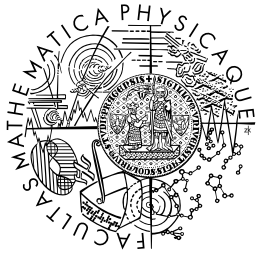


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

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

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

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

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. 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. 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 use methods such as Markov-Chain Monte Carlo (MCMC) to approximate it.

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 close to 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 [Russell and Norvig, 2016]. 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 [ope] 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 [Karras et al., 2018] trained on 1024×1024 images takes almost 7 days on $8\times$ Tesla V100 GPUs. In our experiments (see chapter 6) 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 evaluations to converge [Golovin et al., 2017].

Bayesian optimization [Shahriari et al., 2016] 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** (GP) 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

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 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.

Our implementation of bayesian optimization utilizes the GPy library [GPy, since 2012] 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 chapter 3, 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 chapter 2. 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 $\boldsymbol{x}_{\text{opt}}$ over the set $\mathcal{X}$, that is

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

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 [Golovin et al., 2017].

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 4.1.

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 $\boldsymbol{x}_1$ randomly and evaluate $y_1 = f(\boldsymbol{x}_1), \mathcal{D}_1 = (\boldsymbol{x}_1, y_1)$.
for $i = 1, 2, \ldots$ **do**
  Find $\boldsymbol{x}_{i+1}$ by maximizing the acquisition function.
  Evaluate $y_{i+1} = f(\boldsymbol{x}(x_{i+1})$.
  Add the sample to the dataset $\mathcal{D}_{i+1} = \mathcal{D}_i \cup (\boldsymbol{x}_{i+1}, y_{i+1})$.
  Update the probabilistic model.
end

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

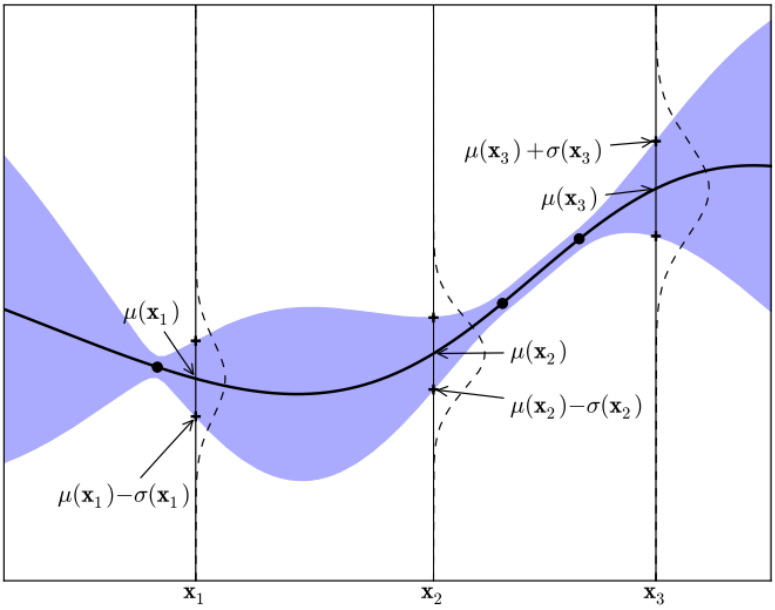
At its core, bayesian optimization only requires two things. A probabilistic regression model which combines prior beliefs $p(f)$ with the data. And an acquistion function which describes the optimality of the next sampling point.

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 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 $\mathrm{x} = (x_1, \ldots, x_n)$ 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

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



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 $k(x_i, x_j)$. 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 Function

TODO

2.3 Hyperparam vs. Architecture Search

Using our earlier definition, any parameter which the model does not learn on its own could be considered a hyperparameter. This definition is broad enough to allow a lot of flexibility, but some hyperparameters are better for the framework of bayesian optimization than others. Discrete and categorical hyperparameters require special consideration. Bayesian optimization itself is flexible in the sense that it allows for an arbitrary probabilistic model, but the specific choice does matter when we consider discrete values. In the case of GPs, the model itself does not have the ability to directly work with anything but continuous real valued vectors.

There has been some recent work [Garrido-Merchán and Hernández-Lobato, 2017] showing better approaches for handling integer valued variables. This is done by simply rounding the appropriate values before computing their covariance. As the kernel will see the values as equivalent, their covariance will be maximized, and the GP will be forced to predict a constant value over each integer region. While this approach does help a little bit with integer valued variables, it does not work for categorical (nominal) variables, which lack any form of ordering. If we simply treat them as integers, the GP prior will enforce relationships which do not exist.

An alternative approach to solving the problem with categorical variables is to use a different model than GPs, namely random forests [Shahriari et al., 2016]. Their main advantage is the ability to naturally handle various types of data. We however do not explore random forests in this work, as many of the hyperparameters of interest when tuning neural networks are either continuous or integer valued. Categorical variables, such as activation functions, are better suited to be tweaked as part of neural architecture search [Zoph et al., 2017].

Regardless of the probabilistic model, categorical variables cause many immediate problems. Consider the choice between SGD with momentum [Ruder, 2016] and Adam [Kingma and Ba, 2014]. While SGD with momentum has a *momentum* hyperparameter, Adam does not, but it has its own two extra hyperparameters, β_1 and β_2. This gives us two different sets of mutually exclusive hyperparameters. Bayesian optimization however does not have any natural way of handling problems like this. Even if we did externally enforce two different modes based on which categorical values was chosen, it would essentially be the same as run-

ning two experiments in parallel, one with SGD, and one with Adam. Another issue arises in visualization, which is one of the goals of this work. Inspecting the samples from two or more different modes of the network at once is challenging at best, and with multiple categorical variables the problem grows exponentially.

For these reasons, we chose to not provide direct support for categorical variables, apart form converting them to integer variables with an ordering and treating them as such. Some categorical variables can be optimized by simply treating them as fixed within a particular experiment, and then running multiple instances of that experiment with a different value each. Other categorical variables, such as the types of layers, activation functions, or even the connections between modules, are better left for the framework of neural architecture search, which treats them in a principled way.

2.4 Related work

There are a few notable mentions of related work in the area of tuning hyperparameters of neural networks. The main motivation for this work, is Google Vizier [Golovin et al., 2017], which is a proprietary service at Google used for black box optimization. There also exist a few implementations of bayesian optimization. Spearmint [Group, 2014] is the most fully featured one, but comes with a very restrictive license, and does not perform any visualization of the results. GPyOpt [authors, 2016] and scikit-optimize [Head et al., 2018] are the most notable libraries for bayesian optimization, but they only provide the basic optimization loop for bayesian optimization, and do not handle long running experiments in a possibly distributed environment.

3. Gaussian Processes

- uvod, proc to delame - ucbnicove - kernely existujou

This chapter goes into the technical details of the Gaussian distribution and its extension the Gaussian Process (GP). We think it is important to have at least a basic understanding of the underlying math to make intuitive claims about the behavior of the model, especially since GPs are a bit different from other parametric machine learning models.

Since our objective is bayesian optimization, we only derive the properties necessary for its implementation. Specifically, we are interested in the conditional and marginal distributions of a multivariate Gaussian. The conditional Gaussian distribution allows us to compute the posterior $p(f|x)$ at an arbitrary point, and the marginal allows us to fit a GP regression model to each hyperparameter separately for additional visualization.

Let us now continue with a more rigorous treatment of the Gaussian distribution. For a more thorough treatment see Bishop [2016] and Murphy and Bach [2012].

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.* $\boldsymbol{a}^T\mathrm{X} = \sum_{i=1}^n \boldsymbol{a}_i \mathbf{X}_i$ *is a Gaussian for all* $\boldsymbol{a} \in \mathbb{R}^n$. *We then write* $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$ *where* $\mathbb{E}[\mathbf{X}_i] = \mu_i$ *and* $\mathrm{cov}(\mathbf{X}_i, \mathbf{X}_j) = \Sigma_{ij}$.

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 $\Sigma = \mathrm{diag}(\sigma_1^2, \dots, \sigma_n^2)$.

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* $\mathrm{cov}(\mathrm{X}_i, \mathrm{X}_j) = 0$. *Note that his is not true for any random variable, as it is a special property of the multivariate Gaussian.*

Proof. TODO □

Theorem 2. *A Gaussian random variable* $\mathrm{X} \sim \boldsymbol{\mathcal{N}}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ *has a density iff it is non-degenerate (i.e.* $\det \boldsymbol{\Sigma} \neq 0$*, alternatively* $\boldsymbol{\Sigma}$ *is positive-definite). And in this case, the density is*

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma})}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right\} \tag{3.1}$$

Remark. The normalizing constant in the denominator is also often in an alternate form as

$$\det(2\pi\boldsymbol{\Sigma}) = (2\pi)^n \det(\boldsymbol{\Sigma})$$

which follows from basic determinant properties. Alternatively we can also put the square root in the exponent $(2\pi)^{n/2}(\det \boldsymbol{\Sigma})^{1/2}$.

Remark. A special case of the multivariate gaussian is when $n = 1$, then Note that if $n = 1$, then $\boldsymbol{\Sigma} = \sigma^2$, meaning $cov(X,X) = \sigma^2$, $\boldsymbol{\Sigma}^{-1} = \frac{1}{\sigma^2}$, and hence the multivariate Gaussian formula becomes the univariate one

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}. \tag{3.2}$$

3.1 Sampling

Even not of immediate interest for bayesian optimization, we will shortly show how to generate samples from a multivariate Gaussian, as this can be useful for visualization purposes with GPs.

Theorem 3. *Given a random variable* $\mathbf{X}$ *with* $cov[\mathbf{X}] = \boldsymbol{\Sigma}$*, it follows from the definition of covariance that* $cov[\mathbf{AX}] = \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T$*.*

Proof.

$$\begin{aligned}
cov[\mathbf{AX}] &= E[(\mathbf{AX} - E[\mathbf{AX}])(\mathbf{AX} - E[\mathbf{AX}])^T] && (3.3)\\
&= E[(\mathbf{AX} - \mathbf{A}E[\mathbf{X}])(\mathbf{AX} - \mathbf{A}E[\mathbf{X}])^T] && (3.4)\\
&= E[\mathbf{A}(\mathbf{X} - E[\mathbf{X}])(\mathbf{X} - E[\mathbf{X}])^T\mathbf{A}^T] && (3.5)\\
&= \mathbf{A}E[(\mathbf{X} - E[\mathbf{X}])(\mathbf{X} - E[\mathbf{X}])^T]\mathbf{A}^T && (3.6)\\
&= \mathbf{A}cov[\mathbf{X}]\mathbf{A}^T && (3.7)\\
&= \mathbf{A}\Sigma\mathbf{A}^T && (3.8)
\end{aligned}$$

□

Theorem 4. *Given a random variable* $\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ *and a positive-definite matrix* $\boldsymbol{\Sigma}$ *with a cholesky decomposition* $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^T$*, then*

$$\mathbf{LX} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}). \tag{3.9}$$

Proof. We can immediately use equation 3.8.

$$\mathbf{LX} \sim N(0, \mathbf{LIL}^T) = N(0, \mathbf{LL}^T) = N(0, \boldsymbol{\Sigma}) \tag{3.10}$$

□

Theorem 5. *Any affine transformation of a Gaussian is a Gaussian. In particular*

$$\mathrm{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \implies \mathbf{A}\mathrm{X} + \boldsymbol{b} \sim \mathcal{N}(\mathbf{A}\boldsymbol{\mu} + \boldsymbol{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T)$$

for any $\boldsymbol{\mu} \in \mathbf{R}^n, \boldsymbol{\Sigma} \in \mathbf{R}^{n \times n}$ *positive semi-definite, and any* $\mathbf{A} \in \mathbf{R}^{m \times n}, \boldsymbol{b} \in \mathbf{R}^m$. *We call this the* **affine property** *of a Gaussian.*

Proof. Follows from the linearity of expectation together with Equation 3.9. □

Since samples from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ can be generated independently, using the affine property we can generate samples from an arbitrary multivariate Gaussian. All that is required is a procedure for cholesky decomposition, and a way of generating independent samples from a univariate gaussian, which can be achieved using the Box-Muller transform [Box, 1958].

3.2 Geometric Properties

If $\boldsymbol{\Sigma}$ is positive-definite, then $\mathbf{Y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ implies $\mathbf{A}^{-1}(\mathbf{Y} - \boldsymbol{\mu}) \sim \boldsymbol{\mathcal{N}}(0, \mathbf{I})$ where $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^T$. The random variable $\mathbf{A}^{-1}(\mathbf{Y} - \boldsymbol{\mu})$ has a spherical shape in n-dimensional space.

Looking further at the density formula for a multivariate Gaussian (Equation 3.1) the term $(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$ is called the Mahalanobis distance between $\boldsymbol{x}$ and $\boldsymbol{\mu}$. If we consider $\boldsymbol{\mu}$ a constant, we can also view it as a quadratic form in x. When $\boldsymbol{\Sigma}$ is an identity matrix, the Mahalanobis distance reduces to Euclidean distance. In general, it can be thought of as a distance on a hyper-ellipsoid. Let us now derive some intuition for this.

Since $\boldsymbol{\Sigma}$ is a covariance matrix, we know it is positive definite, and we can perform its eigendecomposition to get $\boldsymbol{\Sigma} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$, where $\mathbf{U}$ is an orthogonal matrix of eigenvectors, and $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues. Basic matrix algebra gives us

$$\boldsymbol{\Sigma}^{-1} = (\mathbf{U}^T)^{-1}\boldsymbol{\Lambda}^{-1}\mathbf{U}^{-1} = \mathbf{U}\boldsymbol{\Lambda}^{-1}\mathbf{U}^T = \sum_{i=1}^{D} \frac{1}{\lambda_i}\boldsymbol{u}_i\boldsymbol{u}_i^T,$$

where the second to last equality comes from $\mathbf{U}$ being orthogonal ($\mathbf{U}^{-1} = \mathbf{U}^T$). Substituting this in the Mahalanobis distance we get

$$(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}) = (\boldsymbol{x} - \boldsymbol{\mu})^T \left(\sum_{i=1}^{D} \frac{1}{\lambda_i}\boldsymbol{u}_i\boldsymbol{u}_i^T \right) (\boldsymbol{x} - \boldsymbol{\mu}) \tag{3.11}$$

$$= \sum_{i=1}^{D} (\boldsymbol{x} - \boldsymbol{\mu})^T \frac{1}{\lambda_i}\boldsymbol{u}_i\boldsymbol{u}_i^T(\boldsymbol{x} - \boldsymbol{\mu}) \tag{3.12}$$

$$= \sum_{i=1}^{D} \frac{y_i^2}{\lambda_i} \tag{3.13}$$

where $y_i = u_i^T(\boldsymbol{x} - \boldsymbol{\mu})$ which has exactly the same form as a D dimensional ellipse. From this we conclude that the contour lines of a multivariate Guassian will be elliptical, where the eigenvectors determine the orientation of the ellipse, and the eigenvalues determine the length of the principal axes [Bishop, 2016].

3.3 Conditional and Marginal Gaussian Distribution

In this section we derive the conditional $p(x_1|x_2)$ and marginal $p(x_1)$ for a given joint distribution $p(x_1, x_2)$. One of the interesting properties of a multivariate Gaussian is that both the conditional and the marginal are also Gaussian, and we can easily compute their parameters in closed from based on the parameters of the joint distribution.

Before we derive the conditional and marginal distributions, let us state the partitioned inverse formula without proof.

Theorem 6 ([Murphy and Bach, 2012])**.** *Consider a partitioned matrix*

$$\mathbf{M} = \begin{bmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{bmatrix} \tag{3.14}$$

where we assume $\mathbf{E}$ *and* $\mathbf{H}$ *are invertible. We have*

$$\mathbf{M}^{-1} = \begin{bmatrix} (\mathbf{M}/\mathbf{H})^{-1} & -(\mathbf{M}/\mathbf{H})^{-1}\mathbf{F}\mathbf{H}^{-1} \\ -\mathbf{H}^{-1}\mathbf{G}(\mathbf{M}\mathbf{H})^{-1} & \mathbf{H}^{-1} + \mathbf{H}^{-1}\mathbf{G}(\mathbf{M}/\mathbf{H})^{-1}\mathbf{F}\mathbf{H}^{-1} \end{bmatrix} \tag{3.15}$$

$$= \begin{bmatrix} \mathbf{E}^{-1} + \mathbf{E}^{-1}\mathbf{F}(\mathbf{M}/\mathbf{E})^{-1}\mathbf{G}\mathbf{E}^{-1} & -\mathbf{E}^{-1}\mathbf{F}(\mathbf{M}/\mathbf{E})^{-1} \\ -(\mathbf{M}/\mathbf{E})^{-1}\mathbf{G}\mathbf{E}^{-1} & (\mathbf{M}/\mathbf{E})^{-1} \end{bmatrix} \tag{3.16}$$

where

$$\mathbf{M}/\mathbf{H} = \mathbf{E} - \mathbf{F}\mathbf{H}^{-1}\mathbf{G} \tag{3.17}$$

$$\mathbf{M}/\mathbf{E} = \mathbf{H} - \mathbf{G}\mathbf{E}^{-1}\mathbf{F} \tag{3.18}$$

is called the **Schur complement**.

Proof. Since the proof is rather technical and only consists of applying the LDU decomposition and many algebraic manipulations, we leave it out and refer the reader to Murphy and Bach [2012] for details. □

3.3.1 Conditional Distribution is a Gaussian

Suppose $\boldsymbol{x}$ is a D-dimensional random vector with a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and that $\boldsymbol{x}$ is partitioned into two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ such that

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} \tag{3.19}$$

We also partition the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ into a block matrix, and name the inverse of the covariance matrix $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$, which will simplify a few of the equations that follow. We will derive the exact form of $\boldsymbol{\Lambda}$ and of its individual blocks later in this section. For now we simply use the fact that $\boldsymbol{\Sigma}$ is positive-definite, and thus it is invertible. The matrix $\boldsymbol{\Lambda}$ is also known as a **precision matrix**.

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}, \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{bmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{bmatrix} \tag{3.20}$$

Note that since $\mathbf{\Sigma}$ is a symmetric matrix, $\mathbf{\Sigma}_{12}{}^T = \mathbf{\Sigma}_{21}$, and similarly $\mathbf{\Lambda}_{12}{}^T = \mathbf{\Lambda}_{21}$. Similarly, $\mathbf{\Sigma}_{11}$, $\mathbf{\Sigma}_{22}$, $\mathbf{\Lambda}_{11}$, and $\mathbf{\Lambda}_{22}$ are all symmetrical.

Before we derive the parameters of the conditional, we show that the conditional distribution $p(x_1|x_2)$ is a Gaussian. To do this, we take the joint distribution $p(x_1,x_2)$ and fix the value of x_2 [Bishop, 2016]. Using the definition of conditional probability $p(x_1,x_2) = p(x_1|x_2)p(x_2)$ we can see that after fixing the value of x_2, $p(x_2)$ is simply a normalization constant, and the remaining term $p(x_1|x_2)$ is a function of x_1 which together with the normalization constant gives us the conditional probability distribution on x_1. We now use the partitioned form of the multivariate Gaussian defined by equation 3.20 to show that $p(x_1|x_2)$ is actually a Gaussian.

Let us begin by looking at the exponent in equation 3.1:

$$-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\mathbf{\Lambda}(\boldsymbol{x}-\boldsymbol{\mu}) = -\frac{1}{2}\left(\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{bmatrix} - \begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix}\right)^T \begin{bmatrix}\mathbf{\Lambda}_{11} & \mathbf{\Lambda}_{12}\\ \mathbf{\Lambda}_{21} & \mathbf{\Lambda}_{22}\end{bmatrix} \left(\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{bmatrix} - \begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix}\right) \tag{3.21}$$

$$= -\frac{1}{2}\begin{bmatrix}\boldsymbol{x}_1-\boldsymbol{\mu}_1\\ \boldsymbol{x}_2-\boldsymbol{\mu}_2\end{bmatrix}^T \begin{bmatrix}\mathbf{\Lambda}_{11} & \mathbf{\Lambda}_{12}\\ \mathbf{\Lambda}_{21} & \mathbf{\Lambda}_{22}\end{bmatrix} \begin{bmatrix}\boldsymbol{x}_1-\boldsymbol{\mu}_1\\ \boldsymbol{x}_2-\boldsymbol{\mu}_2\end{bmatrix} \tag{3.22}$$

To make the next few equations easier to follow we set $\boldsymbol{y}_1 = \boldsymbol{x}_1 - \boldsymbol{\mu}_1$ and $\boldsymbol{y}_2 = \boldsymbol{x}_2 - \boldsymbol{\mu}_2$.

$$-\frac{1}{2}\begin{bmatrix}\boldsymbol{y}_1\\ \boldsymbol{y}_2\end{bmatrix}^T \begin{bmatrix}\mathbf{\Lambda}_{11} & \mathbf{\Lambda}_{12}\\ \mathbf{\Lambda}_{21} & \mathbf{\Lambda}_{22}\end{bmatrix} \begin{bmatrix}\boldsymbol{y}_1\\ \boldsymbol{y}_2\end{bmatrix} = \tag{3.23}$$

$$= -\frac{1}{2}\begin{bmatrix}\boldsymbol{y}_1\mathbf{\Lambda}_{11}+\boldsymbol{y}_2\mathbf{\Lambda}_{21}\\ \boldsymbol{y}_1\mathbf{\Lambda}_{12}+\boldsymbol{y}_2\mathbf{\Lambda}_{22}\end{bmatrix}^T \begin{bmatrix}\boldsymbol{y}_1\\ \boldsymbol{y}_2\end{bmatrix} \tag{3.24}$$

$$= -\frac{1}{2}\left(\boldsymbol{y}_1^T\mathbf{\Lambda}_{11}\boldsymbol{y}_1 + \boldsymbol{y}_2^T\mathbf{\Lambda}_{21}\boldsymbol{y}_1 + \boldsymbol{y}_1^T\mathbf{\Lambda}_{12}\boldsymbol{y}_2 + \boldsymbol{y}_2^T\mathbf{\Lambda}_{22}\boldsymbol{y}_2\right) \tag{3.25}$$

$$\begin{aligned} = &-\frac{1}{2}(\boldsymbol{x}_1-\boldsymbol{\mu}_1)^T\mathbf{\Lambda}_{11}(\boldsymbol{x}_1-\boldsymbol{\mu}_1)+\\ &-\frac{1}{2}(\boldsymbol{x}_2-\boldsymbol{\mu}_2)^T\mathbf{\Lambda}_{21}(\boldsymbol{x}_1-\boldsymbol{\mu}_1)+\\ &-\frac{1}{2}(\boldsymbol{x}_1-\boldsymbol{\mu}_1)^T\mathbf{\Lambda}_{12}(\boldsymbol{x}_2-\boldsymbol{\mu}_2)+\\ &-\frac{1}{2}(\boldsymbol{x}_2-\boldsymbol{\mu}_2)^T\mathbf{\Lambda}_{22}(\boldsymbol{x}_2-\boldsymbol{\mu}_2) \end{aligned} \tag{3.26}$$

We see that this is a quadratic form in x_1, and hence the corresponding conditional distribution $p(x_1|x_2)$ will be Gaussian. Because we know $p(x_1,x_2) = p(x_1|x_2)p(x_2)$ and that both $p(x_1,x_2)$ and $p(x_1|x_2)$ are multivariate Gaussians, fixing the value of x_1 means that $p(x_1|x_2)$ as a function of x_2 is just a normalization constant, and $p(x_2)$ must have the same form as $p(x_1,x_2)$ and therefore is also a multivariate Gaussian.

tady ten marginal nevim jiste

TODO: x2 je spatne (marignal)

Because the Gaussian distribution is completely defined by its mean and covariance, we do not need to figure out the value of the normalization constant. We simply have to derive the equations for $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$.

We continue with the proof from Murphy and Bach [2012]. We will make use of the partitioned matrix inverse theorem Equation 3.16.

At this point we know that the joint distribution factors into two multivariate Gaussians, that is

$$p(x_1, x_2) = p(x_1 \,|\, x_2) p(x_2) \tag{3.27}$$
$$= \mathcal{N}(x_1 \,|\, \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}) \mathcal{N}(x_2 \,|\, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}) \tag{3.28}$$

and we only need to infer their parameters. To make the equations more readable, we again define

$$\boldsymbol{y}_1 = \boldsymbol{x}_1 - \boldsymbol{\mu}_1 \tag{3.29}$$
$$\boldsymbol{y}_2 = \boldsymbol{x}_2 - \boldsymbol{\mu}_2. \tag{3.30}$$

We then simply take the block definition of a multivariate Gaussian and multiply everything out

$$\mathrm{E} = \exp\left\{ -\frac{1}{2} \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} \right\} \tag{3.31}$$
$$= \exp\left\{ -\frac{1}{2} \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix}^T \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} & \mathbf{I} \end{bmatrix} \begin{bmatrix} (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{22}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} \right\} \tag{3.32}$$
$$= \exp\left\{ -\frac{1}{2} \begin{bmatrix} \boldsymbol{y}_1^T - \boldsymbol{y}_2^T(\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}) \\ \boldsymbol{y}_2 \end{bmatrix}^T \begin{bmatrix} (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{22}^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{y}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{y}_2) \\ \boldsymbol{y}_2 \end{bmatrix} \right\} \tag{3.33}$$
$$= \exp\left\{ -\frac{1}{2} \begin{bmatrix} (\boldsymbol{y}_1^T - \boldsymbol{y}_2^T\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} \\ \boldsymbol{y}_2^T\boldsymbol{\Sigma}_{22}^{-1} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{y}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{y}_2) \\ \boldsymbol{y}_2 \end{bmatrix} \right\} \tag{3.34}$$
$$= \exp\left\{ -\frac{1}{2}(\boldsymbol{y}_1^T - \boldsymbol{y}_2^T\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1}(\boldsymbol{y}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{y}_2) \right\} \times \tag{3.35}$$
$$\times \exp\left\{ -\frac{1}{2}\boldsymbol{y}_2^T\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{y}_2 \right\}$$

We can immediately see that the second term is a quadratic form in $\boldsymbol{x}_2$ and corresponds to $\mathcal{N}(\boldsymbol{x}_2 \,|\, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22})$. Let us now consider the first term in isolation and move the terms around a little bit. We also make use of the fact that because $\boldsymbol{\Sigma}_{22}$ is a positive-definite matrix, its inverse is also symmetric, so $\boldsymbol{\Sigma}_{22}^{-1^T} = \boldsymbol{\Sigma}_{22}^{-1}$. We also know that $\boldsymbol{\Sigma}_{12}^T = \boldsymbol{\Sigma}_{21}$.

$$E_{1|2} = \exp\left\{ -\frac{1}{2}(\boldsymbol{y}_1^T - \boldsymbol{y}_2^T\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1}(\boldsymbol{y}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{y}_2) \right\} \tag{3.36}$$
$$= \exp\left\{ -\frac{1}{2}(\boldsymbol{y}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{y}_2)^T(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1}(\boldsymbol{y}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{y}_2) \right\} \tag{3.37}$$
$$= \exp\{ -\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)^T(\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} \tag{3.38}$$
$$(\boldsymbol{x}_1 - \boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2))\} \tag{3.39}$$

In equation 3.39 we again see a Gaussian density with parameters

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \tag{3.40}$$

$$\boldsymbol{\Sigma}_{1|2} = (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{22})^{-1} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}{\boldsymbol{\Sigma}_{22}}^{-1}\boldsymbol{\Sigma}_{21}. \tag{3.41}$$

This formula extremely important for the use of GPs as a probabilistic model in bayesian optimization. It will allow us to compute the exact parameters of the posterior $p(f|x)$ at any given point, and as a result compute the acquisition function.

3.4 Gaussian Processes

Gaussian Process is a stochastic process (a collection of random variables), such that every subset of those random variables has a multivariate Gaussian distribution. It is defined by a mean function $m(\boldsymbol{x})$ and a covariance funciton $\kappa(\boldsymbol{x}, \boldsymbol{x}')$. Formally, we write

$$p(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), \kappa(\boldsymbol{x}, \boldsymbol{x}')). \tag{3.42}$$

Any finite subset $\boldsymbol{x} = (x_1, \dots, x_n)$ is jointly Gaussian with mean $m(\boldsymbol{x})$ and covariance $\boldsymbol{\Sigma}(\boldsymbol{x})$ where $\boldsymbol{\Sigma}(\boldsymbol{x})_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where κ is any positive definite kernel function [Murphy and Bach, 2012].

The GP defines a prior distribution over functions f, which when combined with data x can be converted into a posterior distribution over functions $p(f|x)$.

3.4.1 GP regression with noise-free observations

Consider the case when we are interested in predicting a function f based on a few observations $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$, and we are interested in predicting the value of $y_\star$ at a new point $x_\star$.

Using the definition of a GP, we know that y and $y_\star$ are jointly Gaussian. We also know, that these are the only points we are interested in. Even though the GP is a distribution over functions, that is over infintiely dimensional vectors, we only need to look at finitely many points and can ignore the rest. This is a crucial property of the GP and essentially makes everything we are about to do possible.

By assuming a GP, we get all of the properties of a multivariate Gaussian for free, including a closed form solution to the conditional and marginal distribution parameters. Let us now write the joint distribution of $\boldsymbol{y}$ and $\boldsymbol{y}_\star$ in a partitioned form

$$\begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{y}_\star \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_\star \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_\star \\ \mathbf{K}_\star^T & \mathbf{K}_{\star\star} \end{pmatrix} \right)$$

where $\mathbf{K} = \kappa(\mathbf{X}, \mathbf{X})$, $\mathbf{K}_\star = \kappa(\mathbf{X}, \mathbf{X}_\star)$ and $\mathbf{K}_{\star\star} = \kappa(\mathbf{X}_\star, \mathbf{X}_\star)$ [Williams and Rasmussen, 2006]. Note that $\boldsymbol{y}$ and $\boldsymbol{y}_\star$ can be either single points, or they can be vectors, as we might be interested in computing the posterior over multiple points at once given an existing dataset. Because this is just a multivariate Gaussian, we can make use of the conditioning formula and compute the posterior

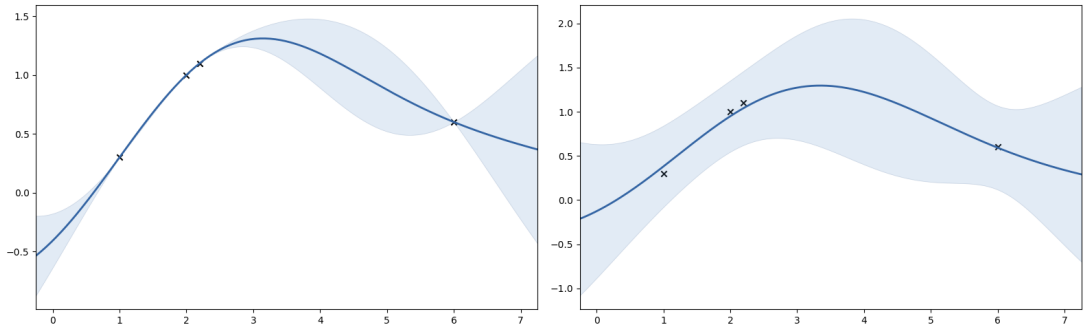


Figure 3.1: GP regression without noise on the left, and with a constant amount of noise added on the right.

$p(\boldsymbol{y}_\star \mid \mathbf{X}_\star, \mathbf{X}, \boldsymbol{y})$ exactly as

$$p(\boldsymbol{y}_\star \mid \mathbf{X}_\star, \mathbf{X}, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{y}_\star \mid \boldsymbol{\mu}_\star, \boldsymbol{\Sigma}_\star) \tag{3.43}$$

$$\boldsymbol{\mu}_\star = \boldsymbol{\mu}(\mathbf{X}_\star) + \mathbf{K}_\star^T \mathbf{K}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}(\mathbf{X})) \tag{3.44}$$

$$\boldsymbol{\Sigma}_\star = \mathbf{K}_{\star\star} - \mathbf{K}_\star^T \mathbf{K}^{-1} \mathbf{K}_\star. \tag{3.45}$$

where $\boldsymbol{\mu}_\star$ is the mean and $\boldsymbol{\Sigma}_\star$ the covariance of the multivariate Gaussian on $\boldsymbol{y}_\star$.

3.4.2 GP regression with noisy observations

Consider the case when f is not a deterministic function, but rather a stochastic function which returns a noisy output y given some fixed input $\boldsymbol{x}$ (meaning $f(\boldsymbol{x})$ is a random variable). GP regression is flexible enough to model Gaussian noise in the output directly. For now, let us consider the noise having a fixed variance σ^2.

In practice, the noise becomes a baseline for the variance of each point of the posterior, as the variance can never be lower than the noise. Since the variance is represented on the diagonal of the covariance matrix computed by the kernel function κ, we can model the noise directly by simply adding a diagonal matrix to the output of the kernel, that is

$$\text{cov}(\boldsymbol{y}) = \kappa(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}.$$

See Figure 3.4.2 for a comparison of noise-less and noisy regression. It should also be noted that in principle nothing is preventing us from specifying a different noise value for each element of the diagonal. This could be useful if we had additional prior information about the function f. It could, for example, be an output of a measurement for which we know exactly the amount of noise for each $\boldsymbol{x}$.

3.4.3 Kernels

So far we have considered the covariance function κ to be an arbitrary positive-definite kernel. Even though in theory there are no restrictions on what kernel we can choose, there are a few more popular choices that are commonly used.

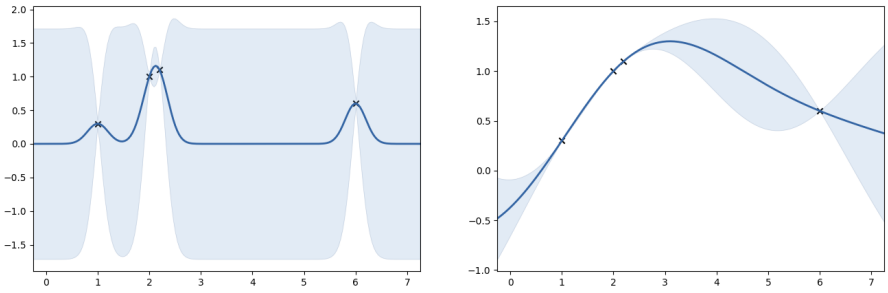


Figure 3.2: Lengthscale $l = 0.2$ on the left, and $l = 2$ on the right.

A prototypical example is the squared exponential (SE) kernel, also called the radial-basis function (RBF) kernel

$$\kappa(x_1, x_2) = \sigma_k^2 \exp\left\{-\frac{1}{2l}(x_1 - x_2)^2\right\}.$$

This kernel, among many others, falls under the category of stationary kernels. A **stationary kernel** is one which is shift-invariant, which means its value does not depend on the absolute values of x_1 and x_2, but only on their distance $d = |x_1 - x_2|$. We can thus write it as

$$\kappa(x_1, x_2) = \sigma_k^2 \exp\left\{-\frac{1}{2l}d^2\right\}.$$

The values σ_k and l are called the variance and lengthscale, and control the behavior of the kernel, where σ_k changes the vertical scale of the function, and l changes the horizontal scale. Changing the lengthscale essentially allows the kernel to re-normalize the data. If $\boldsymbol{x}$ is a vector we can define a lengthscale parameter l_i for each of the components. This becomes very useful in the context of hyperparameters where each hyperparameter can have a very different scale, and the individual lengtscale per component allows the kernel to capture it.

Figure Figure 3.4.3 show how the behavior of the kernel changes based on the changed value of the lengthscale l. When the lengthscale is set too low the values of y become essentially uncorrelated, leading to a function with many spikes. On the other hand, a larger value for the lengthscale yields a much smoother function.

3.4.4 Optimizing GP hyperparameters

So far we have considered the noise, variance and lengthscale parameters to be fixed, but in this section we show how their value can be determined automatically form the data using maximum likelihood estimation.

Since the GP is a probabilistic model, we can ask it directly what is the likelihood of our data. Using the definition of a GP, we know that the likelihood of our data is a multivariate Gaussian, that is $p(\boldsymbol{y}|\mathbf{X}) = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \mathbf{K})$, giving us a log likelihood of

$$\log p(\boldsymbol{y}|\mathbf{X}) = -\frac{1}{2}\boldsymbol{y}\mathbf{K}^{-1}\boldsymbol{y} - \frac{1}{2}\log\det\mathbf{K} - \frac{N}{2}\log(2\pi).$$

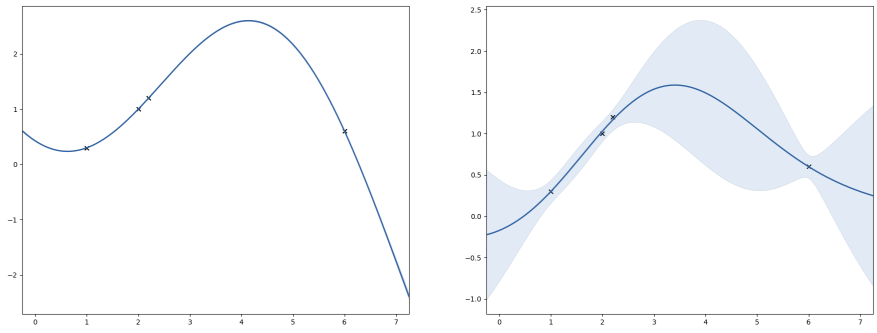


Figure 3.3: GP regression without optimizing kernel parameters on the left, and after optimizing using maximum likelihood on the right. Notice that the noise parameter is also optimized, as the regression model does not go directly through the data points, but instead considers the small variation as due to noise.

We leave out the technical details (see Williams and Rasmussen [2006] for more details) including the gradient of the marginal likelihood with respect to the kernel parameters, as they do not provide any useful insights.

One important detail we want to stress out is that computing K^{-1} takes $O(N^3)$, which puts a serious restriction on the size of the data we can fit with exact GP regression. This does not concern us in the context of hyperparamter optimization as we would always stay within low hundreds of evaluations anyway, but for other tasks it becomes a serious limitation. As a result many workarounds for approximate inference were developed [Williams and Rasmussen, 2006], which again, we omit from this text, because they are not relevant for hyperparameter optimization.

Implementing the kernel parameter optimization is easy in practice. One can simply implement the marginal likelihood formula shown in subsection 3.4.4 and use a package for automatic differentiation with an optimizer like SGD or L-BFGS to optimize the parameters. Since the objective is non-linear, a common practice [GPy, since 2012] is to optimize with mutliple restarts. Figure 3.4.4 shows the effect of optimizing kernel parameters using maximum likelihood.

neni oz-naceno jako rovnice

4. Bayesian Optimization in depth

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

4.1 Acquisition functions

An intuitive choice for an acquisition function is to maximize the probability of improving over our currently best achieved value, which is called the **probability of improvement**. This can be computed in closed form as

$$PI(\boldsymbol{x}) = \Phi\left(\frac{\mu(\boldsymbol{x}) - y_{\max}}{\sigma(\boldsymbol{x})}\right)$$

where $y_{\max}$ is the maximum value achieved by sampling $f(\boldsymbol{x})$.

A natural extension is the **expected improvement** (EI) acquisition function which is simply the expected improvement over the currently achieved maximum. We define it as

$$EI(x) = \mathbb{E}[f(x) - y_{\max}].$$

At first it might seem that the expectation would be an intractable integral, but fortunately even this equation can be computed in closed form as

$$EI(x) = \Delta(x) + \sigma(x)\varphi\left(\frac{\Delta(x)}{\sigma(x)}\right) - |\Delta(x)|\Phi\left(\frac{\Delta(x)}{\sigma(x)}\right)$$

where $\Delta(x) = \mu(x) - y_{\max}$. In practice, improvement shows better results than probability of improvement. For more examples of acquisition functions see Frazier [2018].

In both of these cases, the next sampling point would be chosen by maximizing the acquisition function, that is

$$x_{\text{next}} = \arg\max_{x} EI(x)$$

for the case of expected improvement. This can again be done by any stochastic optimizer, such as the commonly chosen L-BFGS with restarts.

4.2 Parallel evaluations

In practice we might have the ability to evaluate $f(\boldsymbol{x})$ at multiple points in parallel, but the framework we have shown so far only allows for sequential optimization. In the previous section we've shown a few examples of the acquisition functions. A natural extension would be to not optimize with respect to a single x_{next}, but rather multiple points. In the context of EI this is called **parallel expected improvemenet**.

Unfortunately, there is no simple solution [Frazier, 2018]. A common solution is the so called **Constant Liar** approximation, which chooses x_{i+1} assuming x_i was already chosen, and has the corresponding value y_i equal to a constant, often chosen to be the expected value of $f(x_i)$ under the GP posterior.

This allows us to trivially implement parallel evaluations by simply considering the μ prediction for unfinished evaluations as their y value and consider them part of the dataset $\mathcal{D}$.

4.3 Integer parameters

GP regression by itself does not have the ability to model integer values in X directly as some other models do (e.g. random forests chapter 2). A common solution, used by [Group, 2014] and which we also implement in this thesis, is to consider all parameters to be real valued and only round them at the end.

In recently published work by Garrido-Merchán and Hernández-Lobato [2017] they show a more principled approach. The effect of rounding causes the model to see variation and relationships even among constant-valued regions. A possible downside is that the model could predict values different enough so that the acquisition function would obtain a maximum within a constant region which already has an existing sample, and thus wasting an evaluation. A proposed solution to this problem, as mentioned in the paper, is to round the appropriate values right before they are input into the kernel function, such as

$$\kappa'(x_1, x_2) = \kappa(T(x_1), T(x_2))$$

where $T(x)$ is an identity for real valued elements and a rounding function for integers.

Our implementation however does not use this approach, as our GP regression is handled by the GPy [since 2012] library, which did not support it at the time, and implementing it would mean overriding many of the existing kernels. We did instead handle the problem explicitly by detecting the pathological cases, as described in chapter 5.

4.4 bopt algorithm

matematika

4.5 Logscale

When optimizing hyperparameters we might want to distinguish not only between real and integer valued ones, but also based on their scale. Optimizing the number of training epochs or layers are very well modelled by a linear scale, but a learning rate is better modelled with a logarithmic scale.

We provide a simple solution, which can work independently of Bayesian optimization, by simply transforming all of the appropriate value to logscale before inputting them into the model, and then transforming them back after we get a next sample x proposal.

5. Software

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

This chapter describes the implementation part of this thesis. While we do not provide any theoretical extensions to Bayesian optimization, we instead provide a modular and fully working implementation which was tested on multiple experiments. The implementation is provided as a Python (van Rossum [1995]) package called `bopt`(short for Bayesian optimization).

The main features of the package are:

- Flexible experiment configuration with random search and GP backends.
- Parallel execution of evaluations, both on the local machine and on a cluster.
- Robust error handling with duplicate/similar sample detection.
- Command line interface for controlling the experiment evaluations.
- Simple filesystem based storage for manual user intervention.
- Web based visualizations of the whole optimization process, including 1D and 2D slices and marginal plots at all points during the evaluation.

5.1 Architecture

In this section we will explore the high level architecture of `bopt`. Everything is structured around a central class `Experiment`, which represents a single objective function together with a configuration of its hyperparameters, and configuration for the Bayesian optimization itself. The `Experiment` can contain multiple `Samples`, where each sample represents a single evaluation of the objective function.

We assume the function being optimized can be evaluated by running a script file. The hyperparameters will be passed as command line arguments, and the output will be parsed from the standard output using a regular expression provided by the user. This provides the user with maximum flexibility with regards how the actual function is being executed, as `bopt`will simply spawn the process, pass the command line arguments, and then wait for it to terminate to collect the output and parse the result. If the result is not found in the output, or the process exists with an exit code greater than 0 it will mark the evaluation as failed.

5.1.1 Samples

5.1.2 Runners

5.2 GPy

5.3 Random Search

5.4 Visualizations

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

- experimenty
- 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

OpenAI five. https://openai.com/five/. Accessed: 2019-04-22.

The GPyOpt authors. GPyOpt: A bayesian optimization framework in python. http://github.com/SheffieldML/GPyOpt, 2016.

C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer New York, 2016. ISBN 9781493938438. URL https://books.google.cz/books?id=kOXDtAEACAAJ.

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

George EP Box. A note on the generation of random normal deviates. *Ann. Math. Stat.*, 29:610–611, 1958.

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.

Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. Dealing with integer-valued variables in bayesian optimization with gaussian processes. *arXiv preprint arXiv:1706.03673*, 2017.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Elliot Karro, and D. Sculley, editors. *Google Vizier: A Service for Black-Box Optimization*, 2017. URL http://www.kdd.org/kdd2017/papers/view/google-vizier-a-service-for-black-box-optimization.

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

GPy. GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy, since 2012.

Harvard Intelligent Probabilistic Systems Group. Spearmint. https://github.com/HIPS/Spearmint, 2014.

Tim Head, MechCoder, Gilles Louppe, Iaroslav Shcherbatyi, fcharras, Zé Vinícius, cmmalone, Christopher Schröder, nel215, Nuno Campos, Todd Young, Stefano Cereda, Thomas Fan, rene rex, Kejia (KJ) Shi, Justus Schwabedal, carlosdanielcsantos, Hvass-Labs, Mikhail Pak, SoManyUsernamesTaken, Fred Callaway, Loïc Estève, Lilian Besson, Mehdi Cherti, Karlson Pfannschmidt, Fabian Linzberger, Christophe Cauet, Anna Gut, Andreas Mueller, and Alexander Fabisch. scikit-optimize/scikit-optimize: v0.5.2, March 2018. URL https://doi.org/10.5281/zenodo.1207017.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. URL http://arxiv.org/abs/1812.04948.

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.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

G. van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT Press Cambridge, MA, 2006.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. URL http://arxiv.org/abs/1707.07012.

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 First Attachment