and (ii) Constrained Iterated Least Square Method. These are simple baseline methods used for comparison, and they have theoretical guarantees in terms of regret bounds as mentioned in Keskin and Zeevi, 2014

3.1.1 Iterated Least Square Method (ILS)

The ILS method is an optimization technique used to solve non-linear least squares problems. It is an iterative approach that aims to minimize the sum of squared residuals between observed and predicted values. The ILS algorithm estimates a polynomial curve by applying the least square method to a set of prior points and their corresponding values. It then selects points for the next iteration greedily based on the estimated curve.

After formulating the problem, the initial point is chosen, which can be done randomly or based on prior knowledge. This point is queried from the original polynomial, along with some noise, and the best fit polynomial is estimated using these points and values. The ILS algorithm selects the point from the estimated curve that provides the maximum reward. This selected point is then added to the prior set of points for the next step, and the process is repeated for a specified number of time periods (T), resulting in an estimated polynomial. However, it should be noted that ILS has been shown to be sub-optimal.

3.1.2 Constrained Iterated Least Square Method (CILS)

CILS (Constrained Iterated Least Square) is an extension of the ILS method that incorporates constraints on the unknown parameters. This approach ensures that the parameter estimates satisfy the specified limitations. CILS is a framework designed to handle non-linear least square problems with constraints, enabling more flexible and realistic modeling scenarios. CILS achieves asymptotically optimal regret, as mentioned in Keskin and Zeevi, 2014 by integrating forced point-dispersion with ILS. Specifically, CILS with a threshold parameter k suggests that at time t, the algorithm selects points that satisfy the following condition :

$$p_t = \begin{cases} \overline{p}_{t-1} + \mathrm{sgn}(\delta_t) k_{t-1}^{1/4} & \text{if } |\delta_t| < kt^{-1/4}, \\ \text{ILS price} & \text{otherwise.} \end{cases}$$

where $\delta_t = p_{t-1} - \overline{p}_{t-1}$ as the average of the prices suggested over t-1 periods.

3.2 Our Algorithm

Our algorithm shares similarities with the work mentioned in (Xu et al., 2020) but it differs in a few key aspects. In our algorithm, we rely solely on reinforcement learning, specifically a UCB (Upper Confidence Bound) based approach, to approximate the reward-generating function.

Additionally, our algorithm addresses a scenario where the selection of arms $k$is from an infinite set K. This is a departure from the previous work, which considered a discrete and finite number of arms. By extending the problem to include an infinite number of arms, we aim to tackle more complex and realistic scenarios.

The agent in our algorithm has knowledge that the reward-generating function is a polynomial of some degree, although the exact degree is unknown. In order to identify the true degree, the agent assigns a very small weight to the higher degree terms when the model degree is greater than the simulator degree.

To learn the simulator polynomial, we initializes a vector θ which comprises of coefficient of model initialized as $n(0, 1/d)$, where d is the dimension of model. A matrix A_t is is defined based on historical transformed features, initialized as $A_t = \lambda I$, where λ

is regularization parameter and a vector b_t i initialized as 0 vector. These initialization provide the starting point for the learning process.

For time period $t = 1toT$, the algorithm receives a feature vector a $x_{t,k}$, then the agent chooses an arm based on:

$$a_t = \arg\max_{k\in[K]} \left(\langle \theta_{t-1}^\intercal, x_{t,k} \rangle + \alpha_t \|x_{t,k}\|_{(A_{t-1}^{-1})} \right) \tag{3.1}$$

where θ_{t-1} is the point estimate at time $t-1$, $\alpha_t > 0$ is algorithmic parameter controlling the exploration, and A_t is a matrix defined based on historical transformed features:

$$A_t = \lambda I + x_{t,a_t} x_{t,a_t}^\intercal \tag{3.2}$$

and $\lambda > 0$. After pulling an arm a_t, the agent will receive a noisy reward $\hat{r_t} := \hat{r}_{x_{t,a_t}}$ defined as:

$$\hat{r}_{x_{t,k}} = r_{x_{t,k}} + \xi_t \tag{3.3}$$

where ξ_t is an unknown ν-subGaussian random noise for any $\nu > 0$ and $r(.)$ is unknown reward generating function.

After receiving reward $\hat{r_t}$ the algorithm updates its estimate θ_t. In particular, we have:

$$\theta_t = A_t^{-1} b_t \tag{3.4}$$

where $b_t = b_{t-1} + \hat{r}_t x_{t,a_t}$.

Algorithm 8 Our Algorithm

```
1: **Input:** regularization parameter $\lambda > 0$, number of total steps $T$, exploration parameters $\{\alpha_t > 0\}_{t\in[T]}$
2: **Initialization:** $A_0 = \lambda I$, $b_0 = 0$; entries of $\theta_0$ follow $N(0, 1/d)$
3: **for** $t = 1, \ldots, T$ **do**
4:     receive feature vector $\{x_{t,1}, \ldots, x_{t,k}\}$
5:     $a_t = \arg\max_{k\in[K]} \left( \langle \theta_{t-1}^\intercal, x_{t,k} \rangle + \alpha_t \|x_{t,k}\|_{(A_{t-1}^{-1})} \right)$
6:     **Update** $A_t$ and $b_t$:
7:         $A_t = \lambda I + x_{t,a_t} x_{t,a_t}^\intercal$
8:         $b_t = b_{t-1} + \hat{r}_t x_{t,a_t}$
9:     Update $\theta_t = A_t^{-1} b_t$
10: **end for**
11: **Output:** $\theta_T$
```

3.2.1 Our Algorithm with Barycentric Spanner

The use of barycenter in our algorithm allows agent for efficient exploration and exploitation of the context-action space. Unlike the earlier version of our algorithm, which randomly chose a point between the bounds at $t = 0$, the barycenter method helps the agent to identify similar or related contexts.

Depending on the degree of the model, the context-action space is divided, and these points are presented to the agent only during the initial stages. Afterward, the agent selects the best arm according to the algorithm which is basically involves Upper Confidence Bound(UCB) approach. The barycenter method helps in the efficient exploration and finding best areas in the context-action space.

Chapter 4

Results and Future Work

4.1 Result

The algorithm is tested with simulator of different-different degrees along with different ν-subGaussian noise and also in the case when model degree and simulator degree is different. The algorithm is also tested with different baseline algorithms namely, (i) Iterated Least Square Method (ILS) (ii) Constrained Iterated Least Square Method (CILS).

4.1.1 Quadratic Polynomials

We carried out experiment on three quadratic polynomials, one having maxima at middle, second one with eccentric maxima from middle and last one from (Keskin and Zeevi, 2014). Experiment carried out on 100 samples.Implementation of Algorithm.

In Figure 4.1, we consider an environment for simulator as $f(x) = 10 + 100x - x^2 + \epsilon$, having bounds $p \in [0, 100]$, where ϵ is zero mean Gaussian noise sample with $\sigma = 1$.

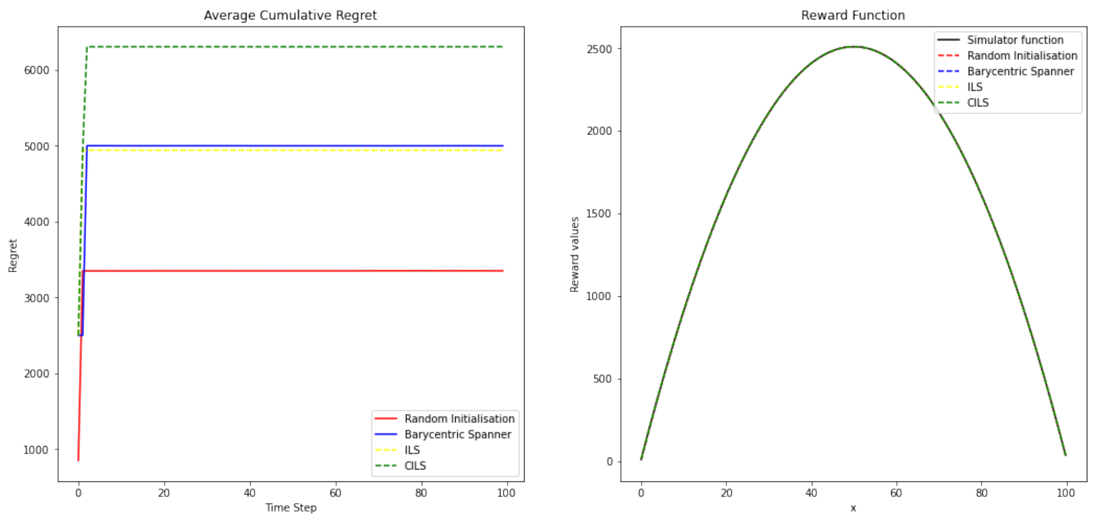


Figure 4.1: Average Cumulative Regret and model plot for f(x) $= 10 + 100x - x^2 + N(0,1)$

For polynomial with eccentric maxima from middle, the environment for simulator is $f(x) = 500 + 400x - 10x^2 + N(0, 1)$ having bounds $p \in [0, 50]$. The regret plot and the model function plot with different algorithms is shown in Figure 4.2

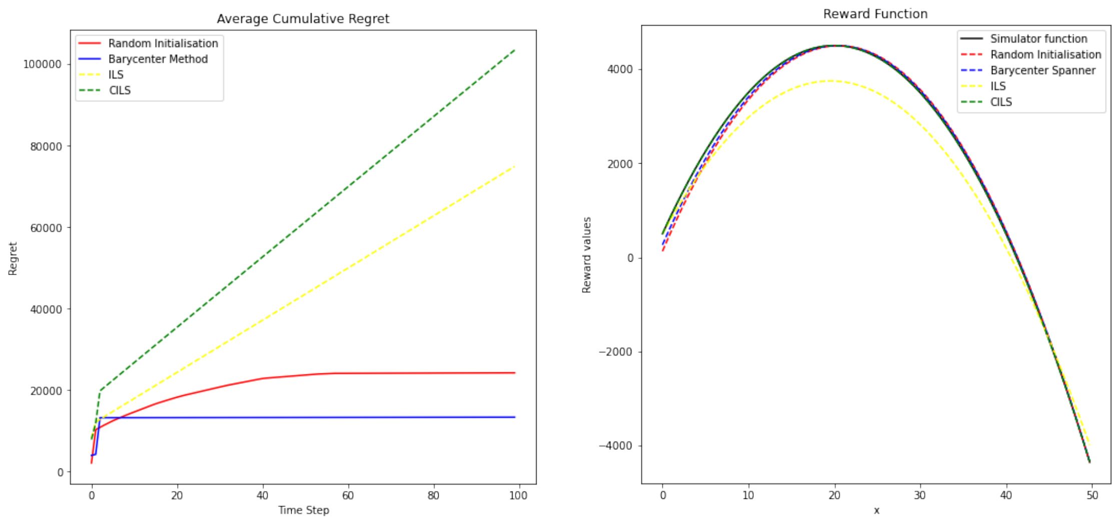


Figure 4.2: Average Cumulative Regret and model plot for f(x) $= 500 + 400x - 10x^2 + N(0, 1)$

Environment of quadratic polynomial mentioned in (Keskin and Zeevi, 2014) is, $f(x) = 1.1x - 0.5x^2 + N(0, 0.01)$ with bounds $p \in [0.75, 2]$. The regret plot and the model function plot with different algorithms is shown in Figure 4.3

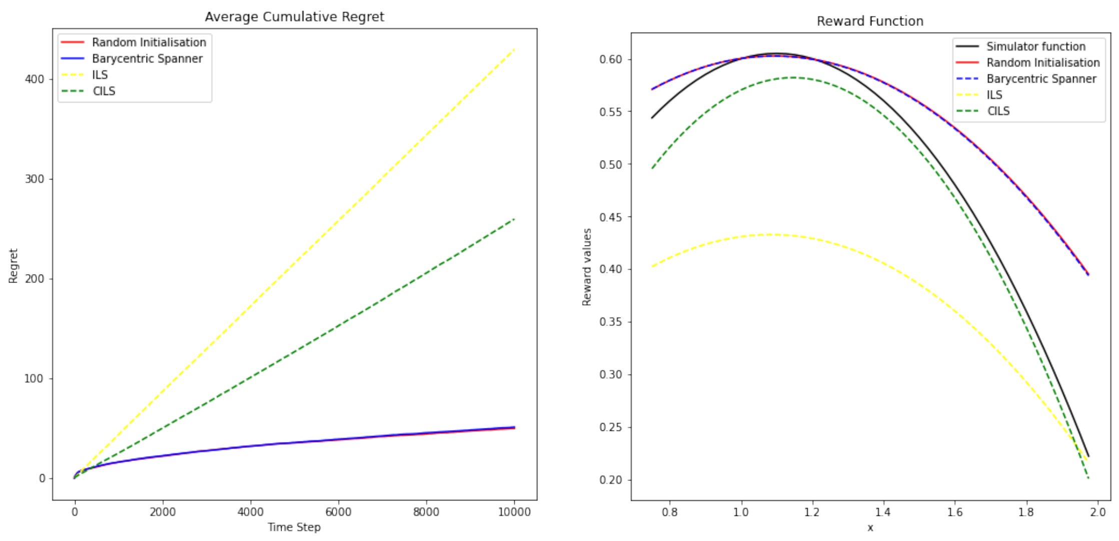


Figure 4.3: Average Cumulative Regret and model plot for $f(x) = 1.1x - 0.5x^2 + N(0, 0.01)$

4.1.2 Higher Degrees Polynomials with multiple maxima

In Figure 4.4, the simulator environment is $f(x) = -150 + 480x - 165x^2 + 22x^3 - x^4 + N(0, 10)$ having bounds $p \in [1, 10]$ mentioned in (Amballa et al., 2021), our algorithm does not give appropriate results. Experiment is carried out over 10 samples.

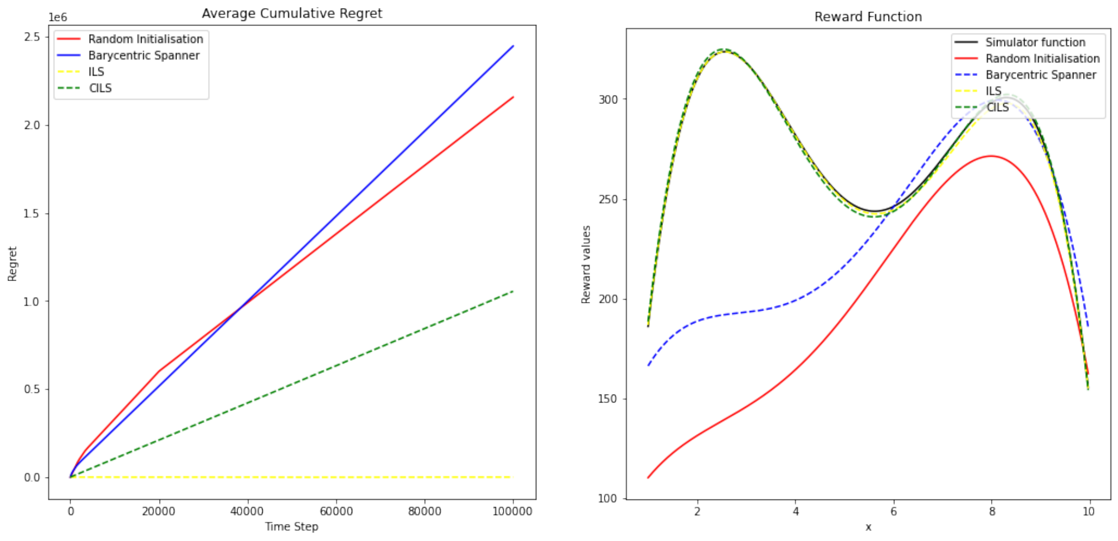


Figure 4.4: Average Cumulative Regret and model plot for $f(x) = -150 + 480x - 165x^2 + 22x^3 - x^4 + N(0, 10)$

4.1.3 Effects of noise

Simulator function $f(x) = 10 + 100x - x^2 + N(0, 5)$ almost had no effect of noise on the results as shown in Figure 4.5

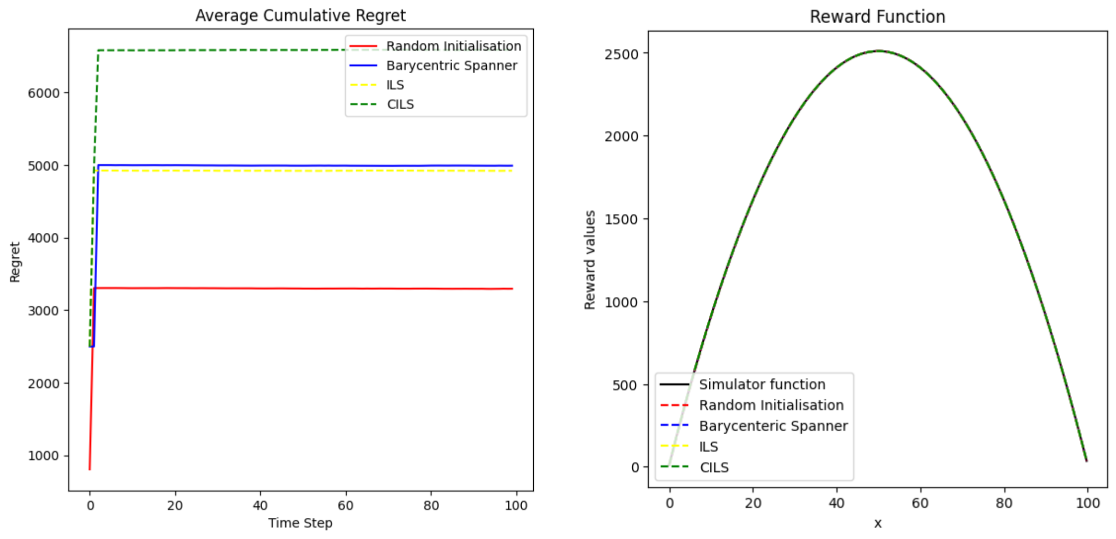


Figure 4.5: Average Cumulative Regret and model plot for f(x) $= 10 + 100x - x^2 + N(0, 5)$

Simulator function $f(x) = 500 + 400x - 10x^2 + N(0, 5)$ almost have significant effect of noise on the results as shown in Figure

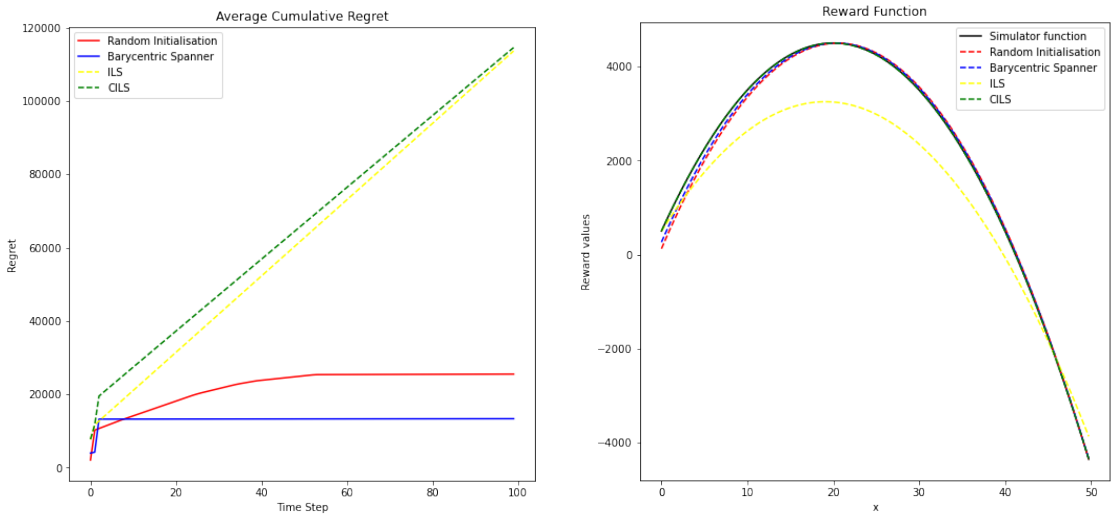


Figure 4.6: Average Cumulative Regret and model plot for $f(x) = 500 + 400x - 10x^2 + N(0,5)$

4.1.4 Mismatch in Simulator and Model Degree

When simulator function $f(x) = 10 + 100x -^2 + N(0,1)$ is learnt with cubic (Figure 4.7) and 4-degree model (Figure 4.8) polynomial, CILS outputs very absurd result when this polynomial is learnt with degree 4 model.

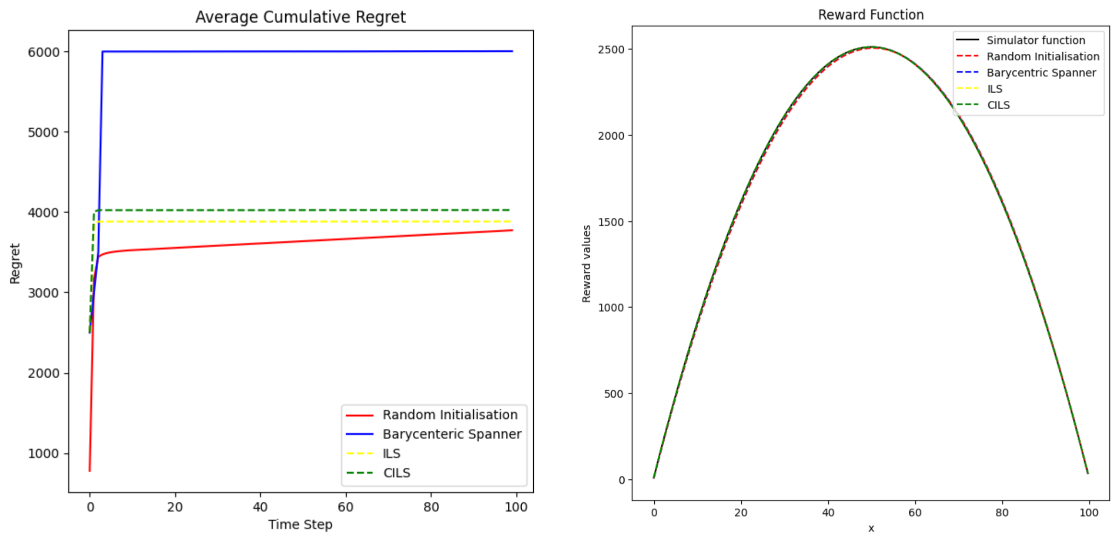


Figure 4.7: Average Cumulative Regret and model plot for f(x) $= 10 + 100x - x^2 + N(0,1)$, when model is assumed cubic polynomial

When simulator function $f(x) = 500 + 400x - 10x^2 + N(0,1)$ is learnt with cubic (Figure 4.9) and 4-degree model (Figure 4.10) polynomial, CILS outputs very absurd result when this polynomial is learnt with degree 4 model.

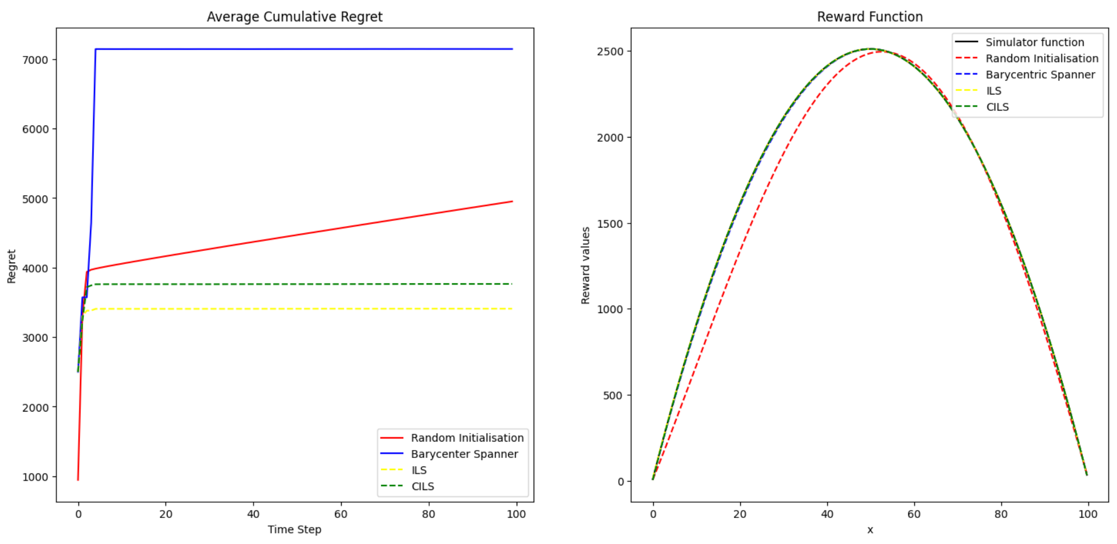


Figure 4.8: Average Cumulative Regret and model plot for f(x) $= 10 + 100x - x^2 + N(0,1)$, when model is degree 4 polynomial

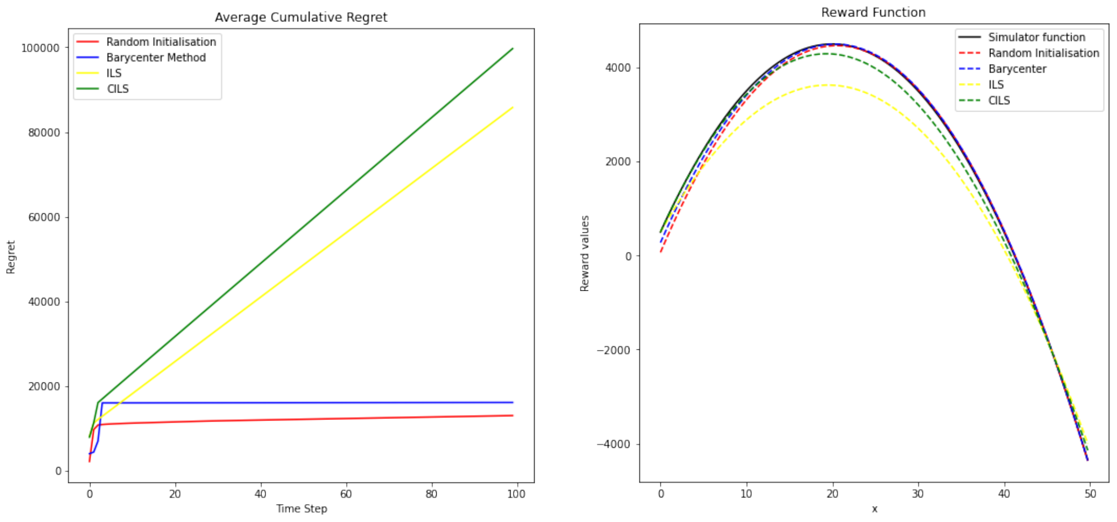


Figure 4.9: Average Cumulative Regret and model plot for f(x) $= 500 + 400x - 10x^2 + N(0,1)$,when model is assumed cubic polynomial

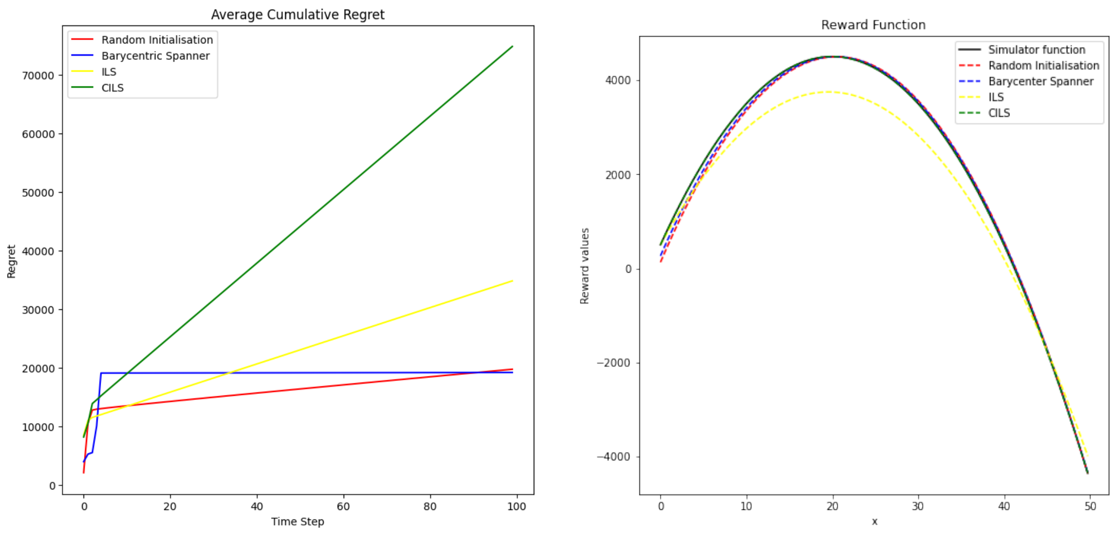


Figure 4.10: Average Cumulative Regret and model plot for f(x) $= 500 + 400x - 10x^2 + N(0,1)$, when model is degree 4 polynomial

4.2 Future Work

Following are the future research which can be done:

- There is need of neural network, the algorithm is not able to optimize the the polynomial with multiple maxima, with the advancement in the field of Artificial Intelligence, neural network become very powerful optimizer which can optimize almost all function. In our algorithm, neural network can be used to assign appropriate weight to the point estimate θ.
- Application of the algorithm is to be done on real-world data. The algorithm can be used for various applications like dynamic pricing, recommender system, etc.

References

Abbasi-Yadkori, Y., D. Pál, and C. Szepesvári (2011). “Improved algorithms for linear stochastic bandits”. In: *Advances in neural information processing systems* 24 (cit. on p. 6).

Allen-Zhu, Z., Y. Li, and Y. Liang (2019). “Learning and generalization in overparameterized neural networks, going beyond two layers”. In: *Advances in neural information processing systems* 32 (cit. on p. 16).

Amballa, C., M. K. Gupta, and S. P. Bhat (2021). “Computing an efficient exploration basis for learning with univariate polynomial features”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 8, pp. 6636–6643 (cit. on p. 26).

Arora, S., S. Du, W. Hu, Z. Li, and R. Wang (2019). “Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks”. In: *International Conference on Machine Learning*. PMLR, pp. 322–332 (cit. on p. 16).

Cao, Y. and Q. Gu (2019). “A generalization theory of gradient descent for learning overparameterized deep relu networks”. In: *arXiv preprint arXiv:1902.01384* 2 (cit. on p. 16).

Chu, W., L. Li, L. Reyzin, and R. Schapire (2011). “Contextual bandits with linear payoff functions”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, pp. 208–214 (cit. on p. 6).

Collier, M. and H. U. Llorens (2018). “Deep contextual multi-armed bandits”. In: *arXiv preprint arXiv:1807.09809* (cit. on p. 10).

Jacot, A., F. Gabriel, and C. Hongler (2018). “Neural tangent kernel: Convergence and generalization in neural networks”. In: *Advances in neural information processing systems* 31 (cit. on p. 16).

Keskin, N. B. and A. Zeevi (2014). “Dynamic pricing with an unknown demand model: Asymptotically optimal semi-myopic policies”. In: *Operations research* 62.5, pp. 1142–1167 (cit. on pp. 21, 22, 25, 26).

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on p. 19).

Li, L., W. Chu, J. Langford, and R. E. Schapire (2010). “A contextual-bandit approach to personalized news article recommendation”. In: *Proceedings of the 19th international conference on World wide web*, pp. 661–670 (cit. on p. 12).

Riquelme, C., G. Tucker, and J. Snoek (2018). “Deep bayesian bandits showdown”. In: *International conference on learning representations*. Vol. 9 (cit. on pp. 13, 15).

Xu, P., Z. Wen, H. Zhao, and Q. Gu (2020). “Neural contextual bandits with deep representation and shallow exploration”. In: *arXiv preprint arXiv:2012.01780* (cit. on pp. ii, 16, 20, 22).

Zhou, D., L. Li, and Q. Gu (2020). “Neural contextual bandits with ucb-based exploration”. In: *International Conference on Machine Learning*. PMLR, pp. 11492–11502 (cit. on pp. 13, 17, 18).