<u>Making A Python Based Package for the Multi Arm Bandit Problem</u>

Group Members

Pankayaraj (E/14/237)
Subasinghe(E/14/335)

Introduction

In probability theory multi arm bandit problem or N-arm bandit problem is a problem in which a gambler at a row machine have to choose which machine to play and how many time to play it given a limited number of turns to choose. When chosen a machine would give a particular amount of reward which is either deterministic or probabilistic. Thus to accumulate an optimal amount of reward the gambler should choose a an optimal solution without knowing the reward structure behind the machine. If stated formally the problem can be considered as a set of real distributions

$$B = \{R_1, \ldots, R_K\}$$

Where each distribution can be associated with the rewards produced by each one of the arms when selected. If we let the mean of those rewards to be

$$\mu_1, \ldots, \mu_K$$

When the gambler iteratively chooses some for T iterations arms, then the regret for that procedure can be measured as

$$\rho = T\mu^* - \sum_{t=1}^{T} \hat{r}_t$$

Where

$$\mu^* = \max_k \{\mu_k\}$$

And $\hat{r}_t$ is the reward associated with a single selection.

Thus the problem can be solved by trying to reduce the regret which will eventually maximise the total amount of reward obtained.

Instead of considering these rewards as a result of a real distribution they can also be considered as a bernoulli distribution to tackle the problem where the reward is a binary variable.

As the problem moved away from the discrete arms got extended as a continuous variable with a K dimension the problem got extended as continuous bandit problem. Since the no of bandits became infinite to reduce the complexity the problem was formulated with deterministic rewards where the rewards of each arm were considered as a correlated function. As the scope of these problems narrowed down to the bayesian thinking they were named as bayesian optimization. They can be considered as a problem where we are supposed to optimize a function with certain bounds with as few samples as possible.

<u>Problem definition and proposed solution</u>

Problem

Due to the large no of variation and vast amount of solutions proposed for this particular problem making a standard open source library for the problem has been an issue. Though the presence of large number of solutions has been an issue that doesn't stop the problem from being important in the practical aspects. Due to the rapid development of machine learning the prominence of optimization has increased greater than before. When it comes to implementing this problem financial institutes are using their own exclusive libraries while the research communities are using the functions written by previous researchers. Thus those functions written by them are not only scattered but also not optimal in performance as they fail to use all the computer hardware available completely.

Proposed Solution

So we decided to make a standard library for this problem. Since the scope of the problem and solutions available are vast we have decided to initially make a library on a narrowed scope and then continue it from there on. As far as the discrete case is concerned for the time being we have decided to make a take care of the problem with a binary reward and to opt the one with a continuous reward out as the solutions proposed for them where large in number. And we decided to concentrate more on the continuous case. The reason for the narrowing strategy is the concern towards the practical need for this solutions. Due the massive improvement in reinforcement learning in the recent years the problem of learning an optimum policy by the agent in an environment within a short period of time became an important problem. Thus the continuous case mentioned above became more important. Thus concentrating on that issue seemed more important to us. So we have decided to concentrate on the bayesian optimization domain in this problem. Though in theory the solutions proposed gave a certain time limits in complexity parallelization of the problem with help of GPUs seemed as an viable option as well. So we have decided to parallelize the code wherever it seems optimal in the process of implementing the library.

Solutions Implemented

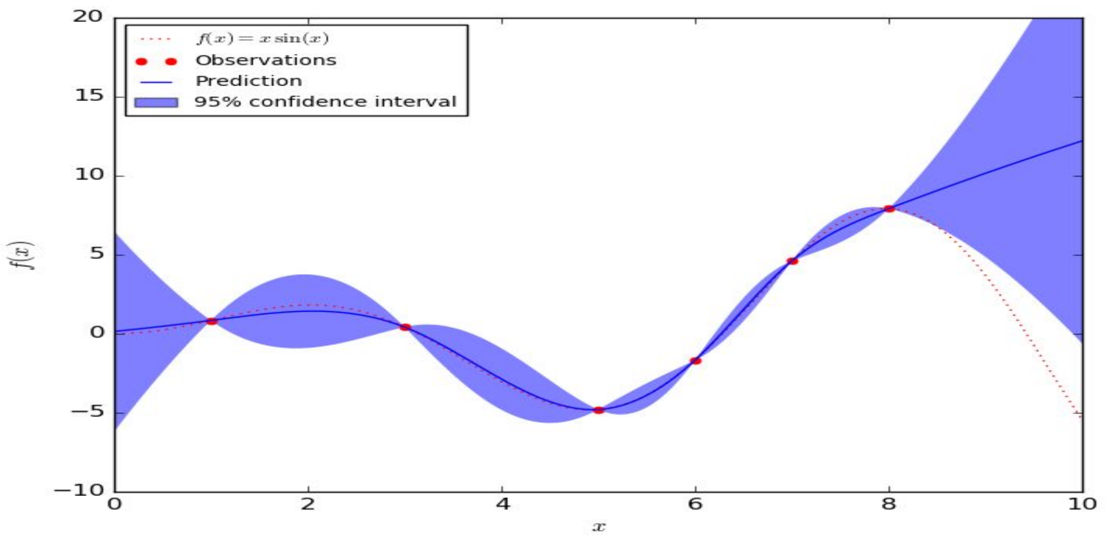
Binary Rewards(Discrete case) : Beta Bernoulli Model

Here the rewards are considered as a bernoulli random variable. Since the bernoulli random variable can be characterized by their mean only a prior belief on those means where modeled and updated as they were sampled along time. A Beta distribution is used to model the initial belief on those means and then the arm with the highest mean is sampled and the belief is updated according to the bayes theorem.

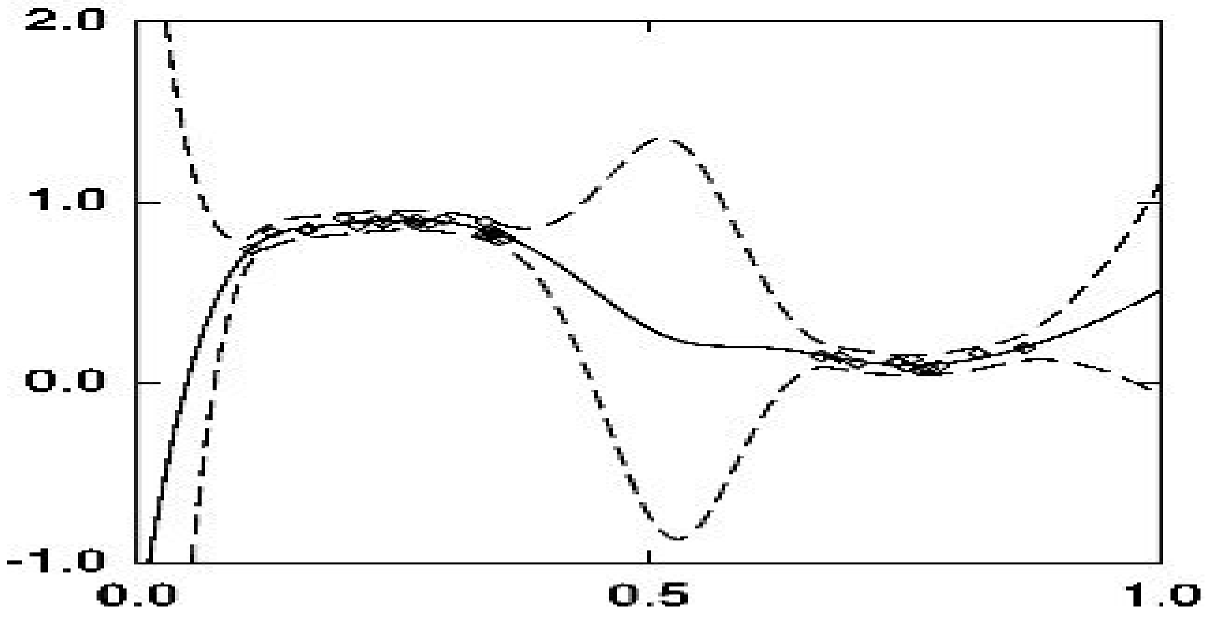
Continuous Bandit Problem

Here three algorithms Probability of improvement, thompson sampling and Expected improvement are implemented with gaussian process as a surrogate function. Since we don’t know the function we are optimizing a belief on that function is built initially and then that belief is updated with every sample we obtain. Thus a faster optimization can be made my a good model of belief

Here the belief/ Surrogate function that is used is a gaussian process. The advantage of a gaussian process is that it gives not only a interpolation of existing points but also a measure on how much uncertain it is about each prediction.



As seen in the figure it give a prediction of the function(blue) and an uncertainty about a point in that function (purple). The prediction is given as the mean of a gaussian distribution while the uncertainty is given as the variance of the same distribution.
The similar concept can also be implemented using a bayesian neural network where instead of obtaining a parametric model we obtain a probabilistic model from a neural network using a technique called dropout.

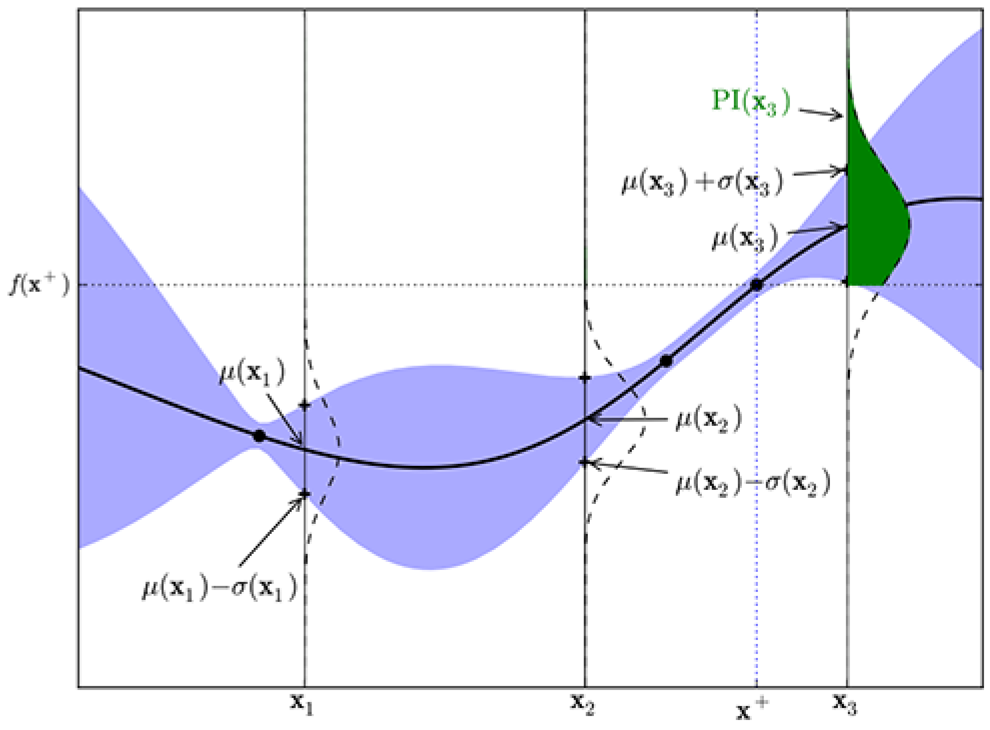


Once such measurements are obtained the mean provides us a way to exploit the recent findings while the variance gives us an indication of the unexplored area. Thus functions that are formulated using the two parameters can be optimized to obtain a sample that will possibly be a better one. Such functions that are formed are the probability of improvement, expected improvement and thompson sampling

Thompson Sampling

The idea behind the thompson sampling is to draw a function from this surrogate function and optimize that function. And the sample at the optimum value and then update the surrogate function. Continuing this way will give us the optimum solution.

Probability of Improvement



$\mathrm{PI}(\mathbf{x}_3)$
$\mu(\mathbf{x}_3)+\sigma(\mathbf{x}_3)$
$\mu(\mathbf{x}_3)$
$f(\mathbf{x}^+)$
$\mu(\mathbf{x}_1)$
$\mu(\mathbf{x}_2)$
$\mu(\mathbf{x}_2)-\sigma(\mathbf{x}_2)$
$\mu(\mathbf{x}_1)-\sigma(\mathbf{x}_1)$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}^+$
$\mathbf{x}_3$

$$\alpha_{\mathrm{PI}}(\mathbf{x}; \mathcal{D}_n) := \mathbb{P}[v > \tau] = \Phi\left(\frac{\mu_n(\mathbf{x}) - \tau}{\sigma_n(\mathbf{x})}\right),$$

In this process the probability of the next sampled point being greater than the existing maximum is considered as the acquisition function to to be optimized to get the new sample. This method is normally used when the point in the proximity of the maximum is known.

Expected Improvement

Expected improvement is an idea that was proposed by Mockus in 1978 based on the idea of bayesian decision process.

$$EI(x) = \underbrace{\underbrace{(y_{PBS} - \hat{y}(x))}_{\boxed{\text{Predicted difference between current minimum and prediction at } x}} \underbrace{\Phi\left(\frac{y_{PBS} - \hat{y}(x)}{s(x)}\right)}_{\boxed{\text{Penalized by area under the curve}}}}_{\boxed{\text{Large when } \hat{y} \text{ is small with respect to } y_{PBS}\text{, promotes exploitation}}} + \underbrace{s(x)\phi\left(\frac{y_{PBS} - \hat{y}(x)}{s(x)}\right)}_{\boxed{\text{Large when } s(x) \text{ is large, promotes exploration}}}$$

Design and implementation of the solution

Choosing the language

First of all we have to choose a programming language as the interface language. Given the prominence of this problem within the research community we had to decide a language that is popular within the community. The language also needed to be resourceful to tackle the issue of parallelizing which is discussed below. Thus we went with python 3 as our interface language for this problem

Finding a code base

Since it was a group project we needed to find an appropriate code base so that group can communicate without any issues. Moreover due to the open sourced nature of this project we needed a popular and easily accessible free base. Thus we have decide the popular github as our code base. The codes were formulated and maintained in a github repository. The URL for that repository is

https://github.com/punk95/Multi-Arm-Bandit-Library

Apart from making the code available in a library we have also decided to make it easy to install as use with python from anywhere. Thus we also choose to upload a wheel of the project to PYPI the largest python repository to be installed from anywhere with python setuptools.

Parallelization of code

As much as we have identified the need for the parallelization of the code we have also identified the need of a stable mechanism for the purpose with a continuous improvement along time. So it became obvious for us to go for an already stable under continuous development python package to do the parallelization. Though we have initially decided to go with the Theano package developed by University of Montreal due to their recent announcement on the discontinuation of the further development we have decided to go with a Google backed package Tensorflow for the parallelization process.

Structuring the functions

We wanted the package to be more flexible and easy for the user to use. Since a particular solution had many dependencies which have their own hyper parameters. For example in the continuous case where the surrogate function Gaussian process and the bayesian neural network had their own hyper parameters to tune such as the learning rate, no of layers, kernel parameters etc. At the meanwhile the acquisition function had its own parameters to care about. At the meanwhile gaussian process had to lead with large matrices in the process.

Thus to make matter easier for the surrogate function we have decided to implement them with the concept of object oriented programming. That way the surrogate functions get to use their own matrices for a prolonged use rather than to create them each time they are needed. With the effectiveness of object oriented programming it was tempting for us to make the acquisition function using the same methodology. But since the problem we were tackling was that of active learning we thought it would be computationally optimal not to create an object every time we use them to make a sample. Furthermore they didn’t have any notably large amount of dependent data. Thus it made us to go with the usage of normal functions while providing options to tune the surrogate functions behind with through these functional argument.

Parallelization

When it came to parallelizing as already decided we went an existing library tensorflow. But the only cost with a library is that it is a static library. That is all the structures of the algorithm must be defined beforehand for it to optimize the algorithm. Thus though it made a heavy process run faster with a good gpu it had an issue of creating significant amount of overhead beforehand. Thus it made sense for us to use the parallelization for the surrogate functions which are going to take long time to produce the results and going to stay in use for a while. Meanwhile using parallelization for the iterative light weighted functions such as acquisition ones seemed like an computationally expensive option rather than an optimal one. So we have decided to make those functions with sequential programming techniques.

Programming Dependencies

As mentioned above for the purpose of parallelization we used tensorflow library. Furthermore for the purpose of handling arrays and random numbers efficiently we used a frequently used package numpy in our implementation. In addition to that we had the problem of optimizing the acquisition function with the luxury of having more inexpensive sampling at each iteration to get the next sample point. Thus we needed a good package to do the traditional optimization. Thus we have used the renowned scipy to get these necessary functions for the optimization. Thus in total three packages tensorflow, scipy, and numpy along with their own dependencies were used to implement this package.

Results

Beta Bernoulli model

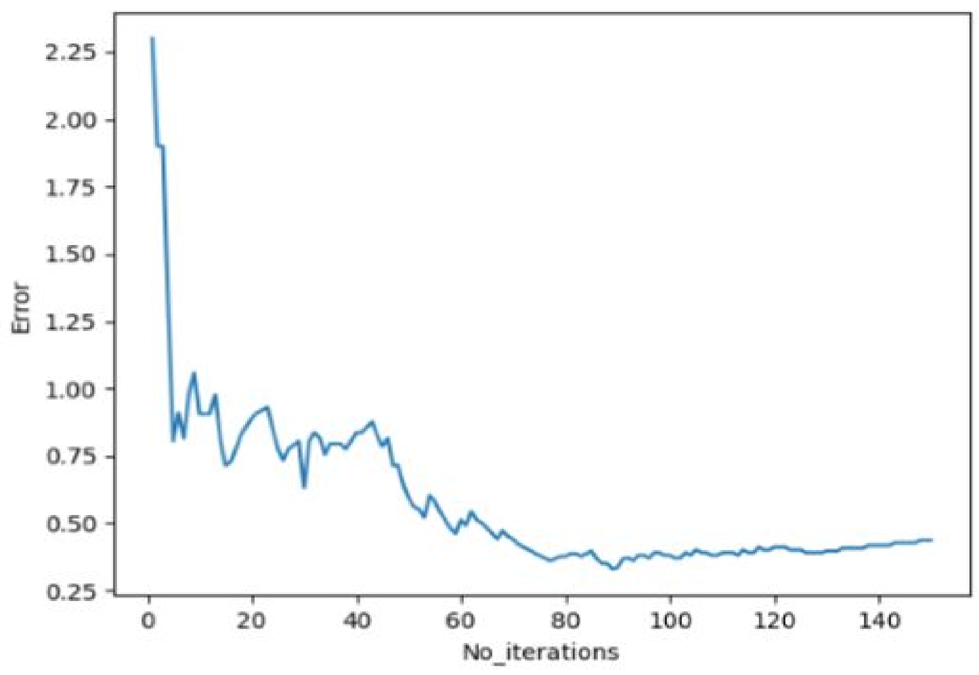
Problem Specification :

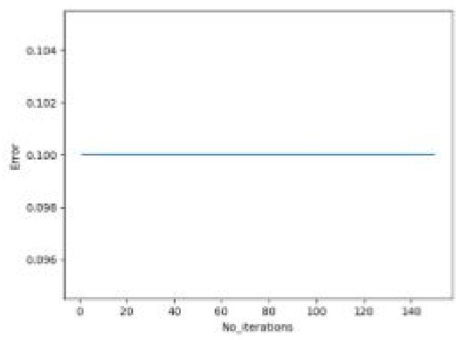
Arms = 5
Reward = Bernoulli Model
Reward Mean = 0.1, 0.5, 0.7, 0.2, 0.8

Model

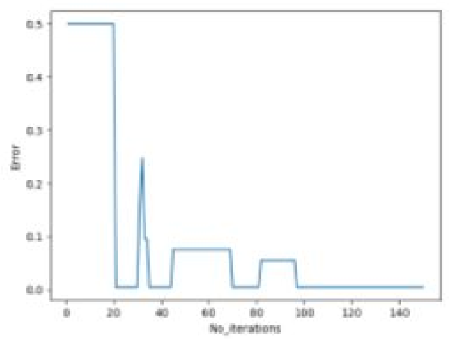
Beta_Bernoulli with Thompson Sampling

Total Error

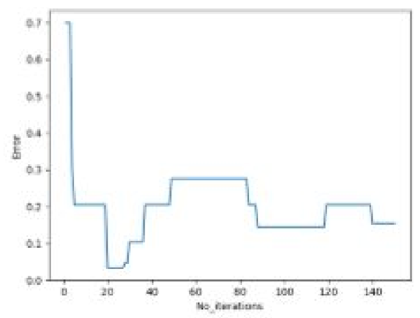




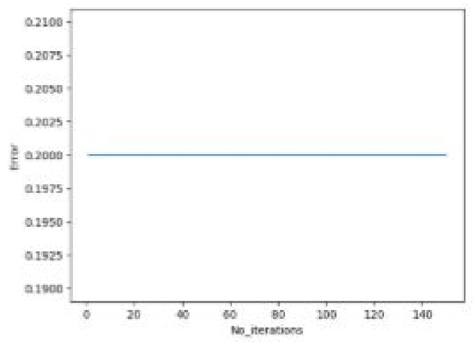
Theta = 0.1



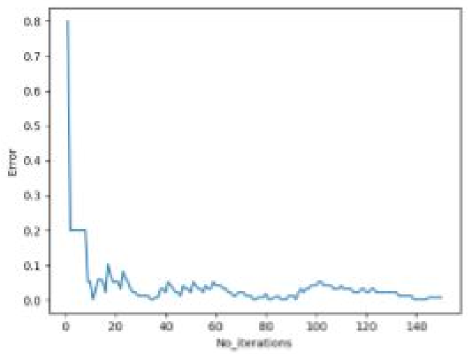
Theta = 0.5



Theta = 0.7



Theta = 0.2



Theta = 0.8

Figure : The results produced by Beta Bernoulli model when 5 arms with binary rewards were given

Expected Improvement with Gaussian Process as a surrogate function

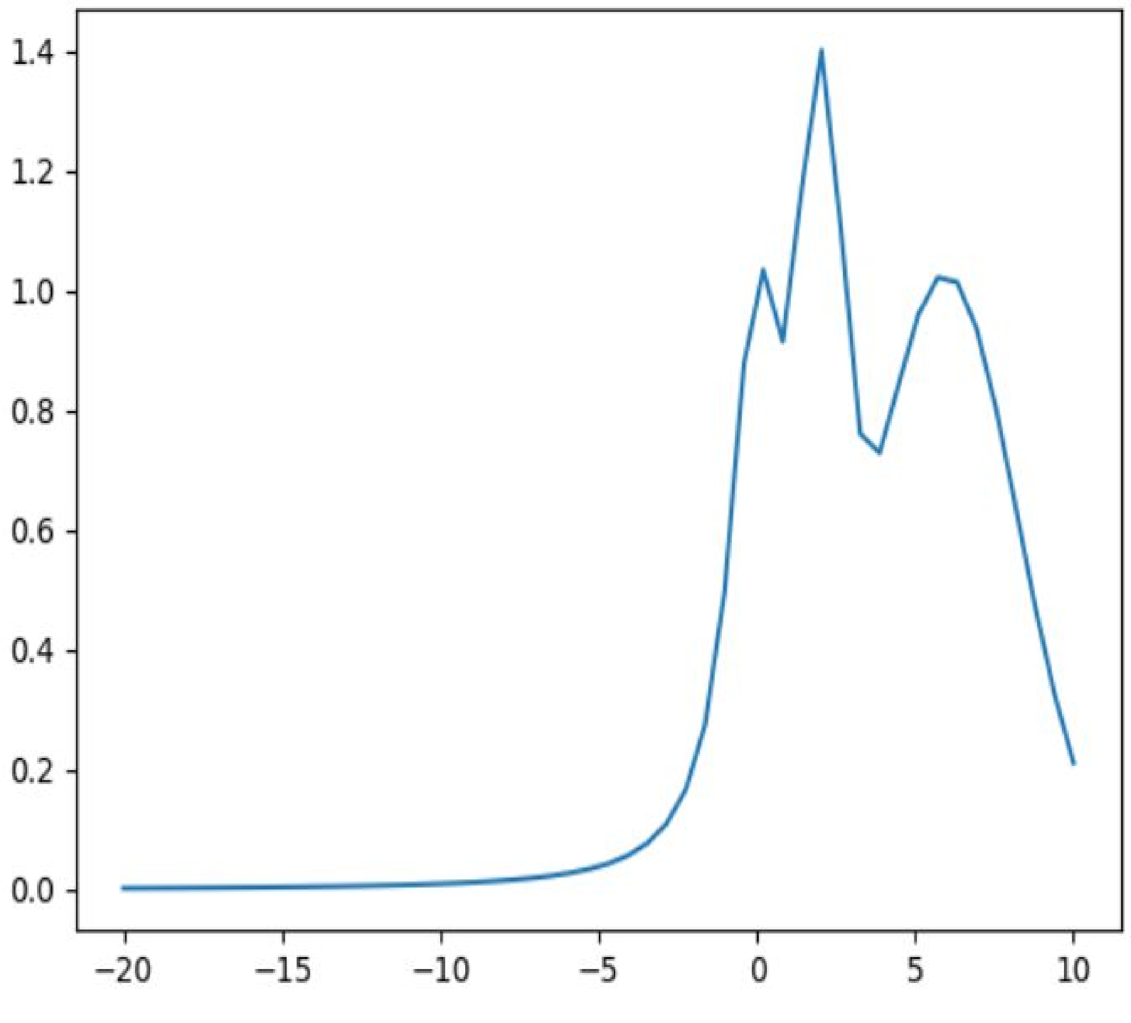
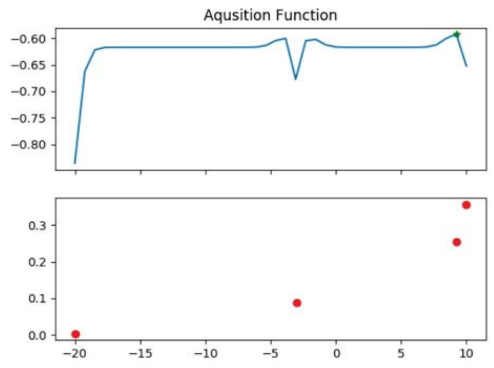
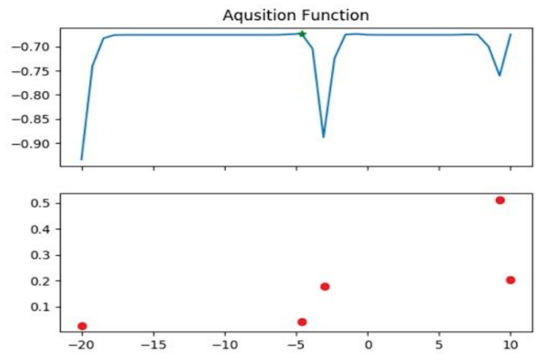


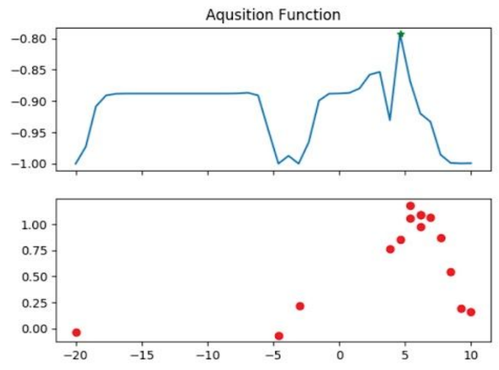
Figure : Six hump Camel Back Function



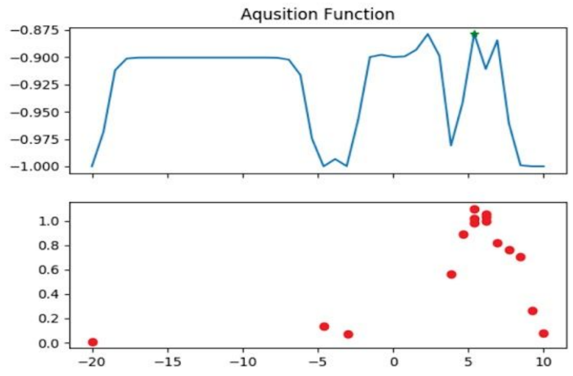
Iteration 1



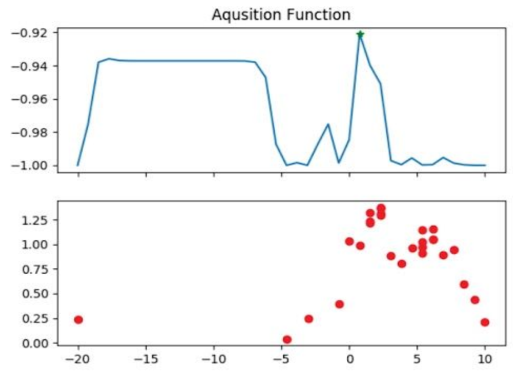
Iteration 2



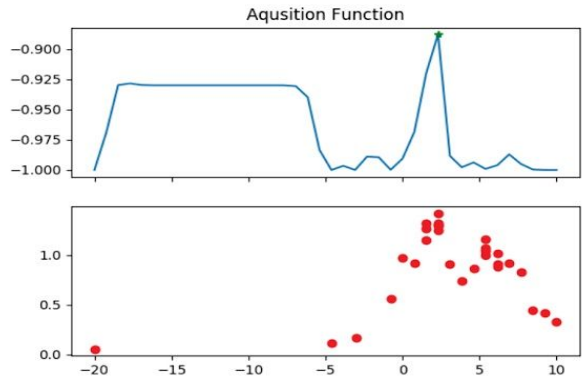
Iteration 12



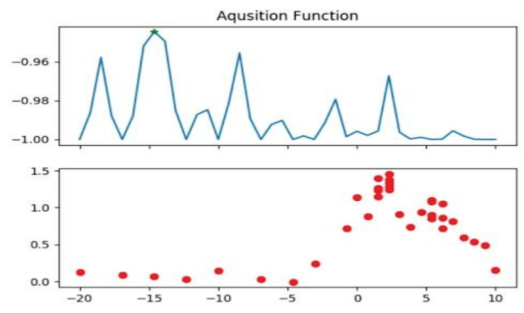
Iteration 13



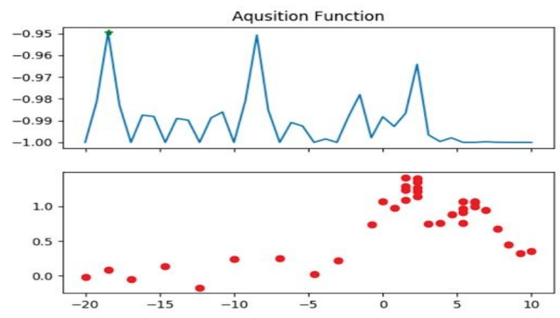
Iteration 25



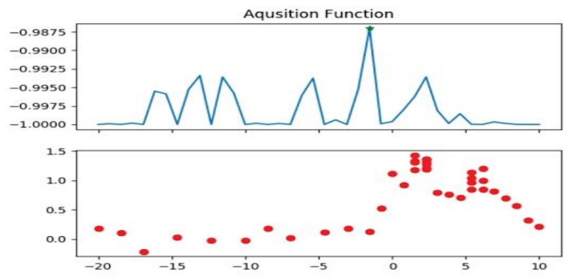
Iteration 26



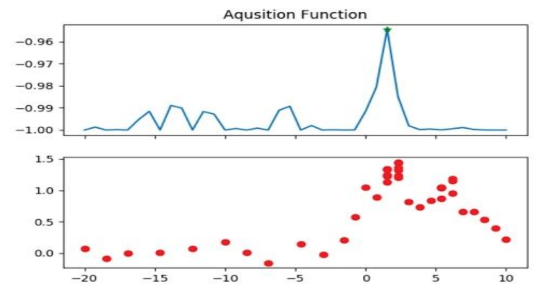
Iteration 32



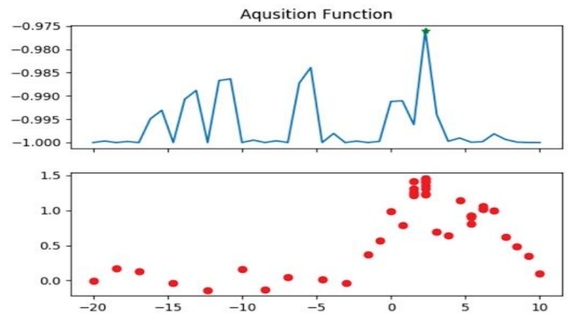
Iteration 33



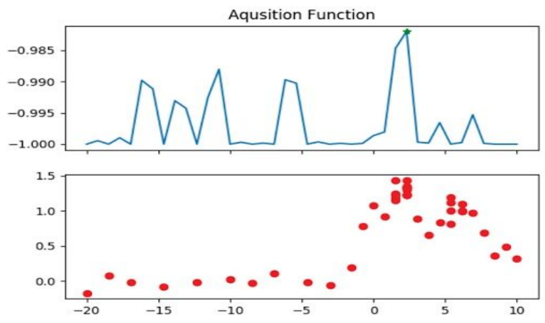
Iteration 36



Iteration 37



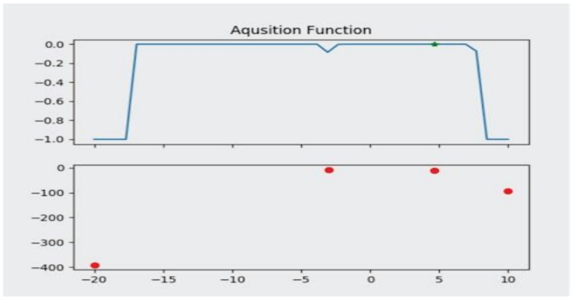
Iteration 39



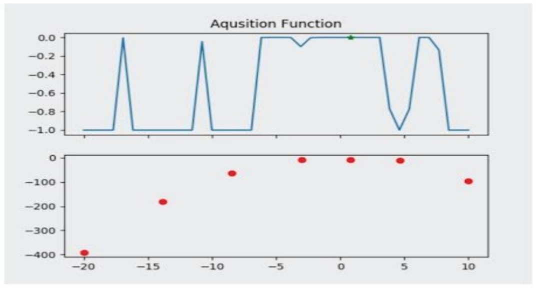
Iteration 40

Figure : Results produced by Expected improvement when it is used to optimize a six hump camel back function. Here red points indicate the actual sampled function while the blue curves on the top denote the acquisition function optimized for a new sample

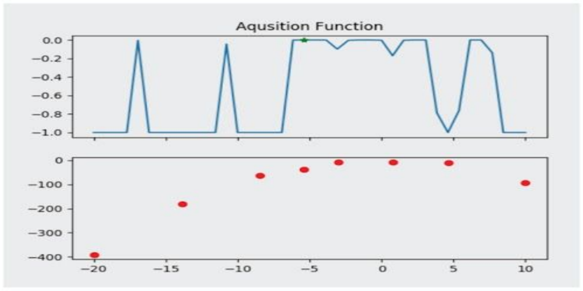
Probability of Improvement with Gaussian Process as a surrogate function



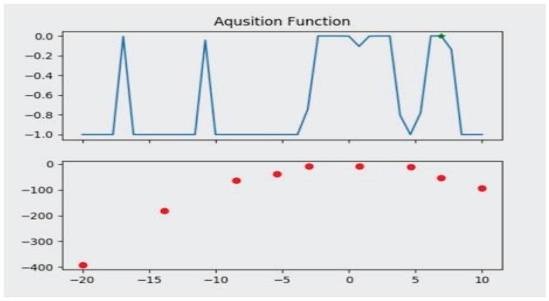
Iteration 1



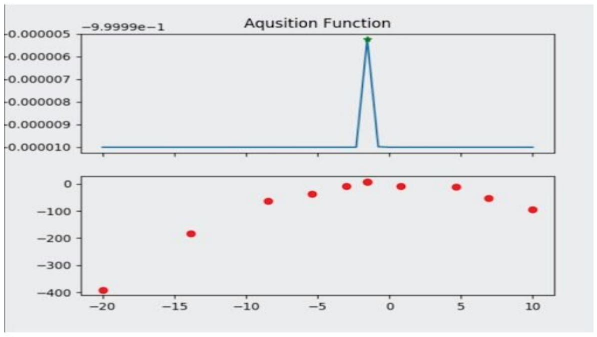
Iteration 3



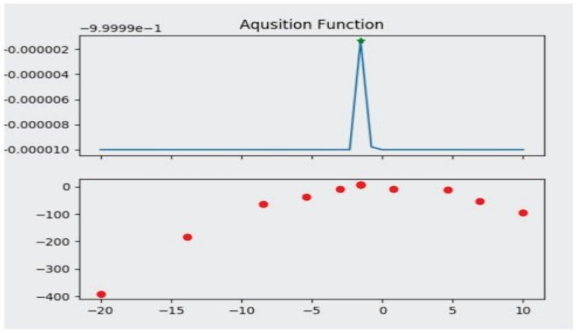
Iteration 4



Iteration 5



Iteration 7



Iteration 8

Figure : Results produced by Probability of improvement method when point near the maximum is known. Here red points indicate the actual sampled function while the blue curves on the top denote the acquisition function optimized for a new sample

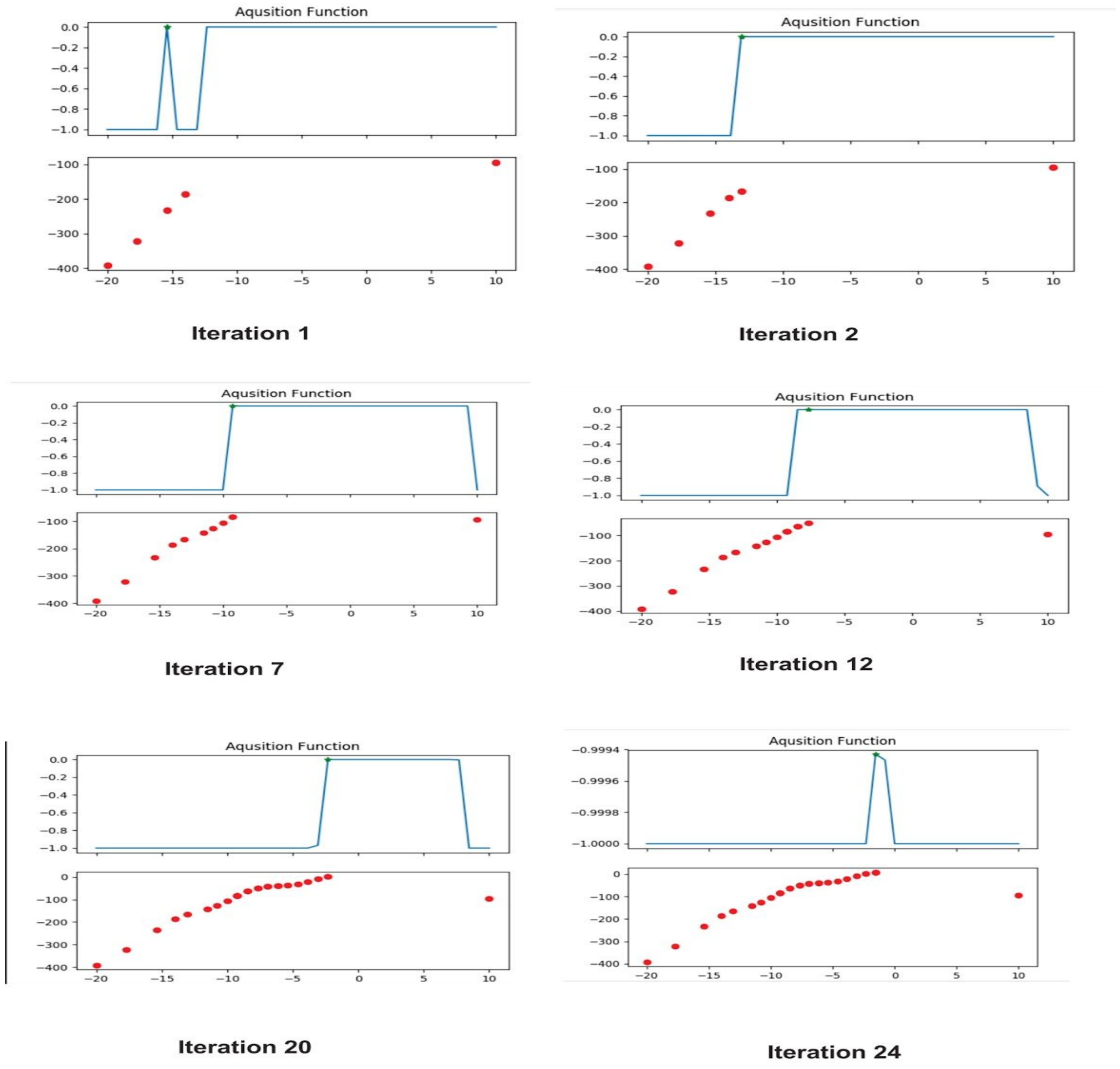


Figure : Results produced by a Probability of Improvement when a point near the actual maximum is unknown. Here red points indicate the actual sampled function while the blue curves on the top denote the acquisition function optimized for a new sample

Future Works

Thus we have concluded our project with the discrete case where the rewards are binary and the continuous case where the gaussian process is used as a surrogate function. But as said earlier the set of solutions available for this problem is vast and we have only achieved a small feat in our vision. Thus we have decided to publish the product made so far and to continue our works aftermath, updating the product in the process. As far as the continuous case is concerned we have decided to first develop the same set of algorithms to work with the already formulated bayesian neural network. The we have decided to make an implementation of the recent information theory base methods such as entropy search, predictive entropy search and max value entropy search.

As far as the discrete case is concerned we have decided to implement the solution for the discrete case where the reward is a real distribution. In that case we have decided to implement the solutions based on UBC algorithm, adversarial bandits, Lipschitz bandits and the contextual bandits. And we have also planned to optimize the existing code. Furthermore we also want to make this library capable of running with many parallelizing libraries at the backend by using library independent functions.

References

1. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning.

   Eric Brochu, Vlad M.Chopra, Nando de Freitas

2. An introduction to the Beta-Binomial model.

   COMPSCI 3016: Computational Cognitive Science Dan Navarro & Amy Perfors University of Adelaide

3. Max-value Entropy Search for Efficient Bayesian Optimization

   Zi Wang, Stefanie Jegelka

4. Predictive Entropy Search for Efficient Global Optimization of Black-box Functions

   Jose Miguel Hernandez-Lobato, Mathew W Hoffman, Zoubin Ghahramani

5. Introduction to multi arm bandits

   Aleksandrs Sivikins