Bayesian Optimization Review - JnJ

Ted Ladas

July 2022

1 Introduction

In this document, I will present a brief introduction to Bayesian optimization, as well as some additional challenges that we face when applying the algorithm to chemical synthesis. The reason for creating this document is to present how the literature is approaching implementations of that algorithm as alongside the Shields et al. paper [7], there are other more generally applicable ideas that could potentially be tried out (or even combined).

Bayesian optimization is a sequential optimization strategy for an *objective* (black box) function $f(x)$, $x \in X$. We call $f(x)$ a black box because (1) evaluations of the function at input x are expensive, (2) we have no access to the gradients of the function, and (3) observations are further corrupted by noise. The algorithm can be broken down into the following steps.

- Initial sample of the parameter space through some space-filling DOE
- Evaluate the objective function
- Train a probabilistic *surrogate* model of the objective function
- Optimize an *acquisition* function $\alpha(x)$
- Goto step 2

Hence we understand that this algorithm has two main components.

1. What is our **Model** - Gaussian process, Bayesian Neural Network, Variational Autoencoder, Random Forest.
2. What is our **Exploration Strategy** given that model - Acquisition function that maps the model to the next query point.

The role of the acquisition function is to balance the exploration (choosing points far away from the current optimum to avoid local optima) and exploitation (choosing points very close to the current optimum to find a better solution). A

common acquisition function would be: $\alpha(x) = f(x) - k \cdot sd(f(x))$. The hyperparameter k is what controls this exploration vs exploitation balance. In general, the acquisition function tries to solve the problem:

$$x_{next} = query(M, D) = argmax_{x \in X} a(x; M, D)$$

Where M is the model and D is the know data points.
Popular acquisition functions are:

- Probability of Improvement (PI)
- Expected Improvement (EI)
- Upper (Lower) Confidence Bound (UCB)
- Thomson Sampler

2 Approaches to reaction optimization

In the field of chemical synthesis, tools like High Throughput Experimentation and Design of Experiments, are used to search the parameter space of each reaction. These methods are very efficient since they can be parallelized and fairly easy to automate. However, they also have substantial downsides. First of all, HTE essentially performs an exhaustive search, and therefore it doesn't scale well with additional parameters/levels of parameters. On the other hand, DOE is limited by the fact that each design is created in batches and therefore we don't gain any information about the optimum when running each experiment.

Orthogonal to that, we could run each experiment sequentially, and formulate an optimization problem. Bayesian optimization is such a strategy. The huge positive is that we can potentially identify the optimum with fewer experiments, albeit not necessarily less time. The last step would be to combine those techniques and create a hybrid approach that performs Bayesian Batch Optimization.

Bayesian optimization is extensively studied in the Machine Learning literature, because of its automated nature. It is mainly used in hyperparameter tuning and especially with a Gaussian process as its surrogate model. This makes implementation fairly straightforward as there are many APIs in popular languages that implement the algorithm, such as GPyOpt, bayesian-optimization, and others. However, there is an important problem when implementing these algorithms to solve chemical synthesis problems. Bayesian optimization takes **continuous** features for inputs. This is a big issue as there are very few continuous features in reaction optimization (such as concentrations, temperatures, etc.) and many categorical ones (such as bases, ligands, solvents, etc). Even more importantly, those categorical features have many possible levels, so one-hot encoding of those features doesn't work as it makes our problem a $d > n$ one.

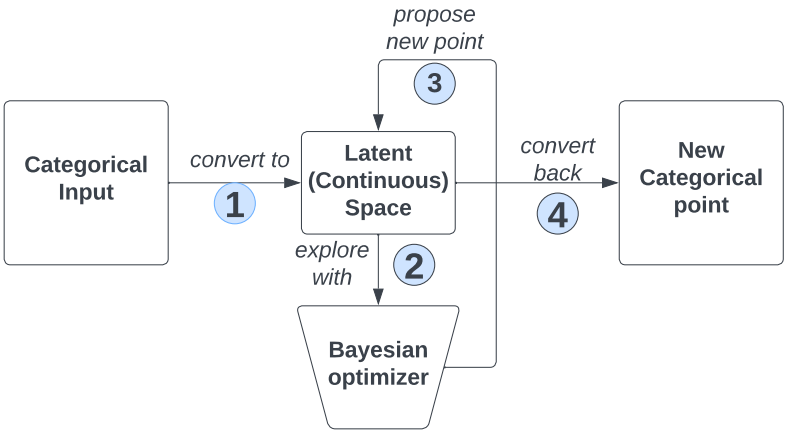

Figure 1: This diagram showcases a simple conversion from categorical to continuous space, as well as where the Bayesian Optimization algorithm fits in that picture.

A preprocessing step of translating the categorical features to a continuous space is needed to use Bayesian optimization in that latent space. A graphic representation of this process is shown in Figure 1. Multiple approaches in the literature perform this conversion and in the next paragraphs, the most promising ones will be briefly presented. After that, I will discuss what are the next steps for the 2 months remaining of that project.

2.1 Descriptors

According to the physical properties of each ligand, we can construct some features in a continuous space that describe each specific ligand [1]. This is a mechanistic way of creating a continuous space for the molecules that could potentially be used in an algorithm to optimize yield. This is the approach of the Shields et al. paper [7]. They claim that they also used other techniques such as one-hot-encoding, but the best results were established by chemical descriptors. However, this approach falls short in some crucial ways.

1. Each ligand can have many descriptors (hundreds) so if we were to include all of them in our datasets, we would again have a $d > n$ problem.

2. Each ligand might have different descriptors that are relevant, making the selection process even harder

3. If we include k (where k *small*) descriptors there is no guarantee that there will be a one-to-one relationship with the existing ligands. In other words,

if we choose to include a small number of descriptors for each ligand, there could be the case that two different ligands have the same descriptors.

4. Most importantly, the reverse operation, from the continuous space back to the ligand is not easy. A ligand might be translated to a specific vector of k ligands, and when we perturb one of those values, we might get no vector that matches the specific vector. The Continuous space has a lot of black spots where no ligand can be translated to these points.

These negatives make the use of descriptors as a continuous representation of the categorical variables difficult to justify. However, it is still a valuable technique, as it can be used to leverage existing domain knowledge as well as for computing distances between ligands if used correctly. For those reasons, we aim to keep them in the loop of the algorithm in some form.

2.2 Variational Auto Encoders

A data-driven way of creating a lower dimensional continuous representation of these molecules is shown in the Bombarelli et al. paper [3]. This approach combines the existing literature in Variational Auto Encoders (VAE) by using the letter sequence of the ligands as its input.

An Autoencoder is a specific architecture of a Neural Network used for unsupervised tasks. In a simplified view, it can be thought of as a function $f : R^n \rightarrow R^n$ that approximately returns the input.

$$f(x) \approx x$$

Without any constraints, this is trivial as we can write:
$f(x) = Wx, \;\; W = I$
The constraint that is introduced in autoencoder architecture is the "bottleneck". For example, a (narrow) one-hidden layer autoencoder could be represented as:

$$\text{h=g}^{(1)}(W^{(1)}x + b^{(1)})$$
$$\text{f=g}^{(2)}(W^{(2)}h + b^{(2)})$$

Where $W^{(1)} : K \times D, \;\; with \;\; K << D$ and $g^{(1)}, g^{(2)}$ non-linear, differentiable functions. This architecture is also shown in a toy diagram in Figure 2.

The key to what makes autoencoders useful is the fact that if this network manages to (almost) reconstruct the input, then we can use the lower representation of that input without suffering much loss. It is a good lossy compression as the bottlenecked layer h contains in fewer dimensions "most" of the information of the input layer. The Bombarelli et al. paper also pushes this technique one step further by using a Variational Autoencoder. This is a similar architecture to the autoencoder, but the latent representation that it learns is not a vector, but the mean and standard deviation of a Normal distribution.

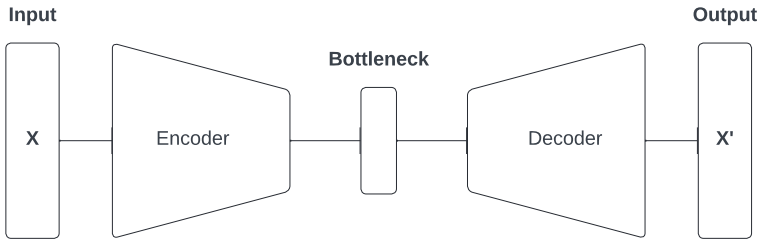

Figure 2: This is a pictorial representation of how the layers of an autoencoder look. The important part is the bottleneck layer and the output layer is the same as the input layer (or at least close to it); an unsupervised learning task.

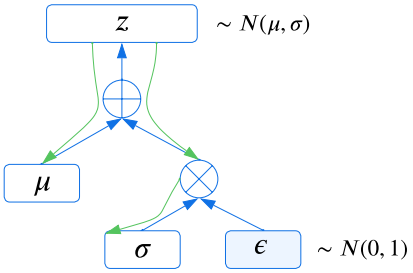

Figure 3: This is a picture of the reparameterization trick. The μ and σ are the nodes that we are interested in learning, so we can combine them with the stochastic node ε and create the node z. This node is used to sample a new point from the latent space. The green backward arrow represents that we can push gradients backward through our deterministic nodes, but not our stochastic node (ε).

In the Variational autoencoder architecture, the aim is the same as before, except now we are trying to learn a latent variable $z = \mu + \sigma \cdot \varepsilon$. By defining $\varepsilon \sim N(0,1)$, we then know that $z \sim N(\mu, \sigma^2)$. This is called the reparameterization trick and it is used to be able to "push" gradients backward in the neural network training phase, as stochastic nodes don't have gradients. A graphic representation of the reparameterization trick is in Figure 3. The reason VAE, is preferred over regular AE, is that they learn a more continuous representation of the latent space, hence leaving fewer dark spots (spots where a point in the latent space doesn't correspond to any point in the original input space).

In conclusion, this technique to solve the conversion problem is shown to work, but it would require building the encoder and decoder using ligand SMILES, which is not immediately obvious from the paper, how to do it, or how to fine-tune the algorithm for better performance. In addition to that, the assumption that the latent

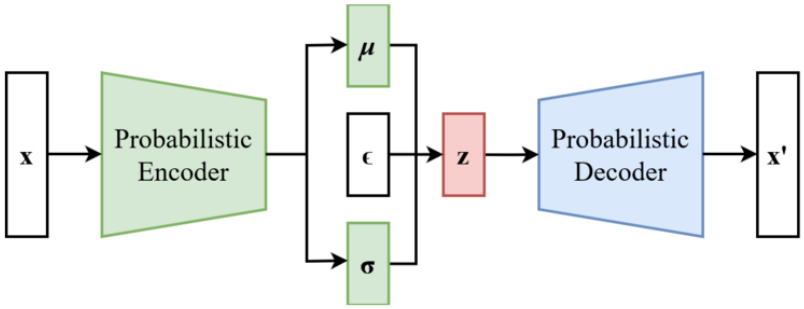

Figure 4: This is a picture of how a Variational Auto Encoder looks. The nodes μ and σ represent the mean and standard deviation of the Normal distribution, and they are deterministic nodes. On the other hand, $\varepsilon \sim N(0,1)$, represents a stochastic node.

variable z follows a Normal distribution is a heavy assumption that is not easy to justify other than that "it just works". The idea of reparameterization however is important and it is used in the next approach.

2.3 Gryffin Algorithm

Gryffin is an algorithm recently published by Florian et al. [4]. It is an extension to the Phoenics [5] algorithm that also accepts categorical variables as its input. The basic idea of that algorithm is to convert the categorical features into their one-hot-encoded version. By doing so, it essentially treats each level of the ligand as a corner of a Simplex, on which the algorithm will try to optimize the response.

$$\zeta \in \Delta = \{\zeta \in R^n|_{\zeta_i \in \{0,1\}} \; and \; \sum_{i=1}^{n} \zeta_i = 1\}$$

This is a bit too restrictive, so we can go a step further and construct a *soft* one-hot-encoded version of the categorical features. That means that instead of optimizing on the Simplex, we can optimize in the Simplex.

$$\zeta \in \Delta = \{\zeta \in R^n|_{\zeta_i \in [0,1]} \; and \; \sum_{i=1}^{n} \zeta_i = 1\}$$

Because this mathematical "trick" doesn't make sense in the real world (there is no ligand that is using 80% ligand A and 20% ligand B), the algorithm uses this continuous space to do the optimization, but at the last step, it **projects** the parameter to the closest hard one-hot-encoded vector (e.g. $p = (0.4, 0.6) \rightarrow p = (0,1)$). We can then define a tractable distribution on the Simplex and use it to construct the surrogate model. This distribution is the Gumbel-Softmax distribution.

The Gumbel-Softmax distribution is an additional step on the Gumbel-Max distribution [6]. This distribution is important as it can be smoothly annealed, and hence used to push gradients backward.

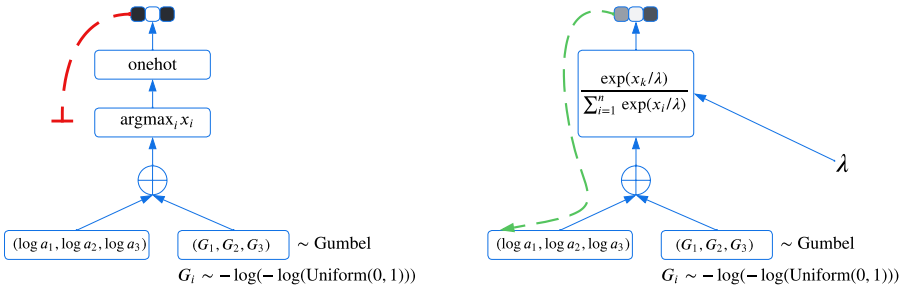

Figure 5: (a) Gumbel-Max distribution, (b) Gumbel-Softmax distribution

The simplest implementation of Gryffin, naïve Gryffin, uses the Gumbel-Softmax distribution to estimate kernel densities of the distribution of the categorical parameters (one-hot encoded). This is being done using a Bayesian Neural Network to estimate the parameters of the Kernels based on data. BNNs are suitable for this task because they can perform quite well in smaller datasets. In addition, since the BNN is used to estimate parameters for the kernels, the estimation of the surrogate model is cost-efficient as it doesn't require a forward pass through the BNN itself.

Gryffin can then be further complicated, if we drop the naïve assumption that all ligands are exactly different. This assumption is not true in cases where a notion of similarity (distance) can be established between two levels of a categorical variable. This is how descriptors can be used to help the algorithm define a better covariance matrix between the input variables and hence perform a better search. Hence this version is a descriptor-guided search with static Gryffin.

In the last step of their paper, they showcase dynamic Gryffin, which in addition to using descriptors like static Gryffin to define the distance between two levels of a categorical variable (e.g. ligands), also learns which descriptors are the most informative as more information is gathered through each experiment.

This is a very promising algorithm and it should be fairly forward to implement, as it is publicly available through their Python API (here). In addition to that, it is also a fairly straightforward implementation using keras and tensorflow-probability, as it is just a Bayesian Neural Network, a Bayesian optimizer, and at the last step, a projection to the closest point of the Simplex.

3 Conclusion and Next Steps

Bayesian optimization is a **straightforward** algorithm, that is shown from various sources that it works in the chemical synthesis setting. The **difficult** task is to construct a reversible operation that translates the categorical to a continuous space. The Bayesian optimization is going to operate on that space to find the next best candidate and for that, the continuous space needs to have as less dead zones as possible.

In the following list, the most promising ideas are presented in order of increasing complexity. This list contains some of my ideas (dimensionality reduction on one-hot-encoded vectors) alongside the academic consensus.

1. Descriptors
2. One-Hot Encoding
3. Naïve Gryffin's Simplex Optimization
4. Descriptor-Guided Gryffin
5. Variational Auto Encoders
6. Sequence-to-Sequence VAE (Seq2Seq)

The aim for the next two months is to implement some of these techniques to the same dataset and get results as to which ones are more promising. A very useful tool for performing such comparisons is SUMMIT [2]. This is another Python library that essentially standardizes the comparison processes and acts like a virtual laboratory. All algorithms are operating in the same dataset, that will be user-defined for the same number of virtual experiments, with the same number of random starts, etc, etc. The only thing that is not straightforward to me is how easy it is to add new algorithms into the mix (for example the one-hot encoded MDS). However, the packages already have the most important models already built in.

After we establish which are the potentially good approaches, the aim is to test the same algorithms in many different scenarios, for example:

- From one categorical variable with a few categories, too many variables with many levels
- From only numerical datasets, to mixed categorical and numerical
- From smooth objective curves to "very" non-convex objective curves, etc

The reason for these tests will be to understand **where the algorithm breaks**. This will be a very important finding as it will show which reactions could be further explored with this technique and which are too complicated to handle with our current tools.

Another important area of exploration is to compare these algorithms with the traditional DOE. As seen in the introduction part, DOE is a parallelizable technique, and thus if Bayesian Optimization doesn't show better performance than the traditional, easy-to-explain DOE approach, then there is no need to explore this technique further.

Assumption

I assume that in simple settings (only a few variables with a few levels) there will be no significant difference between those two algorithms (given that we select a correct model for our RSM), but for larger more complicated spaces, Bayesian optimization will have the advantage. Lastly, very large input spaces, won't be sufficiently optimized by these algorithms, and the chemist's intuition will be the most valuable asset.

References

[1] Derek J Durand and Natalie Fey. Computational ligand descriptors for catalyst design. *Chemical reviews*, 119(11):6561–6594, 2019.

[2] Kobi C Felton, Jan G Rittig, and Alexei A Lapkin. Summit: benchmarking machine learning methods for reaction optimisation. *Chemistry-Methods*, 1(2):116–122, 2021.

[3] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[4] Florian H¨ase, Matteo Aldeghi, Riley J Hickman, Lo¨ıc M Roch, and Alán Aspuru-Guzik. Gryffin: An algorithm for bayesian optimization of categorical variables informed by expert knowledge. *Applied Physics Reviews*, 8(3):031406, 2021.

[5] Florian Hase, Lo¨ıc M Roch, Christoph Kreisbeck, and Alán Aspuru-Guzik. Phoenics: a bayesian optimizer for chemistry. *ACS central science*, 4(9):1134–1145, 2018.

[6] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[7] Benjamin J Shields, Jason Stevens, Jun Li, Marvin Parasram, Farhan Damani, Jesus I Martinez Alvarado, Jacob M Janey, Ryan P Adams, and Abigail G Doyle. Bayesian reaction optimization as a tool for chemical synthesis. *Nature*, 590(7844):89–96, 2021.