Adaptive Knowledge Transfer based on Transfer Neural Kernel Network

Pengfei Wei
National University of Singapore
dcsweip@nus.edu.sg

Xinghua Qu
Nanyang Technological University
XINGHUA001@e.ntu.edu.sg

Yiping Ke
Nanyang Technological University
ypke@ntu.edu.sg

Tze Yun Leong
National University of Singapore
leongty@nus.edu.sg

Yew Soon Ong
Nanyang Technological University
Singapore's Agency for Science Technology and Research
ASYSOng@ntu.edu.sg

ABSTRACT

Transfer agents are widely used in the challenging problems where knowledge is cross-used among different tasks. One popular research approach is to design a transfer kernel that controls the strength of knowledge transfer based on the similarity of tasks. In this paper, we propose a Transfer Neural Kernel Network (TNKN), which enables flexible modeling of the task similarity. The proposed TNKN is constructed by compositions of primitive kernels and represented by a neural network. Two coupled compositional kernel structures are used to characterize data covariance, one for the intra-task data covariance and another for the inter-task one. A sufficient condition that validates the transfer agent using TNKN for any data is given. This condition also discloses the relationship of the two compositional kernel structures, and can be used as a constraint in the agent learning. Since the overall architecture of TNKN is differentiable, the learning of the transfer agent using TNKN is end-to-end trainable with gradient-based optimization. Extensive experiments on various real-world datasets demonstrate the transfer effectiveness of TNKN.

KEYWORDS

Transfer agent; Transfer kernel neural network; Compositional kernel structures

ACM Reference Format:
Pengfei Wei, Xinghua Qu, Yiping Ke, Tze Yun Leong, and Yew Soon Ong. 2020. Adaptive Knowledge Transfer based on Transfer Neural Kernel Network. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

Transfer agents aim to reuse knowledge across different but related tasks, which can be used in many real-world applications, e.g., topic classification problem [29], project scheduling problem [24], sim-to-real robot modelling problem [33] and constraint satisfaction problem [23], etc. Although existing transfer agents are mainly proposed for classification problems, e.g., object identification [17] and sentiment analysis [11], and reinforcement learning problems, e.g., 3D games [21] and robotic jaco arms [14], transfer regression is attracting increasing interests due to its popularity in the real-world applications, e.g., Wi-Fi localization [30], mechanical modeling analysis [16], and aerospace systems design [20]. In this work, we deal with transfer regression problems. We focus on transfer agents based on Gaussian process (GP) modelings, highly regarded for their capability to handle prior beliefs on the characteristics of the underlying functions.

Since brute-force transfer assuming tasks are always related may cause negative transfer [22], a crucial problem for transfer agents is to adaptively control the knowledge transfer strength based on the relatedness of tasks. In GP-based transfer regression problems, this is usually done by designing a transfer kernel. Different from the conventional kernel that treats all instance pairs equally, a transfer kernel discriminates the covariance of intra-task instance pairs (the two instances are from the same task) from that of inter-task instance pairs (the two instances are from different tasks).

Typically, a transfer kernel can be constructed by using two different functions, i.e., using f and f' to calculate the covariance of the intra-task instance pairs and the inter-task ones, respectively. The form of f and f' determines the transfer capacity of the transfer kernel. For instance, a transfer kernel k_λ [3, 6, 7, 28] is developed by using any kernel as f and setting $f' = \lambda f$. This transfer kernel is simple and enables the cheap GP modelling, but only models a compromised similarity on a specific data characteristic across tasks. To improve the expressive power of transfer kernel, Wei et al. [30] propose a more flexible design, k_{mk}, by exploiting multiple kernel learning [2]. Specifically, f and f' are constructed by the linear summation of several shared base kernels, but with different construction coefficients. By allowing heterogeneous base kernel coefficients, k_{mk} can disclose similarities on the data characteristics represented by the linear summation of base kernels.

Although the design of transfer kernel permits certain flexibility, it must follow a fundamental principle. That is, a transfer kernel must be positive semi-definite (PSD) so that the resulting GP based transfer agent is applicable for any data. In some cases [8], this can be done by applying some prior problem information. However, for most real-world applications that are black-box problems, a more general strategy that defines the valid set of a transfer kernel is needed. For instance, Cao et al. [6] provide a value range of λ ($|\lambda| \leq 1$) to guarantee the positive semi-definiteness of k_λ. Wei et al. [30] propose a sufficient and necessary condition, where the

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

relationship of the coefficients are constrained, to ensure a PSD k_{mk}.

In this paper, we propose a more flexible and powerful transfer kernel, called Transfer Neural Kernel Network (TNKN). On the one hand, TNKN can approximate arbitrary stationary kernels, that is, it can express the rich and complex data characteristics. On the other hand, TNKN is capable of capturing the similarity of different tasks on these diverse characteristics in a hierarchical manner. More specifically, we develop an end-to-end neural network architecture to model the fine-grained similarity of different tasks layer by layer. The proposed TNKN consists of two coupled neural kernel network corresponding to f and f' respectively. Each of f and f' is a compositional kernel structure based on the composition rules for kernels. Note that f and f' share the same network architecture but with different network parameters. To ensure that TNKN is a PSD kernel, we propose a sufficient condition that discloses the relationship of f and f'. We theoretically prove that TNKN is always PSD as long as the network parameter of f' never goes beyond the corresponding one of f. We also take such a relationship of f and f' into account in the learning of the transfer agent using TNKN. Since the overall architecture of TNKN is differentiable, the transfer agent using TNKN is end-to-end trainable. We propose to utilize the gradient-based optimization for the learning of the transfer agent. The relationship of f and f' is used as a computationally inexpensive constraint in the learning process. In the experiments, we show the effectiveness of our proposed TNKN by comparing it with several state-of-the-art baselines on 1 time-series and 4 regression real-world datasets.

The proposed TNKN is a general transfer method as it can be applied in different models, e.g., SVM and kernel ridge regression model, etc., and different tasks, e.g., causal inference, decision making and planing, and also classification. In this work, our focus is to show the effectiveness of TNKN by adapting it to GP models. We leave more possible instantiations of TNKN on other models and tasks in future studies.

2 RELATED WORKS

The transfer performance of transfer agents highly depends on the design of transfer kernel. One popular form of transfer kernels is based on the intrinsic coregionalization model (ICM) from geostatistics [13]. This type of transfer kernels defines the covariance of functions as the product of two covariances, one for the tasks and are given by parametric similarity coefficient, and another for the input data and are represented by standard parametric kernels. Works [3, 6, 7, 28, 31] fall within the scope of this type of transfer kernels. More specifically, in the works [3, 7], the authors handle the multi-task problem, and propose a free-form similarity matrix where each element encodes the similarity of two tasks. Cao et al. [6] directly use a single parametric coefficient to model the similarity of two tasks in the single source transfer problem. Wagle et al. [28] follow the idea of the work [6], and improve the transfer kernel by further distinguishing the source-to-source covariance from the target-to-target covariance. Wei et al. [31] extend the transfer kernel of the work [6] to the multiple source transfer problem by assigning each source-target task pair a similarity coefficient. Another line of transfer kernels is based on the linear model of coregionalization (LMC) that is widely used in geostatistics [13], multi-output learning [4], and multi-class learning [1]. It exploits the multiple kernel learning [2] to enable a more flexible modeling on heterogeneous similarities. Wei et al. [30] utilize this form of transfer kernels to capture the sub-similarity of tasks on each base kernel. Another LMC based transfer kernel [26] is proposed to model the relatedness of multiple time sequence data. A compositional kernel structure is used to model the data covariance, and a non-parametric indicator matrix is used to model the task covariance. However, the indicator matrix only allows the binary values 0 and 1, which highly constrains the modeling of the task relatedness.

Another challenge in the design of a transfer kernel is to ensure its positive semi-definiteness. By using the binary values for the similarity matrix, the transfer kernel proposed in the work [26] naturally satisfies the positive semi-definiteness. In the works [3] and [7], Cholesky decomposition [15] is used to decompose the similarity matrix as the product of two low-triangular matrices. This ensures the resulting transfer kernel matrix is PSD, but it lacks of a clear-cut relationship between a given low-triangular matrix and the elements in the original similarity matrix. This hampers the control on the semantic meaning of the similarities during learning. Moreover, the problem is exacerbated by the fact that the factorization is not unique. To overcome these limitations, recent studies explore a more holistic way to ensure the positive semi-definiteness. Cao et al. [6] propose a sufficient condition that constrains the value range of the similarity coefficient between -1 to 1. Such a condition not only guarantees the positive semi-definiteness of the transfer kernel, but also enables a semantic interpretation on the learned similarity coefficient. Following the idea of the work [6], Wei et al. [31] further provide a sufficient and necessary condition for the positive semi-definiteness of the transfer kernel used in the multiple-source transfer scenario, and demonstrates a pathology. Wagle et al. [28] directly use the conclusions of [6] and [19] to construct a PSD transfer kernel. Most recently, Wei et al. [30] prove a sufficient and necessary condition for the complete valid set of the transfer kernel using multiple kernel strategy. In this paper, we are also interested in a similar holistic solution as proposed in the works [6, 30, 31] to guarantee the positive semi-definiteness of our proposed TNKN.

3 PROBLEM SETTING AND PRELIMINARIES

Problem Setting. We consider the transfer regression problem with one source task S and one target task T. Sufficient source labeled data, $\mathbf{X}^S \in \mathbb{R}^{n_S \times d}$ with $\mathbf{y}^S \in \mathbb{R}^{n_S}$, and only limited target labeled data, $\mathbf{X}^T \in \mathbb{R}^{n_T \times d}$ with $\mathbf{y}^T \in \mathbb{R}^{n_T}$, are available in the training phase. Typically, $n_T \ll n_S$. Our objective is to utilize $\mathbf{X} = [\mathbf{X}^S; \mathbf{X}^T] \in \mathbb{R}^{n \times d}$ and $\mathbf{y} = [\mathbf{y}^S; \mathbf{y}^T] \in \mathbb{R}^n$, with $n = n_S + n_T$, to build a transfer agent for the target task T.

Transfer Agent based on Gaussian Process. As stated in [32], *a GP is a collection of random variables, such that any finite number of which have the multivariate Gaussian distribution.* A *GP* model defines a multivariate Gaussian distribution over functions, $\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$, with mean vector $\boldsymbol{\mu}$ and covariance matrix $\mathbf{C}$ that is given by a parameterized kernel. Usually, $\boldsymbol{\mu} = \mathbf{0}$ is assumed, and thus the *GP* model is completely specified by the kernel. Likewise,

A transfer agent based on GP defines a joint Gaussian distribution over the space of source and target functions. With the zero-mean assumption, the GP based transfer agent is completely determined by the transfer kernel. However, different from the conventional kernel that treats all instance pair equally, the transfer kernel differentiates the instance pair based on the tasks where they come from.

Primitive Kernels. Kernels encode all assumptions about the form of function that GP are modelling. In general, a kernel represents some form of data characteristic, e.g., a linear kernel encodes the linear relationship. Some primitive kernels are widely used in GP models including:

- constant kernel: $C(\mathbf{x}, \mathbf{x}') = \sigma^2$,
- linear kernel: $LIN(\mathbf{x}, \mathbf{x}') = \sigma^2 \mathbf{x}^\top \mathbf{x}$,
- polynomial kernel: $POLY(\mathbf{x}, \mathbf{x}') = \sigma^2 (c + \mathbf{x}^\top \mathbf{x})^d$,
- radial basis function: $RBF(\mathbf{x}, \mathbf{x}') = \sigma^2 exp(-\frac{||\mathbf{x}-\mathbf{x}'||^2}{2l^2})$,
- rational quadratic: $RQ(\mathbf{x}, \mathbf{x}') = \sigma^2 (1 + \frac{||\mathbf{x}-\mathbf{x}'||^2}{2\alpha l^2})^{\frac{1}{\alpha}}$,
- periodic: $PER(\mathbf{x}, \mathbf{x}') = \sigma^2 exp(-\frac{2sin^2(\pi||\mathbf{x}-\mathbf{x}'||/p)}{l^2})$.

Neural Kernel Network. A neural kernel network [25] (NKN) is a neural net that computes compositional kernel structures. It is based on the well-known composition rules for kernels [12]: for any two kernels k_1 and k_2, (1) $\lambda_1 k_1 + \lambda_2 k_2$ with the compositional weights $\lambda_1, \lambda_2 \geq 0$ is a kernel, and (2) the product $k_1 k_2$ is a kernel. A NKN consists of several linear and product layers. By setting all the compositional weights non-negative, every unit of the network is a kernel based on the composition rules. As proved by [12], NKN has very powerful ability to approximate arbitrary stationary kernels.

4 TRANSFER NEURAL KERNEL NETWORKS

In this section, we introduce the transfer neural kernel network (TNKN), a more flexible and powerful transfer kernel that can model the fine-grained similarity of tasks in a hierarchical manner. TNKN consists of two compositional kernel structures, one for the covariance of intra-task instance pairs and another for the covariance of inter-task ones. It is formally defined as:

$$k_{tnkn}(\mathbf{x}, \mathbf{x}') = \begin{cases} k^d_{cks}(\mathbf{x}, \mathbf{x}'), & \delta(\mathbf{x}, \mathbf{x}') = 0, \\ k^s_{cks}(\mathbf{x}, \mathbf{x}'), & \delta(\mathbf{x}, \mathbf{x}') = 1, \end{cases} \tag{1}$$

where $\delta(\mathbf{x}, \mathbf{x}') = 1$ if $\mathbf{x}$ and $\mathbf{x}'$ are from the same task, otherwise $\delta(\mathbf{x}, \mathbf{x}') = 0$. Regarding Eq. (1), we have the following remarks:

- Each individual k^d_{cks} or k^s_{cks} is a compositional kernel structure. However, either of them is not necessarily to be a NKN since we do not specially constrain the compositional weights to be non-negative.
- k^d_{cks} and k^s_{cks} are coupled to one single network, k_{tnkn}, by sharing the same compositional structure but with different compositional weights. The data covariance is given by k^d_{cks} if two instances are from different tasks, otherwise it is given by k^s_{cks}. The task covariance is modeled by the difference of the combination weights of k^d_{cks} and k^s_{cks}.
- The positive semi-definiteness of k_{tnkn} is not guaranteed. In another words, to obtain a PSD k_{tnkn}, we need to carefully design the way of coupling k^d_{cks} and k^s_{cks}, specifically the compositional weights of k^d_{cks} and k^s_{cks}.

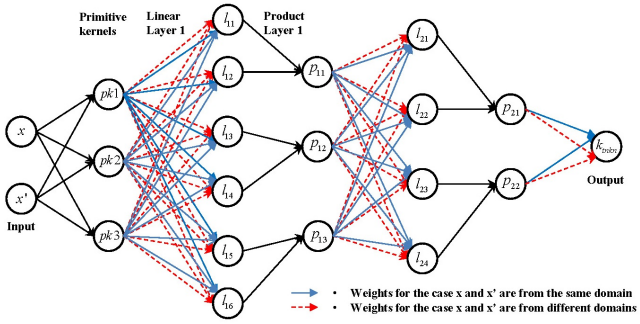


Figure 1: The architecture of a k_{tnkn}. This k_{tnkn} contains 3 primitive kernels, 3 linear layers and 2 product layers. The red dash line arrow denotes the weights of the linear layer for k^d_{nkn}, and the blue solid line arrow denotes the weights of the linear layer for k^s_{nkn}. The black solid line are connections shared by k^d_{nkn} and k^s_{nkn}.

4.1 Architecture

The first layer of k_{tnkn} is a set of primitive kernels. Afterwards, linear layer and product layer stack alternately. The k^d_{cks} and k^s_{cks} share all product layers, but differentiate from each other on linear layers. More specifically, each linear layer contains two sets of compositional weights, one set for k^d_{cks} and another set for k^s_{cks}. The full architecture of one k_{tnkn} example is shown in Figure 1.

The first layer of k_{tnkn} consists of several primitive kernels. These primitive kernels usually express the fundamental structural motifs for GPs, e.g., linearity and periodicity. Every primitive kernel is characterized by some hyper-parameters, and we propose to learn these hyper-parameters together with the rest of network parameters. Note that we utilize multiple copies of every primitive kernel to enable diverse parametrization. The linear layer of k_{tnkn} is a fully-connected layer associated with sets of learnable weights. Precisely, every linear layer includes two sets of compositional weights, one set for k^s_{cks} and another set for k^d_{cks}. This is significantly different from NKN that contains only one set of compositional weights in every linear layer. Moreover, different from NKN that can simply use non-negative compositional weights to ensure the positive semi-definiteness, k_{tnkn} needs to explore more complex relationship of the weights between k^s_{cks} and k^d_{cks}. Note that simply using non-negative weights for both k^s_{cks} and k^d_{cks} does not ensure a PSD k_{tnkn}. We present more details on the positive semi-definiteness of k_{tnkn} in the next section. The product layer introduces multiplications to k_{tnkn}. The connectivity pattern is fixed and there is no trainable parameters in the product layers. As proved in the work [25], this fixed structure does not restrict the expressiveness of the network. Finally, we also include a nonlinear activation function for every layer as the conventional deep neural network does. We use the exponential function $f(\mathbf{x}) = e^{\mathbf{x}}$ where $\mathbf{x}$ is the result of a linear combination or product.

4.2 Positive Semi-definiteness

To ensure the GP based transfer agent using k_{tnkn} are applicable for any data, k_{tnkn} must be PSD. We propose a sufficient condition for the PSD k_{tnkn}. The condition is given by Theorem 4.1.

Theorem 4.1. *A transfer neural kernel network k_{tnkn} consists of n_k primitive kernels ($k_1, ..., k_{n_k}$), n_l linear layers, and n_p product layers. The l-th ($l = 1, 2, ..., n_l$) linear layer includes a non-negative weight matrix $\mathbf{W}^{s_l} \in \mathbb{R}^{n_I^l \times n_O^l}$ for k^s_{cks}, and a weight matrix $\mathbf{W}^{d_l} \in \mathbb{R}^{n_I^l \times n_O^l}$ for k^d_{cks}, where n_I^l and n_O^l are the number of input and output units for the l-th linear layer, respectively. Such a k_{tnkn} is positive semi-definite for any input data if for $l = 1, 2, ..., n_l$, $|w_{ij}^{d_l}| \leq w_{ij}^{s_l}$ with $i = 1, 2, ..., n_I^l$ and $j = 1, 2, ..., n_O^l$ where $w_{ij}^{d_l}$ ($w_{ij}^{s_l}$) is the element of $\mathbf{W}^{d_l}$ ($\mathbf{W}^{s_l}$) in the i-th row and j-th column.*

Proof. To prove k_{tnkn} is a PSD kernel under the above condition, we first prove every unit of the network is a PSD kernel. Then the entire network is a PSD kernel according to the composition rules for kernels. We start from the first linear layer, which is a linear combination of the primitive kernels. The number of input units is equal to that of the primitive kernels, i.e., $n_I^1 = n_k$. For every output unit $k_j^1, j = 1, ..., n_O^1$, it has the following form:

$$k_j^1(\mathbf{x}, \mathbf{x}') = \begin{cases} \sum_{i=1}^{n_I^1} w_{ij}^{d_1} k_i(\mathbf{x}, \mathbf{x}'), \ \delta(\mathbf{x}, \mathbf{x}') = 0, \\ \\ \sum_{i=1}^{n_I^1} w_{ij}^{s_1} k_i(\mathbf{x}, \mathbf{x}'), \ \ \delta(\mathbf{x}, \mathbf{x}') = 1. \end{cases} \tag{2}$$

The Gram matrix of eq. (2) is:

$$\mathbf{K}_j^1 = \begin{bmatrix} \sum_{i=1}^{n_I^1} w_{ij}^{s_1} \mathbf{K}_{SS}^i & \sum_{i=1}^{n_I^1} w_{ij}^{d_1} \mathbf{K}_{ST}^i \\ \sum_{i=1}^{n_I^1} w_{ij}^{d_1} \mathbf{K}_{TS}^i & \sum_{i=1}^{n_I^1} w_{ij}^{s_1} \mathbf{K}_{TT}^i \end{bmatrix},$$

where

$$\mathbf{K}^i = \begin{bmatrix} \mathbf{K}_{SS}^i & \mathbf{K}_{ST}^i \\ \mathbf{K}_{TS}^i & \mathbf{K}_{TT}^i \end{bmatrix}$$

is the kernel matrix calculated by the primitive kernel k_i over the input source and target data. Since when $w_{ij}^{s_1} = 0$, $w_{ij}^{d_1}$ must be 0 ($|w_{ij}^{d_l}| \leq w_{ij}^{s_l}$), we can rewrite:

$$\mathbf{K}_j^1 = \sum_{i=1}^{n_I^1} w_{ij}^{s_l} \begin{bmatrix} \mathbf{K}_{SS}^i & w_{ij}^{d_l}/w_{ij}^{s_l} \mathbf{K}_{ST}^i \\ w_{ij}^{d_l}/w_{ij}^{s_l} \mathbf{K}_{TS}^i & \mathbf{K}_{TT}^i \end{bmatrix}.$$

We now consider:

$$\begin{aligned} \mathbf{N} &= \mathbf{K}_{TT}^i - (w_{ij}^{d_l}/w_{ij}^{s_l})^2 (\mathbf{K}_{ST}^i)^\top (\mathbf{K}_{SS}^i)^{-1} \mathbf{K}_{ST}^i \\ &= [1 - (w_{ij}^{d_l}/w_{ij}^{s_l})^2] \mathbf{K}_{TT}^i \\ &\quad + (w_{ij}^{d_l}/w_{ij}^{s_l})^2 [\mathbf{K}_{TT}^i - (\mathbf{K}_{ST}^i)^\top (\mathbf{K}_{SS}^i)^{-1} \mathbf{K}_{ST}^i]. \end{aligned}$$

Since $\mathbf{K}^i$ is PSD, according to Schur complement theorem [34] we have:

$$\mathbf{K}_{TT}^i \succeq \mathbf{0} \text{ and } \mathbf{K}_{TT}^i - (\mathbf{K}_{ST}^i)^\top (\mathbf{K}_{SS}^i)^{-1} \mathbf{K}_{ST}^i \succeq \mathbf{0}.$$

Furthermore, $[1 - (w_{ij}^{d_l}/w_{ij}^{s_l})^2] \geq 0$ since $|w_{ij}^{d_l}| \leq w_{ij}^{s_l}$. We then derive: $\mathbf{N} \succeq \mathbf{0}$. Considering $\mathbf{K}_{SS}^i \succeq \mathbf{0}$ and $w_{ij}^{s_l} \geq 0$, based on Schur complement theorem [34], we can derive: $\mathbf{K}_j^1 \succeq \mathbf{0}$. Equivalently, k_j^1 is a valid kernel.

Since the above proof is applicable for every $k_j^1, j = 1, ..., n_O^1$, we obtain that all the output units of the first linear layer are valid kernels. These output units are the input units for the next product layer. According to the composition rules for kernels, we induce that all the output units of the product layer are also valid kernels. Then these outputs are taken as the inputs for the next linear layer. The similar proof can be applied to the following alternate linear and product layers. To conclude, we can prove that all the units of k_{tnkn} are valid kernels when $|w_{ij}^{d_l}| \leq w_{ij}^{s_l}$ for $l = 1, 2, ..., n_l$, $i = 1, 2, ..., n_I^l$ and $j = 1, 2, ..., n_O^l$, and thus k_{tnkn} is a valid kernel under this condition. □

Note that we omit the exponential operation for every unit in the proof since it does not change the positive semi-definiteness of a unit. We now look into the semantic interpretation of Theorem 4.1. Given a linear layer, an output unit is a linear combination of several input kernels. The weight represents the contribution of an input kernel to the output unit, and thus implies how much knowledge is exploited from this input kernel. Theorem 4.1 shows that when the weights of k^d_{cks} do not go beyond that of k^s_{cks}, k_{tnkn} is PSD. This is reasonable as k^s_{cks} is used to calculate the covariance of two instances from the same task, and thus its weights imply the maximum transfer capacity. It is worth noting that k_{tnkn} greatly enriches the width and depth of the similarity modeling across tasks. Regarding the depth, k_{tnkn} stacks multiple linear and product layers to enable the similarity modeling from the low level to the high level. Regarding the width, k_{tnkn} uses multiple linear combinations to diversify the fine-grained modeling of the nuances across tasks. All of these make k_{tnkn} have a flexible and meticulous expressive power, as well as similarity modeling, on the complex and heterogenous data, and thus perform effective knowledge transfer in the real-world problems.

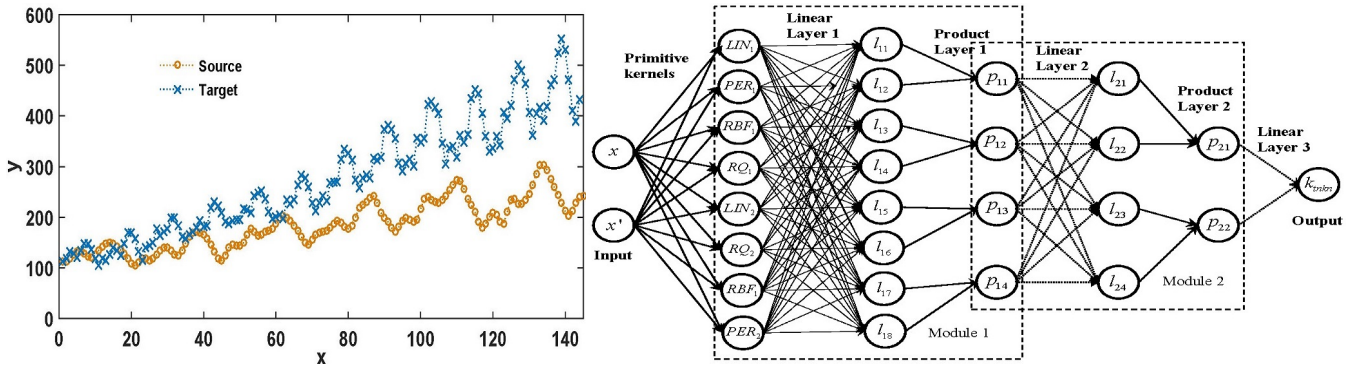
4.3 Significance

In this section, we highlight the significance of k_{tnkn}. Given the primitive kernels $\{k_j\}_{j=1}^{n_k}$, k_{tnkn} can be reformulated as follows:

$$k_{tnkn}(\mathbf{x}, \mathbf{x}') = \begin{cases} \sum_{t=1}^{T} \omega_t^d \prod_{j=1}^{n_k} k_j(\mathbf{x}, \mathbf{x}')^{p_{tj}}, \ \delta(\mathbf{x}, \mathbf{x}') = 0, \\ \\ \sum_{t=1}^{T} \omega_t^s \prod_{j=1}^{n_k} k_j(\mathbf{x}, \mathbf{x}')^{p_{tj}}, \ \delta(\mathbf{x}, \mathbf{x}') = 1, \end{cases} \tag{3}$$

where each of k^s_{cks} and k^d_{cks} is represented as a weighted polynomial of primitive kernels. The number of polynomials T and the degree p_{tj} are determined by the architecture of k_{tnkn}. The coefficients $\{\omega_t^s, \omega_t^d\}_{t=1}^T$ can be inferred from the weight matrices of the linear layers. This reformulation shows that k^s_{cks} and k^d_{cks} lie in the same kernel function space expanded by some base kernels (i.e., the polynomials of the primitive kernels), but they perform differently on these base kernels. The coefficients $\{\omega_t^s\}_{t=1}^T$ and $\{\omega_t^d\}_{t=1}^T$ fully characterize k^s_{cks} and k^d_{cks}, and their element-wise distance indicates the similarity of the tasks on the corresponding base kernel.

Moreover, k^s_{cks} is a NKN with $\{\omega_t^s \geq 0\}_{t=1}^T$. Based on the Theorem 3 in [25], k^s_{cks} can approximate arbitrary stationary kernels, which again verifies the expressive power of k_{tnkn}. Theorem 4.1



(a) The source and target data include similar patterns.

(b) The TNKN structure: 3 linear layers and 2 product layers.

Figure 2: The data and the TNKN architecture for time series task.

also shows us ω_t^d can take negative values. In this case, the individual k_{cks}^d is not a NKN, and even may be not PSD at all. However, by coupling with k_{cks}^s, k_{tnkn} become a PSD transfer kernel. This demonstrates that k_{tnkn} is not a simple extension of two NKNs, but a specialized NKN-based design for the transfer purpose.

It is worth noting that the existing popular transfer kernels motivated from ICM, e.g., [6, 28, 31] or LMC. e.g., [26, 30], can be taken as special cases of our k_{tnkn}. For the transfer kernel used in [6, 28, 31], it is the k_{tnkn} with only one primitive kernel, $\omega_t^s = 1$, and $\omega_t^d = \lambda$. For the transfer kernel proposed by [30], it is the k_{tnkn} where k_{cks}^s and k_{cks}^d are the weighted primitive kernels. For the transfer kernel in [26], it is the k_{tnkn} where the combination coefficients $\{\omega_t^s, \omega_t^d\}_{t=1}^T$ are constrained to be 0 or 1. Compared with the existing kernels, k_{tnkn} outperforms on both the expressive power (allowing diverse polynomial of primitive kernels) and the capacity of similarity modelling (allowing rich and flexible combination coefficients). In this sense, k_{tnkn} is a more general and powerful transfer kernel.

4.4 Learning

The overall architecture of k_{tnkn} is differentiable, and thus the GP based transfer agent using k_{tnkn} is trainable using gradient-based optimization. All the parameters include three categories: (1) hyper-parameters in the primitive kernels, (2) weights matrices of the linear layers, and (3) the noise variances of the source and target tasks. We group all the parameters together, denoted as $\mathbf{\Theta}$, and learn them jointly. Following [6], we focus on the target task, and optimize the conditional distribution $p(\mathbf{y}^T|\mathbf{X}, \mathbf{y}^S)$. By denoting δ_S^2 and δ_T^2 as the noise variance for the source and target tasks and $\mathbf{K}^*$ as the kernel matrix calculated from k_{tnkn}, we have $p(\mathbf{y}^T|\mathbf{X}, \mathbf{y}^S) \sim \mathcal{N}(\boldsymbol{\mu}_*, \mathbf{C}_*)$ where

$$\boldsymbol{\mu}_* = \mathbf{K}_{TS}^*(\mathbf{K}_{SS}^* + \delta_S^2\mathbf{I})^{-1}\mathbf{y}^S$$
$$\mathbf{C}_* = (\mathbf{K}_{TT}^* + \delta_T^2\mathbf{I}) - \mathbf{K}_{TS}^*(\mathbf{K}_{SS}^* + \delta_S^2\mathbf{I})^{-1}\mathbf{K}_{ST}^*.$$

Then, the log-likelihood is defined as:

$$\mathcal{L}(\mathbf{\Theta}) = -\frac{1}{2}ln|\mathbf{C}_*| - \frac{1}{2}(\mathbf{y}^{\mathcal{T}} - \boldsymbol{\mu}_*)^{\mathrm{T}}\mathbf{C}_*^{-1}(\mathbf{y}^{\mathcal{T}} - \boldsymbol{\mu}_*) - \frac{n}{2}ln(2\pi).$$

Note that we also need to take Theorem 4.1 into account in the learning. Regarding the non-negative weights of k_{nkn}^s, we use $f(x) = log(1 + exp(x))$ to enforce the nonnegativity constraint. Regarding the inequality constraints on the weights of k_{nkn}^s and k_{nkn}^d, we can check the fulfilment of these constraints in each epoch, and map the violated $w_{ij}^{d_l}$ to a value that is close to the feasible boundary. Alternatively, we can further constrain $w_{ij}^{d_l}$ as non-negative to enforce k_{nkn}^d also a NKN. Then, we satisfy $|w_{ij}^{d_l}| \leq w_{ij}^{s_l}$ by setting $w_{ij}^{s_l} = w_{ij}^{d_l} + {\epsilon_{ij}^l}^2$. In this case, $w_{ij}^{d_l}$ and ϵ_{ij}^l are the optimization variables.

4.5 Complexity

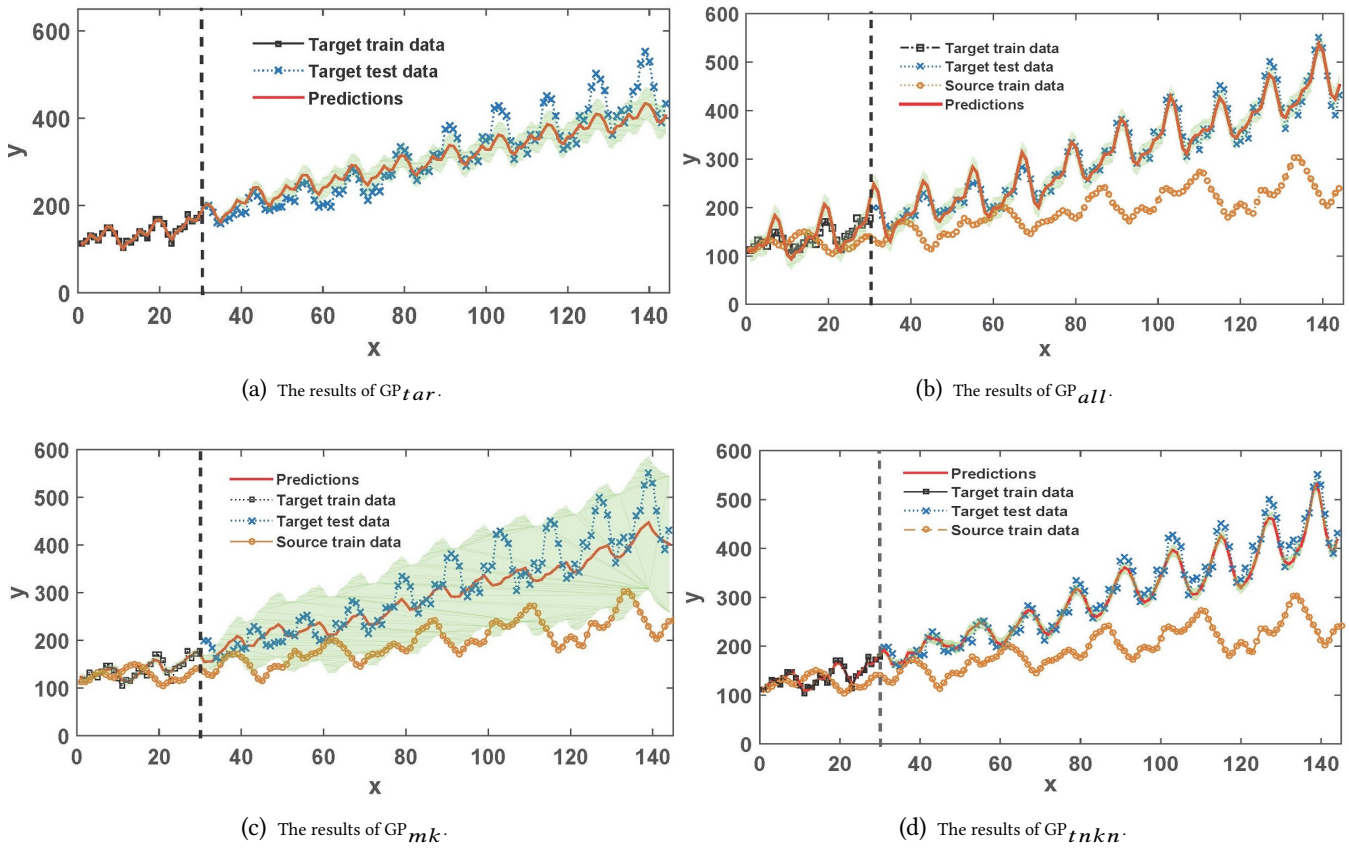
The usage of k_{tnkn} does not deteriorate the computation cost of the transfer agent learning. Considering a k_{tnkn} with m connections, the computational cost of the forward pass is $O(n^2m)$. Note that the computational cost of the kernel matrix inversions is $O(n^3)$. By adequately designing the architecture of k_{tnkn} (as what we have done in the experiments), the number of parameters, m, is usually much smaller than that of training data, n (in our experimental studies, k_{tnkn} has only tens of parameters, but the training points are in hundreds and even thousands). In this case, the computational cost of the transfer agent learning is still dominated by the kernel matrix inversions, which is the same as the conventional GP learning.

5 EXPERIMENTAL STUDIES

We evaluate the transfer performance of the GP based transfer agent using k_{tnkn}, denoted as GP_{tnkn}, on 5 real-world datasets including 1 time series dataset and 4 regression datasets. We compare it with several state-of-the-art baselines with respect to the transfer performance. All the experiments are done on a 64-bit operating system with the processor Intel(R) Xeon(R) CPU E5-1650 0 @ 3.20GHz.

5.1 Time Series Extrapolation

We first conduct experiment on a time series dataset [12] to demonstrate the transfer capacity of GP_{tnkn} on the data extrapolation.



(a) The results of GP$_{tar}$. (b) The results of GP$_{all}$.

(c) The results of GP$_{mk}$. (d) The results of GP$_{tnkn}$.

Figure 3: The black squares are the target training data. The yellow circles are the source training data. The blue 'x's are the ground-truth of the target test points. The red line is the prediction mean and the green region represents the standard deviation.

The dataset includes the airline passenger volume data, and we construct two sets of data with different patterns. The task is to predict the future data patterns. As shown in Figure 2(a), both the source and target data include the periodic and increasing waves [1]. Specifically, the wave appears periodically along a slope. The amplitude of the wave is also growing with the period. However, it is worth noting that the pattern characteristics of the waves are significantly different between the source and target. The data in the source task has longer period, flatter slope, and slower amplitude growth than the the data in the target task. Our objective is to utilize sufficient source data and limited target data to extrapolate the subsequent unseen patterns for the target. In the experiment, we take the first 30 target points and 288 source points as the training data. The rest target data (from 31 to 144) are used as the test points.

We design a TNKN with 3 linear layers and 2 product layers for this problem. We use 2 LIN kernels, 2 RBF kernels, 2 PER kernels, and 2 RQ kernels as the primitive kernels. The number of the input and output units for the 3 linear layers are (8,8), (4,4), and (2,1). Correspondingly, the numbers of the input and output units for the two product layers are (8,4) and (4,2). The overall architecture is shown in Figure 2(b). For the sake of brevity, we only show the connections of the network without distinguishing between k^s_{cks} and k^d_{cks}. For this TNKN, the maximum degree of the polynomials of primitive kernels is 4, and the number of the polynomials of degree 4 is 330 (C^{8-1}_{8+4-1}). We can see that such a TNKN can model rich and complex data characteristics by these diverse polynomials of primitive kernels, e.g., LIN[4] models the long-term smooth trend, PER[4] models the periodicity, and the more complex LIN×PER×RBF×RQ models the periodic variation.

We implement our GP$_{tnkn}$ based on the GPflow package [18] using python 3.6.5. We use 'Adamoptimizer' with the learning rate as 0.001, and set the epoch number as 20000. We compare with several baselines including GP$_{tar}$ that only uses 30 target data to train a GP model, GP$_{all}$ that just simply combines the 30 target training data and 288 source training data, and GP$_{mk}$ [30]. For GP$_{tar}$ and GP$_{all}$, we exploit a NKN with the same network structure as TNKN but only 1 set of weights as the kernel. For GP$_{mk}$, we use the default model configurations as used in [30]. Root mean squared error (RMSE) is used as the evaluation metric.

The numerical RMSE results are shown in table 1. It can be seen that GP$_{tnkn}$ achieves the best performance, followed by GP$_{all}$,

[1] A wave is a curve with the approximate contour '∧'.

Table 1: Numerical results for time series extrapolation.

Transfer Agents	GP_{tar}	GP_{all}	GP_{mk}	GP_{tnkn}
RMSE	51.32	27.87	44.05	24.41

GP_{mk}, and lastly GP_{tar}. To have a more clear view of the extrapolation, we plot the predictions of each method in Figure 3. As can be seen from 3(a), GP_{tar} can well fit the target training points, but it fails to fit the subsequent target test points. Since no information is available except for the 30 target training points, the predictions proceed with the target training pattern. However, the amplitude growth pattern is completely lost. Regarding GP_{all} in Figure 3(b), it also fails to do accurate extrapolation on the target test data, and even worse, it fails to fit the target training data. Interestingly, we observe that GP_{all} generates a new pattern. That is, from the first wave, every 3 waves form a period. In such a period, the amplitude of the wave slowly grows larger, and achieves the largest at the third wave. Afterwards, a new period starts, but the amplitude of the first wave in the new period decreases compared with that of the last wave in the former period. Such a new pattern is a negative result of the brute-force data integration that completely ignores the divergence between different tasks. For GP_{mk}, which is a recent adaptive transfer GP model, Figure 3(c) shows that it yields almost the same results as GP_{tar}. This implies that GP_{mk} does not transfer any knowledge from the source to the target task. The bad performance is mainly due to the poor expressive power of GP_{mk}. GP_{mk} uses a linear summation of the primitive kernels, and thus fails to capture and transfer the knowledge that are represented by the product of the primitive kernels. Moreover, GP_{mk} obtains much larger standard deviation than GP_{tar} since the source data pull down the prediction confidence. Finally, from Figure 3(d), we can see that GP_{tnkn} not only fits the target training data well, but also generates sensible extrapolation. The predictions capture all the patterns including the periodicity, linear increase, and amplitude growth. Meanwhile, the standard deviations of the predictions are small, which implies the source data indeed provide help for the target task. These results demonstrate that GP_{tnkn} can effectively and adaptively transfer the knowledge across tasks. We also observe that GP_{tnkn} models the amplitude growth much better in the starting periods than in the subsequent ones. This is reasonable since the source data has a slow amplitude growth pattern, and it is transferred to the target task.

We further analyze the similarity of the weights directly learned from GP_{tnkn}. There are 3 linear layers in this TNKN, and we focus on the weights of the first linear layer, i.e., the combination weights of the primitive kernels. For each node, we calculate the ratio of the weights of k^d_{cks} over the corresponding ones of k^s_{cks} as the similarity of the primitive kernels to construct this node. Figure 4 shows the heat map of the learned similarities on each primitive kernel for the 8 nodes of the first linear layer. The y-axis denotes node index, and the x-axis denotes primitive kernel. Thus, each row represents the learned combination weights of the 8 primitive kernels for one node. In a horizontal view, it can be observed that different nodes have different similarity combinations of the primitive kernels. This demonstrates the rich expressive power of k_{tnkn}. In a vertical view, we find that the two tasks are more similar on the linear kernel,

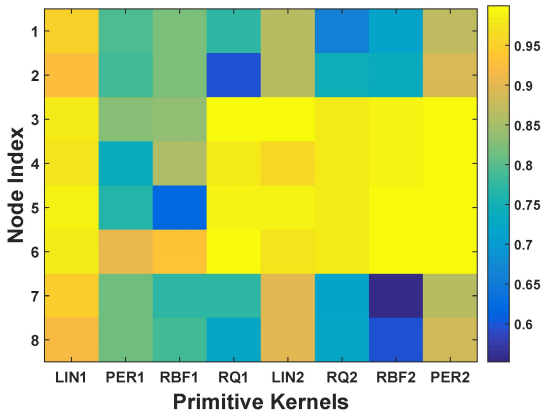


Figure 4: The heat map of the similarity for the first linear layer.

Table 2: Similarity of some polynomial of primitive kernels.

Polynomials	$LIN_1{}^4$	$PER_1{}^4$	$RBF_1{}^4$	$RQ_1{}^4$	$LIN_1 \times PER_1 \times RBF_1 \times RQ_1$
Similarity	0.9130	0.4354	0.4056	0.9453	0.6385

but less similar on the periodic kernel. This well corresponds to the source and target patterns as shown is Figure 2. Moreover, we also note that the similarities of the same type of primitive kernels can be very different due to the different parametrization, e.g., PER1 vs. PER2. All of these demonstrate the flexility of k_{tnkn} in modelling diverse similarities across tasks.

Except for the analysis of the learned linear layer weights, we also investigate the weights of the polynomial of primitive kernels, i.e., ω_s and ω_d in eq. (3), which can be inferred from the learned weights of GP_{tnkn}. Recall that this TNKN can express 330 polynomials with degree 4, and we only investigate some representative polynomials of primitive kernels. Specifically, we calculate ω^s_t and ω^d_t for $LIN_1{}^4$, $PER_1{}^4$, $RBF_1{}^4$, $RQ_1{}^4$, and $LIN_1 \times PER_1 \times RBF_1 \times RQ_1$. We use the ratio ω^d_t/ω^s_t to represent the similarity of different tasks on the corresponding polynomial. Table 2 shows the similarity results. The high similarity on $LIN_1{}^4$ and $RQ_1{}^4$ correspond to the similar long-term smooth trend and small-scale variations of the source and target functions in Figure 2(a). In contrast, the low similarity on $PER_1{}^4$ and $RBF_1{}^4$ reflects the dissimilar periodicity and variation speed. Regarding the $LIN_1 \times PER_1 \times RBF_1 \times RQ_1$ that models both the long-time trend and the short-time derivation, the similarity is an intermediate value. All these results show that GP_{tnkn} can adaptively capture the fine-grained similarities across tasks.

5.2 Real-world Regression Problems

We further verify the effectiveness of k_{tnkn} on four real-world regression applications. We test on the following 4 real-world datasets.

Wine Quality. The wine quality dataset includes red and white wine samples [9]. Physicochemical variables, e.g. volatile acidity, are used as features and the sensory variable is used as label. We use the quality prediction on white wine (**W**) as the source task and the quality prediction on red wine (**R**) as the target one. Specifically,

Table 3: Comparison results (RMSE) on four real-world regression tasks.

Task	GP_{tar}	GP_{all}	GP_{λ}	$GP_{\lambda_{var}}$	GP_{mk}	GP_{tnkn}
Wine Quality	1.7720 ± 0.0563	1.2634 ± 0.0156	1.0672 ± 0.0236	1.0635±0.0178	0.8395 ± 0.0125	**0.7233 ± 0.0021**
Energy Use	1.8939 ± 0.0154	1.9030 ± 0.0089	1.5120 ± 0.0108	1.4996±0.0069	1.2389 ± 0.0067	**1.1660 ± 0.0027**
Air Quality	0.9329 ± 0.0819	1.0134 ± 0.0059	0.9246 ± 0.0522	0.9180±0.0437	0.8172 ± 0.0268	**0.7915 ± 0.0046**
Building Location	0.2163 ± 0.0531	0.1177 ± 0.0046	0.1339 ± 0.0073	0.1296±0.0025	0.1023 ± 0.0421	**0.0963 ± 0.0026**

W includes 500 training data points, and **R** includes 10 training data points and 500 test data points.

Appliance Energy Prediction. This dataset is used to predict the appliance energy use in different months [5]. The house conditions data, e.g. humidity, monitored by the sensors are used as features, and the amount of appliance energy use is used as label. The data of May (500 for training) is used as the source task, and the data of January (5 for training and 500 for test) is used as the target task.

Air Quality. The air quality dataset [10] contains responses of gas multisensor devices from different seasons. The averaged responses from the metal oxide chemical sensors are features, and the ground-truth provided by a co-located reference certified analyzer is used as label. Data from Summer and Winter are used for the source and target, respectively. The data configuration is the same as that of wine quality dataset.

UJIIndoorLoc. This dataset covers signals from different buildings of Universitat Jaume I [27]. The signal strength intensity from 520 wireless access points is used as features, and the location is used as label. We use building 1 including 1000 training data as the source, and we use building 2 including 25 training data and 1000 test data as the target.

We design a TNKN with 2 linear layers and 1 product layer for the regression problems. Two RBF kernels and 1 linear kernel are used as the primitive kernels. The number of the input and output units for the two linear layers is (3,4) and (2,1). Correspondingly, the number of the input and output units for the product layer is (4,2). We use 'Adamoptimizer' with the learning rate as 0.001 for the training, and set the epoch number as 5000 for all the regression problems. We run five restarts with different initializations, and take the average root mean squared error (RMSE) as the final result. We compare with state-of-the-art baselines, namely GP_{tar}, GP_{all}, GP_{λ} [6], $GP_{\lambda_{var}}$ [28], and GP_{mk} [30].

The comparison results are shown in Table 3. As can be seen from the table, the four adaptive transfer methods GP_{λ}, $GP_{\lambda_{var}}$, GP_{mk} and GP_{tnkn} generally achieve better results than the other two naive methods. Among these 4 methods, GP_{tnkn} is the best, followed by GP_{mk}, and lastly $GP_{\lambda_{var}}$ and GP_{λ}. The deviation of the transfer performance across these methods is due to the difference of their expressive power and capacity in modeling the task similarity. GP_{λ} uses a single similarity coefficient and a single primitive kernel, which highly constrain its expressive power and transfer capacity. $GP_{\lambda_{var}}$ is a variant of GP_{λ}, and it is only applicable for RBF kernel to ensure the positive semi-definiteness. GP_{mk} uses a single linear layer, and thus can not express the data characteristics represented by the product of the primitive kernels. In contrast, our GP_{tnkn} establishes a deep kernel neural network that can approximate arbitrary stationary kernels. All the results demonstrate our GP_{tnkn} is a very promising method for transfer regression problems.

6 CONCLUSIONS

In this paper, we propose a transfer neural kernel network (TNKN). A transfer kernel k_{tnkn} that is represented by a neural network is presented to model the task similarity in a layer-to-layer manner. A sufficient condition that validates k_{tnkn} as a valid kernel is proposed, and is then used as a constraint in the corresponding GP based transfer agent learning. Experimental results on 1 time series extrapolation task and 4 real-world regression tasks demonstrate the effectiveness of the proposed TNKN compared with several state-of-the-art baselines. We emphasize that TNKN is a general transfer method, although in this paper we demonstrate its effectiveness by applying it to GP in regression problems. We will study the potential application of TNKN to other models, e.g., SVM and kernel ridge regression model, etc., and more problems, e.g., dynamic decision making and classification, etc., in future studies.

7 ACKNOWLEDGEMENT

This work was partially done when the first author worked in Nanyang Technological University. This work was partially supported by an Academic Research Grant No. MOE2016-T2-2-068 from the Ministry of Education in Singapore.

REFERENCES

[1] Mauricio A Alvarez, Lorenzo Rosasco, Neil D Lawrence, et al. 2012. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning* 4, 3 (2012), 195–266.

[2] Francis R. Bach, Gert R. G. Lanckriet, and Michael I. Jordan. 2004. Multiple Kernel Learning, Conic Duality, and the SMO Algorithm. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML '04)*. ACM, 6–16.

[3] Edwin V. Bonilla, Kian M. Chai, and Christopher Williams. 2008. Multi-task Gaussian Process Prediction. In *Advances in Neural Information Processing Systems 20*. 153–160.

[4] Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larrañaga. 2015. A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5, 5 (2015), 216–233.

[5] Luis M. Candanedo, Véronique Feldheim, and Dominique Deramaix. 2017. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings* 140 (2017), 81–97.

[6] Bin Cao, Sinno Jialin Pan, Yu Zhang, Dit-Yan Yeung, and Qiang Yang. 2010. Adaptive Transfer Learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. 407–712.

[7] Kian M Chai. 2009. Generalization errors and learning curves for regression with multi-task Gaussian processes. In *Advances in neural information processing systems*. 279–287.

[8] Trevor Cohn and Lucia Specia. 2013. Modelling annotator bias with multi-task gaussian processes: An application to machine translation quality estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 32–42.

[9] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. Modeling Wine Preferences by Data Mining from Physicochemical Properties. *Decision Support Systems* 47, 4 (2009), 547–553.

[10] Saverio De Vito, Grazia Fattoruso, Matteo Pardo, Francesco Tortorella, and Girolamo Di Francia. 2012. Semi-supervised learning techniques in artificial olfaction: A novel approach to classification problems and drift counteraction. *IEEE Sensors Journal* 12, 11 (2012), 3215–3224.

[11] Xin Dong and Gerard de Melo. 2018. A helping hand: Transfer learning for deep sentiment analysis. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2524–2534.

[12] David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B Tenenbaum, and Zoubin Ghahramani. 2013. Structure discovery in nonparametric regression through compositional kernel search. *ICML* (2013), 1–9.

[13] Timothy C Haas. 1996. Multivariate Geostatistics: An Introduction With Applications. *J. Amer. Statist. Assoc.* 91, 435 (1996), 1375–1377.

[14] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. 2017. Darla: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1480–1490.

[15] Nicholas J Higham. 1990. *Analysis of the Cholesky decomposition of a semi-definite matrix*. Oxford University Press.

[16] Min Jiang, Zhongqiang Huang, Liming Qiu, Wenzhen Huang, and Gary G Yen. 2018. Transfer learning-based dynamic multiobjective optimization algorithms. *IEEE Transactions on Evolutionary Computation* 22, 4 (2018), 501–514.

[17] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. 2018. Conditional adversarial domain adaptation. In *Advances in Neural Information Processing Systems*. 1647–1657.

[18] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. 2017. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research* (2017), 1–6.

[19] Arman Melkumyan and Fabio Ramos. 2011. Multi-kernel Gaussian processes. In *Twenty-Second International Joint Conference on Artificial Intelligence*. 1408–1403.

[20] Alan Tan Wei Min, Ramon Sagarna, Abhishek Gupta, Yew-Soon Ong, and Chi Keong Goh. 2017. Knowledge transfer through machine learning in aircraft design. *IEEE Computational Intelligence Magazine* 12, 4 (2017), 48–60.

[21] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342* (2015).

[22] David N Perkins, Gavriel Salomon, et al. 1992. Transfer of learning. *International encyclopedia of education* 2 (1992), 6452–6457.

[23] Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. 2019. Learning Variable Ordering Heuristics for Solving Constraint Satisfaction Problems. *arXiv preprint arXiv:1912.10762* (2019).

[24] Wen Song, Donghun Kang, Jie Zhang, Zhiguang Cao, and Hui Xi. 2019. A sampling approach for proactive project scheduling under generalized time-dependent workability uncertainty. *Journal of Artificial Intelligence Research* 64 (2019), 385–427.

[25] Shengyang Sun, Guodong Zhang, Chaoqi Wang, Wenyuan Zeng, Jiaman Li, and Roger Grosse. 2018. Differentiable compositional kernel learning for Gaussian processes. *arXiv preprint arXiv:1806.04326* (2018).

[26] Anh Tong and Jaesik Choi. 2019. Discovering Latent Covariance Structures for Multiple Time Series. In *International Conference on Machine Learning*. 6285–6294.

[27] Joaquín Torres-Sospedra, Raúl Montoliu, Adolfo Martínez-Usó, Joan P. Avariento, Tomás J. Arnau, Mauri Benedito-Bordonau, and Joaquín Huerta. 2014. UJIIndoorLoc: A new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems.. In *2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 261–270.

[28] Neeti Wagle and Eric W Frew. 2017. Forward adaptive transfer of gaussian process regression. *Journal of Aerospace Information Systems* 14, 4 (2017), 214–231.

[29] Pengfei Wei, Yiping Ke, and Chi Keong Goh. 2018. Feature Analysis of Marginalized Stacked Denoising Autoenconder for Unsupervised Domain Adaptation. *IEEE transactions on neural networks and learning systems* (2018), 1–14.

[30] Pengfei Wei, Ramon Sagarna, Yiping Ke, and Yew Soon Ong. 2018. Uncluttered Domain Sub-Similarity Modeling for Transfer Regression. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1314–1319.

[31] Pengfei Wei, Ramon Sagarna, Yiping Ke, Yew-Soon Ong, and Chi-Keong Goh. 2017. Source-Target Similarity Modelings for Multi-Source Transfer Gaussian Process Regression. In *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 3722–3731.

[32] Christopher Williams and Edward Rasmussen. 1996. Gaussian processes for regression. In *Advances in neural information processing systems*. 514–520.

[33] Sergey Zakharov, Wadim Kehl, and Slobodan Ilic. 2019. DeceptionNet: Network-Driven Domain Randomization. *ICCV* (2019), 1627–1634.

[34] Fuzhen Zhang. 2005. *The Schur Complement and Its Applications*. Vol. 4. Springer Science & Business Media.