Meta-learning Hyperparameter Performance Prediction with Neural Processes

Ying Wei [1] **Peilin Zhao** [2] **Junzhou Huang** [2]

Abstract

The surrogate that predicts the performance of hyperparameters has been a key component for sequential model-based hyperparameter optimization. In practical applications, a trial of a hyperparameter configuration may be so costly that a surrogate is expected to return an optimal configuration with as few trials as possible. Observing that human experts draw on their expertise in a machine learning model by trying configurations that once performed well on other datasets, we are inspired to build a trial-efficient surrogate by transferring the meta-knowledge learned from historical trials on other datasets. We propose an end-to-end surrogate named as Transfer Neural Processes (TNP) that learns a comprehensive set of meta-knowledge, including the parameters of historical surrogates, historical trials, and initial configurations for other datasets. Experiments on extensive OpenML datasets and three computer vision datasets demonstrate that the proposed algorithm achieves state-of-the-art performance in at least one order of magnitude less trials.

1. Introduction

In the pipeline of a machine learning system, model configuration poses daunting challenges: 1) how to choose the optimal algorithm among hundreds to thousands of machine learning algorithms? 2) how to configure the optimal hyperparameters after an algorithm is specified? Brute-force exploration, obviously, is prohibitively expensive and impractical. Though experts pinpoint a configuration relatively quickly, practitioners outside machine learning possibly get bogged down in the meticulous design. These challenges highlight the critical importance of automating model configuration where automated hyperparameter optimization (HPO) is investigated in this paper.

[1]Department of Computer Science, City University of Hong Kong, Hong Kong [2]Tencent AI Lab, Shenzhen, China. Correspondence to: Ying Wei <yingwei@cityu.edu.hk>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

Free from the limitations of exhaustive search (Bergstra & Bengio, 2012) being computationally expensive and model-specific methods (Keerthi et al., 2007) being too customizable to be applied in general, sequential model-based optimization (SMBO) has been the current state-of-the-art for HPO. The core of SMBO is to learn from observed hyperparameter performances a surrogate model which maps a hyperparamter configuration to the measured performance on a dataset. Sequentially, in each trial, a promising configuration is selected by optimizing the surrogate and thereupon evaluated; the new observation is incorporated to further improve the surrogate. While existing surrogate models including Gaussian Processes (GPs) (Snoek et al., 2012), parzen estimators (Bergstra et al., 2011), random forest (Hutter et al., 2011), and neural networks (Snoek et al., 2015; Springenberg et al., 2016) have shown their effectiveness provided with sufficient observations, it is imperative to return an optimal configuration in very few trials in real-world applications where a trial on huge datasets is costly.

Transferring knowledge from historical trials on other datasets, which we focus on, has been a promising approach to speed up HPO. The inspiration comes from how machine learning experts hone their skills in a model – a set of hyperparameter configurations that perform well on some explored datasets, especially those bearing striking similarity with the target dataset of interest, are likely qualified candidates for the target. Following the convention of transfer learning, the most important research question to address is *what to transfer*. Existing studies instantiate what to transfer as observations (Schilling et al., 2015; Swersky et al., 2013; Yogatama & Mann, 2014), parameters of a surrogate model (Bardenet et al., 2013; Feurer et al., 2018; Perrone et al., 2018; Wistuba et al., 2016), and initial configurations to warmstart SMBO (Feurer et al., 2015; Lindauer & Hutter, 2018; Wistuba et al., 2015). Nevertheless, we argue the necessity of transferring all of them to expedite HPO. Figure 1 shows that transferring only observations, only parameters, or only initial configurations fails to return a satisfactory surrogate close to the groundtruth response function.

Unfortunately, simultaneously transferring all these knowledge is non-trivial for GP-based surrogate models. For example, it is particularly challenging to leverage the parameters of previous multitask GP surrogates (Swersky et al., 2013) where only observations are transferred. Even con-

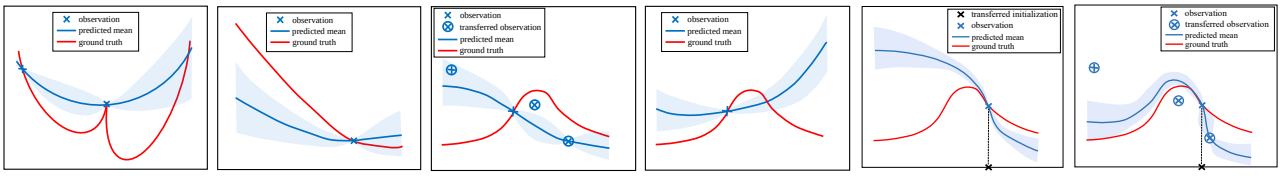


Figure 1. Illustration of the necessity of simultaneously transferring observations, parameters, and initial configurations for the surrogate: (a) historical dataset 1, (b) historical dataset 2, (c) transferring observations only, (d) transferring parameters only, (e) transferring initial configurations only, and f) transferring the joint of observations, parameters, and initial configurations.

sidering observations to be transferred only, GP surrogates with cubic scaling are highly inefficient and even impractical to incorporate abundant past observations, say 10 datasets each of which has 100 observations. Meanwhile, Neural Processes (NPs) (Garnelo et al., 2018a), by combining the best of both GP and neural networks, is known to be efficiently trained with backpropagation and preserve the property reminiscent of GP, i.e., defining distributions over functions. Motivated by its recent success, for the first time, we propose a novel end-to-end hyperparameter optimization algorithm with NPs as the surrogate, called Transfer Neural Processes (TNP). TNP sequentially consists of an encoder learning the representation of each observation, a dataset-aware attention unit which attentively aggregates representations of all observations to infer the latent distribution of hyperparameter performances, and a decoder which predicts the performances for target hyperparameter configurations with uncertainties. More remarkably, empowered by the flexibility and simplicity of NPs, TNP is trained with a meta-learning strategy so that an ensemble of knowledge is gracefully transferred from all historical datasets, including observations via the attention unit, parameter initializations for TNP, and a well-generalized set of initial configurations to warm-start SMBO.

We summarize our contributions as the following. First, we introduce NPs to serve as the surrogate model for HPO. Second, thanks to the simplicity and flexibility of NPs, we propose a hyperparameter meta-learning algorithm to achieve the joint transfer of observations, parameters, and initial configurations. Finally, through comprehensive experiments on three tasks and more than 100 datasets, we consolidate the effectiveness and efficiency of TNP.

2. Related Work

One influential line of research to accelerate HPO is to leverage knowledge from historical trials on other datasets that are similar to the target dataset of interest. To measure the similarity between the target and previous datasets, the majority have resorted to manually defined meta-features of a dataset. Feurer et al. (Feurer et al., 2015) proposed to initialize a hyperparameter search with the best configurations from similar datasets. Similarly, observations from k nearest neighbour datasets in the meta-feature space are incorporated to train the surrogate model together with those in the target (Schilling et al., 2015; Yogatama & Mann, 2014). Assuming a globally shared GP model, Bardenet et al. (Bardenet et al., 2013) optimized the model with observations from all datasets. Each observation is described as the concatenation of hyperparameters and meta-features. The downside of these methods comes with the challenge of meta-features, i.e., being hand-crafted and loosely correlated to the behaviors of hyperparameter performance.

There have been several attempts towards eliminating meta-features. For example, Wistuba et al. (2015) adopted a meta-loss to learn a set of initial configurations from past observations to maximize the performance at the very beginning of SMBO. In (Swersky et al., 2013), a multitask GP borrows observations of similar datasets where the similarity as a kernel is learned. Besides, multiple GP experts each of which is trained on a previous dataset are combined to be the surrogate for the target dataset, where the ensemble weight is learned as the generalization error of each expert on the target (Feurer et al., 2018; Wistuba et al., 2016). Perrone et al. (2018) conducted Bayesian linear regression with a feature map learned by a neural network. The shared feature map is believed to improve knowledge generalization across datasets. Unfortunately, all these works require a kernel to be explicitly defined, which gives rise to either poor scaling for GP-based approaches (Feurer et al., 2018; Swersky et al., 2013; Wistuba et al., 2015; 2016) or unfeasible algorithm deployment using standard deep learning libraries for linear kernel (Perrone et al., 2018). A more recent work (Law et al., 2019) considers not only the similarity in hyperparameter performances, but also the similarity in training data distributions. Unfortunately, the feature representation network requires all datasets to be in the same dimensionality, which limits its practical use. What is more, unlike ours, these works assuming the kernel as prior to be globally shared across datasets fail to accommodate heterogeneous datasets.

There is another line of works transferring the acquisition function (Volpp et al., 2020; Wistuba et al., 2018), while our work focuses on speeding up the learning of an accurate

surrogate model via knowledge transfer. The improved surrogate is compatible with any acquisition function, and we will investigate the collective power of the improved surrogate and an advanced acquisition function in the future.

3. Background and Problem Setup

Given a probability distribution $\mathcal{P}_D$ and a dataset $\mathcal{D}$ in which all examples are sampled from $\mathcal{P}_D$, *hyperparameter optimization (HPO)* aims to identify optimal values for hyperparameters $\mathbf{x}$ so that the generalization metric is maximized (e.g., accuracy) or minimized, i.e.,

$$\mathbf{x}^* = \arg\max_{\mathbf{x}\in\mathcal{X}} \mathbb{E}_{\mathbf{d}\sim\mathcal{P}_D}[\mathcal{L}(\mathbf{d}, A_{\mathbf{x}}(\mathcal{D}))] = \arg\max_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}), \tag{1}$$

where $\mathbf{d}$ is a sample drawn from $\mathcal{P}_D$, and $A_{\mathbf{x}}(\mathcal{D})$ represents the model produced by training an algorithm A equipped with hyperparameters $\mathbf{x}$ on the dataset $\mathcal{D}$. The hyperparameter space $\mathcal{X}$ could be continuous or discrete. Considering the difficulty of evaluating the expectation over an unknown distribution $\mathcal{P}_D$ and optimizing it, a *hyperparameter response function* f $w.r.t.$ the hyperparameters $\mathbf{x}$ is maximized instead. HPO, in this case, is equivalent to maximizing the black-box function f over $\mathcal{X}$, as there is no knowledge of the response function f and the search space $\mathcal{X}$.

Sequential Model-based Bayesian Optimizaion (SMBO) (Jones et al., 1998) has been a dominant framework for global optimization of black-box functions. SMBO consists of two components, i.e., a surrogate model Φ to approximate the response function and an acquisition function a to determine the next hyperparameter configuration to evaluate. Provided with n_I initial configurations $\mathbf{x}_{I1},\cdots,\mathbf{x}_{In_I}$, SMBO starts by querying the function f at these configurations to constitute the initial set of history observations $\mathcal{H}_0 = \{(\mathbf{x}_{I1}, y_{I1}),\cdots,(\mathbf{x}_{In_I}, y_{In_I})\}$. Afterwards, it iterates the following four stages: 1) in the t-th iteration (trial), fit the surrogate Φ_t on the observations $\mathcal{H}_t$; 2) use the surrogate Φ_t to make predictions $\{\hat{\mu}_j\}_{j=1}^{n_\mathcal{X}}$ with uncertainties $\{\hat{\sigma}_j\}_{j=1}^{n_\mathcal{X}}$ for $n_\mathcal{X}$ target configurations $\{\hat{\mathbf{x}}_j\}_{j=1}^{n_\mathcal{X}}$; 3) based on the predictions and uncertainties, the acquisition function a decides the next configuration $\mathbf{x}_t \in \{\hat{\mathbf{x}}_j\}_{j=1}^{n_\mathcal{X}}$ to try; 4) evaluate the function f at $\mathbf{x}_t$, and update the history set $\mathcal{H}_{t+1} = \mathcal{H}_t \cup \{(\mathbf{x}_t, y_t)\}$.

In the t-th iteration, there are $n_I + t$ observations in the history set $\mathcal{H}_t$. In this paper, additionally, we leverage knowledge from M history sets, i.e., $\mathcal{H}^1_{T^1},\cdots,\mathcal{H}^M_{T^M}$, of HPO on M datasets, i.e., $\mathcal{D}^1,\cdots,\mathcal{D}^M$. In the m-th dataset $\mathcal{D}^m$, there are T^m observations available in the history set $\mathcal{H}^m_{T^m} = \{(\mathbf{x}^m_t, y^m_t)\}_{t=1}^{T^m}$. The goal of this paper lies that by borrowing strength from these M history sets on M datasets, the surrogate model can be quickly maximized (equivalent to maximizing the response function f) with the optimal hyperparameter configuration returned in less trials.

4. Transferable Neural Processes

In this section, we will detail the Transferable Neural Processes (TNP) as the surrogate Φ_t. We start by illustrating the neural process model and how we fit the model on the current observation set $\mathcal{H}_t$ without transferring from HPO trials on previous datasets. Next, we highlight how the TNP meta-learns the knowledge, including parameters for the surrogate, observations, and the initial set of configurations, from other datasets to accelerate maximizing the surrogate.

4.1. The Neural Process Model

Neural Processes (NPs) (Garnelo et al., 2018a;b; Kim et al., 2019), as an alternative to GPs, approaches regression by learning a distribution over functions that map inputs to outputs As a result, NPs can support the predictions with uncertainties and rapidly adapt to a newly incorporated observation in SMBO. Meanwhile, NPs enjoys the desirable advantages of scalability with linear scaling.

The neural process model here, based on NPs (Garnelo et al., 2018a;b; Kim et al., 2019), involves three components. The encoder, shown in Figure 2, learns an embedding $\mathbf{r}_{t'} \in \mathbb{R}^r$ for each observation $(\mathbf{x}_{t'}, y_{t'})$, i.e., $\mathbf{r}_{t'} = E_{\theta_e}(\mathbf{x}'_t, y'_t)$, $\forall t' \in \{1,\cdots,n_I + t\}$. Note that the encoder E_{θ_e} is parameterized with a neural network. The dataset-aware attention unit as the second component summarizes all observations and produces an order-invariant representation of historical observations. Mathematically, $\mathbf{r}_* = A_{\theta_a}(\mathbf{r}_1,\cdots,\mathbf{r}_{n_I+t};\hat{\mathbf{x}}_j)$. This representation, $\mathbf{r}_* \in \mathbb{R}^r$, is expected to encode the latent distribution of hyperparameter performances conditioned on the whole set of observations $\mathcal{H}_t$. We will detail this unit A_{θ_a} with an attention scheme later in Section 4.2. Last but not the least, the decoder takes the representation $\mathbf{r}_*$ as well as a target configuration $\hat{\mathbf{x}}_j$ as input, and outputs $\hat{\mathbf{y}}_j \in \mathbb{R}^2$ as predictions on values of f, i.e., $\hat{\mathbf{y}}_j = D_{\theta_d}(\mathbf{r}_*, \hat{\mathbf{x}}_j)$. The two values of $\hat{\mathbf{y}}_j$ represent the mean $\hat{\mu}_j$ and variance $\hat{\sigma}_j$ of a Gaussian distribution $\mathcal{N}(\hat{\mu}_j, \hat{\sigma}_j)$, respectively. We also parameterize the decoder D_{θ_d} with a neural network. More architectural details of the neural process model can be found in Appendix A.1.

We denote the neural process model as $\text{TNP}_\theta = E_{\theta_e} \circ A_{\theta_a} \circ D_{\theta_d}$, where $\theta = \theta_e \cup \theta_a \cup \theta_d$. Drawing inspiration from (Garnelo et al., 2018b), we train the parameters of the neural process model, i.e., θ, by following three steps: 1) randomly shuffle observations in $\mathcal{H}_t$ and divide them into two parts, e.g., $\mathcal{H}_{t,h} = \{(\mathbf{x}_{t'}, y_{t'})\}_{t'=1}^{t_h}$ and $\mathcal{H}_{t,\bar{h}} = \{(\mathbf{x}_{t'}, y_{t'})\}_{t'=t_h+1}^{n_I+t}$; 2) predict the observations $\mathcal{H}_{t,h}$ conditioned on $\mathcal{H}_{t,\bar{h}}$; 3) maximize the conditional log likelihood,

$$\mathcal{L}(\mathcal{H}_{t,h}|\mathcal{H}_{t,\bar{h}},\theta) = \mathbb{E}_{f\sim P}[\mathbb{E}_{t_h}[\log p_\theta(\{y_{t'}\}_{t'=1}^{t_h}|\mathcal{H}_{t,\bar{h}}, \{\mathbf{x}_{t'}\}_{t'=1}^{t_h})]], \tag{2}$$

where the gradient of the loss is empirically estimated by

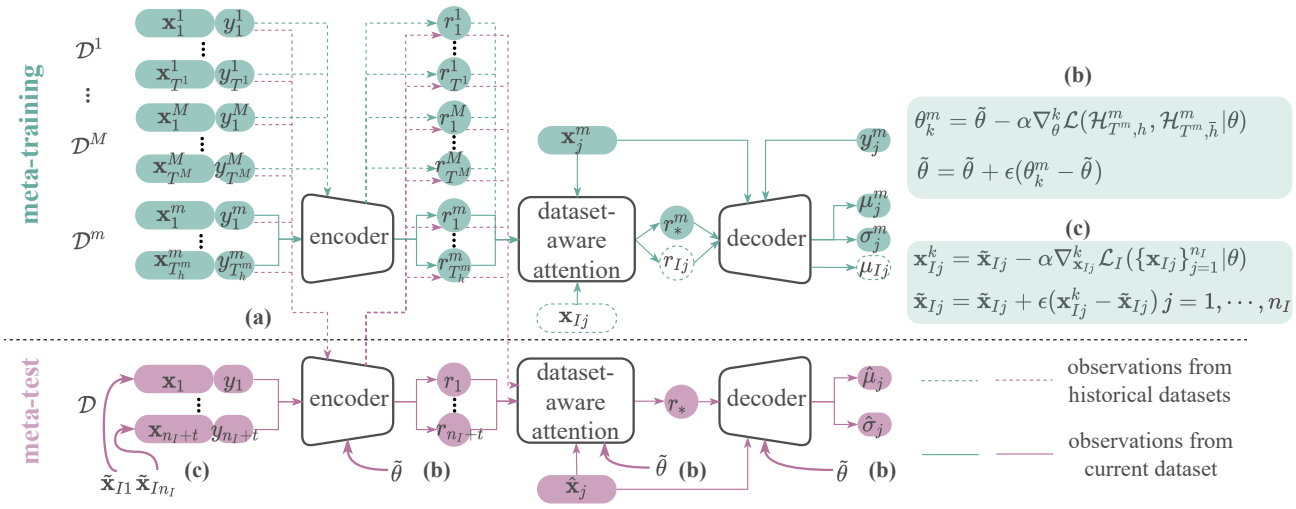


Figure 2. The **Transferable Neural Processes** consists of two stages. In the meta-training stage (colored as green), HPO trials on M historical datasets are leveraged to **(b)** learn the transferable initializations for parameters of TNP (i.e., $\tilde{\theta}$) and **(c)** optimize the well-generalized initial configurations for SMBO (i.e., $\{\tilde{\mathbf{x}}_{I1},\cdots,\tilde{\mathbf{x}}_{In_I}\}$). During the meta-test stage (colored as purple), besides drawing on the initializations for TNP and the initial configurations for SMBO learned in meta-training, TNP also **(a)** takes all historical observations from M datasets into consideration. Remarkably, TNP further fine-tunes the parameters initialized with $\tilde{\theta}$ by training on the current observation set $\mathcal{H}_t$, which allows the prior to be quickly tailored for the target dataset of interest.

sampling f and sampling different values of t_h.

4.2. Knowledge Transfer

Dataset-aware attention for leveraging observations The crux of GPs lies in modelling the similarity between a target configuration and historical observations – if a target configuration $\hat{\mathbf{x}}_j$ is close to the configuration of the t'-th observation $\mathbf{x}_{t'}$, its prediction is expected to be close to $y_{t'}$. To leverage observations from other datasets, we have to accommodate another desiderata: discrimination between datasets. Even if a target configuration $\hat{\mathbf{x}}_j$ stays close to $\mathbf{x}^m_{t'}$, it is likely that the t'-th observation from the m-th dataset contributes little if the m-th and the target dataset are wildly different. Based on (Kim et al., 2019) modelling the similarity between in-dataset configurations with the multihead attention mechanism (Vaswani et al., 2017), we design our dataset-aware attention unit A_{θ_a} as,

$$\begin{aligned}\mathbf{r}_* &= A_{\theta_a}(\mathbf{r}_{t_h+1},\cdots,\mathbf{r}_{n_I+t},\mathbf{r}^1_1,\cdots,\mathbf{r}^M_{T^M};\hat{\mathbf{x}}_j),\\ &=\textbf{MultiHead}(g(\hat{\mathbf{x}}_j),g(\mathbf{X}^{0:M}),\mathbf{R}^{0:M},\mathbf{s}),\\ &=[\textbf{head}_d(g(\hat{\mathbf{x}}_j),g(\mathbf{X}^{0:M}),\mathbf{R}^{0:M},\mathbf{s})]^{\lfloor r/d\rfloor}_{d=1},\end{aligned}$$

where $\hat{\mathbf{x}}_j \in \mathcal{H}_{t,h}$ when we train the parameters θ of the neural process model by conditioning on $\mathcal{H}_{t,\bar{h}}$ to predict the observations $\mathcal{H}_{t,h}$. $g(\hat{\mathbf{x}}_j)\in\mathbb{R}^r$ is the query, and $g(\mathbf{X}^{0:M})$ serves as the keys with $\mathbf{X}^{0:M}=[\mathbf{X};\mathbf{X}^1;\cdots;\mathbf{X}^M]$ including both in-dataset observations $\mathbf{X}=\{\mathbf{x}_{t'}\}^{n_I+t}_{t'=t_h+1}$ and cross-dataset ones $\mathbf{X}^m=\{\mathbf{x}^m_{t'}\}^{T^m}_{t'=1}$ $(\forall m=1,\cdots,M)$. $\mathbf{R}^{0:M}=[\mathbf{R};\mathbf{R}^1;\cdots;\mathbf{R}^M]$ with $\mathbf{R}=\{\mathbf{r}_{t'}\}^{n_I+t}_{t'=t_h+1}$ and $\mathbf{R}^m=\{\mathbf{r}^m_{t'}\}^{T^m}_{t'=1}$ $(\forall m=1,\cdots,M)$ provides the values to be attentively aggregated. Each head of the multihead attention follows $\textbf{head}_d=\text{softmax}\big(\mathbf{s}\circ[g(\hat{\mathbf{x}}_j)]^T\mathbf{W}^q_d[\mathbf{W}^k_d]^Tg(\mathbf{X}^{0:M})/\sqrt{r}\big)\mathbf{R}^{0:M}\mathbf{W}^v_d$, where $\mathbf{W}^q_d$, $\mathbf{W}^k_d$, $\mathbf{W}^v_d\in\mathbb{R}^{r\times d}$ are parameters. The final values $\mathbf{r}_*$ are produced by concatenating all $\lfloor r/d\rfloor$ heads.

We especially highlight $\mathbf{s}=\text{softmax}([\mathbf{1}^{(1\times(n_I+t-t_h))},s^1\mathbf{1}^{(1\times T^1)},\cdots,s^M\mathbf{1}^{(1\times T^M)}])$ which measures the similarity between the target and all datasets. Note that $\mathbf{1}^{1\times T^m}$ denotes a row vector of all ones in length T^m. The similarity is estimated as $s^m=\frac{1}{n_I+t}\sum_{t'}\cos(\mathbf{r}_{t'},\frac{1}{Q}\sum^Q_{t''}\mathbf{r}^m_{t''})$ where we condition on the mean embedding of the Q nearest observations from the m-th dataset to each configuration $\mathbf{x}_{t'}$. $\cos(\cdot,\cdot)$ refers to the cosine similarity between vectors. For more details about the dataset-aware attention and the similarity between datasets, please kindly refer to Appendix A.2. Apart from liberating practitioners from manually defining meta-features of a dataset, the mean embedding is more descriptive and pertinent to the HPO behaviours.

Note that when we finally make predictions for target configurations $\{\hat{\mathbf{x}}_j\}^{n_{\mathcal{X}}}_{j=1}$, we have

$$\begin{aligned}\mathbf{r}_*&=A_{\theta_a}(\mathbf{r}_1,\cdots,\mathbf{r}_{n_I+t},\mathbf{r}^1_1,\cdots,\mathbf{r}^M_{T^M};\hat{\mathbf{x}}_j)\\ &=\textbf{MultiHead}(g(\hat{\mathbf{x}}_j),g(\mathbf{X}^{0:M}),\mathbf{R}^{0:M},\mathbf{s}),\end{aligned}$$

where in this case the keys $\mathbf{X}^{0:M}=[\mathbf{X};\mathbf{X}^1;\cdots;\mathbf{X}^M]$ include both in-dataset observations $\mathbf{X}=\{\mathbf{x}_{t'}\}^{n_I+t}_{t'=1}$ and cross-dataset ones $\mathbf{X}^m=\{\mathbf{x}^m_{t'}\}^{T^m}_{t'=1}$, and $\mathbf{R}^{0:M}=$

Algorithm 1 Transferable Neural Processes (TNP) for Hyperparameter Optimization

Input: Observations on M datasets $\mathcal{H}^1_{T^1}, \cdots, \mathcal{H}^M_{T^M}$; # of trials T; acquisition function a; target configurations $\{\hat{\mathbf{x}}_j\}_{j=1}^{n_{\mathcal{X}}}$; fine-tuning rate α; meta update rate ϵ; # of initial configurations n_I.
Output: The best hyperparameter configuration $\mathbf{x}^*$.
Randomly initialize $\tilde{\theta}$, $\{\tilde{\mathbf{x}}_{Ij}\}_{j=1}^{n_I}$, and set $y^* = 0$
for $m = 1$ **to** M **do**
  Perform k gradient steps on :
    $\theta_k^m = \tilde{\theta} - \alpha\nabla_\theta^k \mathcal{L}(\mathcal{H}^m_{T^m,h}|\mathcal{H}^m_{T^m,\bar{h}}, \theta)$
    $\mathbf{x}_{Ij}^k = \tilde{\mathbf{x}}_{Ij} - \alpha\nabla_{\mathbf{x}_{Ij}}^k \mathcal{L}_I(\{\mathbf{x}_{Ij}\}_{j=1}^{n_I}|\theta)$,
    $\forall j = 1, \cdots, n_I$
  Update $\tilde{\theta}$ and $\{\tilde{\mathbf{x}}_{Ij}\}_{j=1}^{n_I}$:
    $\tilde{\theta} = \tilde{\theta} + \epsilon(\theta_k^m - \tilde{\theta})$, $\tilde{\mathbf{x}}_{Ij} = \tilde{\mathbf{x}}_{Ij} + \epsilon(\mathbf{x}_{Ij}^k - \tilde{\mathbf{x}}_{Ij})$
end for
Query the values of f at $\{\tilde{\mathbf{x}}_{Ij}\}_{j=1}^{n_I}$, and obtain the initial observation set $\mathcal{H}_0 = \{(\tilde{\mathbf{x}}_{Ij}, \tilde{y}_{Ij})\}_{j=1}^{n_I}$.
for $t = 1$ **to** T **do**
  Fine-tune TNP by k gradient steps:
    $\theta_k = \tilde{\theta} - \alpha\nabla_\theta^k \mathcal{L}(\mathcal{H}_{t-1,h}|\mathcal{H}_{t-1,\bar{h}}, \theta)$
  Fit TNP_{θ_k} to $\mathcal{H}_{t-1}$:
    $\mathbf{x}_t \leftarrow \arg\max_{\mathbf{x}\in\{\hat{\mathbf{x}}_j\}_{j=1}^{n_{\mathcal{X}}}} a(\text{TNP}_{\theta_k}(\mathbf{x}))$
  Evaluate $y_t = f(\mathbf{x}_t)$ and update the observation set $\mathcal{H}_t = \mathcal{H}_{t-1} \cup \{(\mathbf{x}_t, y_t)\}$
  if $y_t > y^*$ **then**
    $\mathbf{x}^*, y^* \leftarrow \mathbf{x}_t, y_t$
  end if
end for
return $\mathbf{x}^*$

$[\mathbf{R}; \mathbf{R}^1; \cdots; \mathbf{R}^M]$ with $\mathbf{R} = \{\mathbf{r}_{t'}\}_{t'=1}^{n_I+t}$ and $\mathbf{R}^m = \{\mathbf{r}_{t'}^m\}_{t'=1}^{T^m}$ ($\forall m = 1, \cdots, M$). The similarity accordingly takes $\mathbf{s} = \text{softmax}([\mathbf{1}^{(1\times(n_I+t))}, s^1\mathbf{1}^{(1\times T^1)}, \cdots, s^M\mathbf{1}^{(1\times T^M)}])$.

To conclude, the proposed dataset-aware attention unit allows a target configuration to attend those similar observed configurations of related datasets. Though this dataset-aware attention raises the time complexity to $\mathcal{O}(n_{\mathcal{X}}(n_I + t + \sum_{m=1}^M T^m))$, the training can be approximately linear by conducting the attention in parallel. Moreover, TNP searches hyperparameters significantly faster in terms of SMBO iterations, i.e., a small value of t.

Transferring parameters As mentioned above, analogous to GPs, TNP with dataset-aware attention learns an implicit kernel characterizing the similarity between configurations of historical observations and target configurations. While GPs requires an analytic kernel, e.g., Matérn-5/2, to be specified by experts, the kernel of TNP is data-driven. Under the assumption that different response functions f sampled to optimize Eqn. (2) by sampling different datasets are from the same underlying distribution, the kernel is globally shared and progressively improved as training proceeds. Unfortunately, this assumption runs counter to practical scenarios where a global kernel cannot accommodate a wide range of datasets. The hierarchical Bayesian model is qualified to alleviate the problem: there is a global kernel on which each dataset-specific kernel is statistically dependent. Without loss of scalability and end-to-end training of neural networks, we follow the strategy of model agnostic meta-learning (Finn et al., 2017) which has been proved its equivalence to hierarchical Bayesian inference (Grant et al., 2018). Given the transferable parameters $\tilde{\theta}$ for TNP characterizing the global kernel, the dataset-specific parameters θ_k^m dictating the customized kernel for the m-th dataset are further optimized (fine-tuned) in k gradient steps, i.e.,

$$\theta_k^m = \tilde{\theta} - \alpha\nabla_\theta^k \mathcal{L}(\mathcal{H}^m_{T^m,h}|\mathcal{H}^m_{T^m,\bar{h}}, \theta),$$

where α is the learning rate for fine-tuning. The transferable parameters $\tilde{\theta}$ is afterwards updated by

$$\tilde{\theta} = \tilde{\theta} + \epsilon(\theta_k^m - \tilde{\theta}),$$

where ϵ denotes the meta-learning rate. As shown in the meta-training stage of Figure 2, each time we sample the m-th dataset as the target and the rest of M datasets as historical datasets. First, initialized with $\tilde{\theta}$, TNP optimizes $\mathcal{L}(\mathcal{H}^m_{T^m,h}|\mathcal{H}^m_{T^m,\bar{h}}, \theta)$ in k gradient steps, where we follow Section 4.1 by dividing the observation set $\mathcal{H}^m_{T^m}$ into two parts. In turn, θ_k^m updates the transferable initialization $\tilde{\theta}$. During meta-testing, it is straightforward to first fine-tune TNP on $\mathcal{H}_t$, i.e., $\theta_k = \tilde{\theta} - \alpha\nabla_\theta^k \mathcal{L}(\mathcal{H}_{t,h}|\mathcal{H}_{t,\bar{h}}, \theta)$, and then make predictions for the j-th target configuration $\hat{\mathbf{x}}_j$ using TNP_{θ_k}, namely the TNP equipped with the parameters θ_k.

Initializing SMBO with well-generalized configurations The initial configurations have been demonstrated crucial to the success of SMBO (Lindauer & Hutter, 2018; Wistuba et al., 2015) – those configurations which achieve larger values of f (here we discuss the maximization of f in Eqn. (1)) are prone to speed up the SMBO. Fortunately, we are provided with M observation sets $\mathcal{H}^1_{T^1}, \cdots, \mathcal{H}^M_{T^M}$ which offer a treasure of the configurations with higher f values. Therefore, we again formulate the problem of learning initial configurations as a hierarchical Bayesian inference problem. Similar to inferring $\tilde{\theta}$ in Eqn. (3), we learn the set of well-generalized initial configurations $\{\tilde{\mathbf{x}}_{Ij}\}_{j=1}^{n_I}$ which are fine-tuned for each m-th dataset. The only difference is the loss with regard to $\{\tilde{\mathbf{x}}_{Ij}\}_{j=1}^{n_I}$, which enforces the predictions and the uncertainties of at least one of the initial configurations are maximized, i.e., $\mathcal{L}_I(\{\mathbf{x}_{Ij}\}_{j=1}^{n_I}|\theta) = \sum_{j=1}^{n_I} \frac{e^{\alpha\mu_{Ij}}}{\sum_{j'}^{n_I} e^{\alpha\mu_{Ij'}}}\mu_{Ij} + \frac{e^{\alpha\sigma_{Ij}}}{\sum_{j'}^{n_I} e^{\alpha\sigma_{Ij'}}}\sigma_{Ij}$. The reason why we impose the softmax with $\alpha > 0$ is to ensure the diversity of the initial configurations, so that TNP as the surrogate

benefits from those initial configurations with either large predictions to exploit or high uncertainties to explore. The overall learning algorithm is presented in Algorithm 1.

5. Experiments

5.1. Experimental Setup

Datasets First of all, we consider the OpenML (Vanschoren et al., 2014) platform which contains a large number of datasets covering a wide range of applications. The OpenML datasets are heterogeneous in both feature and label spaces. For example, the meta-datasets (wine and oh5.wc) in Figure 5c have 13 and 3,012 numeric features, while the target dataset (kr-vs-kp) has 36 nominial features. Wine and oh5.wc have 3 and 10 classes, respectively, while kr-vs-kp is a binary classification task. Due to time constraint, we select 100 supervised classification datasets that have fewer than 100,000 instances and no missing values. The training, validation, and test sets of each dataset are exactly the same as OpenML provides. The hyperparameters are optimized on validation sets, while we compare different HPO methods by reporting the performance of the best configuration returned on test sets. For comparison with those baselines using meta-features to measure the similarity between datasets, we extract a list of meta-features for each dataset following Table 1 in (Wistuba et al., 2015). We improve the classification accuracy of Logistic Regression (LR) and fully connected networks (FCN) on all OpenML datasets. For LR, the dimension of the hyperparameter space is four, including the learning rate $\eta \in [10^{-6}, 10^{0}]$ for SGD, the l2-regularization coefficient $r_2 \in [0, 1]$, the batch size $B \in [20, 2000]$, and the dropout ratio $\gamma \in [0, 0.75]$. For FCN, a total of ten hyperparameters for FCN are tuned, including the learning rate $\eta \in [10^{-6}, 10^{0}]$, the $\ell 2$ regularization strength $\beta \in [10^{-8}, 10^{-1}]$, the batch size $B \in [32, 512]$, the two parameters gamma $\gamma \in [10^{-3}, 10^{-1}]$ and power $\epsilon \in [0, 1]$ for decaying the learning rate, the momentum $m \in [0.3, 0.999]$ for optimization, the number of hidden units for the two layers $d_1, d_2 \in [2^5, 2^{12}]$, and the dropout rates for the two layers $p_1, p_2 \in [0, 0.99]$.

Besides OpenML, we also investigate the effectiveness of TNP on three popular computer vision datasets, including CIFAR-10 (Krizhevsky & Hinton, 2009), MNIST (LeCun et al., 1995), and SVHN (Netzer et al., 2011). We take the last 10,000, 10,000, and 6,000 training instances as the validation set for CIFAR-10, MNIST, and SVHN, respectively. Each of them is also described with meta-features. Here we focus on a three-layer convolutional neural network in which each layer consists of a convolution with batch normalization and ReLU activation functions followed by max pooling. All convolutions have the filter size of 5×5. We tune five hyperparameters including the learning rate $\eta \in [10^{-6}, 10^{0}]$ for Adam, the batch size $B \in [32, 512]$, and the number of hidden units for the three layers, $d_1, d_2, d_3 \in [2^4, 2^8]$, respectively. Note that we do not perform data augmentation in this experiment.

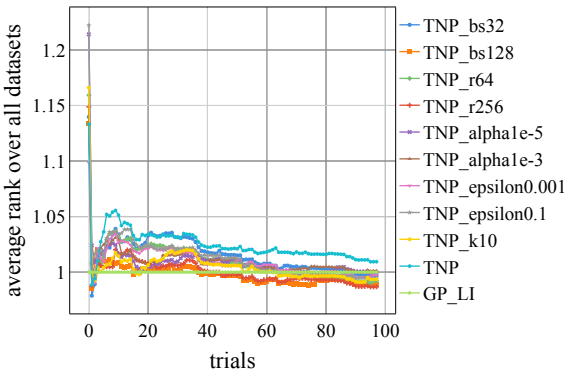
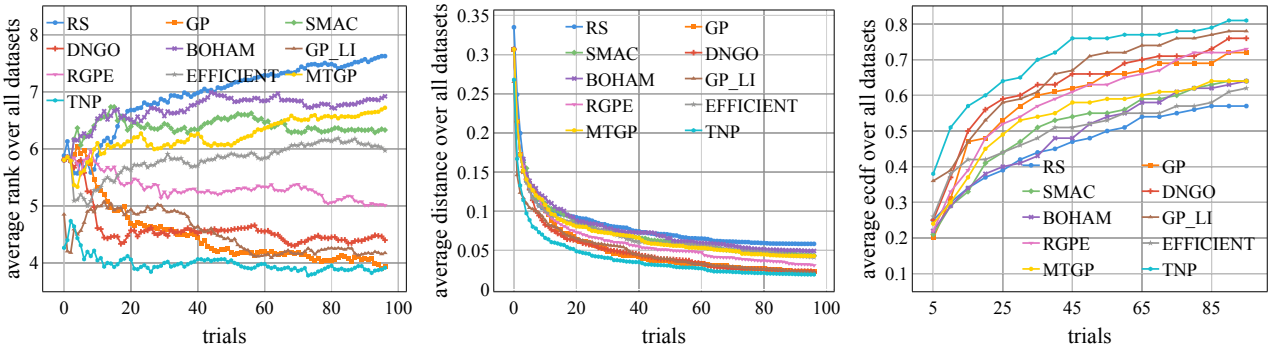


Figure 3. Comparing the average improvement of all TNPs over GP_LI with different hyperparameters.

Baselines We consider 9 baseline methods. All methods including ours are based on the SMBO framework, and use the expected improvement (EI) as the acquisition function. The baselines fall into four groups based on the surrogate model and whether knowledge is leveraged from other datasets. 1) *No surrogates*: random search (**RS**) (Bergstra & Bengio, 2012); 2) *Surrogates without neural networks*: Gaussian Processes with a Matérn-5/2 kernel (**GP**) (Snoek et al., 2012) and random forests (**SMAC**) (Hutter et al., 2011); 3) *Surrogates with neural networks*: **DNGO** (Snoek et al., 2015) and **BOHAM** (Springenberg et al., 2016); 4) *Surrogates with knowledge transfer*: multitask GPs (**MTGP**) (Swersky et al., 2013) and **EFFICIENT** (Yogatama & Mann, 2014) that transfer observations without and with meta-features, respectively, ranking-weighted Gaussian Process ensemble (**RGPE**) (Feurer et al., 2018) that transfers parameters from past GPs, and **GP_LI** (Wistuba et al., 2015) that leverages past observations to learn initial configurations. Hyperparameter settings of the baselines can be found in Appendix B.1.

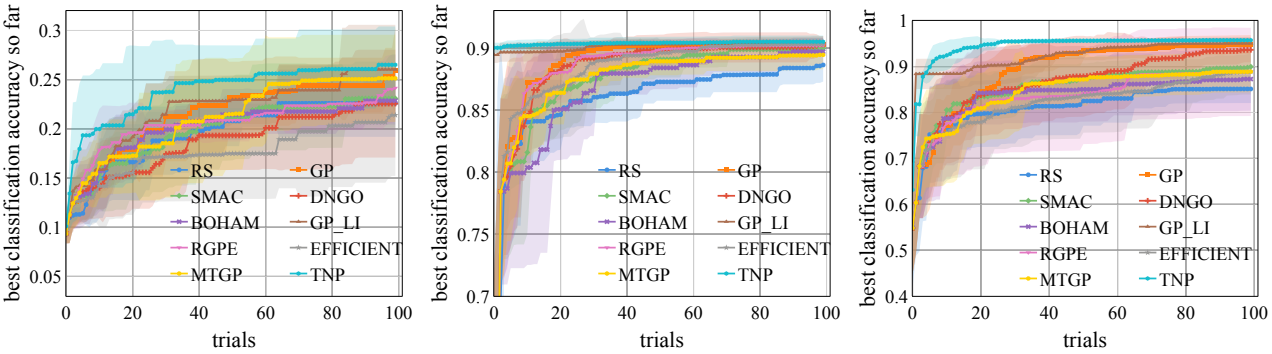
Evaluation Metrics We compare in terms of the maximum classification accuracy achieved so far, the average rank over all datasets indicating the rank of a method, the scaled average distance to the maximum, and the empirical cumulative distribution function (ECDF) of the number of datasets that have achieved the maximum in a trial. In Appendix B.2, we give details of the last three metrics.

Network Setup The encoder, the decoder, and the attention embedding function g are all implemented as a two-layer multilayer perceptron with $r = 128$ hidden units. Following (Garnelo et al., 2018a), we first pre-train the network by sampling $30,000$ batches of GP functions with length



(a) Average rank (b) Average distance to the max (c) ECDF of # of datasets to the max

Figure 4. Comparison of the performances while optimizing the hyperparameters of LR on 100 OpenML datasets.



(a) vineyard + molecular-biology promoters → mfeat-morphological (b) new3s.wc + monks-problems-1 → nursery (c) wine + oh5.wc → kr-vs-kp

Figure 5. Comparison of the maximum accuracies achieved on three randomly selected datasets.

scale $l \sim U[0.3, 1.0]$ and kernel scale $\sigma = 1.0$, where the dimension of each GP function amounts to that of the hyperparameter space. We set the batch size, the number of gradient steps k, the learning rate α for Adam, and the meta update rate ϵ to be 64, 10, 1e-5, and 0.01, respectively. We summarize all hyperparameter settings of TNP in Appendix C.1. In Figure 3, we study the hyperparameter sensitivity. We vary the values of the hyperparameters and report the improvement ratios of TNPs with different hyperparameters over the most competitive baseline GP_LI. Despite the variance, TNPs with different hyperparameters still outperform GP_LI, especially in few trials. More results on hyperparameter sensitivity could be found in Appendix C.1.

5.2. Results on OpenML Datasets

Effectiveness of hyperparameter optimization For each OpenML dataset, we obtain 100 historical observations by running GPs to optimize the hyperparameters of LR on it in 100 trials. Taking each of the 100 datasets as a target, we first randomly sample $M = 2$ of the 99 others as historical meta-datasets for ours as well as other transfer learning baselines. Figure 4 shows the performance comparison of all methods, where TNP consistently and significantly outperforms other baselines. Note that the average rank is averaged over all 100 OpenML datasets each of which is run 10 times repeatedly. The similar applies to computing the average distance to the maximum and the ECDF. Unsurprisingly, random search without a surrogate model performs the worst. GPs proves itself almost the most robust algorithm without knowledge transfer, as long as a sufficient number of observations have been collected. Consequently, despite the superiority of some baselines at the beginning (less than 40 trials), e.g., DNGO, GPs becomes increasingly powerful. Since all transfer learning baselines fail to simultaneously transfer parameters, observations and initial configurations, they seem to be competent only at the very beginning and bring about negative transfer afterwards. All methods share the same $n_I = 3$ initial configurations, except that GP_LI and TNP learn the initial configurations from historical datasets. Though GP_LI approaches TNP within $n_I = 3$ initial configurations, afterwards, it lags behind TNP which also transfers parameters and observations. As shown in Figure 4c, in the 10-th trial, TNP has achieved the maximum accuracy on 50% of the datasets, which significantly

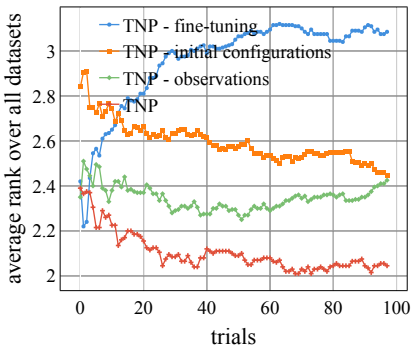


Figure 6. Removing different components

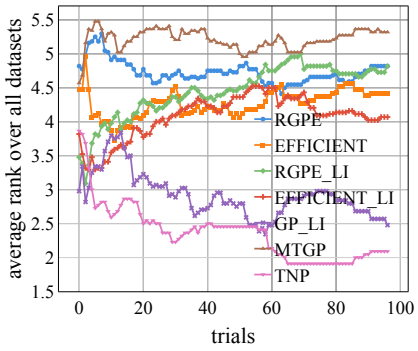


Figure 7. Comparison of transfer methods when $M = 50$

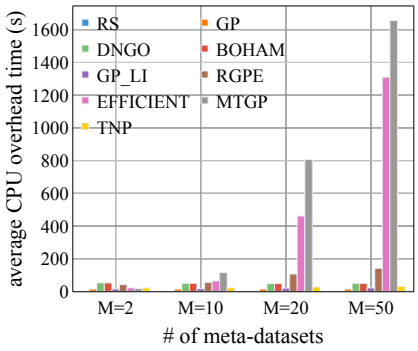


Figure 8. Comparison in CPU overhead over datasets/trials

improves the baselines by 25%. All these results consolidate the effectiveness of TNP, especially in the first few trials.

We also randomly select three datasets and compare the maximum classification accuracies achieved so far in Figure 5. The performance of baselines varies from dataset to dataset. EFFICIENT outperforms GPs in the first ten trials in Figure 5b, while it is almost the worst in Figure 5a. MTGP which learns the similarity between datasets and thereupon transfers observations is a little more robust than EFFICIENT relying on meta-features, which suggests the vulnerability of the similarity learned by meta-features. Instead, by precisely learning the similarity as defined in Section 4.2, TNP promotes more transferable knowledge between similar datasets and simultaneously alleviates negative transfer between wildly dissimilar ones. In Appendix C.7, we give a detailed ablation study to demonstrate that the learned similarity by TNP is more effective than that based on meta-features. In particular, the most similar source dataset to the target according to TNP is vineyard (0.243) for Figure 5a, new3s.wc (0.286) for Figure 5b, and oh5.wc (0.2599) for Figure 5c. TNP benefits the most in Figure 5b by leveraging new3s.wc with the greatest similarity (0.286).

While optimizing the hyperparameters of a fully connected network, as shown in Figure 9, the behaviors of the baselines are completely different from those during HPO for the logistic regression model. GP_LI, which is almost the most competitive during optimizing LR, is inferior especially in the beginning – it is likely that the predictive power of GP which learning the initial configurations depends on becomes less effective as the dimension of hyperparameters gets larger. TNP by transferring parameters and observations, however, remains highly predictive towards higher dimensional HPO and thereby learns more generalized initial configurations. We also observe that two baselines that transfer observations, including MTGP and EFFICIENT, are more qualified in this case. In conclusion, we highlight TNP's consistent superiority, even though the dimension of

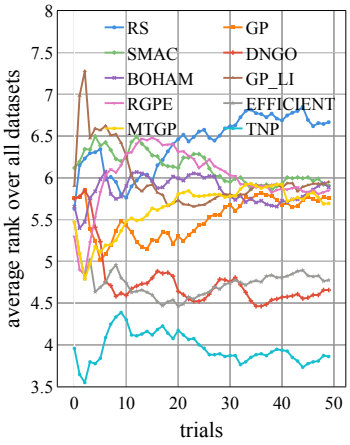
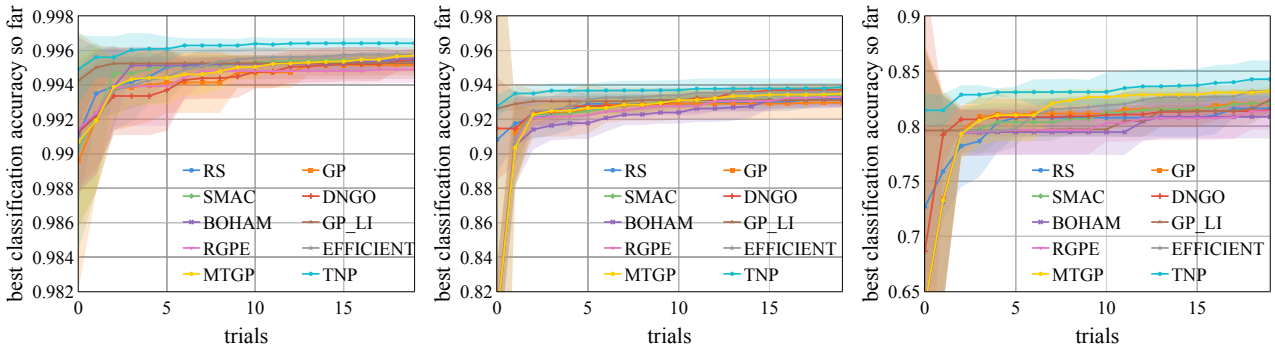


Figure 9. The average ranks of all algorithms while optimizing the hyperparameters of FCN on 100 OpenML datasets.

the hyperparameter search space increases to 10.

Ablation Studies First, we aim to study the *influence of different components* on the performance of TNP. As shown in Figure 6, the performance of TNP without fine-tuning a globally-shared kernel to each target dataset drops the most. The set of well-generalized initial configurations is also crucial to warm-start HPO. Leveraging observations from other datasets also plays an important part in the effectiveness of TNP. More empirical evidence for mutual reinforcement of these compoenents can be found in Appendix C.2. Second, we increase *the number of historical datasets*, i.e., M. Figure 7 tells that even if M increases to 50, TNP still outperforms the other transfer learning baselines. By virtue of the joint transfer of observations, parameters, and initial configurations, TNP is robust against more noisy datasets as M and the difficulty of knowledge transfer increases. Note that here we also enable RGPE and EFFICIENT to be capable of learning initial configurations, denoted as RGPE_LI and EFFICIENT_LI, and compare with them. More com-



(a) MNIST (b) SVHN (c) CIFAR-10

Figure 10. Comparison of the maximum accuracies achieved on three computer vision datasets.

parison results when $M = 10$ and $M = 20$ can be found at Appendix C.3. We also investigate *the influence of the number of initial configurations* in Appendix C.4 and *the insensitivity to random seeds* in Appendix C.5, respectively.

Efficiency In Figure 8, we compare the average CPU time overhead by different algorithms. Note that the CPU time overhead characterizes the time taken by a HPO algorithm to determine the next configuration to evaluate, and does not include the time to evaluate a selected configuration which varies significantly from configuration to configuration. Since both MTGP and EFFICIENT transfer observations and scale cubically with the number of observations, the computational costs for them become extraordinarily expensive when M increases. The two Bayesian neural network methods with MCMC sampling, i.e., DNGO and BOHAMIANN, are also very slow. Though TNP simultaneously transfers observations and parameters, it does not sacrifice the computational efficiency too much. This paves the way for wide application of TNP in the real world. Figure 8 is averaged over datasets and trials, while more results across trials are reported in Appendix C.6.

5.3. Results on Computer Vision Datasets

In light of the computational cost of each trial, here we perform HPO within only 20 trials, being more practical for real-world deployment. Regarding each of the three datasets as the target, the other two serve as $M = 2$ historical datasets. Again TNP is effective on all the three datasets by achieving competent classification accuracies even within the first 5 trials. As the most challenging dataset, CIFAR-10 benefits the most from the other two. Compared with diverse OpenML datasets, the three datasets fall in the same category of computer vision and are much more similar. As a consequence, most of transfer learning baselines, e.g., MTGP, outperform GPs, while it is not the case on OpenML datasets. Another reason is that the number of hyperparameters increases to five in this experiment, which requires more observations for GPs to succeed. The HPO results of FCN with 10 hyperparameters on OpenML datasets in Figure 9 also provide the justification.

6. Conclusion

We introduced TNP as a novel end-to-end hyperparameter optimization algorithm. Equipped with NPs as the surrogate, for the first time, TNP harnesses the collective power of observations, parameters for the surrogate, and initial configurations for SMBO from previous datasets. Besides, TNP enjoys the advantages of neural processes with high scalability, which lays the foundation for practical use in real-world applications. In the future, we are committed to conquer the challenge of comparing datasets in heterogeneous feature spaces, so that the between-dataset similarity takes into account not only the hyperparameter performances but also the dataset distribution.

Acknowledgements

This work was supported in part by the start-up grant from City University of Hong Kong (9610512). Ying Wei would also like to acknowledge the support from the Tencent AI Lab Rhino-Bird Gift Fund. We would also sincerely thank the reviewers for their constructive comments.

References

Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. Collaborative hyperparameter tuning. In *ICML*, pp. 199–207, 2013.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *JMLR*, 13(Feb):281–305, 2012.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *NeurIPS*, pp. 2546–2554, 2011.

Feurer, M., Springenberg, J. T., and Hutter, F. Initializing bayesian hyperparameter optimization via meta-learning. In *AAAI*, 2015.

Feurer, M., Letham, B., and Bakshy, E. Scalable meta-learning for bayesian optimization. *arXiv preprint arXiv:1802.02219*, 2018.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pp. 1126–1135, 2017.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. Conditional neural processes. In *ICML*, pp. 1690–1699, 2018a.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. Recasting gradient-based meta-learning as hierarchical bayes. In *ICLR*, 2018.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *LION*, pp. 507–523, 2011.

Jones, D. R., Schonlau, M., and Welch, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

Keerthi, S. S., Sindhwani, V., and Chapelle, O. An efficient method for gradient-based adaptation of hyperparameters in svm models. In *NeurIPS*, pp. 673–680, 2007.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Whye Teh, Y. Attentive neural processes. In *ICLR*, 2019.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, 2009.

Law, H. C. L., Zhao, P., Chan, L., Huang, J., and Sejdinovic, D. Hyperparameter learning via distributional transfer. In *NeurIPS*, 2019.

LeCun, Y., Jackel, L., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., Simard, P., et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

Lindauer, M. and Hutter, F. Warmstarting of model-based algorithm configuration. In *AAAI*, 2018.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Perrone, V., Jenatton, R., Seeger, M. W., and Archambeau, C. Scalable hyperparameter transfer learning. In *NeurIPS*, pp. 6846–6856, 2018.

Schilling, N., Wistuba, M., Drumond, L., and Schmidt-Thieme, L. Hyperparameter optimization with factorized multilayer perceptrons. In *ECML/PKDD*, pp. 87–103, 2015.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *NeurIPS*, pp. 2951–2959, 2012.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. Scalable bayesian optimization using deep neural networks. In *ICML*, pp. 2171–2180, 2015.

Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. Bayesian optimization with robust bayesian neural networks. In *NeurIPS*, pp. 4134–4142, 2016.

Swersky, K., Snoek, J., and Adams, R. P. Multi-task bayesian optimization. In *NeurIPS*, pp. 2004–2012, 2013.

Vanschoren, J., Van Rijn, J. N., Bischl, B., and Torgo, L. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NeurIPS*, pp. 5998–6008, 2017.

Volpp, M., Fröhlich, L. P., Fischer, K., Doerr, A., Falkner, S., Hutter, F., and Daniel, C. Meta-learning acquisition functions for transfer learning in bayesian optimization. In *ICLR*, 2020.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. Learning hyperparameter optimization initializations. In *DSAA*, pp. 1–10, 2015.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. Two-stage transfer surrogate model for automatic hyperparameter optimization. In *ECML/PKDD*, pp. 199–214, 2016.

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

Yogatama, D. and Mann, G. Efficient transfer learning method for automatic hyperparameter tuning. In *AISTATS*, pp. 1077–1085, 2014.