UCL DEPARTMENT OF
COMPUTER SCIENCE

Research Note
RN/12/15

Dynamic Learning Gaussian Process Bandits

26 November 2012

Diogo Dubart Norinho

Abstract

Numerous applications require the optimisation of unknown functions, which are costly to evaluate and potentially too complex to allow for strong simplifying assumptions (e.g. linearity, differentiability, etc.). May it be in a clinical trial, online advertisement or oil reserves location, it is not always economically feasible to sample the objective functions at whim. Gaussian Process (GP) bandits efficiently deal with such situations by learning probability distributions over the black-box functions favouring their sampling in areas with higher prospects.

GP–UCB is a type of GP-bandit algorithm that trades off exploration and exploitation by using Upper Confidence Bounds (UCB). We perfect it by introducing a hyperparameter learning stage. This leads to a more efficient algorithm whose performance is tested against that of the GP–UCB over a set of benchmark functions.

Dynamic Learning Gaussian Process Bandits

Diogo Dubart Norinho
University College London

14 October 2012

Abstract

Numerous applications require the optimisation of unknown functions, which are costly to evaluate and potentially too complex to allow for strong simplifying assumptions (e.g. linearity, differentiability, etc.). May it be in a clinical trial, online advertisement or oil reserves location, it is not always economically feasible to sample the objective functions at whim. Gaussian Process (GP) bandits efficiently deal with such situations by learning probability distributions over the black-box functions favouring their sampling in areas with higher prospects.

GP–UCB is a type of GP-bandit algorithm that trades off exploration and exploitation by using Upper Confidence Bounds (UCB). We perfect it by introducing a hyperparameter learning stage. This leads to a more efficient algorithm whose performance is tested against that of the GP–UCB over a set of benchmark functions.

1 Introduction

An increasing number of scenarios require an agent to take the best possible decision, by maximising his rewards. So far, the main body of the literature has focused on solving well defined optimisation problems in which the function to be optimised or *objective function* has a closed-form mathematical formulation [2]. Moreover, that function is considered cheap to evaluate, in the sense that it can be extensively measured at different inputs in order to find its optimum.

Nevertheless, in everyday situations those requirements are hardly ever met. In *online assignment learning* problems [16] a service provider has to choose which ads to display on a webpage for each user to view. The provider is paid depending on the number times the shown ads are clicked on, the Click-Through-Rates (the reward function). He is not given any information about the reward function other than if the user has clicked on the ad or not. As a result, bandits were used to find which ads maximise the payoffs, without having to make unreasonably strong assumptions over the nature of a reward function, about which so little is know.

Other possible applications include training multi-armed bandits to rank web documents by relevance and exhibit the k best [11]. Bandits can also be used to increase the statistical power of clinical trial results where the population under study is divided into many subpopulations (e.g. different ethnicities) [10]. They also have applications in reservoir location in the oil industry and can even play games [5].

Gaussian Process (GP) bandits, in particular, construct a probabilistic representation of the objective function, called its *surrogate*. Then they sample the objective function in areas corresponding to an optimum of the surrogate. This action increases our knowledge of the objective function and modifies the surrogate accordingly. That modification is dictated by the covariance kernel of the GP, which depends on a certain number of fixed parameters called *hyperparameters*. They can substantially influence the performance of the GP-bandit and yet their values is difficult to set a priori, specially when optimising black-box functions.

To respond to that issue the author of this paper has developed, during his MSc. thesis [4], a new framework named *Dynamic Learning Gaussian Process* (DLGP) *bandit* where the GP bandit learns the hyperparameters. This learning process is none trivial though, as many complications arise from the high correlation between sampled observations. This new framework ultimately allows us to make more informed choices about which values to sample, making it converge to the optimum after fewer function evaluations.

2 Gaussian Process (GP)

Gaussian Processes are a generalisation of multivariate normal distributions, instead of just being a distribution over vectors they define a distribution over functions. Conceptually GPs are based on the naive, yet effective, idea that a function $f(\mathbf{x})$ can be regarded as an infinite vector whose values correspond to $f(\mathbf{x})$ evaluated at every possible input $\mathbf{x}$.

Let $\mathcal{X}$ be a non-empty index set, which can either be countable (e.g.. $\mathbb{N}^d$) or uncountable (e.g.. $\mathbb{R}^d$). Moreover, let $\mathbf{x}_i$ be a specific value taken by $\mathbf{x}$.

Definition 2.1 *[8] A Gaussian Process $\{f(\mathbf{x}), \mathbf{x} \in \mathcal{X}\}$, is a family of random variables $f(\mathbf{x})$, all defined in the same probability space. In addition, for any finite subset $\mathcal{F} \subset \mathcal{X}$, with $\mathcal{F} := \{\mathbf{x}_{\pi_1}, \ldots, \mathbf{x}_{\pi_n}\}$, the random vector $\mathbf{f} := [f(\mathbf{x}_{\pi_1}), \ldots, f(\mathbf{x}_{\pi_n})]^\top$ has a (possibly degenerate) Gaussian distribution.*

A GP is fully characterised by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$. They are in turn defined as $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$:

$$\begin{aligned} m(\mathbf{x}) &:= \mathbb{E}[f(\mathbf{x})], \\ k(\mathbf{x}, \mathbf{x}') &:= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))\,(f(\mathbf{x}') - m(\mathbf{x}'))]. \end{aligned}$$

The GP is denoted as,

$$f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right). \tag{2.1}$$

In the literature the mean function is usually disregarded for notational simplicity, hence considering $m(\mathbf{x}) = \mathbf{0}(\mathbf{x})$, where $\mathbf{0}(.)$ is the zero function.

The main requirement of the covariance function or kernel is to be positive semidefinite. Amongst the best known examples of covariance functions there is the *Automatic Relevance Determination* (ARD) kernel, defined as

$$k_{ARD}(f(\mathbf{x}), f(\mathbf{x}')) := \sigma_0^2 \exp\left(-\tfrac{1}{2}(\mathbf{x} - \mathbf{x}')^\top M(\mathbf{x} - \mathbf{x}')\right), \tag{2.2}$$

where M is a positive semidefinite matrix generally set as $M = \mathrm{diag}(\mathbf{h})^{-2}$. The non-null hyperparameters σ_0 and $\mathbf{h}$, can either be set by the user or tuned using some optimisation techniques. The elements of vector $\mathbf{h}$ are called *length-scale* parameters.

In practice, when analysing a finite set $\mathcal{F}$ we find a new random variable $\mathbf{f} \in \mathbb{R}^{|F|}$ by evaluating the GP only at the points in $\mathcal{F}$. Now, consider that we have n input values drawn from the GP in (2.1), gathered in a matrix $X_* = (\mathbf{x}_1^*, \ldots, \mathbf{x}_n^*)^\top$, with $*$ indicating that we have not observed $f(\mathbf{x}_i^*)$ (with or without measurement noise). Then, the generated random variable $\mathbf{f}_* \in \mathbb{R}^n$ follows a multivariate gaussian distribution,

$$\mathbf{f}_* \sim \mathcal{N}(\mathbf{0}, K(X_*, X_*)), \tag{2.3}$$

where $[K(X_*, X_*)]_{i,j} := k(\mathbf{x}_i^*, \mathbf{x}_j^*)$ is a covariance matrix in $\mathbb{R}^{n \times n}$.

We assume that each observation $f(\mathbf{x}_i)$ is tainted with some gaussian noise $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$

$$y_i = f(\mathbf{x}_i) + \varepsilon_i. \tag{2.4}$$

As a result the n-training set is composed of pairs $\{(\mathbf{x}_i, y_i)\}_{i=1:n}$, in matrix notation $(X, \mathbf{y})$, where with $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$ and $\mathbf{y} = (y_1, \ldots, y_n)^\top$.

As the GP process is a natural conjugate prior over functions is is trivial to find the conditional distribution of the test cases given the training examples, in other words $\mathbf{f}_*|X, \mathbf{y}, X_*$. The predictive distribution simply becomes,

$$\mathbf{f}_*|X, \mathbf{y}, X_* \sim \mathcal{N}(\bar{\mathbf{f}}_*, \Sigma_*), \quad \text{where} \tag{2.5}$$

$$\bar{\mathbf{f}}_* := \mathbb{E}[\mathbf{f}_*|X, \mathbf{y}, X_*] = K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y}, \tag{2.6}$$

$$\Sigma_* := K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}K(X, X_*). \tag{2.7}$$

We see from (2.5) that the GP posterior at a given test input $\mathbf{x}_*$ depends on the training data X, on the choice covariance function and on the set of hyperparameters in the kernel. Those randomly chosen hyperparameters are to some extend a nuisance.

2.1 Evidence maximisation

Here we introduce an *evidence maximisation* based method to learning hyperparameters, also known as *marginal likelihood maximisation*. Its first step consists in determining the likelihood of the data given the collection of hyperparameters θ. Since, when training the Gaussian Process, we only work with a finite dataset then the likelihood corresponds to that of a multivariate Normal distribution.

We want to compute the evidence $P(\mathbf{y}|\mathbf{X}, \theta)$. We set the prior $P(\mathbf{f}|\mathbf{X}, \theta) \sim \mathcal{N}(\mathbf{0}, K)$, with $K := K(X, X)$. We obtain the evidence by noticing that in (2.4) we have $P(\mathbf{y}|\mathbf{X}, \theta) = \mathcal{N}(\mathbf{f}|\mathbf{0}, K) + \mathcal{N}(\varepsilon|\mathbf{0}, \sigma^2 I) = \mathcal{N}(\mathbf{0}, \Sigma)$. with $\Sigma := K(X, X) + \sigma^2 I$. It is also named

marginal likelihood. We find the optimal values of every hyperparameter $\theta_j \in \theta$ by taking the derivatives of the evidence with respect to each of them.

Although, evidence maximisation seems to be a good method to find the hyperparameters θ, it fails to take overfitting into account, hence the model fails to generalise to unobserved data. One way of avoiding this issue is to use leave–one–out cross–validation (LOO). This technique assumes that for input i in our dataset, renamed $\mathbf{x}_{*_i}$ for the occasion, we have not observed the corresponding output y_{*_i}. It then computes the log likelihood L_i of observing y_{*_i} given all other pairs of observations; this operation is repeated for every pair. Finally, we choose θ to maximise the joint likelihood of observing the entire n sample (L_{LOO}).

Let μ_i and σ_i^2 be the predictive mean and variance $\mathbf{x}_{*_i}$, respectively, computed from (2.6) and (2.7). Notice that these values are scalars since there is a only single element in the test set $X_* = \mathbf{x}_{*_i}^\top$. Also let the subscript $_{-i}$ designate matrices/vectors containing all the observations in the sample with the exception of the i^{th}. Consequently, we get

$$\begin{aligned} L_i &:= \log P(y_{*_i}|\mathbf{x}_{*_i}, y_{-i}, X_{-i}, \theta) \\ &= -\tfrac{1}{2}\log\sigma_i^2 - \frac{(y_{*_i}-\mu_i)^2}{2\sigma_i^2} - \tfrac{1}{2}\log(2\pi) \end{aligned}$$

The hyperparameters are found by maximising the following,

$$L_{LOO}(y, X, \theta) := \sum_{i=1}^{n} L_i. \tag{2.8}$$

Finally, for more technical details on how to maximise (2.8) in a computationally efficient way the reader is directed to [13] Chapter 5.4.2.

3 Gaussian Process Bandit

Bandits are a formal framework for solving reinforcement learning problems where a learner has to explore an unknown environment to try to get the highest possible cumulative reward. Bandits are an efficient way of trading off *exploration and exploitation* and are generally best described by the classical slot machine example. In this scenario the agent is in a casino and tries to maximise is cumulative earnings while playing at the slot machines.

The agent has to chose between playing N slot machines with different distributions over earnings, with expected payoffs μ_i. His aim is to find machine $i^* := \text{argmax}_{\text{i}\in\text{N}}\,\mu_\text{i}$ while avoiding wasting money on the machines with lower expected earning, along the way. To achieve that goal he will mainly play the slot machine that has been giving him the highest payoffs - *exploitation* - while sporadically playing other machines to verify that their payoffs have not been underestimated - *exploration*.

In the bandit context we consider each machine to be an *arm* of a *multi-armed bandit*. By choosing an appropriate algorithm to solve this task the hope is to minimise the average cumulated regret. The regret at time t being the gap between our maximal possible earning μ_{i^*} and our actual reward r_t, i.e.

$$R_t := \mu_{i^*} - r_t. \tag{3.1}$$

3.1 Context

GP-bandits rely on the assumption that the arms of the bandit are dependent, hence the reward obtained by pulling one arm delivers information about neighbouring arms. For example, if a person likes a movie that gives us indications about what other types of movies she might like.

We formalise this scenario by assuming that a reward function y is obtained from a latent function $f(.)$, which is a sample path generated from a $\mathcal{GP}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}'))$ distribution. It is evaluated at arms $\mathbf{x} \in \mathcal{F}$. Typically, $\mathcal{F}$ is taken to be an axis parallel hyper-rectangle of finite size. The observations are believed to be subject to some Gaussian noise ε, as in (2.4).

In this article, to trade off exploration versus exploitation we will only focus one of the most theoretically understood method, which has also been shown to give good empirical results, GP–UCB a version of the UCB algorithm created by [1]. Finally, without loss of generality we will only consider the case were we are looking for the maximum of $f(\,.\,)$.

3.2 GP–UCB

Gaussian Process - Upper Confidence Bound (GP–UCB) applies the principle of *optimism in the face of uncertainty* by valuing the elements $\mathbf{x}$ that potentially hold a great reward r_t but that at the same time lie in unsampled areas of the objective function's input space $\mathcal{D}$.

$$\mathbf{x}_t = \text{argmax}_{\mathbf{x}\in\mathcal{D}}\{\,\mu_{\text{t}-1}(\mathbf{x}) + \sqrt{\beta_\text{t}}\sigma_{\text{t}-1}(\mathbf{x})\,\} \tag{3.2}$$

The GP–UCB in (3.2) combines a greedy strategy $\mathbf{x}_t = \text{argmax}_{\mathbf{x}\in\mathcal{D}}\{\mu_{t-1}(\mathbf{x})\}$ taking the elements with highest expected value under the predictive distribution, computed from data gathered up to evaluation $t-1$, and balances it with an exploratory component. The term that allows for deviations from the greedy optimum is $\sqrt{\beta_t}\sigma_{t-1}(\mathbf{x})$, where $\sigma_{t-1}(.)$ is the predictive variance and β_t is the parameter that establishes the level of exploration.

In [15], it is shown that, for finite $\mathcal{D}$, the value of the tradeoff parameter,

$$\beta_t = 2\log\left(\frac{|\mathcal{D}|\,t^2\,\pi^2}{6\,\delta}\right), \quad \text{with} \quad \delta \in (0,1), \qquad (3.3)$$

holds good convergence properties.

Let μ, $k_0(.,.)$ and θ_0 be the prior mean, variance and hyperparameters respectively and $X_t := (\mathbf{x}_1,\ldots,\mathbf{x}_t)^\top$, then the GP–UCB algorithm can be stated as,

input: Input space $\mathcal{D}$; GP prior $\mu_0 = 0$, σ_0, $k_0(.,.)$, θ_0

for *t=1,2,...* **do**

- Generate a set of m arms $U := \{\mathbf{u}_1,\ldots,\mathbf{u}_m\} \subset \mathcal{D}$;
- Choose $\mathbf{x}_t \leftarrow \text{argmax}_{\mathbf{u}\in U}\{\mu_{t-1}(\mathbf{u}) + \sqrt{\beta_t}\sigma_{t-1}(\mathbf{u})\}$;
- Sample $y_t \leftarrow f(\mathbf{x}_t) + \varepsilon_t$;
- $\mu_t \leftarrow$ GP posterior mean with θ_0 and inputs X_t using (2.6);
- $\sigma_t^2 \leftarrow$ GP posterior variance with θ_0 and inputs X_t using (2.7);

end

Algorithm 1: GP–UCB algorithm

4 Dynamic Learning Gaussian Process Bandit

The case bellow illustrates one of the main weaknesses of GP–UCB. Consider a GP with an ARD kernel in which the length-scale of dimension x is $M_{1,1} = 0.4$ and the one along dimension y is $M_{2,2} = 1$, depicted in figure 1. As a result the GP varies more smoothly along y than x.

Now suppose that we wish to maximise this function. To do so we shall need to sample it more along x then y, because the probability of observing a sudden peak, holding the maximum, is higher with x than y. Nevertheless, GP–UCB will not be able to take this factor rapidly into account during the optimisation, since it never updates the prior hyperparameters.

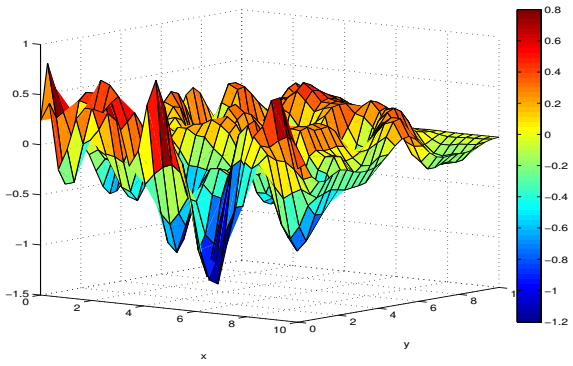

Figure 1: axis x on the left and y on the right

For instance, consider a petroleum company looking to increase its oil reserves by finding new reservoirs. For the company to know if a reservoir is exploitable expensive analysis are required (e.g. drilling, satellite imagining, etc.). Bandits can be used here to reach the best reservoirs while limiting costly analysis. Here X_t represents the coordinates of all areas analysed so far. Assuming peaks in figure 1 represent bigger reservoirs the company would prefer to focus on sampling along the x axis, hence a method allowing it to realise this could save the company a lot of money.

That is when the true strength of the DLGP is revealed as it is capable of dynamically learning the hyperparameters and more quickly avoid excessively sampling areas with low prospects. This is particularly important in high-dimensional problems where many dimensions may have low relevance to the objective, i.e. noise variables.

4.1 DLGP-bandit definition

The DLGP as presented in this section is just a general framework that introduces a hyperparameter learning stage onto GP–UCB without specifying how to learn them in practice. Consequently, that entire procedure is represented by a function call $h(\theta_0, \mathcal{A}_{t-1}, ...)$ in algorithm 2. This way practitioners will be able to improve DLGP further and tailor it to their needs, yet a very detailed algorithm for solving

$h(.)$ is presented in section 5.

input: Input space $\mathcal{D}$; GP prior $\mu_0 = 0$, σ_0, $k(.,.)$, θ_0 and ϕ

1 $\theta^* \leftarrow \theta_0$ and $\mathcal{A}_0 \leftarrow \{\emptyset\}$;
2 **for** *t=1,2,...* **do**
3 **if** $t \in \phi$ **then**
4 $\theta_t \leftarrow h(\theta_0, \mathcal{A}_{t-1}, ...)$ `select new hyperparameters`;
5 $\theta^* \leftarrow \theta_t$;
6 **end**
7 Generate a set of m arms $U := \{\mathbf{u}_1, \ldots, \mathbf{u}_m\} \subset \mathcal{D}$;
8 Choose $\mathbf{x}_t \leftarrow \text{argmax}_{\mathbf{u}\in U}\{\mu_{t-1}(\mathbf{u}) + \sqrt{\beta_t}\sigma_{t-1}(\mathbf{u})\}$;
9 $\mathcal{A}_t \leftarrow \{\mathcal{A}_{t-1}, \mathbf{x}_t\}$ `adding the last arm pulled to the set of pulled arms`;
10 Sample $y_t \leftarrow f(\mathbf{x}_t) + \varepsilon_t$;
11 $\mu_t \leftarrow$ GP posterior mean with θ^* at inputs X_t using (2.6);
12 $\sigma_t^2 \leftarrow$ GP posterior variance with θ^* at inputs X_t using (2.7);
13 **end**

Algorithm 2: DLGP algorithm

At first sight DLGP is not fundamentally different from the GP–UCB. In appearance it only includes a few extra commands, lines 3-6 in Algorithm 2, however they result in a behaviour substantially differently from that of its predecessor.

In essence, the DLGP starts off like the GP–UCB up until t reaches the first element in ϕ denoted ϕ_1. Indeed, it is only possible to start learning the hyperparameters when there is data on which to train them. That is why it is advised to have ϕ_1 larger then the number of hyperparameters in the kernel.

We define a set ϕ instead of just reevaluating the hyperparameters each time because the computational cost of learning θ increases more than linearly with the sample size but not the gain of recomputing θ. Indeed, as the sample grows the contribution of a single additional observation on determining θ will become smaller and not worth the extra computation.

4.2 Dependent data issues

In the next section we shall suggest one way of constructing $h(.)$, but first we discuss a certain number of general issues related to hyperparameter learning. These are specific to the GP-bandits and that the reader should be aware off. These include the complications caused by the fact that data in the set of pulled arms $\mathcal{A}_t$ is not *iid* and why normal cross-validation fails to prevent overfitting in this situation.

Indeed, when using bandits, the arm to pull next $\mathbf{x}_{t+1}$ always depends on all past arms $\mathcal{A}_t$. As we perform k-fold cross-validation, we have a test set $\mathcal{X}_\mathcal{V} := \{\mathbf{x}_i : i \in \mathcal{V}\}$, where $\mathcal{V} \subset \{1, \ldots, t\}$, $|\mathcal{V}| = t/k$ and a training set $\mathcal{X}_{-\mathcal{V}} := \{\mathbf{x}_i : i \in \mathcal{V}^c\}$. If the data were *iid* then $\mathcal{X}_\mathcal{V} \perp\!\!\!\perp \mathcal{X}_{-\mathcal{V}}$, but when working with dependent data neighbouring observations contain information about each other and the relation no longer holds.

When applying the DLGP, sampled input points tend to cluster around the argument of local optima, i.e. the value $\mathbf{x}$ such that $f(\mathbf{x})$ is a local optima. It would seem that as the bandit learns the hyperparameters it compresses space and considers data points $\mathbf{x}_i$ to be closer then they really are. A similar problem was described in kernel regression [6], with positively correlated data, where the kernel bandwidth tended to be overestimated with correlated data more than with uncorrelated data.

Consequently, the learner tends to overestimate the length-scale hyperparameters typically considering the objective objective function to be too smoother. This is most probably due to the fact that the hyperparameter estimator ignores the dependence in the measurements and attributes their positive correlation to the smoothness of the underlying function.

A method to cross-validate dependent data, commonly used in time series, is *modified cross-validation* or *h-blockage* [3]. This approach involves selecting a term $h > 0$ that controls for the distance of the blockage and then removing the observations less then h periods away from the test value.

The method that we shall use relies on this concept, while adding a geometric component to the technique. This method will require the use of GMM's and silhouette coefficients that we will quickly review.

4.2.1 Gaussian Mixture Model (GMM)

We assume that the generative model for observations is a Gaussian mixture,

$$P(\mathbf{x}_i|\Theta) = \sum_{k=1}^{K} p_k \, \mathcal{N}(\mathbf{x}_i|\mu_k, \Sigma_k),$$

where $\Theta := (\Theta_1, \ldots, \Theta_K)^\top$ and $\Theta_k := (\alpha_k, \mu_k, \sigma_k)^\top$. The p_k's are the *mixing probabilities*, that is the prior probabilities on any observation $\mathbf{x}$ belonging to a cluster $\mathcal{C}_k$, so $p_k := P(\mathbf{x} \in \mathcal{C}_k) = P(\mathbf{x}_1 \in \mathcal{C}_k) \ldots P(\mathbf{x}_n \in \mathcal{C}_k)$.

Let the superscript (t) indicate the t^{th} estimation of a parameter. The Expectation Maximisation (EM) algorithm solves the Gaussian mixture model by recursively performing the following steps.

- E step (at recursion t):

$$r_{ik}^{(t)} := \frac{p_k^{(t-1)}\, \mathcal{N}(\mathbf{x}|\mu_k^{(t-1)}, \Sigma_k^{(t-1)})}{\sum_{k'} p_{k'}^{(t-1)}\, \mathcal{N}(\mathbf{x}|\mu_{k'}^{(t-1)}, \Sigma_{k'}^{(t-1)})},$$

- M step (at recursion t):

$$\mu_k^{(t)} = \frac{\sum_{i=1}^n r_{ik}^{(t)} \mathbf{x}_i}{\sum_{i=1}^n r_{ik}^{(t)}},$$

$$\Sigma_k^{(t)} = \frac{\sum_{i=1}^n r_{ik}^{(t)} (\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^\top}{\sum_{i=1}^n r_{ik}^{(t)}},$$

$$p_k^{(t)} = \frac{1}{n} \sum_{i=1}^n r_{ik}^{(t)}.$$

We notice that there is one parameter in the model that was considered known up to now but which must in fact be learned, that is the number of clusters K.

4.2.2 K means and silhouette coefficient

Let us consider the K means algorithm, that determines cluster means $\mu_1, \ldots, \mu_K$, such that:

$$\min_{\{\mu_1, \ldots, \mu_K\}} \sum_{k=1}^K \sum_{i \in \mathcal{C}_k} \|x_i - \mu_k\|^2.$$

We find the number of clusters K by using the *silhouette coefficient* $s(i)$ [14], obtained for each observation i in the sample.

Let $a_i := \frac{1}{|\mathcal{C}_i|} \sum_{j \in \mathcal{C}_i} \|\mathbf{x}_i - \mathbf{x}_j\|$ and $b_i := \min_{\mathcal{C} \in \mathcal{C}_i^c} \frac{1}{|\mathcal{C}|} \sum_{j \in \mathcal{C}} \|\mathbf{x}_i - \mathbf{x}_j\|$. Then

$$s(i) := \frac{b_i - a_i}{\max(a_i, b_i)},$$

and by construction $s(i) \in [-1, 1]$. In the case where the cluster $\mathcal{C}_i$ only contains i we set $s(i) = 0$. Then, computing the average silhouette coefficient over the entire sample indicates the quality of the clustering. K is chosen so as to obtain the maximal average.

5 Defining $h(.)$

In this section we present an algorithm that can be implemented to learn the hyperparameters in the DLGP framework, in other words we are defining $h(\theta_0, \mathcal{A}_t, ...)$. Conceptually, it is based on the fact that dependent observations tend to group in clusters instead of being randomly spread across the axis parallel hyper-rectangle. As the correlation typically decreases with the Euclidean distance between the inputs, then two observations in the same cluster generally contain more information about each other then two observations in two different clusters.

input: $\mathcal{A}_t$, θ_0, w
for *k=2,3,...* **do**
  run k means;
  $\bar{sc}(k) \leftarrow$ average silhouettes $s(i)$
  if $\max\{\bar{sc}(k), \ldots, \bar{sc}(k-w+1)\} < \bar{sc}(k-w)$ **then**
    $K \leftarrow k - w$;
    leave loop;
  end
end
$t \leftarrow 0$;
while *EM not converged* **do**
  $t \leftarrow t + 1$;
  $r_{ik}^{(t)} \leftarrow$ E step (for K clusters);
  $(\mu_k^{(t)}, \Sigma_k^{(t)}, p_k^{(t)}) \leftarrow$M step(for K clusters);
end
for *i=1,2,...,n* **do**
  $P(\mathbf{x}_i) = \sum_{k=1}^K p_k^{(t)}\, \mathcal{N}(\mathbf{x}_i|\mu_k^{(t)}, \Sigma_k^{(t)})$;
  $\gamma \leftarrow \mathcal{U}(0, 1)$;
  $f(\mathbf{x}_i) \leftarrow \gamma P(\mathbf{x}_i)$
end
$\Delta \leftarrow$ only keep the $n - h$ lowest values in $\{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)\}$;
perform LOO on Δ and use conjugate gradient to maximise θ;

Algorithm 3: clustered cross-validation algorithm

If we chose to perform LOO cross–validation, we keep in the training set all the observations that belong to the cluster containing the observation in the test set, then the method will fail to prevent overfitting. This is because the training set contains too much information about the test set. Therefore, the

idea is that before starting the cross–validation we compute the probability $P(\mathbf{x}_i)$ of each input being sampled under a Gaussian mixture model (GMM) and we then trim h arms whose *modified probability* under that model is highest. $P(\mathbf{x}_i)$ is modified by multiplying it by a random number generated from a Uniform $\mathcal{U}(0,1)$. Those observations are removed from both the training and test sets, so it can be view as a sort of geometry based *h–block*.

Note that, the trimmed observations do not take part in learning the hyperparameters at time ϕ_i, but they are not lost so they can possibly be used at time ϕ_{i+1}. Furthermore, although they are not used to learn the hyperparameters, they are always used to compute the GP predictive distribution.

As the DLGP converges to its target, samples will be drawn increasingly close to the argument of the optimum and to each other. This will cause them to form a cluster, but this time a desirable one. So to avoid penalising this situation too much we use modified probabilities. The removed observations will tend to be at the centre of clusters but not always of the largest one. This modification has proven to give better results in terms of empirical convergence speed of the algorithm.

Finally, the number clusters in the GMM are determined by iteratively running K means with different K values and evaluating the silhouette coefficient for the model with different K's [9]. The number of clusters corresponding to the K with the highest coefficient.

This algorithm should be put in lieu of line 4 on the DLGP (algorithm 2). With w representing how many times the loop can increase K, while the silhouette coefficient of the corresponding model decreases, before it stops and settles for a given value of K.

6 Comparison

In this section we will compare the performances of both the GP-bandit and the DLGP-bandit algorithms against a test suite of standard benchmark functions, the optimisation literature, [7] and [17]. This was implemented in MatLab with the help of the GPML toolbox [12].

The algorithms will be judged according two main criteria, one being the speed of convergence to the global optimum and the second, more in tune with reinforcement learning, is the regret.

6.1 Methodology

At every step t of every experience a set of arms to pull, U_t, is randomly generated and then U_t is presented to both algorithms. Once they have chosen an arm the same random noise value ε_t will blur the observation, (2.4).

The speed of convergence does not solely depend on the quality of the algorithms, it also depends on the sequence of possible arms (U_t) that is presented to it and mostly for GP–UCB on the values of the hyperparameters. That is why the algorithms are rerun a few times to marginalise over those random components. At each run the hyperparameters start off as being the same and their values is randomly initialised in the interval $(0, 1)$.

The performance of the DLGP and GP–UCB algorithms will be evaluated according to two metrics. The first metric is the Euclidean distance of the j^{th} closest element to the optimum,

$$\mathcal{D}_t^{(j)} := \min\{\|\mathbf{x}_i - \mathbf{x}_{opt}\| : \mathbf{x}_i \in \mathcal{A}_t^{(j)}\}$$

where $\mathcal{A}_t^{(j)}$ corresponds to $\mathcal{A}_t$ once the $j-1$ closest elements to the optimum were removed. Then $\bar{\mathcal{D}}_t^{(i)}$ is the average $\mathcal{D}_t^{(i)}$ over all generated sample paths under the same distribution. The bigger j is the more robust the metric.

The second measure we will use is the average over all sample paths of the individual average regret over time, $\bar{R}_t := \frac{1}{t}\sum_{i=1}^{m} R_{t,i}$ where (3.1) with an additional i indicates which of the m sample paths it corresponds to. Note that again we average $\bar{R}_t$ over different runs to gain additional robustness.

6.2 Test suite

The functions that are used as benchmarks have different characteristics that may disrupt an optimisation algorithm, such as being multimodal, trapping the method in local optima. All these issues tend to get exacerbated with the dimensionality of the function. However, one of the characteristics that often dominates the complexity of the search is known as separability. Separability indicates if a n-dimensional optimisation problem can be solved as n separate optimisations.

The functions in our test suit include the Rastigin F_1, Griewank F_2 and Ackley's F_3. The search domain for F_1 is $[-5, 5]^D$ while it is $[-100, 100]^D$ for F_2 and

F_3. All of them are multimodal and only F_2 is non-separable.

$$F_1(\mathbf{x}) = 10\,D + \sum_{i=1}^{D} [\mathbf{x}_i^2 + \cos(2\pi \mathbf{x}_i)]$$

$$F_2(\mathbf{x}) = \sum_{i=1}^{D} \frac{x_i}{4000} - \prod_{i=1}^{D} \cos \frac{x_i}{\sqrt{i}} + 1$$

$$F_3(\mathbf{z}) = -20 \exp\left(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^{D} z_i^2}\right) - \exp\left(\frac{1}{D} \sum_{i=1}^{D} \cos(2\pi z_i)\right) + 20 + e^1$$

Their global optimum can be found at $\mathbf{x}^\star = \mathbf{0}$ and $F(\mathbf{x}^\star) = 0$. However, here we do not use the Ackley's function itself but a shifted version. : $\mathbf{z} := \mathbf{x} + \mathbf{o}$, with $\mathbf{o}$ being a vector that shifts the functions coordinates with respect to standard function. The global optimum of F_3 is at $\mathbf{z}^\star = \mathbf{o}$ and $F(\mathbf{z}\star) = 0$

7 Results

For the 2 dimensional functions simulations a Gaussian noise $\mathcal{N}(0, 0.3)$ was used. The parameters of the DLGP were set as follows: $\phi_1 = 20$ and then increases by steps of 50, i.e. $\phi_{i+1} = \phi_i + 50$.

The figures on the next page show the the behaviour of the algorithms on each objective function across function evaluations. The first image of each figure represents the progression of $\bar{\mathcal{D}}_t^{(5)}$ and the lower right the progression of $\bar{R}_t$.

7.1 Comments

- **Rastirgin function**: the measure $\mathcal{D}_t^{(5)}$ converges more quickly to the optimum using the DLGP than the GP–UCB, however the latter has less regret then the former. This is because the GP–UCB gets more easily stuck at local minima not far from the optimum, which is at the centre of a quadratic-like function. The DLGP though keeps on exploring the area surrounding the optimum but since there are some peaks in that area, each time it evaluated the function there that increased its regret. This tends to justify that regret is not as appropriate as $\mathcal{D}_t^{(5)}$ in this situation.

- **Ackley's function**: the difference between the results given between the DLGP and the GP–UCB is due to the fact that the GP–UCB is almost doing a random search. Indeed, the surface of the function is almost flat with the exception of a pit that contains the global minimum. As a result the only few time that the GP–UCB is close to the optimum is when it lands nearby by chance and since we are using a robust measure that kind of random guesses is penalised. The DLGP on the other hand manages to cluster around the solution get increasing closer to it at each evaluation.

8 Conclusion

Bandits allow for a very flexible non-parametric approach to black-box function optimisation that does not require many constraining assumptions. The ease of application of GP-bandits and the existence of numerous freely available toolboxes that support GPs make them a viable optimisation techniques.

The new Dynamic Learning Gaussian Process (DLGP) setup that was introduced here shows the GP-bandit's true potentialities, with performances up to 20 times faster on some benchmark functions than current state of the art GP–UCB.

Nevertheless, the perspectives for their improvement are wide. For instance, using more sophisticated hyperparameter optimisation algorithms, instead of just performing conjugate gradient. Furthermore, we have restricted model selection to hyperparameter optimisation, but that could be extended to kernel learning.

In conclusion, DLGP is an algorithm that has proven to be better then GP–UCB in a variety of situations. Although further analysis is needed with higher dimensional functions, intuitively if the hyperparameters are learnt consistently DLGP should always be better (or at worst as good as) than GP–UCB. This means that some sort of clustering penalisation should always be included. Consequently, it appears that the DLGP should be preferred to GP–UCB, specially if processor time is not a constraint.

	DLGP					GP–UCB				
	Best	Median	Worst	Mean	Std	Best	Median	Worst	Mean	Std
F_1	0.0724	0.157	0.256	0.144	0.0757	0.429	0.454	1.27	0.607	0.368
F_2	0.725	0.881	1.38	0.954	0.262	6.82	6.96	8.08	7.16	0.515
F_3	0.291	0.529	0.773	0.521	0.176	8.74	10.2	10.4	9.75	0.821

Table 1: comparison of $\mathcal{D}_{500}^{(5)}$ for DLGP and GP–UCB on 2D functions

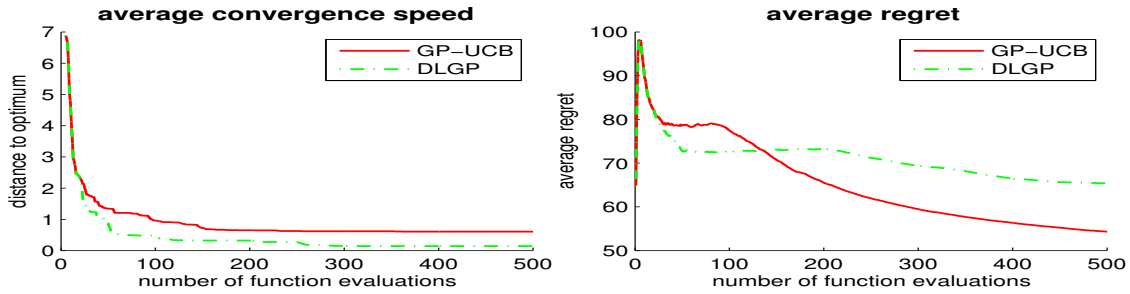

Figure 2: Rastirgin 2D

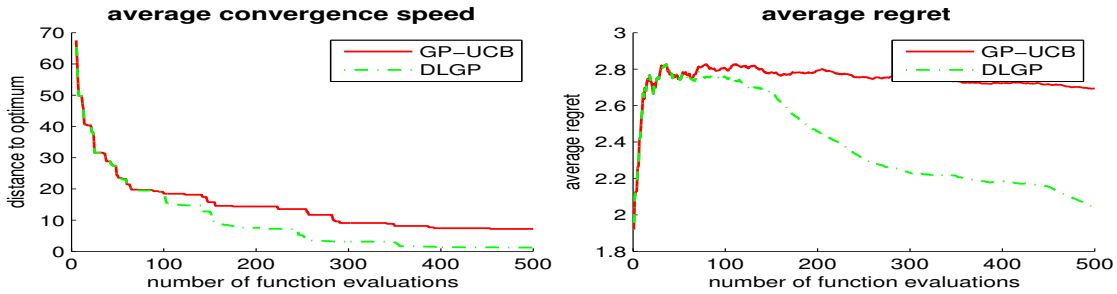

Figure 3: Griewank 2D

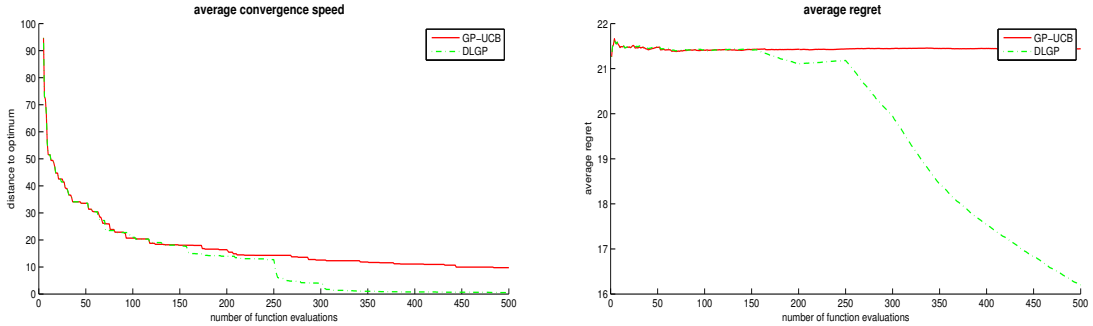

Figure 4: Ackley's 2D

References

[1] P. Auer. Using Confidence Bounds for Exploitation-Exploration Trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

[2] E. Brochu, V. M. Cora, and N. D. Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions , with Application to Active User Modeling and Hierarchical Reinforcement Learning. Technical report, University of British Columbia, 2010.

[3] P. Burman, E. Chow, and D. Nolan. A cross-validatory method for dependent data. *Biometrika*, 81(2):351–358, 1994.

[4] D. Dubart Norinho, D. Silver, and J. Shawe-Taylor. Dynamic Learning Gaussian Process bandits. Master's thesis, University College London, 2012.

[5] Y. Fan. Bandit Algorithms in Game Tree Search : Application to Computer Renju. Technical report, University of British Columbia, June 2012.

[6] D. Hart and T. E. Wehrly. Using Repeated Estimation Regression Kernel Data Measurements. *Journal of the American Statistical Association*, 81(396):1080–1088, 1986.

[7] F. Herrera, M. Lozano, and D. Molina. Test Suite for the Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems. Technical report, University of Granada, 2010.

[8] S. P. Lalley. Gaussian processes; kolmogorovchentsov theorem. Technical report, University of Chicago, 2008.

[9] Y. Lee, K. Y. Lee, and J. Lee. The Estimating Optimal Number of Gaussian Mixtures Based on Incremental k-means for Speaker Identification. *International Journal of Information Technology*, 12(7):13–21, 2006.

[10] W. H. Press. Bandit solutions provide unified ethical models for randomized clinical trials and comparative effectiveness research. *Proceedings of the National Academy of Sciences of the United States of America*, 106(52):22387–92, Dec. 2009.

[11] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*, pages 784–791, New York, New York, USA, 2008. ACM Press.

[12] C. E. Rasmussen and H. Nickisch. Gaussian Processes for Machine Learning (GPML) Toolbox. *Journal of Machine Learning Research*, 11:3011–3015, 2010.

[13] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning.* MIT Press, 2 edition, Apr. 2006.

[14] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, Nov. 1987.

[15] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian Process Optimization in the Bandit Setting : No Regret and Experimental Design. In *27th International Conference on Machine Learning*, Haifa, 2010.

[16] M. Streeter, A. Krause, and D. Golovin. Online Learning of Assignments. Technical report, Carnagie Melon University, 2007.

[17] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise. Benchmark Functions for the CEC ' 2010 Special Session and Competition on Large-Scale Global Optimization. Technical Report 1, Nature Inspired Computation and Applications Laboratory, 2010.