# Semester project addendum: Black box optimization modeling epistemic uncertainty for Bayesian optimization

Romeo Valentin
ETH, Fall 2021

Supervised by
Max B. Paulus
Andisheh Amrollahi

**Abstract**

In this work we present and explore a new algorithm applying Bayesian optimization to sequential decision making problems with discrete and graph-based inputs. We recall the basic ideas of Bayesian optimization and then showcase how epistemic uncertainty modeling with Deep Ensembles and uncertainty calibration can be used to employ a common idea in Bayesian optimization to neural networks. Finally we present a case study, applying the algorithm to the *configuration task* of the 2021 NeurIPS competition on Machine Learning for Combinatorial Optimization[1].

## Foreword

This work is an addendum to the main part of the semester project, which was the participation at the aforementioned ML4CO competition where we scored 3rd on the overall leaderboard and 1st on the student leaderboard. A related project report will be published in January 2022 at the NeurIPS workshop for ML4CO. The code and a poster are available at github.com/RomeoV/ml4co-competition.

## 1 Introduction

Mixed-Integer Linear Programming (MILP) problems are a powerful mathematical framework, suitable for example for modeling and optimizing problems arising in operations research, logistics or process systems engineering. Sophisticated solvers like SCIP [1] have been developed for many years and

[1]See ecole.ai/ml4co-competition.

have accumulated a large number of hyperparameters to tune. Although good defaults are generally chosen, often a large performance increase can be gained when looking at problems at a case by case basis.

In this work we will consider the problem of looking at a group of MILP problem instances sharing some structure (usually by arising from the same fundamental problem, for example a tight packing problem) and trying to predict good hyperparameters for each instance. To this end we reformulate the prediction problem as a supervised learning problem and show how a graph neural network based model can be trained given enough informative training data.

Interestingly though, MILP instances can generally be written as a bipartite graph, which is a structure not generally supported by many learning algorithms. Furthermore, we consider the case where we do not have an initial dataset of instances, hyperparameters and solver performances, but are free to generate one ourselves. To this end we argue that a Bayesian optimization approach can yield very informative datasets while keeping the necessary exploration low, and introduce a graph neural-network architecture for this problem. Then we show how neural networks can be adapted to model epistemic uncertainty and be used in a Bayesian optimization framework. Finally, we present a new algorithm suitable for the Bayesian optimization setting and present a case study based on the ML4CO competition data.

**Contributions:**

- A new graph neural-network based architecture for MILP problems,
- a supervised-learning formulation for the case of incomplete datasets,
- an uncertainty calibration method for using epistemic uncertainty in a Bayesian optimization setting, and
- a new algorithm for solving the sequential decision making problem when exploring MILP instances and hyperparameters.

## 2 Mathematical preliminaries & setup

In this section we introduce the mathematical setup for learning on the combination of MILP instances, which are modeled as bipartite graphs [4], and a (possibly large) configuration space. Further we show how a new informative dataset can be generated efficiently by phrasing it as a *sequential decision making problem* and will present how this can be solved.

Given instances $i \in \mathcal{I}_{\text{train}} \cup \mathcal{I}_{\text{valid}}$, a configuration space $\mathcal{C}$, and normally distributed labels

$$\hat{\gamma}(i,c) \sim \mathcal{N}\left(\hat{\gamma}_\mu(i,c), \hat{\gamma}_\sigma^2(i,c)\right), \tag{1}$$

we want to find an assignment $i \mapsto \mathit{model}(i) = c$ that minimizes the expected cost metric $\hat{\gamma}_\mu(i,c)$ on a validation set. Let $\phi_\theta$ be some prediction model parametrized by $\theta$. Then we can formalize this as: Find

$$\theta = \arg\min_\theta \mathbb{E}_{i \sim \mathcal{I}_{\text{valid}}} \left[\hat{\gamma}_\mu \left(i, \phi_\theta(i)\right)\right]. \tag{2}$$

## 2.1 Formulating a supervised learning problem

Suppose that $|\mathcal{I}|$ and $|\mathcal{C}|$ are reasonably small. Further, assume that we have access to the full labeled dataset $\mathcal{D}^{\text{full}}_{\text{train}}$, i.e. $\left\{\left(i, c, \hat{\gamma}(i,c)\right)\right\}_{\forall i \in \mathcal{I}_{\text{train}}, c \in \mathcal{C}}$. In the case where $\hat{\gamma}(i,c)$ is deterministic, we can simply create a mapping

$$i \mapsto \phi_\theta(i) = \arg\min_{c \in \mathcal{C}} \hat{\gamma}(i,c), \tag{3}$$

or estimate $\hat{\gamma}_\mu(i,c)$ in the non-deterministic setting, i.e

$$i \mapsto \phi_\theta(i) = \arg\min_{c \in \mathcal{C}} \mathbb{E}\left[\hat{\gamma}(i,c)\right] \tag{4}$$

where the expectation is estimated using the available data.

If we now want to model $\phi_\theta$ to generalize to new instances we can try to use the cost metric $\hat{\gamma}(i,c)$ as a label for supervised learning. Let

$$\phi'_\theta : \mathcal{I} \to \mathbb{R}^{|C|}_{\geq 0},\ i \mapsto \phi'_\theta(i) = \begin{bmatrix} \gamma(i,c_1) \\ \gamma(i,c_2) \\ \vdots \\ \gamma(i,c_{|C|}) \end{bmatrix} \tag{5}$$

be an auxiliary model. Then we can simply formulate

$$\phi_\theta : \mathcal{I} \to \mathcal{C},\ \arg\min_{c \in \mathcal{C}} \phi'_\theta(i)[c], \tag{6}$$

where $\phi'_\theta(i)[c]$ denotes indexing the prediction vector with the index of $c$.

The supervised learning problem is then easily derived by using $\phi'_\theta$ and training it with the full vector of labels and, for example, a simple $\mathcal{L}_2$-loss. Note that this formulation was used in the competition submission.

## 2.2 Supervised learning with an incomplete dataset

Let's consider now the setting where $\mathcal{D}_{\text{full}}$ is not available, but we have a reduced dataset $\mathcal{D}_{\text{train}} \subset \mathcal{D}^{\text{full}}_{\text{train}}$ that contains labels only for some combinations of $i$ and $c$. In order to set up a supervised learning problem for this setting, the formulation in Eqs. (5) and (6) can not be used, as we do not have access to the full labeled vector $\left[\hat{\gamma}(i,c_1), \ldots, \hat{\gamma}(i,c_{|C|})\right]^\intercal$. Instead, we consider a new model

$$\phi'_\theta : \mathcal{I} \times \mathcal{C} \to \mathbb{R}_{\geq 0},\ (i,c) \mapsto \phi'_\theta(i,c) = \gamma(i,c) \tag{7}$$

that uses both the instance and configuration as inputs to map to a single performance prediction. Similarly to Eq. (6) we can phrase our final prediction as

$$\phi_\theta : \mathcal{I} \to \mathcal{C},\ i \mapsto \arg\min_{c\in\mathcal{C}} \phi'_\theta(i, c) = c. \tag{8}$$

Note that this formulation allows for an easier setup of the supervised learning problem, namely also with an "incomplete" dataset. On the other hand a higher burden is placed on the model compared to the formulation in Section 2.1, since the model not only has to generalize to new instance, but also to unseen combinations of $i$ and $c$. This can be restrictive, but can also allow for much larger configuration spaces $\mathcal{C}$, which might become prohibitively large for the formulation in Eq. (5).

## 2.3 Setting up the sequential decision making problem

In Section 2.2 we discussed how one can set up an appropriate learning problem when a dataset $\mathcal{D}_{\text{train}} \subset \mathcal{D}^{\text{full}}_{\text{train}}$ is available, containing only some combinations of $i$ and $c$. We will now discuss how such a dataset can be created. For this we assume that we have a data generating function $G$ available with

$$G : \mathcal{I} \times \mathcal{C} \to \mathbb{R},\ (i, c) \mapsto G(i, c) = \hat{\gamma}. \tag{9}$$

Note again that $G$ can be stochastic, but that we assume an underlying Gaussian distribution (see Eq. (1)).

In theory, we can use $G$ to construct $\mathcal{D}^{\text{full}}_{\text{train}}$, but consider $G$ to be prohibitively expensive. Instead we are interested in procedurally building up $\mathcal{D}_{\text{train}}$ in order to help the model make good predictions, but without sampling the entire data space. This can be seen as a *sequential decision making problem*, where each decision is which data point(s) to generate next.

To formalize this problem, let $\mathcal{D}^{(t)}$ be the collected dataset after generating data for $t$ steps using $G$ (where each step may generate one or more samples). Further, we will recall the notion of *regret* and show how we can address the problem.

**Definition 2.1** (*Instantaneous* and *cumulative regret*)**.** Suppose that for a dataset $\mathcal{D}^{(t)} = \mathcal{D}^{(t)}_{\text{train}} \cup \mathcal{D}^{(t)}_{\text{valid}}$ we have a "good" way of fitting our model parameters $\theta^{(t)}$ to $\mathcal{D}^{(t)}_{\text{train}}$. Further, let $c^*(i) = \arg\min_c \hat{\gamma}_\mu(i, c)$ be an (unknown) oracle for the best config for a given instance. Then we can define the *relative instantaneous regret* at step $t$ as

$$r^{(t)} = \mathbb{E}_{i\sim\mathcal{I}^{\text{full}}_{\text{valid}}} \frac{\hat{\gamma}_\mu\big(i, c^*(i)\big) - \hat{\gamma}_\mu\big(i, \phi^{(t)}_\theta\big)}{\hat{\gamma}_\mu\big(i, \phi^{(t)}_\theta\big)}, \tag{10}$$

and the *relative cumulative regret* for $T$ time steps as

$$R_T = \sum_{t=1}^{T} r^{(t)}. \tag{11}$$

A desirable asymprotic property of an algorithm is to be *no-regret*, i.e.

$$\lim_{T\to\infty} R_T/T \to 0, \tag{12}$$

which implies that our model $\phi_\theta^{(t)}$ converges to $c^*$.

## 2.4 The Bayesian optimization approach to SDMP

A common way to solve sequential decision making problems (SDMPs) is by employing Bayesian optimization (BO) powered by Gaussian Processes (GPs). In essence, GPs allow for (i) learning a regression function from noisy (continuous) data, and (ii) explicitly modeling the uncertainty about unseen data points in a principled way. To achieve that, GPs make certain assumptions about local correlation of data points. For an in-depth discussion of GPs, we refer the reader to [8].

When GPs are applicable, the sequential decision making problem can then be solving by sampling new data points as guided by a combination of the GP regression and uncertainty prediction. In particular, the BO algorithms samples new points which have both a good regression prediction, but also a high uncertainty associated. Formally, let $x \in \mathbb{R}^n, y \in \mathbb{R}$ be input and output variables, and let

$$GP : \mathbb{R}^n \to \mathbb{R} \times \mathbb{R},\ x \mapsto GP(x) = \big(y(x), \sigma(x)\big) \tag{13}$$

be the GP prediction, including the predicted uncertainty $\sigma(x)$ which can be interpreted as the standard deviation of a Gaussian around $y(x)$. With this setup, the new points to be sampled can then be chosen as

$$x = \arg\min_x y(x) - \lambda \cdot \sigma(x), \tag{14}$$

where $\lambda$ denotes a hyperparameter, a sort of "exploration parameter". Note that for special cases the optimal parameter $\lambda$ can be determined analytically (see e.g. [10]), but this is not the case for our problem.

**Drawbacks of GPs.** While GPs solve the continuous case in a principled way, they are generally restricted to data $\subset \mathbb{R}^n$. Since we are considering data both in $\mathbb{Z}^n$ as well as (bipartite) graph data, GPs can not easily be used for our problem.

Instead, in the following we will discuss how we can use the ideas from Bayesian optimization and adapt them to our neural network-based approach.

## 2.5 Modeling uncertainty with neural networks

In this section we will first review some fundamentals about uncertainty estimation and then show how we can adapt our neural networks for the Bayesian optimization setting (Section 2.4), and what challenges lie therein.

### 2.5.1 (Some) Types of uncertainties

Before diving into how to model uncertainty, we will first recall two important types of uncertainty. Der Kiureghian et. al [6] first introduce a differentiation between *aleatoric* and *epistemic* uncertainty, which has since been made more precise (see e.g. [9]). Im summary, we differentiate them as:

**Aleatoric uncertainty** The (irreducible) uncertainty inherent to the data. In our case this corresponds to $\hat{\gamma}_\sigma(i, c)$.

**Epistemic uncertainty** The (reducible) uncertainty of the model about a data point, or about a data distribution (which might be disjunct from the training data distribution). In our case, this corresponds to the $\sigma$ in Eq. (14), which is what we want to model.

Note that while GPs do not explicitly output different uncertainty estimates, the aleatoric uncertainty is modeled by a model prior and the epistemic uncertainty is included in the output $\sigma$.

### 2.5.2 Modeling epistemic uncertainty in neural networks

Different approaches for modeling epistemic uncertainty (EU) in neural networks exist, although most of them are based on heuristics and are only validated empirically. A more principled approach to modeling EU is given by Bayesian Neural Networks [2, 3], but they are generally harder to apply for any given task. Instead we focus on a simple Bayesian approximation using Deep Ensembles [7] which simply consists of training multiple models and computing their "disagreement" about any given point. One way for that is simply computing the standard deviation of the predictions, i.e.

$$\sigma(i, c) = 1/E \sum_{e=1}^{E} \left( \phi_{\theta,e}(i, c) - 1/E \sum_{e=1}^{E} \phi_{\theta,e}(i, c) \right)^2 . \tag{15}$$

This method can give a decent estimate of the epistemic uncertainty, and can for example be used for out-of-distribution detection. Crucially though, the estimated uncertainties tend to be *uncalibrated* in the sense that they lie in a (often very) different order of magnitude than the regression predictions. In Section 3.1 we will introduce how to remedy this problem and adapt it for use in Eq. (14).

## 3 Method

In this section we introduce a method for solving the sequential decision making problem for the introduced task, namely sampling a meaningful dataset consisting of bipartite graphs and discrete data. To this end, we will first present how we calibrate the epistemic uncertainty in order to use the data sampling strategy from Eq. (14). Then we will present our algorithm and comment on the individual steps.

## 3.1 Calibrating the epistemic uncertainty

As outlined in Section 2.5.2, a crucial problem with most methods estimating the epistemic uncertainties in neural networks is that they are *uncalibrated.* Unlike in GPs, where the magnitude of $\sigma$ is governed by the magnitude of $y$, in Dense Ensembles $\sigma$ can be multiple orders of magnitude away from $y$. This problem is further accentuated by the fact that for epistemic uncertainty it is hard to come up with a general definition of calibration[2].

To solve this problem, we introduce a sort of "normalization procedure" that aims to bring the regression values and the estimated uncertainties on the same scale manually. For this, let's consider again the setting introduced in Section 2.2. The normalization procedure then consists of two steps:

**1) Data normalization.** First, we normalize the data as follows: Given a regression label $\hat{\gamma}$, we normalize it with

$$\hat{\gamma} \leftarrow \frac{\hat{\gamma} - mean\left(\hat{\boldsymbol{\gamma}}_{\text{train}}\right)}{std\left(\hat{\boldsymbol{\gamma}}_{\text{train}}\right)} \tag{16}$$

where $\hat{\boldsymbol{\gamma}}_{\text{train}}$ denotes all labels in $\mathcal{D}_{\text{train}}$, such that the labels are zero mean and have a standard deviation of one.

**2) Uncertainty normalization.** Now that the labels are on a known scale, we try to bring the uncertainty estimations onto the same scale. For this, we first set a target calibration number, say $C_{\text{target}} = 0.5$, and then try to calibrate the parameter $\lambda$ such that the average epistemic uncertainty is equal to $C_{\text{target}}$:

$$\mathbb{E}_{(i,c)\sim\mathcal{D}_{\text{train}}}\big(\lambda \cdot \sigma(i,c)\big) = C_{\text{target}} \tag{17}$$

$$\Rightarrow \lambda = C_{\text{target}}/\mathbb{E}_{(i,c)\sim\mathcal{D}_{\text{train}}}\sigma(i,c). \tag{18}$$

The computed $\lambda$ can then be used in equation Eq. (14).

Note that we seem to have just replaced the hyperparameter $\lambda$ with a new hyperparameter $C_{\text{target}}$. But unlike $\lambda$, $C_{\text{target}}$ can be interpreted on the scale of normally distributed data with standard deviation one.

**Algorithm 1:** Sequential decision making algorithm

```
Input: Data generator $G$, configuration space $\mathcal{C}$, instance set $\mathcal{I}_{\text{train}}$,
       trainable model $\phi_\theta$, number of sample to generate per step
       $N$, and total number of steps $T$.
Output: Model parametrization $\theta^{(T)}$ and generated dataset $\mathcal{D}^{(T)}_{\text{train}}$.
$\mathcal{D}^{(1)}_{\text{train}} \leftarrow$ evaluate $N$ random samples $(i, c) \in \mathcal{I}_{\text{train}} \times \mathcal{C}$;
for $t$ in $1, \ldots, T$ do
    $\tilde{\mathcal{D}}^{(t)}_{\text{train}} \leftarrow normalizeLabels\left(\mathcal{D}^{(t)}_{\text{train}}\right)$;
    $\theta^{(t)} \leftarrow fitModel\left(\tilde{\mathcal{D}}^{(t)}_{\text{train}}\right)$;
    $\lambda \leftarrow computeCalibrationConstant\,(\phi_\theta, \mathcal{I}_{\text{train}})$;
    $\mathcal{D}_{\text{new}} \leftarrow selectNewSamples\left(\phi_\theta, \theta^{(t)}, \lambda, \mathcal{I}_{\text{train}}, \mathcal{C}\right)$;
    $\mathcal{D}^{(t+1)}_{\text{train}} \leftarrow \mathcal{D}^{(t)}_{\text{train}} \cup generateNewLabels\,(\mathcal{D}_{\text{new}})$;
end
return $\theta^{(T)}$;
```

## 3.2 The algorithm

In Algorithm 1 we present the algorithm which puts together the pieces discussed in Sections 2 and 3.1. In addition, we comment on some technicalities for each step of the algorithm.

***normalizeLabels*** Instead of normalizing with a "global" mean, we normalize "instance-wise" by computing the mean for each instance separately. For the standard deviation this would also be beneficial, but does not work in early iterations where only one sample for an instance might be available, which leads to $std([x]) = 0$ and subsequently a division (normalization) by zero. Therefore we normalize instead with a "global" standard deviation.

***fitModel*** See Appendix A for an overview of the model. In each algorithm step, we train for three minutes with a learning rate of $5e{-}4$ and the `RMSProp` optimizer.

***computeCalibrationConstant*** We approximate

$$\mathbb{E}_{(i,c)\sim\mathcal{D}_{\text{train}}}\sigma(i,c) \approx 1/256 \sum_{(i,c)\overset{256}{\sim}\mathcal{D}_{\text{train}}} \sigma(i,c) \tag{19}$$

where $\overset{N}{\sim}$ denotes randomly sampling $N$ times.

[2]For aleatoric uncertainty on the other hand recall that the ground truth is given by $\hat{\gamma}_\sigma$. See also [5] for further discussion of calibration

***selectNewSamples*** For $N = 200$ we repeat 10 times:

- Sample 512 samples for $(i, c)$ (i.e. $(i, c) \overset{512}{\sim} \mathcal{D}_{\text{train}}$),
- sort the samples by $\gamma(i, c) - \lambda \cdot \sigma(i, c)$, and
- pick the best 20.

## 4 Experimental results

### 4.1 Experimental setup

We present experimental results on the configuration task for problem 1 (item placement) in the ML4CO competition. For this problem $\mathcal{I}_{\text{train}}$ ($\mathcal{I}_{\text{valid}}$) consist of the first 9900 (last 100) problem instances, which are modeled as bipartite graphs. We restrict the configuration space to $\mathcal{C} = \{0, 1, 2, 3\}^3$ with $|\mathcal{C}| = 64$ where $0, \ldots, 3$ denote the settings `OFF`, `DEFAULT`, `FAST`, and `AGGRESSIVE` for the SCIP meta-configs `presolve`, `heuristic`, and `separating`, respectively. When generating new data, we let SCIP run with a timeout of 5 minutes and then record the primal-dual-integral.

### 4.2 Results

Fig. 1 showcases the relative instantaneous regret (Eq. (10)) for three different seeds, evaluated on the (unseen) validation set. Recall that $r^{(t)} = 0$ would mean predicting the best config for every instance, or in other words $\phi_\theta(i) = c^*(i)\ \forall i \in \mathcal{I}_{\text{valid}}$ where $c^*$ is again the "perfect" oracle. Additionally, the regret for the default config is given. We can see that the default config is on average about 60% worse compared to $c^*(i)$.

Surprisingly, for seeds 2 and 3 the model does very well after only one or two iterations, having a $15-30\%$ smaller regret than the default config. But then the models actually start to deteriorate over time, some more quickly than others.

One possible explanation for the good early performance and later performance degradation could be as follows: The algorithm randomly samples a config which tends to be good for many instances. The model then simply learns that this config is good, and does not take instance-level information (which is much more complex) into account. Then, over time, the model might start to overfit to the MILP instances instead of learning to generalize from them, therefore the performance drops gradually.

We conclude that further investigation is necessary, in particular looking into the performance of the instance-level information retrieval. We also note that this timespan only explored a maximum of 3000/9900 training instances, so a longer experiment might be necessary.

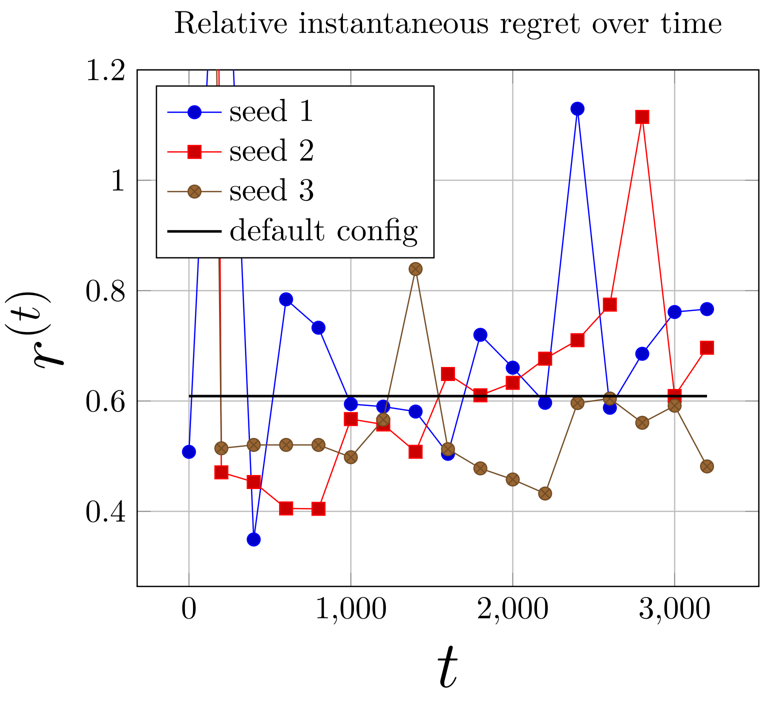


Figure 1: The relative instantaneous regret over time for three different seeds. Note that $\Delta t = 1$ means one more data point generated, but we move in steps of $\Delta t = 200$.

## 5 Conclusion and outlook

In this work we have presented a mathematical framework and implementation to generalize the ideas of Bayesian optimization to discrete and graph-based inputs, and applied them to a specific use case related to black box optimization and sequential decision making. To this end, we have proposed a new algorithm that recalibrates the uncertainty estimation of the model in every iteration in order to use an exploration objective function commonly used in Gaussian processes. When considering the results it is clear that the algorithm is not performing too strongly yet, although it has shown some promises. We therefore suggest further investigation, in particular in the generalization capabilities of the graph neural-network.

## References

[1] Tobias Achterberg. “SCIP: Solving Constraint Integer Programs”. In: *Mathematical Programming Computation* 1.1 (July 1, 2009), pp. 1–41. ISSN: 1867-2957. DOI: `10.1007/s12532-008-0001-1`. URL: `https://doi.org/10.1007/s12532-008-0001-1` (visited on 12/01/2021).

[2] Christopher M. Bishop. “Bayesian Neural Networks”. In: *Journal of the Brazilian Computer Society* 4 (July 1997), pp. 61–68. ISSN: 0104-6500, 1678-4804. DOI: `10.1590/S0104-65001997000200006`.

[3] Charles Blundell et al. “Weight Uncertainty in Neural Networks”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, July 6, 2015, pp. 1613–1622.

[4] Maxime Gasse et al. “Exact Combinatorial Optimization with Graph Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[5] Chuan Guo et al. “On Calibration of Modern Neural Networks”. In: *International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, July 17, 2017, pp. 1321–1330.

[6] Armen Der Kiureghian and Ove Ditlevsen. “Aleatory or Epistemic? Does It Matter?” In: *Structural Safety*. Risk Acceptance and Risk Communication 31.2 (Mar. 1, 2009), pp. 105–112. ISSN: 0167-4730. DOI: `10.1016/j.strusafe.2008.06.020`.

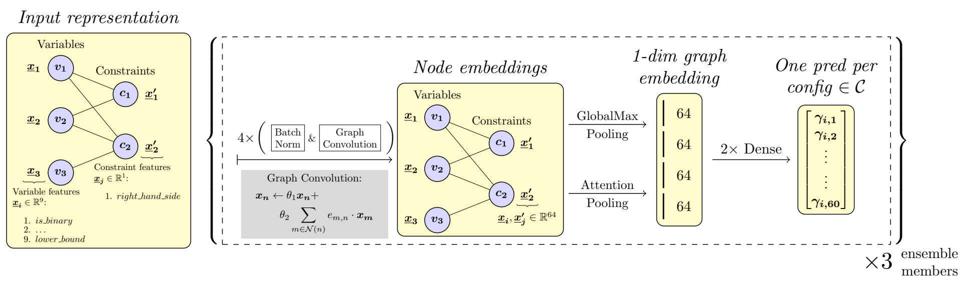
[7] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. “Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles”. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.

[8] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Red. by Francis Bach. Adaptive Computation and Machine Learning Series. Cambridge, MA, USA: MIT Press, Nov. 23, 2005. 272 pp. ISBN: 978-0-262-18253-9.

[9] Lewis Smith and Yarin Gal. *Understanding Measures of Uncertainty for Adversarial Example Detection*. Mar. 22, 2018. arXiv: `1803.08533 [cs, stat]`. URL: `http://arxiv.org/abs/1803.08533` (visited on 12/01/2021).

[10] Niranjan Srinivas et al. “Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML’10. Madison, WI, USA: Omnipress, June 21, 2010, pp. 1015–1022. ISBN: 978-1-60558-907-7.

## A Model overview



Input representation as in [4].