Scalable Meta-Learning for Bayesian Optimization using Ranking-Weighted Gaussian Process Ensembles

Matthias Feurer[*] FEURERM@CS.UNI-FREIBURG.DE
University of Freiburg

Benjamin Letham BLETHAM@FB.COM
Facebook

Eytan Bakshy EBAKSHY@FB.COM
Facebook

Abstract

Bayesian optimization has become a standard technique for hyperparameter optimization of machine learning algorithms. We consider the setting where previous optimization runs are available, and we wish to use their results to warm-start a new optimization run. We develop a new ensemble model for Bayesian optimization that can incorporate the results of past optimization runs, while avoiding the poor scaling that comes with putting all results into a single Gaussian process model. Our experiments show that the ensemble can substantially reduce optimization time compared to standard Gaussian process models and improves over the current state-of-the-art model for warm-starting Bayesian optimization.

1. Introduction

Bayesian optimization is a technique for solving black-box optimization problems with expensive function evaluations which has been successfully applied to optimizing the hyperparameters of machine learning algorithms, where a function evaluation involves training the model (Snoek et al., 2012). Given a small initial set of function evaluations, Bayesian optimization proceeds by fitting a surrogate model to those observations, typically a Gaussian process (GP), and then optimizing an acquisition function that balances exploration and exploitation in determining what point to evaluate next.

The "black-box" nature of the optimization assumes that nothing is known about the problem besides the observed function values, but there are settings in which ancillary information is available in the form of prior optimizations. Such prior optimization can be a result of re-optimizations of a production machine learning model, which is constantly re-trained as new data become available. The optimal hyperparameters may change as the data changes, so they should be frequently re-optimized. Another setting for continuous re-optimizations is machine learning services, where the underlying model stays the same, but is constantly working on different data.

Suppose we have access to the outcomes of previous optimization runs of the same model on different data, or of similar models. If we can identify similar runs we can use their results to guide and speed up the optimization process (i.e., require less function

[*] Research was performed while at Facebook.

evaluations). Re-using the results from past optimization runs can result in reduced time for hyperparameter optimization and thus faster development cycles, reduced CPU usage, or may enable hyperparameter optimization at all for models with very long training times.

To facilitate meta-learning in Bayesian optimization we specify several design goals that allow the reuse of existing infrastructure and minimize the additional resources required to deploy warm-starting. First, we may have a large number of potentially related past optimization runs, and so we need a method that is able to handle both a large number of observations in total across all tasks and a large number of tasks. Second, we need to be able to easily update the surrogate model after adding a new observation on the current target task. Third, the model needs to be directly pluggable into existing Bayesian optimization machinery so that we can reuse established techniques for acquisition function optimization, parallelization, and handling of uncertainty in the target function. Finally, the method needs to be easily adaptable to different hyperparameter spaces or tasks. In particular, it should not depend on task-specific hyperparameter settings, and also not depend on numerical task descriptors which have to be defined by experts. Several Bayesian optimization methods have been developed to borrow strength across runs—we describe these in Appendix A and compare them to these stated design goals.

The contribution of this paper is an ensemble method for warm-starting Bayesian optimization using past runs, called the ranking-weighted Gaussian process ensemble (RGPE). The method fits a GP to the outcome of each prior optimization run and combines all base GPs into a single GP. RGPE neither requires the existence of meta-features nor depends on specific hyperparameter settings, allowing it to be applied to a broad set of optimization problems. We evaluate its performance on a large collection of SVM hyperparameter optimization benchmark problems and provide further experiments in Appendix C. Our method outperforms its closest competitor and is able to improve over the current optimization method for a computer vision platform at Facebook.

2. Background and Problem Setup

The goal of Bayesian optimization is to find a minimizer $\mathbf{x}^*$ of a black-box function in a bounded space by iteratively querying the function at input configurations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ and observing the corresponding outputs $y_1, y_2, \ldots, y_n$. In each iteration we first fit a probabilistic model f on observations $\mathcal{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^n$ made so far. We then use an acquisition function $\alpha(\mathbf{x})$ to select a promising configuration to evaluate next, balancing exploration and exploitation.

In general y_j may be a noisy estimate of the function value, and the noise standard deviation may also be known. We estimate the underlying function with GP regression, yielding a posterior $f(\mathbf{x}|\mathcal{D})$ that has mean $\mu(\mathbf{x})$ and variance $\sigma^2(\mathbf{x})$, which are known analytically (Rasmussen and Williams, 2006). These quantities depend on the GP kernel, which has several hyperparameters that are inferred when the model is fit. The GP posterior at a collection of points $[f(\mathbf{x}_1|\mathcal{D}), \ldots, f(\mathbf{x}_n|\mathcal{D})]$ has a multivariate normal distribution with mean and covariance matrix denoted as $\mu(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and $\Sigma(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

In our experiments in Section 4 and Appendix C.2 we use the expected improvement (EI) acquisition function (Jones et al., 1998), a common choice because it can be computed in closed form and optimized with gradient-based methods. Let $f(\mathbf{x}_{\text{best}})$ be the value of the

best point observed so far: $f(\mathbf{x}_{\text{best}}) = \min_{j=1,\dots,n} f(\mathbf{x}_j)$. The EI is

$$\alpha(\mathbf{x}|\mathcal{D}) = \mathbb{E}_{y\sim f(\mathbf{x}|\mathcal{D})}\left[\max(0, f(\mathbf{x}_{\text{best}}) - y)\right] = \sigma(\mathbf{x})z\Phi(z) + \sigma(\mathbf{x})\phi(z),$$

with $z = \frac{f(\mathbf{x}_{\text{best}})-\mu(\mathbf{x})}{\sigma(\mathbf{x})}$. A thorough introduction to Bayesian optimization is given by Shahriari et al. (2016).

We suppose that $t-1$ runs of Bayesian optimization have been completed. Let $\mathcal{D}_i = \left\{(\mathbf{x}_j^i, y_j^i)\right\}_{j=1}^{n_i}$ be the function evaluations made for past optimization runs i. We fit a GP model to the observations of each past run i and refer to these models as *base models*. They have posterior $f^i(\mathbf{x}|\mathcal{D}_i)$, with mean and variance $\mu_i(\mathbf{x})$ and $\sigma_i^2(\mathbf{x})$ respectively. They remain fixed throughout the optimization, inasmuch as we do not obtain new observations for them. The current optimization problem we are trying to solve is run t. We fit a GP to observations from run t and call it the *target model*. The target model is refit after each new function evaluation. We overload notation and define $\mathcal{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_t\}$. Our goal is to minimize the target function using the base models and the target model.

3. Ranking-Weighted Gaussian Process Ensemble

Our strategy here is to estimate the target function as a weighted combination of the predictions of each base model and the target model itself:

$$\bar{f}(\mathbf{x}|\mathcal{D}) = \sum_{i=1}^{t} w_i f^i(\mathbf{x}|\mathcal{D}_i).$$

A model of this form is preferred for several practical reasons. First, this ensemble model remains a GP, and in particular

$$\bar{f}(\mathbf{x}|\mathcal{D}) \sim \mathcal{N}\left(\sum_{i=1}^{t} w_i\mu_i(\mathbf{x}), \sum_{i=1}^{t} w_i^2\sigma_i^2(\mathbf{x})\right).$$

This means that all of the usual computational machinery for Bayesian optimization with GPs remains valid, such as a closed-form expression for EI and the ability to draw joint samples for parallel optimization. Additionally, each base model remains unchanged throughout the optimization and can be loaded directly from the previous runs. The fitting cost is only the cost of fitting the target model and inferring the weights w_i. Finally, predictions are made independently for each GP and we obtain $\mathcal{O}(tn^2 + n^3)$ complexity and a linear slowdown relative to no warm-starting. Following Yogatama and Mann (2014), we standardize each model prior to inclusion in the ensemble.

Our approach for computing the ensemble weights w_i follows the agnostic Bayesian ensemble of Lacoste et al. (2014), which weights predictors according to estimates of their generalization performance. Given a desired loss function, each predictor in the ensemble is weighted according to the probability that it is the best predictor in the ensemble. We use a ranking loss to compute weights, and so call this method the ranking-weighted Gaussian process ensemble (RGPE).

We now discuss our loss function and approach for estimating generalization of each model. Practical details regarding EI optimization can be found in Appendix B.2.

3.1. Computing Ensemble Weights

Our goal in Bayesian optimization is to find the minimum function value. A model will be useful for optimization if it is able to correctly order observations according to their function value. For meta-learning, we wish to assess the ability of model i to generalize to the target function, and so construct a loss function that measures the degree to which each model is able to correctly rank the target observations $\mathcal{D}_t$. Given $n_t > 1$ target function evaluations, we define the loss as the number of misranked pairs:

$$\mathcal{L}(f, \mathcal{D}_t) = \sum_{j=1}^{n_t} \sum_{k=1}^{n_t} \mathbb{1}((f(\mathbf{x}_j^t) < f(\mathbf{x}_k^t)) \oplus (y_j^t < y_k^t)), \tag{1}$$

where $\oplus$ is the exclusive-or operator.

For base models, this measures their ability to generalize to the target function. For the target model, this is an estimate of in-sample error and does not accurately reflect generalization. We estimate generalization in the target model using cross-validation, in practice with leave-one-out models. Let f_{-j}^t indicate the target model with observation $(\mathbf{x}_j^t, y_j^t)$ left out. The loss for the target model is then computed by using f_{-j}^t in (1) in the place of f. The leave-one-out model is constructed by removing data point j from the GP; kernel hyperparameters are not re-estimated. Fig. 2 provides an illustration of how misrankings are computed, showing the inner sum of (1).

Ranking loss is more appropriate for estimating optimization performance than other choices such as squared error or model log-likelihood because the actual values of the predictions do not matter for optimization—we only need to identify the location of the optimum. Figure 2 (bottom right) illustrates why the squared error is not an appropriate loss function for our purpose.

We weight each model with the probability that it is the model in the ensemble with the lowest ranking loss. The posterior for f^i at the target observations is a multivariate normal with mean $\mu_i(\mathbf{x}_1^t, \ldots, \mathbf{x}_{n_t}^t)$ and covariance $\Sigma_i(\mathbf{x}_1^t, \ldots, \mathbf{x}_{n_t}^t)$. We draw samples from this posterior at the configurations evaluated so far and then obtain posterior samples of the ranking loss by evaluating (1) on the GP samples. We draw S such samples: $\ell_{i,s} \sim \mathcal{L}(f^i, \mathcal{D}_t)$ for $s = 1, \ldots, S$ and $i = 1, \ldots, t$. Weight for model i is then computed as

$$w_i = \frac{1}{S} \sum_{s=1}^{S} \mathbb{1}\left(i = \arg\min_{i'} \ell_{i',s}\right). \tag{2}$$

If the argmin is not unique, the weight is given to the target model if it is part of the tie, otherwise the tie is broken randomly. By using samples from the posterior distributions we take the overall uncertainty of the base and target models into account when calculating the weights, which is not done by any other ensemble method for Bayesian optimization.

3.2. Preventing Weight Dilution

One challenge within this type of ensemble is preventing weight dilution by a large number of noisy models. Suppose a GP has high variance at the points in $\mathcal{D}_t$ and so is able to produce arbitrary rankings of the points. This model is clearly not useful for making predictions,

and if better models are present in the ensemble it will have a low probability of being the argmin in Equation (2). However, if we have a very large number of such models in the ensemble, the chance of any one of them producing the correct ranking in a sample goes to 1 as the number of noisy models increases.

We prevent weight dilution by discarding models that are substantially worse than the target model. Model i is discarded from the ensemble if the median of its loss samples $\ell_{i,s}$ is greater than the 95^{th} percentile of the target loss samples $\ell_{t,s}$. In addition to preventing weight dilution, this strategy has computational benefits in that it results in fewer GP predictions for each ensemble model prediction. The choice of the 95^{th} percentile for the exclusion threshold could be considered a hyperparameter of the method, however, early experiments showed no sensitivity to the exact hyperparameter value.

This thresholding strategy is very flexible in comparison to other ensemble strategies. Models are only removed from the ensemble if they actually perform worse than the target model, while TST-R (Wistuba et al. (2016b), see Appendix A) removes models if they perform badly as evaluated on the observed function evaluations without taking the performance of the target model into account. Other methods based on stacking require additional tuning of a regularization hyperparameter to set the weight of a base model to zero at all.

4. Experiments

We provide a comprehensive study of RGPE's performance using a large set of hyperparameter optimization problems for support vector machines (SVM). Further experiments using a synthetic benchmark function and the computer vision platform at Facebook can be found in Appendix C. All GPs in these experiments used GPy and the ARD Matérn 5/2 kernel (GPy, since 2012). Kernel hyperparameters were set to their posterior means, inferred via MCMC with the NUTS sampler (Hoffman and Gelman, 2014).

Our main experimental validation of RGPE uses a large set of hyperparameter optimization benchmark problems from Wistuba et al. (2015b), which was also used by Wistuba et al. (2016b). They did hyperparameter searches for an SVM on a diverse set of 50 datasets, with sizes ranging from 35 to 250000 training examples, and from 2 to 7000 features. For each dataset, test-set accuracy was measured on a grid of six parameters: three binary parameters indicating a linear, polynomial, or RBF kernel; the penalty parameter C; the degree of the polynomial kernel (0 if unused); and the RBF kernel bandwidth (0 if unused). The grid search resulted in 288 points for each dataset. Note that this is a harder problem than the common 2-dimensional RBF SVM problem. The goal is to optimize the hyperparameters over this grid of 288 points for each problem, while treating the remaining 49 problems as past runs for meta-learning.

Each optimization run was initialized with three randomly selected points, after which it proceeded sequentially for a total of 20 function evaluations. For meta-learning methods, we fit base models for each of the 49 other problems on a random sample of 50 points. Optimization was repeated 20 times for each of the 50 problems, for a total of 1000 optimization runs. Each optimization run used a different random initialization of three samples for the target model, and a different random sample of 50 function evaluations for the base models. Five methods were evaluated on this problem: random search, standard Bayesian optimization with a GP fit only to observations made on the target task (GP), the proposed RGPE

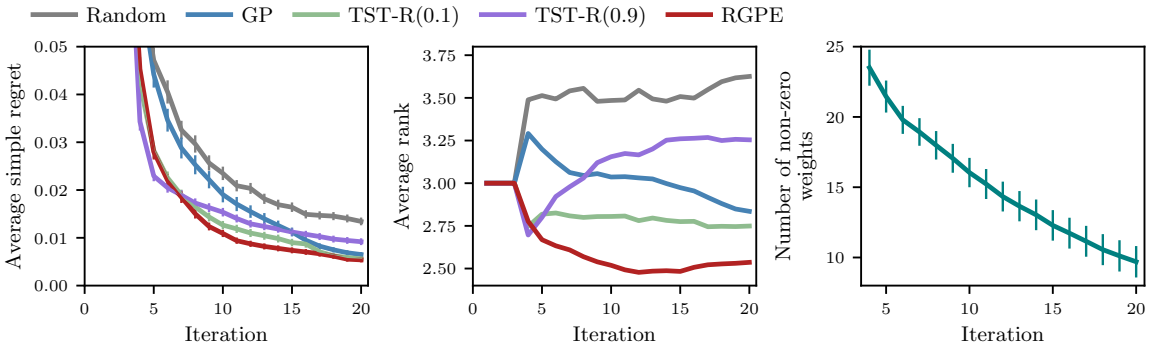

Figure 1: Optimization performance on the SVM hyperparameter optimization benchmarks, evaluated over 20 runs for each of 50 problems. (Left) Simple regret averaged over runs, with bars showing standard error. (Middle) The average rank of each method, ranked by simple regret (lower is better). (Right) The number of non-zero weights in the RGPE ensemble (out of 49).

and TST-R (Wistuba et al., 2016b) as it is the method closest to ours, with bandwidth hyperparameters set to $\rho = 0.1$ and $\rho = 0.9$ (Wistuba, 2016).

Figure 1 shows the results of these experiments. The left panel shows that warm-starting provided RGPE with a significant, early drop in regret compared to GP, which it was able to sustain throughout the optimization. TST-R with $\rho = 0.9$ had the quickest initial drop in regret, however by iteration 7 it was passed by RGPE, and by iteration 15 it was passed even by GP. We also compared the performance of the methods by computing at each iteration the average rank of each method, ranked by simple regret and averaged over optimization runs. Ranks were averaged in the case of ties. Figure 1 shows that from iteration 5 and on, RGPE outperformed the other methods and had the lowest rank. TST-R initially outperformed GP, but by iteration 8 GP achieved a lower average rank than TST-R with $\rho = 0.9$, and was close to TST-R with $\rho = 0.1$ by iteration 20.

The right panel of Figure 1 shows the number of non-zero weights in the RGPE. There were 50 models in the ensemble (49 base models and the target model), but on average less than half of them were ever used. Computationally, this means that instead of a RGPE function evaluation requiring 50 GP evaluations, it actually required many fewer—only about ten by the end of the optimization.

5. Conclusion

Our goal in developing RGPE was to have a meta-learning method that avoids the $\mathcal{O}(t^3n^3)$ scaling of putting all observations into a single GP, while maintaining the nice distributional properties of a GP. This allows RGPE to be directly substituted for a GP in a Bayesian optimization system. Closed-form acquisition functions and parallelization methods that have been developed for GPs can be used directly with RGPE. Our experiments showed that RGPE performed better than the alternative ensemble approach, and that it is a scalable and effective method for warm-starting GP-based Bayesian optimization.

Acknowledgements

Thanks to Till Varoquaux for support in developing the method, and to Alex Chen for supporting the Lumos experiments.

References

Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michèle Sebag. Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 199–207, 2013.

Matthias Bauer, Mark van der Wilk, and Carl Edward Rasmussen. Understanding probabilistic sparse Gaussian process approximations. In *Advances in Neural Information Processing Systems 29*, pages 1533–1541, 2016.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24*, pages 2546–2554, 2011.

Pavel Brazdil, João Gama, and Bob Henery. Characterizing the applicability of classification algorithms using meta-level learning. In *Proceedings of the European Conference on Machine Learning*, pages 83–102, 1994.

Lehel Csató and Manfred Opper. Sparse on-line Gaussian processes. *Neural Computation*, 14(3):641–668, 2002.

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 4134–4142, 2015.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1487–1496, 2017.

GPy. GPy: A Gaussian process framework in Python. http://github.com/SheffieldML/GPy, since 2012.

Matthew D. Hoffman and Andrew Gelman. The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15: 1351–1381, 2014.

Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th Conference on Learning and Intelligent Optimization*, pages 507 – 523, 2011.

Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimization problems. *International Journal of Mathematical Modeling and Numerical Optimisation*, 4(2):150–194, 2013.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.

Alexandre Lacoste, Hugo Larochelle, Mario Marchand, and François Laviolette. Agnostic Bayesian learning of ensembles. In *Proceedings of the 31st International Conference on Machine Learning*, pages 611–619, 2014.

Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. Selecting classification algorithms with active testing. In *Machine Learning and Data Mining in Pattern Recognition*, pages 117–131, 2012.

Benjamin Letham, Brian Karrer, Guilherme Ottoni, and Eytan Bakshy. Constrained Bayesian optimization with noisy experiments. *arXiv:0706.1234 [stats.ML]*, 2017.

M. Lindauer and F. Hutter. Warmstarting of model-based algorithm configuration. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.

David Pardoe and Peter Stone. Boosting for regression transfer. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 863–870, 2010.

Matthias Poloczek, Jialei Wang, and Peter I. Frazier. Warm starting Bayesian optimization. In *Winter Simulation Conference*, 2016. arXiv:1608.03585.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, Massachusetts, 2006. ISBN 0-262-18253-X.

Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Scalable hyperparameter optimization with products of Gaussian process experts. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 33–48, 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.

Alistair Shilton, Sunil Gupta, Santu Rana, and Svetha Venkatesh. Regret bounds for transfer learning in Bayesian optimisation. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pages 307–315, 2017.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2951–2959, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable Bayesian optimization using deep neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2171–2180, 2015.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust Bayesian neural networks. In *Advances in Neural Information Processing Systems 29*, 2016.

Kevin Swersky, Jasper Snoek, and Ryan P. Adams. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems 26*, pages 2004–2012, 2013.

Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. Batched large-scale Bayesian optimization in high-dimensional spaces. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, pages 745–754, 2018.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Sequential model-free hyperparameter tuning. In *2015 IEEE International Conference on Data Mining*, pages 1033–1038, 2015a.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Hyperparameter optimization machines. In *2016 IEEE International Conference on Data Science and Advanced Analytics*, pages 41–50, 2016a.

Martin Wistuba. TST-R implementation, 2016. https://github.com/wistuba/TST.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Learning hyperparameter optimization initializations. In *Proceedings of the IEEE International Conference on Data Science and Advanced Analytics*, pages 1–10, 2015b.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Two-stage transfer surrogate model for automatic hyperparameter optimization. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 199–214, 2016b.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1): 43–78, 2018.

Dani Yogatama and Gideon Mann. Efficient transfer learning method for automatic hyperparameter tuning. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, pages 1077–1085, 2014.

Appendix A. Related Work

Borrowing strength from past runs is a form of meta-learning, and in the context of Bayesian optimization is often called transfer learning. A key requirement is to determine which past runs are similar to the current task. Currently, there are five techniques in the literature: 1) a single model which learns task similarities, 2) an initial design learned from previous optimization runs, 3) changing the acquisition function to include past models, 4) learning an adaption of each base task to the current task, and 5) learning a separate model for each base task and combining those.

Several past methods have used manually defined meta-features to measure task similarity (Brazdil et al., 1994) and then adapted GP-based Bayesian optimization (Bardenet et al., 2013; Yogatama and Mann, 2014; Schilling et al., 2016). Besides the drawback of requiring additional hand-designed features, these methods share the issue that meta-features are non-adaptive throughout the optimization process and do not make use of the new observations (Leite et al., 2012; Wistuba et al., 2018). Another approach is to learn the task similarity without the use of meta-features in order to fit a joint model. Swersky et al. (2013) use a multitask GP to jointly model all past runs and the current task. Multitask GPs suffer from the same poor scaling as putting all observations into a single GP, and cannot be used for the problem of Section 4. Furthermore, Swersky et al. (2013) sample a $t \times t$ lower triangular matrix describing task correlations, which prohibits a large number of past runs.

Orthogonal to optimization is learning the initial design based on past observations, which can be done both with (Feurer et al., 2015) and without (Wistuba et al., 2015a) meta-features. Using an adapted initial design is complementary to our strategy and would benefit our proposed method as well. It does, however, introduce a new hyperparameter, the size of the initial design.

Another approach to handling meta-data in Bayesian optimization is to adapt the acquisition function (Wistuba et al., 2016a, 2018). We do not consider it here because the use of a modified acquisition function would not allow us to directly use established techniques for Bayesian optimization, such as parallelization via Monte Carlo integration and proper handling of uncertainty.

A number of papers propose to learn an adaptation of each past optimization run to the current run. Schilling et al. (2016) learn a joint model for each past run and the target task using meta-features as similarity descriptors. Shilton et al. (2017) model the difference between the past run and the target task with a GP, which is then used to adjust the past run observations for inclusion in the current model. Such approaches are not scalable as they require re-fitting a model for each meta-task in each iteration of the optimization process. Poloczek et al. (2016) and Golovin et al. (2017) take a similar approach for an ordered set of past runs. Rather than fitting a GP to each run separately, a GP is fit to the residuals of each run relative to the predictions of the previous model in the stack. This essentially uses the outcomes of previous runs as a prior for the next run. However, this method assumes an ordering to the runs, which would not be the case for meta-learning from a collection of unrelated problems.

The computationally least demanding strategy is fitting a separate model to each past optimization run and then learning a combination of those, which is known as transfer

stacking (Pardoe and Stone, 2010). Wistuba et al. (2016b) develop the two-stage transfer surrogate model with rankings (TST-R), which uses a Nadaraya-Watson kernel weighting to linearly combine the predictions from each GP by defining a distance metric across tasks. In particular, they consider the pairwise preferences of all observations for the current task, which is a strategy known as relative landmarking (Leite et al., 2012). The distance between the past task and the current task is taken as the proportion of discordant pairs when the past model is evaluated on configurations tested on the current task (Wistuba, 2016). Weights for each model are then computed using a quadratic kernel with bandwidth parameter ρ, which serves as a threshold for the similarity required to borrow strength from any prior task. The bandwidth ρ must be chosen by the user. The kernel is used to combine mean predictions of past models with the mean prediction of the current model, but variances are not combined—the combined model is given the variance of the current model and variances of past models are ignored. This means that the TST-R model is no longer a GP, and in particular does not have a valid posterior from which joint samples can be drawn. Furthermore, the TST-R assigns a constant weight to the target model, therefore the actual weight of the target model in the ensemble depends on the magnitude of the weights of the base models. Due to the high similarity to our proposed model we use it for comparison in our experiments of Section 4 and Appendix C.

In the related field of algorithm configuration (Hutter et al., 2009), Lindauer and Hutter (2018) use stochastic gradient descent to learn a weighted combination of random forests for runtime prediction, with each random forest being fitted on a previous algorithm configuration run. They consider a setting in which only a few auxiliary tasks are available, but the number of observations per task is on the order of several hundreds, far beyond what a GP can handle. While the usage of a simple stacking approach is appealing, it introduces a new model selection problem to find a stacking regressor. Without proper tuning, obtaining a sparse solution (sparse in terms of taking only a small subset of related tasks into account) is not possible. Furthermore, minimizing the least squares error is not necessarily a good criterion when our goal is to obtain a good model for Bayesian optimization.

A drawback which both methods share is that they do not take the uncertainty of the base models into account.

Bayesian optimization has been done with models that scale better with the number of observations, such as random forest (Hutter et al., 2011), parzen estimators (Bergstra et al., 2011) and neural networks (Snoek et al., 2015; Springenberg et al., 2016). These models come with challenges of their own, such as poor uncertainty extrapolation with limited observations and hyperparameter sensitivity. There are also extensions of GPs for large datasets, most notably sparse GPs (Csató and Opper, 2002). Sparse GPs can also have poor uncertainty extrapolation (Bauer et al., 2016; Wang et al., 2018) and do not easily produce joint samples. GPs remain the standard model for practical Bayesian optimization, especially since all methods to jointly model several tasks require the definition of meta-features or expensive calculation of task similarities.

Finally, in the setting of algorithm selection, Leite et al. (2012) introduced *active testing* which uses relative landmarking (pairwise ranking of all observations so far) to decide which algorithm configuration, from a finite set of choices, to run next. For each algorithm that was not run on the target dataset, they do a table lookup to determine whether and by how much that algorithm has improved over the current best observed algorithm on each

meta-dataset, and weight its improvement by a task similarity measure computed using the pairwise ranking of all observations so far. In a Bayesian optimization framework, this can be seen as first computing the probability of improvement criterion for each potential configuration and then weighting those by task similarity. Active testing cannot be directly applied here since it requires a finite set of configurations and requires that each of those was executed on every past dataset.

Appendix B. Further methodological details

B.1. Illustration of the Ranking Loss

Figure 2 provides an illustration of the ranking loss.

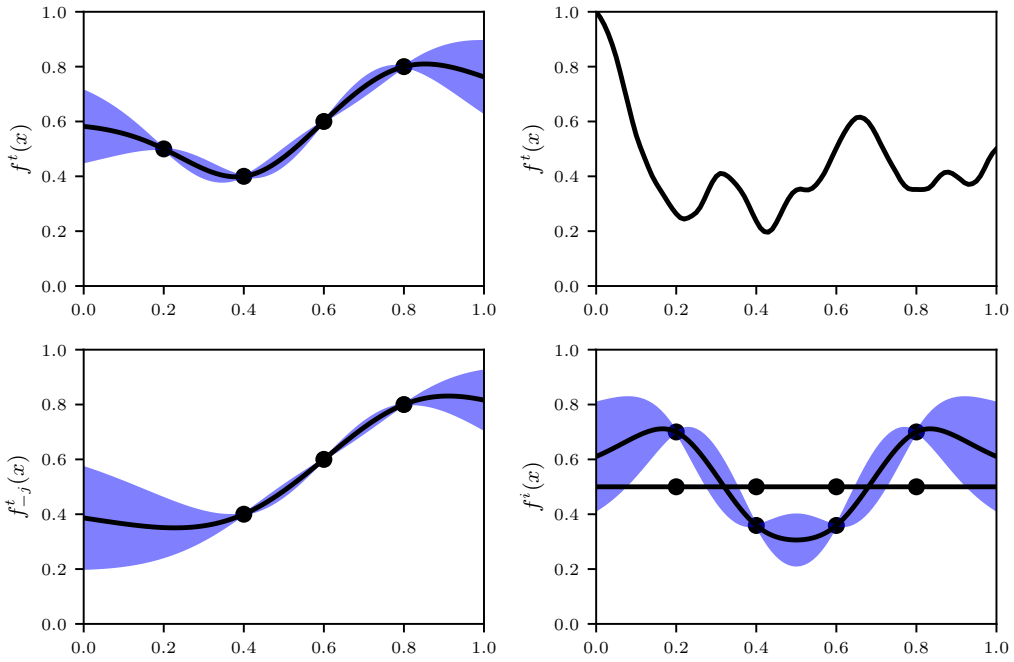

Figure 2: Illustration of the ranking loss. (Top left) The target model with four observations. (Top right) A posterior draw from a base model has one misranking, at $x = 0.8$. (Bottom left) A posterior draw from the leave-one-out target model misranks $x = 0.2$. (Bottom right) two base models which are clearly not helpful for optimization together with the target model, but which have the same mean squared error regressing the observations, and have a lower mean squared error than the model which was used to sample the function in the top right plot.

B.2. Optimization with the Gaussian Process Ensemble

The RGPE retains the distributional properties of a GP, and can therefore be used with standard acquisition functions for Bayesian optimization. More specifically, instead of μ we use $\bar{\mu}(\mathbf{x}) = \sum_{i=1}^{t} w_i \mu_i(\mathbf{x})$, and instead of σ we use $\bar{\sigma}^2(\mathbf{x}) = \sum_{i=1}^{t} w_i^2 \sigma_i^2(\mathbf{x})$, to compute EI.

In many applications of Bayesian optimization we have the ability to run multiple function evaluations in parallel. We use the technique of Snoek et al. (2012) to parallelize EI by integrating over the posterior for the outcomes at pending function evaluations. Suppose we have b pending evaluations at points $\mathbf{x}_1, \ldots, \mathbf{x}_b$, and a worker is available to evaluate an additional point. This point is chosen as the one that maximizes EI when integrated over the pending outcomes $y_1, \ldots, y_b$:

$$\hat{\alpha}(\mathbf{x}) = \int_{\mathbb{R}^b} \alpha\left(\mathbf{x} | \mathcal{D} \cup \{(\mathbf{x}_j, y_j)\}_{j=1}^{b}\right) p(\mathbf{y} | \mathbf{x}_1, \ldots, \mathbf{x}_b, \bar{f}, \mathcal{D}) d\mathbf{y}.$$

In practice, this is done using a Monte Carlo approximation. We jointly sample "fantasies" $\mathbf{y}$ from the posterior at $\{\mathbf{x}_1, \ldots, \mathbf{x}_b\}$, and then add these simulated observations to the GP and compute EI with the conditioned model. EI is averaged over several such fantasies. For RGPE, we sample from each model in the ensemble independently and condition each model on its sample to obtain the conditioned ensemble. We used 30 samples in our experiments here.

If observations are noisy or if there is uncertainty in base models at the current best, f_{best} may be a random variable. This can occur when the locations of the observations in the base models do not overlap with those of the target model. Typical approaches for computing expected improvement with noisy observations can be used with the RGPE; we follow the strategy of Letham et al. (2017) and integrate over uncertainty in f_{best} in the same way that we integrate over pending outcomes for parallelization.

Appendix C. Additional Experiments

C.1. Synthetic Function

We use a modification of the Alpine 1 function (Jamil and Yang, 2013) as a synthetic test case for warm-starting:

$$f(x, s) = x \sin(x + \pi + s) + \frac{x}{10},$$

where s is a shift parameter that is used for generating similar datasets. We used $s = 0$ as the target function, and then created five base functions with varying degrees of similarity for meta-learning: $s = \frac{k\pi}{12}, k = 1, \ldots, 5$. The target and base functions are shown in Figure 3.

Base models were fit to 20 randomly selected points from each base function. Minimization of the target function began with three quasirandom points from a scrambled Sobol sequence, and then proceeded sequentially for a total of 20 function evaluations. The optimization was repeated 100 times, each with a different random selection of the points for the base functions. Five methods were evaluated on this problem: quasirandom points with no model (Sobol), standard Bayesian optimization with a GP fit only to observations made on the target task (GP), TST-R with bandwidth $\rho = 0.1$, TST-R with bandwidth $\rho = 0.9$, and RGPE. Figure 3 shows the simple regret averaged over the 100 runs of the simulation.

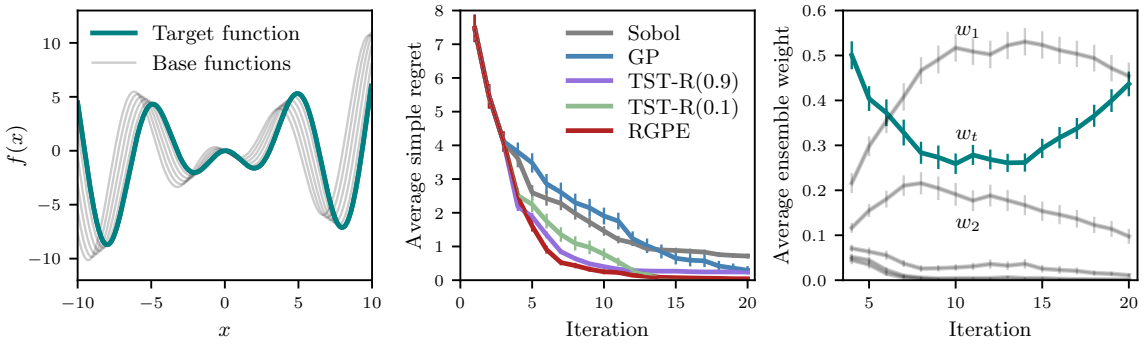

Figure 3: (Left) The target function and base functions for the synthetic test problem. (Center) Optimization performance on the test problem, averaged over 100 runs with quasirandom initializations. Error bars show standard error of the mean. Warm-starting provided a clear benefit early on and RGPE quickly converged to the global optimum. (Right) Average weights on each model in the ensemble, averaged over runs. Weight w_t is of the target model, and w_1 and w_2 correspond to the base functions with the smallest shifts from the target. RGPE relied heavily on the most similar base models.

RGPE used the warm-start provided by the base models to immediately begin sampling near the global optimum and quickly converge. The base model corresponding to the smallest shift ($s = \frac{\pi}{12}$) received most weight of all base models in the ensemble (w_1 in Figure 3), and the two models with the largest shifts received no weight after iteration 7. Weight on the target model, w_t, increased later in the optimization as it gained more predictive power. TST-R also provided benefit over GP, but its performance depended on the value of the kernel bandwidth ρ and was not able to reach the preformance of RGPE.

C.2. Optimizing a Computer Vision Platform

Lumos is a computer vision platform at Facebook that is used to train image classification models for a large variety of tasks and datasets. The final stage of the model is a logistic regression on top of convolutional neural network (CNN) features, for which hyperparameter optimization is done with each training. We used RGPE to accelerate the hyperparameter optimization by borrowing strength from previous runs on different datasets.

We optimized eight image classifiers trained on Lumos, each for a different task and to a different dataset. We optimized three parameters: learning rate for stochastic gradient descent, and two regularization parameters. Datasets ranged from ten thousand to two million images, for the largest of which GP Bayesian optimization required around 2,500 core hours. As base runs, we used the results of nine earlier GP Bayesian optimization sweeps on different datasets. These sweeps had 30 iterations each, for a total of 270 base iterations. Each optimization was begun with an initialization of three quasirandom points, after which it proceeded with two workers asynchronously in parallel for a total of 20 function evaluations. Figure 4 shows the results of these optimization runs. The true minimum is

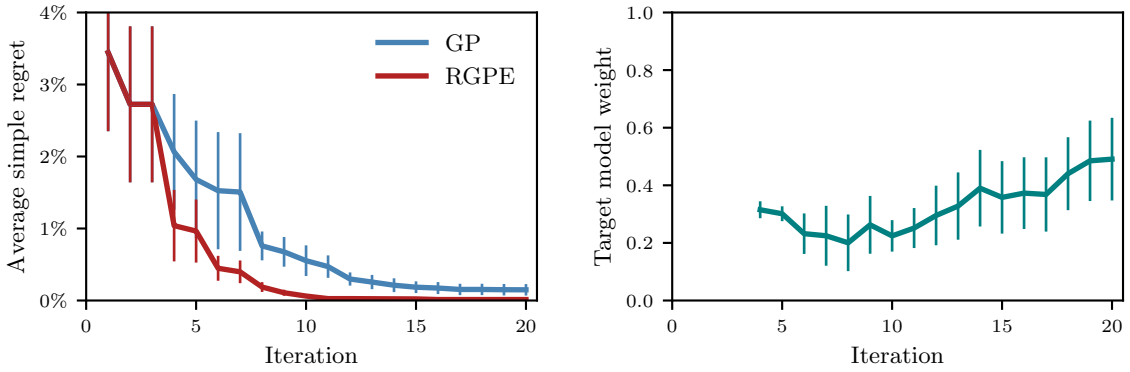

Figure 4: (Left) Optimization performance on the computer vision models, averaged over eight models (standard error in bars). With warm-starting, eight iterations were sufficient to achieve lower regret than the GP. (Right) Early on the RGPE borrowed heavily from base models, and later in the optimization began to concentrate weight on the target model.

not known for these problems, so regret was measured as a percentage of the best point found by either method.

As in the SVM benchmark problems, warm-starting provided a substantial boost in performance starting with the first optimized configuration in iteration 4. By iteration 9 RGPE achieved lower regret than the GP achieved with 20 iterations.

Figure 4 also shows the RGPE target weight throughout the iterations. In early iterations the target model was unable to generalize and so most of the ensemble weight went to base models. With more iterations, the target model improved and was given more weight, capturing 50% by iteration 20.