Supplementary Material: A Quantile-based Approach for Hyperparameter Transfer Learning

David Salinas [1] **Huibin Shen** [2] **Valerio Perrone** [2]

1. Code

Code to reproduce the results of the paper will be made available on this github repository https://github.com/geoalgo/A-Quantile-based-Approach-for-Hyperparameter-Transfer-Learning/. Offline evaluations of blackboxes are already available on this repository https://github.com/icdishb/hyperparameter-transfer-learning-evaluations/.

2. Baselines Details

WS-GP. This method uses the best-performing hyperparameter configuration from each related task to warm start the GP on the target task (Feurer et al., 2015). We also compared to two variants of WS-GP: the first-one uses dataset meta-features to detect the most closely related task and warm-starts the GP with the best s evaluations from that task; a second variant takes the best-performing s evaluations from each task, with $s > 1$. As both variants were outperformed by taking only the best evaluation from each task (i.e., $s = 1$), we show results against this version in the paper.

ABLR. The same algorithm settings as in Perrone et al. (2018) are used. The shared neural network consists of three fully connected layers, each with 50 units and `tanh` activation functions. Its weights, as well as the task-specific scale and noise parameters, are learned by optimizing the marginal log-likelihood by L-BFGS. To run BO, ABLR is combined with the EI acquisition function. We restrict the total number of evaluations for transfer learning to be around $2,000$ so that it is computationally feasible to train with L-BFGS.

[1]NAVER LABS Europe (work started while being at Amazon) [2]Amazon Web Services. Correspondence to: David Salinas <david.salinas@naverlabs.com>, Huibin Shen <huibishe@amazon.com>, Valerio Perrone <vperrone@amazon.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

SGPT. The method fits independent GPs on each related task and weights tasks based on rank matching between the objective values from the target task and the predictions of these GPs on the evaluated hyperparameter configurations (Wistuba et al., 2018). The weights are further included in the acquisition function to scale the predictive improvement on every relevant task. Due to the cubical scaling of GPs, we subsampled $1,000$ hyperparameter evaluations from each related task. We found the results to be very sensitive to the choice of ρ, that is the bandwidth of the ranking-based distance defined in Eq. (42) of Wistuba et al. (2018). We report all SGPT results with $\rho = 0.01$, which gives the best overall performance across tasks among $\rho \in \{1.0, 0.1, 0.01\}$.

R-EA. As in Dong & Yang (2020), the initial population size is 10, the number of cycles is set to infinity, and the sample size is set to 3.

REINFORCE. As in Dong & Yang (2020), the architecture encoding is optimized with ADAM. The learning rate is set to 0.001 and the momentum for exponential moving average is set to 0.9.

BOHB. As in Dong & Yang (2020), the number of samples for the acquisition function is set to 4, the random fraction is set to 0, the minimum bandwidth is set to 0.3 and the bandwidth factor to 3.

AutoGP. This method transfers information by learning a compact search space from other tasks (Perrone et al., 2019). First, a bounding box containing the best hyperparameter configuration from each other task is fit to obtain a smaller search space, which is defined by the learned coordinate-wise lower and upper bounds. Then, standard random search (RS) or GP-based BO (GP) is run in the learned search space. This method comes with no hyperparameters.

GPareto. We use the four different criteria, namely EHI, SMS, SUR and EMI, from GPareto (Binois & Picheny, 2019). When considering the new candidate with EI, we select the best possible option over the known grid of candidates. Importantly, for GPareto this search becomes prohibitively slow so we maximize EI over a random sample

of 2000 candidates out of 15625. To compute the Hypervolume error, the maximum of latency and error is used as a reference point.

3. Look-up Tables

Table 1 describes the hyperparameters considered for each tuning problem. For DeepAR and XGBoost, evaluations were obtained by sampling hyperparameters (log) uniformly at random from their search space. For FCNET and NAS, all possible configurations were evaluated.

For DeepAR, we used the following 10 public datasets from GluonTS (Alexandrov et al., 2019): {electricity, traffic, solar, exchange-rate, m4-Hourly, m4-Daily, m4-Weekly, m4-Montly, m4-Quarterly, m4-Yearly}[1]. The method was evaluated with the Sagemaker version (Januschowski et al., 2018). For FCNET and NAS, we used evaluations from Klein & Hutter (2019) and Dong & Yang (2020) which contains evaluations on {parkinson, protein, naval, slice} for FCNET and {cifar10, cifar100, Imagenet16} for NAS. For XGBoost, we used the tree boosting implementation from Chen & Guestrin (2016) and evaluated $5,000$ hyperparameter configurations against 9 LIBSVM binary classification datasets, namely {a6a, australian, german.numer, heart, ijcnn1, madelon, spambase, svmguide1, w6a}.[2]

Table 1 describes the hyperparameters considered for each tuning problem. For DeepAR and XGBoost, evaluations were obtained by sampling hyperparameters (log) uniformly at random from their search space. For FCNET and NAS, all possible configurations were evaluated.

References

Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., Maddix, D. C., Rangapuram, S., Salinas, D., Schulz, J., Stella, L., Türkmen, A. C., and Wang, Y. GluonTS: Probabilistic Time Series Modeling in Python. *arXiv preprint arXiv:1906.05264*, 2019.

Binois, M. and Picheny, V. Gpareto: An r package for gaussian-process-based multi-objective optimization and analysis. *Journal of Statistical Software*, 89(1), 2019.

Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[1]Datasets available at https://github.com/awslabs/gluon-ts/blob/master/src/gluonts/dataset/repository/datasets.py.

[2]Datasets from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

Dong, X. and Yang, Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. *International Conference on Learning Representations (ICLR)*, 2020.

Feurer, M., Springenberg, T., and Hutter, F. Initializing Bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Januschowski, T., Arpin, D., Salinas, D., Flunkert, V., Gasthaus, J., Stella, L., and Vazquez, P. Now available in amazon sagemaker: Deepar algorithm for more accurate time series forecasting, 2018.

Klein, A. and Hutter, F. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019.

Perrone, V., Jenatton, R., Seeger, M., and Archambeau, C. Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Perrone, V., Shen, H., , Seeger, M., Archambeau, C., and Jenatton, R. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. Scalable Gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1): 43–78, 2018.

Table 1. Search spaces description for each blackbox.

tasks	hyperparameter	search space	scale
DeepAR	# layers	$[1, 5]$	linear
	# cells	$[10, 120]$	linear
	learning rate	$[10^{-4}, 0.1]$	log10
	dropout rate	$[10^{-2}, 0.5]$	log10
	context_length_ratio	$[10^{-1}, 4]$	log10
	# bathes per epoch	$[10, 10^{4}]$	log10
XGBoost	num_round	$[2, 2^{9}]$	log2
	eta	$[0, 1]$	linear
	gamma	$[2^{-20}, 2^{6}]$	log2
	min_child_weight	$[2^{-8}, 2^{6}]$	log2
	max_depth	$[2, 2^{7}]$	log2
	subsample	$[0.5, 1]$	linear
	colsample_bytree	$[0.3, 1]$	linear
	lambda	$[2^{-10}, 2^{8}]$	log2
	alpha	$[2^{-20}, 2^{8}]$	log2
FCNET	initial_lr	$\{0.001, 0.005, 0.01, 0.05, 0.1\}$	categorical
	batch_size	$\{8, 16, 32, 64\}$	categorical
	lr_schedule	{cosine, fix}	categorical
	activation layer 1	{relu, tanh}	categorical
	activation layer 2	{relu, tanh}	categorical
	size layer 1	$\{16, 32, 64, 128, 256, 512\}$	categorical
	size layer 2	$\{16, 32, 64, 128, 256, 512\}$	categorical
	dropout layer 1	$\{0.0, 0.3, 0.6\}$	categorical
	dropout layer 2	$\{0.0, 0.3, 0.6\}$	categorical
NAS	edge_1	{zeroize, skip, 1x1 conv, 3x3 conv, 3x3 avg pool}	categorical
	edge_2	{zeroize, skip, 1x1 conv, 3x3 conv, 3x3 avg pool}	categorical
	edge_3	{zeroize, skip, 1x1 conv, 3x3 conv, 3x3 avg pool}	categorical
	edge_4	{zeroize, skip, 1x1 conv, 3x3 conv, 3x3 avg pool}	categorical
	edge_5	{zeroize, skip, 1x1 conv, 3x3 conv, 3x3 avg pool}	categorical
	edge_6	{zeroize, skip, 1x1 conv, 3x3 conv, 3x3 avg pool}	categorical