FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Jakub Arnold

Bayesian Optimization of Hyperparameters Using Gaussian Processes

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	RNDr. Milan Straka, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2019

Dedication.

Title: Bayesian Optimization of Hyperparameters Using Gaussian Processes

Author: Bc. Jakub Arnold

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Milan Straka, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Abstract.

Keywords: gaussian process bayesian optimization global optimization neural network

Contents

Introduction

1. Introduction

intro (2 stranky) - co jsou hyperparam, ze to model neumi nastavit, atd. - proc chceme delat black box, ...

A machine learning algorithm is an algorithm that can learn from data. A commonly used definition is: "A computer program is said to learn form experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [Mitchell, 1997]. We only provide this definition for completeness and our following definitions will assume an intuitive translation to this definition.

Machine learning models are traditionally split into two categories, parametric and non-parametric. Parametric models have a fixed number of learnable parameters which are trained on a subset of the data called the training set. On the other hand, non-parametric models typically have a variable number of parameters which depends on the size of the data. This could mean that there is a parameter associated with each data point. Non-parametric does not mean that there are no parameters.

Most machine learning algorithms also have hyperparameters, which are parameters that the learning algorithm can not learn itself, and they usually control the its behavior. A parameter could be chosen to be a hyperparameter because it would not be reasonable to infer its value from the training data. An example could be the type of optimizer being used to train a neural network (e.g. SGD or Adam [Kingma and Ba, 2014]). The machine learning practicioner would not consider the optimizer to be a property of the data distribution, and as a reasult treat it as a hyperparameter set externally, rather than trying to infer it.

It could also be set as a hyperparameter simply for practical reasons, where in theory the parameter could be learned from the data, but optimizing it directly would be too difficult. An example here could be the learning rate of a stochastic gradient descent optimizer. Even automatic differentiation [Maclaurin et al., 2015] allows us to compute the gradient flow through arbitrary code, in practice it is not being used to compute the gradients of hyperparameters like the parameters of an optimizer, e.g. the learning rate. Instead, the learning rate is treated as a hyperparameter and is set ahead of time, or according to a fixed schedule. Even when early stopping or other heuristic methods are employed, it would still be treated as a hyperparameter from the perspective of the learning algorithm.

Bayesian statistics allows us to take a principled approach to setting hyperparameters. We would simply set a prior distribution on each hyperparameter and marginalize over them, but unfortunately this has two problems. Firstly, the prior distribution almost always has a parameter of its own, such as the rate of a poisson distribution. We could go one step further and specify a prior on these parameters, which are often called hyperpriors, and this is actually sometimes done **??**. But the real problem of bayesian methods is that the integral in the marginalization often ends up being intractable, and in the case of more complicated as neural networks, we are already computatationally bottlenecked and can not call upon methods such as Markov-Chain Monte Carlo (MCMC) to approximate it (*tuhle vetu napsat jinak lol*).

There are however cases when the bayesian approach does work. Either the model is small and simple enough so that the integral can be actually computed, or its properties (such as conditional independence in LDA [Blei et al., 2003]) allow us to perform more efficient MCMC sampling, or the model could actually have an analytic solution to the marginalization. The last case is something we will make use of later on when we introduce Gaussian Processes.

Unfortunately, in the field of deep learning [Goodfellow et al., 2016] and neural networks, our models are almost always so large that just computing a point estimate of the parameters is right at the edge of our computational limits **??**. For that reason we do not usually employ bayesian methods, but look for alternative – less computationally heavy – approaches.

A simple approach to hyperparameter tuning is either via random search, where each parameter is sampled randomly from a given prior distribution, or via grid search, where a fixed grid over the hyperparameter space is chosen, and then each point on the grid is evaluated, and the best parameter combination is chosen. The evaluation can be done using an arbitrary metric of performance, called the **objective function** . This could simply be the loss achieved by the model on the validation set.

It is not difficult to see that neither of these approaches are optimal. Both do not take into account the already computed values of the loss. If the model is evaluated on one set of hyperparameters and achieves a high loss, we would like the hyperparameter tuning mechanism to take this into account, and possibly try a different combination. This process is similar to that of an agent trying to balance the exploration-exploitation tradeoff **??**. On one hand, we would like to explore different combinations of hyperparameters and explore as much of the search space as possible. But once we find a combination with a high value of the objective function we would want to explore the space around that combination to exploit the high value and possibly find a close combination that is even better.

Therefore, we'd like our hyperparameter optimization procedure, also called the **meta-optimizer** , to balance the exploration-exploitation tradeoff, and take previous evaluation into account when choosing which point to evaluate next. Since the training of machine learning models – and neural networks in particular – can be computationally costly, we need to pick an optimization procedure that is sample efficient. Many modern deep learning models take on the order of days, weeks, or even months to train on high end hardware. To give a few examples, the recent OpenaI Five **??** has consumed 800 PETAFLOPs-days over the course of 10 months. In comparison, a consumer grade GPU GTX 1080 Ti achieves just over 11 TFLOPs. A smaller and more realistic example would be the NVIDIA StyleGAN **??** trained on 1024×1024 images takes almost 7 days on $8\times$ Tesla V100 GPUs. In our experiments (see **??**) we train a tagger and lemmatizer on Czech and English treebanks, where each evaluation takes about four GPU hours. Evaluating the objective function would mean training one such model from scratch with a given set of hyperparameters to obtain an unbiased estimate of its performance.

najit nejaky nazev a pouzivat konzistentne od zacatku

Even though we have the ability to evaluate the objective function, we have no way of computing its gradients, or obtain its analytic form. As a result we have to treat it as a black box and use optimization techniques which don't require either.

But even with the smaller models we have just shown, it is easy to see that we can not perform more than a few hundred of evaluations without spending unreasonable costs on computational resources. This immediately disqualifies many of the common black box optimization techniques, such as evolution strategies or simmulated annealing, which require on the order of thousands or even millions of evaluations to converge **??**.

Bayesian optimization is a black box optimization technique which utilizes a probabilistic model to take a set of evaluations of the objective function and computes a posterior over all possible functions give the observed data. As the next step, it computes an **acquisition function** , which is a function of the posterior, and represents the potential usefulness of the next sample. A popular example is the **expected improvement** function (**??**), which is roughly defined as *the expected improvement over the maximum obtained so far.* By sampling at the maximum of the acquisition function we obtain the point that is most likely going to help us the most. A key insight here is that the model is probabilistic. It does not simply fit a regression line through the data points and find the maximum. Instead, we fit a **Gaussian Process** **??** which allows us to capture the uncertainty in the predictions. This uncertainty is taken into account by the acquisition function, and as a result it ends up balancing the exploration-exploitation tradeoff by both taking into account the predicted value, and our uncertainty in that value.

1.1 Our Contributions

- co jsme udelali

We implemented a tool for optimizing arbitrary programs' hyperparameters using bayesian optimization, including a scheduler which runs the evaluations on a cluster, and a web interface visualizing the results. We do not provide any theoretical extensions to bayesian optimization – the benefit is only in implementation. This work however also serves as a thorough introduction to the theoretical background, specifically on gaussian processes (GP). Understanding the theoretical aspects of GPs allows the user to interpret the behavior of the optimizer, as well as better understand why some result visualizations might look the way they do.

zkratku zacinat na zacatku

Our implementation of bayesian optimization utilizes the GPy library **??** which implements the basic GP regression model. We chose to use the library mainly for the reasons of numerical stability. In theory, as we will show later **??**, impelmenting GP regression is simple for a small toy example. But with realistic data it is easy to run into numerical issues, some of which we will go over in the following chapters **??**. Apart from numerically stable code, the GPy allows for more control over the hyperparameters of the GP using constrained optimization.

Fitting a GP model and optimizing the acquisition function is just the beginning. A non-trivial amount of the work is devoted to evaluating the function in a flexible way. In our case, we expect the user to provide a script which encapsulates the function, accepts its parameters in the form of command line arguments, and prints the result of the function on its standard output. This approach allows us to run the evaluations in parallel, or even run them on a cluster. The implementation is flexible enough so that a user could provide their own way of running the evaluations, should they have specific needs. The experimental data is also stored in an easy to access text format, with command line utilities that allow the user a fine grained control over the experiment.

Lastly, running real-life experiments can be a time consuming process, and having the ability to monitor the process and interfere if needed is an important feature. This is why we provide a web interface that can visualize the progress of the optimization. The user can look at the evaluated samples, as well as the regression model at any point in time during the optimization.

2. Bayesian Optimization

Consider the problem of optimizing an arbitrary continuous function $f : \mathcal{X} \to \mathbb{R}$ where $\mathcal{X} \subset \mathbb{R}^d, d \in \mathbb{N}$. We call f the *objective function* and treat it as a black box, making no other assumptions on its analytical form, or on our ability to compute its derivatives. Our goal is to find the global minimum $\mathbf{x}_{\text{opt}}$ over the set $\mathcal{X}$, that is

$$\mathbf{x}_{\text{opt}} = \arg\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}).$$

We also assume that the evaluation of f is expensive, as the goal of bayesian optimization is to find the optimum in as few evaluations as possible. Consider the case when evaluating f means performing a computation that is not only time consuming, but for example also costs a lot of money. We might only have a fixed budget which puts a hard limit on the number of evaluations we can perform. If the function can be evaluated cheaply, other global optimization approaches such as simulated annealing or evolution strategies could potentially yield better results **??**.

Bayesian optimization techniques are some of the most efficient approaches in terms of the number of function evaluations required. Much of the efficiency stems from the ability to incorporate prior belief about the problem and to trade of exploration and exploitation of the search space [Brochu et al., 2010]

It is called bayesian because it combines the prior knowledge $p(f)$ about the function together with the data in the form of the likelihood $p(x|f)$ to formulate a posterior distribution on the set of possible functions $p(f|x)$. We will use the posterior distribution to figure out which point should be evaluated next to give a likely improvement over the currently obtained maximum. This improvement is defined by an **acquisition function** , which represents our optimization objective. A simple example of an acquisition function is the **probability of improvement** , which simply represents the probability of improving our objective compared to the previously achieved maximum. We will show a few other examples of acquisition functions in a later section **??**.

mezera pred carkou

The optimization procedure is sequential, using a bayesian posterior update at each step, refining its model as more data are available. At each step the we maximize the acquisition function in order to obtain the next sample point x_{i+1}. We then evaluate $f(x_{i+1})$ to obtain y_{i+1}, and incorporate it into the dataset. This processs is repeated for as many evaluations as we can perform. See algorithm 1 for pseudocode.

```
Initialize $\mathrm{x}_1$ randomly and evaluate $y_1 = f(\mathrm{x}_1), \mathcal{D}_1 = (\mathrm{x}_1, y_1)$.
for $i = 1, 2, \dots$ do
    Find $x_{i+1}$ by maximizing the acquisition function.
    Evaluate $y_{i+1} = f(\mathrm{x}(x_{i+1})$.
    Add the sample to the dataset $\mathcal{D}_{i+1} = \mathcal{D}_i \cup (\mathrm{x}_{i+1}, y_{i+1})$.
    Update the Gaussian Process.
end
```

Algorithm 1: Bayesian Optimization, Brochu et al. [2010]

At its core, bayesian optimization only requires two things. A probabilistic regression model which combines prior beliefs $p(f)$ with the data. And an acquistion function which describes the optimality of the next sampling point.

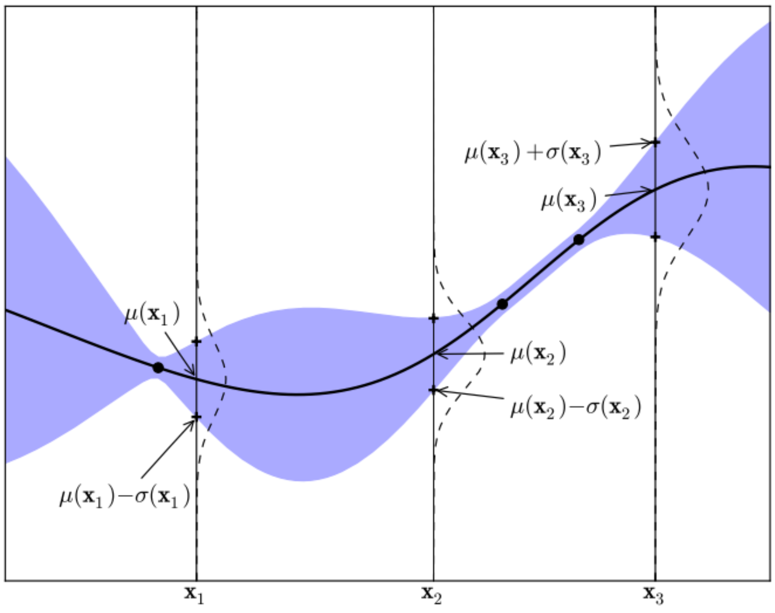
2.1 Prior Distribution over Functions

Bayesian methods by definition require a prior distribution over the quantity of interest. Since we are building a probabilistic model over functions, we need to obtain a prior distribution over functions, which will capture our general beliefs about the properties of the objective function. For example, if we knew that our function was periodic, we would want a prior distribution over periodic functions. But in the case of hyperparameter optimization we will generally only consider continuous functions.

We will follow the general consensus of using a Gaussian process (GP) as a prior distribution over functions [Brochu et al., 2010], as it provides many nice theoretical properties, as well as tractable posterior inference. A GP is an extension of the multivariate Gaussian distribution to infinitely dimensional random variables. Just as a multivariate Gaussian can be thought of as a distribution over vectors, a GP can be thought of as a distribution over infinitely dimensional vectors, which when indexed by the real numbers are equivalent to functions. A GP assumes that any finite subset $\mathrm{x} = (x_1, \dots, x_n)$ is jointly Gaussian with some mean $m(\mathrm{x})$ and covariance $\Sigma(\mathrm{x})$. A GP is defined by its mean function m and covariance function k [Murphy and Bach, 2012]. We write

zkratkovat od zacatku

$$\mathcal{GP}(m(\mathrm{x}), k(\mathrm{x}, \mathrm{x}')).$$



ukradeny obrazek, referencovat nekde

By convention, we assume that the prior mean is a constant zero function, that is $m(x) = 0$. Since the data is often normalized in practice, this does not reduce the flexibility of the model. The power of a GP comes from its covariance function, which for any two points x_i and x_j defines their covariance $k(\mathrm{x}_i, \mathrm{x}_j)$. If x_i and x_j have a high covariance, the values of the function at those points will be similar.

Intuitively, it is often useful to think of a GP as a function, which instead of returning a scalar $f(x)$ returns the mean and standard deviation of a Gaussian distribution over the possible values, centered at x. We leave a formal treatment of GPs until chapter 3.

—

2.2 Acquisition Functions

2.3 Related work

2.4 Hyperparam vs. Architecture Search

2.5 Diskretni hyperparam vs onehot vs ...

3. Gaussian Processes

- uvod, proc to delame - ucbnicove - kernely existujou

Definition 1. *A random variable* X *has a* **univariate Gaussian distribution** *, written as* $\mathrm{X} \sim \mathcal{N}(\mu, \sigma^2)$*, when its density is*

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}.$$

The parameters μ *and* σ *are its* mean *and* standard deviation*.*

Definition 2. *We say* X *has a* **degenerate Gaussian distribution** *when* $\mathrm{X} \sim \mathcal{N}(\mu, 0)$*.*

Definition 3. *A random variable* $\mathbf{X} \in \mathbb{R}^n$ *has a* **multivariate Gaussian distribution** *if any linear combination of its components is a univariate Gaussian, i.e.* $\mathbf{a}^T\mathrm{X} = \sum_{i=1}^{n} \mathbf{a}_i\mathbf{X}_i$ *is a Gaussian for all* $\mathbf{a} \in \mathbb{R}^n$*. We then write* $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$ *where* $\mathbb{E}[\mathbf{X}_i] = \mu_i$ *and* $\mathit{cov}(\mathbf{X}_i, \mathbf{X}_j) = \Sigma_{ij}$*.*

Remark. The parameters μ and Σ uniquely determine the distribution $\mathcal{N}(\mu, \Sigma)$.

Definition 4. *A random variable* $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$ *has a* **degenerate multivariate Gaussian distribution** *if* $\det \Sigma = \mathbf{0}$*.*

Remark. Given a random variable $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$, random variables $\mathbf{X}_1, \dots, \mathbf{X}_n$ are independent with distributions $\mathbf{X}_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ if and only if $\mu = (\mu_1, \dots, \mu_n)$ and $\Sigma = \mathit{diag}(\sigma_1^2, \dots, \sigma_n^2)$.

Theorem 1. *If a random variable* $\mathbf{X} \in \mathbb{R}^n$ *is a multivariate Gaussian, then* $\mathrm{X}_i, \mathrm{X}_j$ *are independent if and only if* $\mathit{cov}(\mathrm{X}_i, \mathrm{X}_j) = 0$*. Note that his is not true for any random variable, as it is a special property of the multivariate Gaussian.*

Proof. TODO □

4. Bayesian Optimization in depth

- bopt alg - jaky kernely v bayes opt., co pouzivame, proc (ref) - paralelni evaluace

5. Software

- co umime - vizualizace - jak se to pousti, runnery, serializace

6. Experiments

- toy tasky - porovnani existujici fuj fce na optimalizaci - srovnani acq/kernel, random search (ze to neco dela)

- maly ulohy
- velka uloha
- interpretace vysledku

- parser - tokenizer/segmentace - speech recognition - opennmt lemmatizace - $reinforce_with_baseline$

—

Let $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i \in 1 : n\}$ denote a set of n samples (evaluations) of the function f, that is $y_i = f(\mathbf{x}_i)$. Our goal is to pick the next $\mathbf{x}_{n+1}$ to maximize our chance of finding the optimum quickly.

Consider the set of all continuous functions $f \in \mathcal{F}$ with a prior distribution $p(f)$. Conditioning on our samples gives us a posterior distribution over possible functions $p(f \mid \mathcal{D})$.

—

Conclusion

Bibliography

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 2010. URL `http://arxiv.org/abs/1012.2599`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, 2015.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1997. ISBN 0070428077. URL `https://www.amazon.com/Machine-Learning-Tom-M-Mitchell/dp/0070428077?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0070428077`.

K.P. Murphy and F. Bach. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machi. MIT Press, 2012. ISBN 9780262018029. URL `https://books.google.cz/books?id=NZP6AQAAQBAJ`.

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 First Attachment

Index