Exploring Approaches for Heterogeneous Transfer Learning in Dynamic Networks

Fernando García Sanz*, Masoumeh Ebrahimi*‡, and Andreas Johnsson*†
* Ericsson Research, Sweden, Email: {*fernando.a.garcia.sanz, masoumeh.ebrahimi, andreas.a.johnsson*}*@ericsson.com*
† Uppsala University, Department of Information Technology, Sweden, Email: *andreas.johnsson@it.uu.se*
‡ KTH, Division of Electronics and Embedded Systems, Sweden, Email: *mebr@kth.se*

NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium | 978-1-6654-0601-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/NOMS54207.2022.9789761

***Abstract*—Maintaining machine-learning models for prediction of service performance is challenging, especially in dynamic network and cloud environments where route changes occur, and execution environments can be scaled and migrated. Recently, transfer learning has been proposed as an approach for leveraging already learned knowledge in a new environment. The challenge is that the new environment may be significantly different from the one the model is trained in, and transferred from, with respect to data distributions and dimensionality.**

In this paper, we introduce heterogeneous transfer learning in the context of dynamic environments and show its efficiency in predicting service performance. We propose two heterogeneous transfer-learning approaches and evaluate them on several neural-network architectures and scenarios. The scenarios are a natural consequence of network and cloud infrastructure re-orchestration. We quantify the transfer gain, and empirically show positive gain in a majority of cases for both approaches. Furthermore, we study the impact of neural-network configurations on the transfer gain, providing tradeoff insights. The evaluation of the approaches is performed using data traces collected from a cloud testbed that runs two services under multiple realistic load conditions.

***Index Terms*—Service Performance, Machine Learning, Heterogeneous Transfer Learning**

I. Introduction

A promising approach enabling intelligent telecom network and service management is the use of machine-learning models that can analyze and predict service performance based on measurements and other observations in the network infrastructure, as well as taking reactive and proactive reconfiguration actions [1] [2]. The ability to learn service performance models, which is the scenario targeted in this paper, simplifies and improves a number of management and operational tasks such as service on-boarding, network resource adaptation, proactive service assurance, energy optimization, and root-cause analysis.

Traditionally, many machine-learning applications operate under the assumption that train and test data are both under the same feature space, and that they are extracted from the same distribution [3]. Hence, when the data distribution or its dimensionality changes, the previously crafted models must be rebuilt from scratch, and trained with the new data. Collecting additional data for training new models is expensive and sometimes infeasible. This gets more accentuated in environments with real-time monitoring requirements, or limited computational or networking resources. Training models from scratch, specifically for large and complex variants, also has a footprint in terms of power consumption stemming from extensive use of compute resources [4].

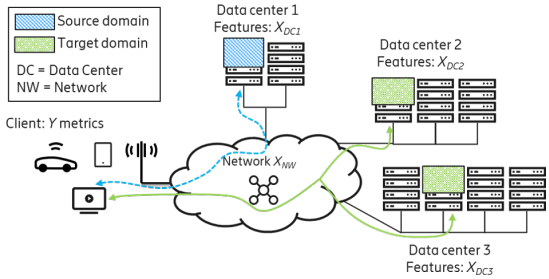


Fig. 1. Modeling of service metrics Y at the client given observations of the infrastructure features in the source domain (X_{NW}, X_{DC1}). When the execution environment changes to the target domain, the model must be adapted to the new feature set $(X_{NW}, X_{DC2}, X_{DC3})$.

Figure 1 illustrates a management example that could lead to a change in data distributions and dimensionality. Clients are accessing a cloud service over the network, where the service is being migrated and horizontally scaled from the source to the target domain. A model for service performance Y trained using features in the source domain (X_{NW}, X_{DC1}) is no longer compatible with the larger set of features describing the target domain $(X_{NW}, X_{DC2}, X_{DC3})$. Another reason for a changed feature set may be due to data-collection costs in certain parts of the network. That is, the cost of collecting a specific feature may not match the expected performance improvement of the model, and thus should be removed.

Transfer learning is a mitigation approach where knowledge learned in one environment can be leveraged in another. This enhances the training process as it is not necessary to develop new models from scratch. Previously built models can rather be adapted to a new environment, which also reduces requirements on monitoring and power consumption.

In this paper, which extends our previous work on homogeneous transfer learning [5] and source selection [6] [7] for improved performance prediction, we explore two approaches for heterogeneous transfer learning on neural-network models in environments where source and target domains do not share a common input feature space. Such scenarios may occur due to changes in monitoring configurations and feature

availability, as exemplified above. More specifically, we study the impact of adding, removing, and changing input features in the target domain on the transfer gain.

The main contributions of this paper are: (1) analysis of heterogeneous transfer learning, providing empirical evidence of feasibility and robustness in modelling performance of networked services, (2) description and evaluation of two heterogeneous transfer learning approaches for feed-forward neural-network models to improve the transfer gain, and (3) investigating the impact of neural-network configurations on the transfer gain using multiple data traces.

The rest of the paper is organized as follows. Section II provides related work. Section III provides a formulation of transfer learning for homogeneous and heterogeneous scenarios. In Section IV, we describe two approaches for heterogeneous transfer learning and various neural network model alternatives. Section V describes the testbed and data traces used for evaluation, whereas Section VI contains results. Conclusions are located in Section VII.

II. Related Work

In a seminal paper by Yosinski et al. [8], the authors studied and highlighted the importance of feature transfer, providing insights that neural networks learn general and specific latent features, and that the former can be applied to multiple image datasets. Further, in [9] the authors studied transfer learning in the field of natural language processing, and showed that semantic similarity impacts the transferability of neural-network models.

Transfer learning is also becoming an increasingly important approach for improving machine-learning performance in dynamic network, IT, data centers, and cloud environments. Examples include methods for fault localization on IT infrastructures [10] where transfer learning is incrementally enriching the models, and optimization and configuration of software systems [11] where transfer learning is used as part of a framework to learn from the most relevant sources. Further, in [12] the authors use transfer learning for solving problems related to cross-platform performance prediction of different hardware architectures. Our own works on homogeneous transfer learning [5] and source selection [6] [7] are extended with the study of scenarios with feature heterogeneity.

There are multiple ways of classifying transfer learning approaches [3], and one key factor is the distinction between homogeneous and heterogeneous transfer learning. Homogeneous transfer learning corresponds to the case when the source and target domain feature spaces are represented by the same attributes or features [13]. The above-mentioned works mainly fall into this class.

As it was already mentioned, in heterogeneous transfer learning, which is the focus of this paper, the source and target domain do not share a common feature space. Heterogeneous transfer learning has mainly been addressed using *mapping-based* approaches [14], where features in respective source domains are transformed to a common subspace, thereafter used by a common predictive model in the target domain; examples include [15] [16] [17]. Other mapping-inspired approaches can be found in [18] [19], where the mapping is done via random forests instead of neural networks. Our work in this paper differentiates itself by using a neural-network model [14] with first layer replacement or modification.

There are also some other recent works in other research domains that leverage neural-network models, for example in [20] [21], where pre-trained models are inserted into a bigger pipeline in order to take advantage of their embedded knowledge. Nonetheless, these works do not specifically investigate feature heterogeneity and the impact of network configuration.

III. Homogeneous and Heterogeneous Transfer Learning Formulation

Figure 1 illustrates the scenario that is considered in this work. Clients are interacting with services that are executing in the cloud. In this paper, we consider data traces originating from testbed experiments where clients access two network services executing in one data center: a Video-on-Demand (VoD) service and a Key-Value Store (KVS) service.

The learning task is to predict the service-level metrics Y_t, e.g., response time, at time t on the clients accessing the services based on knowing the infrastructure metrics X_t, e.g., CPU and memory utilization, at time t. We train and evaluate models $M : X_t \rightarrow \hat{Y}_t$, such that $\hat{Y}_t$ closely approximate Y_t for a given X_t using supervised machine learning.

A. Homogeneous transfer learning

Based on the definition of transfer learning in [3], a domain $D = \{X, P(X)\}$ consists of two components: (1) a feature space X, and (2) a marginal probability distribution $P(X)$, where X corresponds to the infrastructure metrics. The number of features in X is denoted $|X|$. Further, a task $T = \{Y, M\}$ consists of two components: a target space Y corresponding to service-level metrics, and an objective predictive model M. Transfer learning is then defined as follows. Given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T, transfer learning aims at reducing the cost of learning the predictive model M in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$. A model that is transferred from D_S to D_T is denoted $M_{S \rightarrow T}$, to separate it from a model M_S or M_T that is trained in isolation (i.e., from scratch) in the source or target domain. The number of training samples in D_T at time t is denoted N_t.

Homogeneous transfer learning is used to mitigate model degradation stemming from execution environment changes when $X_T = X_S$ as explored in previous work [5].

B. Heterogeneous Transfer Learning

Heterogeneous transfer learning (HTL) corresponds to a scenario where the available input features X are not identical when the execution environment changes. This may occur due to changes in monitoring configurations, re-orchestration of the infrastructure, and policies for feature privacy, for example.

This paper studies, quantifies, and explains the impact of transferring a source domain to a target domain where

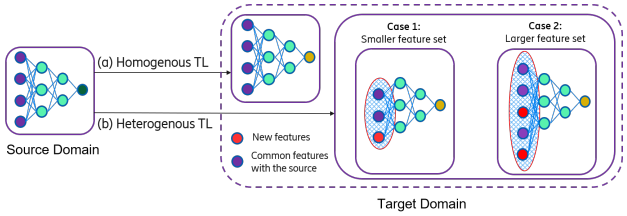


Fig. 2. (a) Homogeneous transfer learning where input feature space in the target domain is similar to that of the source domain, (b) heterogeneous transfer learning where the target domain contains lower or larger number of features than the source domain.

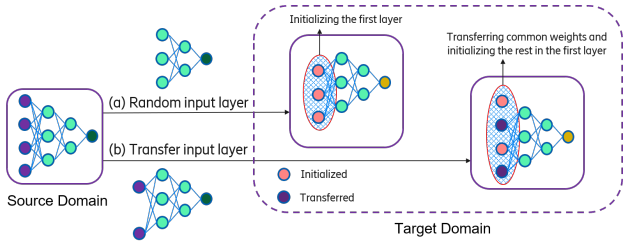


Fig. 3. Two approaches for heterogeneous transfer learning for service-performance prediction: (a) *Random input layer* strategy where the weights of the first layer are randomly initialized, (b) *Transfer input layer* strategy where the weights of common features are transferred, and the rest is initialized.

$X_S \neq X_T$. We define two heterogeneity cases: (1) the number of features in the source is larger than or equal to the target domain $|X_S| \geq |X_T|$, and (2) the number of features in the target is larger than or equal to the source domain $|X_S| \leq |X_T|$. The two cases are illustrated in Figure 2. In the result section, we examine, among other things, how much heterogeneity is acceptable while still getting a positive transfer gain.

IV. Approaches for Transfer Learning in Heterogeneous Scenarios

In this paper, we address scenarios for transfer learning for service performance prediction where we relax the homogeneity of features spaces in the source and target domains. Thus, we describe and compare two transfer learning approaches with regard to the heterogeneity in the input feature space. In addition to the heterogeneity aspect, we also elaborate on the impact of feed-forward neural network architecture choices. We vary the configuration in terms of the number of layers and neurons per layer. Further, in the result section, we investigate the impact of a varying number of available samples N_t (N at time t) in D_T, as the number of samples in the target domain may be limited due to overhead or time constraints.

A. Source and target models

The source and target feed-forward neural-network models can be generated separately using the available samples in their respective domains. The models are given as follows:

Source model (M_S): The model consists of an input layer, corresponding to features X_S, $n-1$ hidden layers $L_1, ..., L_{n-1}$, weights $w_1, ..., w_n$, and an output layer L_n corresponding to Y_t. The weights w_i for a model M_S are trained using backpropagation [22] with samples in the source domain D_S.

Target model (M_T): The same neural-network architecture is used in the target domain but all weights are randomly initialized, and trained using the available features (where $X_T \neq X_S$) and samples in the target domain. This model is thus trained with no prior knowledge and a limited, but increasing over time, set of samples.

A domain defines the input layer to the model, which is dependent on the number of available features. The traditional approach of homogeneous transfer learning does not consider the miss-match in the input layer, and thus other more advanced transfer approaches must be considered to cope with the heterogeneity.

B. Heterogeneous transfer approaches

We propose two approaches for transferring a feed-forward neural-network model from the source to the target domain. Both approaches are based on partial weight transfer but has significant differences, described below.

Random input layer ($M^r_{S\to T}$): This approach builds upon re-training of a feed-forward neural-network model M, that is transferred from a source to a target domain [8]. For heterogeneous transfer learning, the knowledge transfer corresponds to training a target model where the weights of layers $L_2, ..., L_n$ are initialized with the weights from the source model M_S. Further, layer L_1, which now has a different number of weights, is randomly initialized to encompass the feature heterogeneity as illustrated in Figure 3. After knowledge transfer, the model is denoted $M^r_{S\to T}$. The target model is then fine tuned using available samples from the target domain D_T.

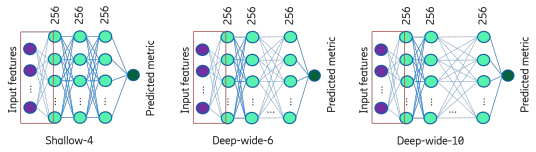
Transfer input layer ($M^t_{S\to T}$): In this approach, the model is transferred and trained similarly to $M^r_{S\to T}$, except for the handling of L_1. Removing and randomly initializing L_1 may lead to loss of encoded information, and thus, in this second approach the shared features across domains are identified, i.e., $X_S \cap X_T$. Once these features are identified, the position in the input feature vector is recorded. The weights in L_1 corresponding to the input feature positions are copied to the corresponding position in the target model. This results in a permutation-invariant weight-transfer approach. For the remaining target domain features, random initialization is performed. The approach is illustrated in Figure 3. An automatic mapping procedure will be investigated in future work.

The hypothesis is that the second approach leverages the fact that the domains are not completely different in terms of features. The more similar the domains are, the more effective the *transfer input layer* approach should be.

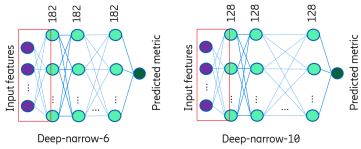
C. Quantifying the transfer gain

In addition to studying the performance of M_S, M_T, $M^r_{S\to T}$, and $M^t_{S\to T}$, we use the concept of *transfer gain* $\mathcal{G}$ to quantify the impact of D_T on the transferred models, which we define as:

$$\mathcal{G} = e_T - e_{S\to T}, \quad (1)$$



(a) Shallow and deep neural network alternatives



(b) Narrow deep neural network alternatives

Fig. 4. Neural-network model architectures considered in this paper.

where e_T and $e_{S\to T}$ are the model errors for M_T and $M_{S\to T}$, respectively. The transfer gain is inspired by the negative transfer gap defined in [23]. Transfer gain occurs when the loss of the transferred model $M_{S\to T}$ is higher than M_T, whereas a negative transfer gap means the opposite.

D. Neural network architectures

Heterogeneous transfer learning directly affects the first layer of the transferred model. This is due to the fact that the number of input features in the source domain is different from the target domain. In this subsection, we discuss the impact of the neural network architectures on the transfer gain in heterogeneous transfer learning. For this purpose, we examine two neural-network settings: 1) shallow vs. deep neural networks, 2) narrow vs. wide neural networks.

Shallow vs. deep neural network: To study the impact of neural network depth on the transfer gain, we examine three cases when the number of layers is 4, 6, and 10 as shown in Figure 4a. These networks are labelled shallow-4, deep-wide-6, and deep-wide-10. The number of parameters in each of these networks is listed in Table I. Our hypothesis is that deeper networks lead to larger transfer gain as more information is embedded in middle layers, which are not affected by the change in the input feature space. We also expect that there should be further gain as a result of increasing the network depth.

Narrow vs. wide neural network: Simply increasing the network depths (i.e., deep-wide neural networks) comes at the cost of increased number of parameters and training time. Thereby, we investigate the case where we keep the number of parameters similar to the shallow network, and just spread neurons over more layers, reaching a deep-narrow-6 and deep-narrow-10 neural network alternatives (Figure 4b). This configuration helps us to understand the impact of the number of parameters per layer on the transfer gain. The number of parameters for each network is listed in Table I.

V. Testbed and Data Traces

The evaluation in this paper is based on realistic traces [24] obtained from a testbed. A brief overview of the scenarios and experimental infrastructure are provided below, and additional details are available in our previous work [25].

TABLE I
Number of parameters in each of the networks employing 18 features in the source domain and 18 in the target domain.

Network	L_1 parameters	$L_2, ..., L_n$ parameters	Total parameters	Training time (sec)
shallow-4	4864	131841	136705	1.48
deep-wide-6	4864	263425	268289	1.62
deep-wide-10	4864	526593	531457	1.79
deep-narrow-6	3458	133407	136865	1.45
deep-narrow-10	2432	132225	134657	1.53

The testbed consists of a server cluster and a set of clients, all deployed on a rack with ten high-performance machines interconnected by a Gigabit Ethernet. On the server cluster we run two network services: Video-on-Demand (VoD) and Key-Value Store (KVS). The *VoD service* uses a modified VLC media player software, which provides single-representation streaming with a varying frame rate. Further, the *KVS service* uses the Voldemort software. The two services are installed on the same machines and can execute in parallel. The client machines act as load generators for both services.

A. Generating load on the testbed

Two load generators are running in parallel, one for the VoD application and another for the KVS application, aiming at emulating real traffic scenarios. The VoD load generator controls the number of active VoD sessions, spawning and terminating VLC clients. The KVS load generator controls the rate of KVS operations issued per second.

Both generators produce load according to two distinct load patterns: (1) *periodic-load* where the load generator produces requests following a Poisson process whose arrival rate is modulated by a sinusoidal function, and (2) *flashcrowd-load* where the requests are following a Poisson process whose arrival rate is modulated by a flashcrowd model [26].

B. Collected data and traces

Data traces, collected on the testbed, contain an input feature set X and the service-level metrics Y_{VoD} and Y_{KVS}. A trace is generated by extracting, and collecting, these statistics during execution of experiments with different configurations.

Features X are extracted from the Linux kernels that run on the machines. To access the kernel data, we use System Activity Report (SAR), a popular open-source Linux library [27], which provides approximately 1700 features per server. Examples of such statistics are CPU utilization per core, memory utilization, and disk I/O.

For the purpose of this paper, we focus on modeling the KVS service where the service-level metrics Y_{KVS} are measured on the clients. During an experiment two main metrics are captured, namely the Read Response Time (*RAvg*) as the average read latency for obtaining responses over a set of operations performed per second, and a corresponding

TABLE II
EXAMPLES OF TRACES OBTAINED FROM TESTBED.

Trace ID	Service(s)	Load pattern	Target Y	# samples
K1P	KVS	Periodic	*RAvg, WAvg*	28962
K1F	KVS	Flashcrowd	*RAvg, WAvg*	19444
K2P	KVS + VoD	Periodic	*RAvg, WAvg*	26488
K2F	KVS + VoD	Flashcrowd	*RAvg, WAvg*	24225

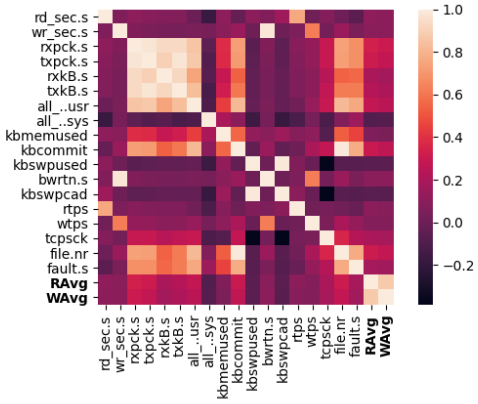


Fig. 5. Feature correlation with *RAvg* and *WAvg*.

metric for the Write Response Time (*WAvg*). The metrics are computed using a customized benchmark tool of Voldemort.

A summary of selected traces captured in the testbed are available in Table II. The trace ID is encoded according to service under investigation (KVS/VoD), number of concurrent services (1 or 2), and load pattern (periodic/flashcrowd).

C. Creation of heterogeneous traces

To study heterogeneous transfer learning for the scenarios considered in this paper, the set of features in the source and target domains must be different. In a first step, we reduced the feature space to 18 features, using domain knowledge, following the approach in [28]. Then we artificially reduce the number of available features utilizing the Pearson correlation (see Figure 5) between the features and *RAvg* and *WAvg*. Details of each feature are available in [27].

For the case with lower number of features in the target, features have been iteratively removed, starting with features with a higher correlation with the given task (*RAvg* or *WAvg*). In the case of having more features in the target domain, features in the target are selected from the least to the most correlated, with the intention of creating challenging scenarios. For instance, the most correlated feature with both *RAvg* and *WAvg* is *rxpck.s*, and the second most correlated ones are *file.nr* and *txpck.s*, respectively.

VI. RESULTS AND DISCUSSION

We evaluate the two approaches for heterogeneous transfer learning, explained in Section IV, on a set of transfer scenarios described in Table III, and quantify the impact of the feature heterogeneity, and architectural neural-network design choices.

TABLE III
TRANSFER SCENARIOS CONSIDERED. NOTE THAT $P(X_S) = P(X_T)$. MODEL ARCHITECTURES ARE IDENTICAL, EXCEPT FOR L_1.

#	D_S		D_T		Trace	Model arch.
	$\|X_S\|$	T_S	$\|X_T\|$	T_T		
1	18	*RAvg*	18, 10, 2	*WAvg*	K2P	$M_S = M_T$
2	2	*RAvg*	18, 10, 2	*WAvg*	K2P	$M_S = M_T$
3	18	*RAvg*	18, 10, 2	*WAvg*	K2F	$M_S = M_T$
4	2	*RAvg*	18, 10, 2	*WAvg*	K2F	$M_S = M_T$

Table III lists four main scenarios examined in this section. For each source domain (D_S) in Table III, multiple models (M_S) are created using the 5 different neural-network model architectures defined in Table I. The chosen neural-network model is assumed to be the same in the source and target domain (i.e., $M_S = M_T$). In scenarios concerning heterogeneous transfer learning, a target domain (D_T) corresponds to a change in feature space, i.e., $X_S \neq X_T$, and in this paper, we set the number of target-domain features to 18, 10, or 2. Further, the prediction task always changes, i.e., $T_S \neq T_T$. This corresponds to a scenario where the service provider for example would like to predict either read or write response time for a KVS service in a domain where the set of features has changed. The model performance is evaluated for an increasing set of samples in the target domain ($N = [25, 50, 75, 100, 250, 500]$).

The *Normalized Mean Absolute Error (NMAE)* is used for quantifying model performance, and is defined as:

$$e = \frac{1}{\bar{y}}\left(\frac{1}{m}\sum_{t=1}^{m}|y_t - \hat{y}_t|\right), \tag{2}$$

where $\hat{y}_t$ is the model prediction for the measured performance metric y_t, and $\bar{y}$ is the average of the samples y_t of the test set of size m. Other metrics lead to similar conclusions.

The neural network architectures considered in this work are listed in Table I and shown in Figure 4. Further, the implementation of the neural networks has been performed by means of the Keras library [29] running on top of TensorFlow [30]. For the source models, the neural networks are initialized with random weights and are trained on samples (X, Y). We use the rectified linear unit (ReLU) activation function for all the layers, Adam optimizer [31] with a learning rate starting at 0.001 using exponential decay with decay rate 85, 1000 decay steps and staircase function, and mean absolute error (MAE) as the loss function. Each experiment was run for a maximum of 200 epochs, with early stopping using a patience of 10 epochs and weight restoring. Regarding the data, a batch size of 32 was used, performing a validation split of 20% of the training set, composed by the 80% of the trace samples, being the test set constituted by the remaining 20%.

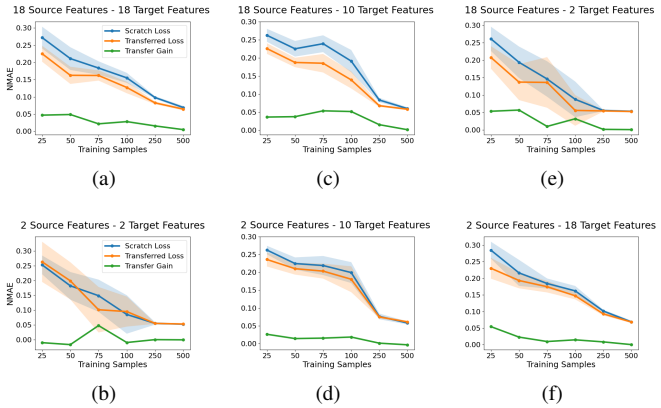


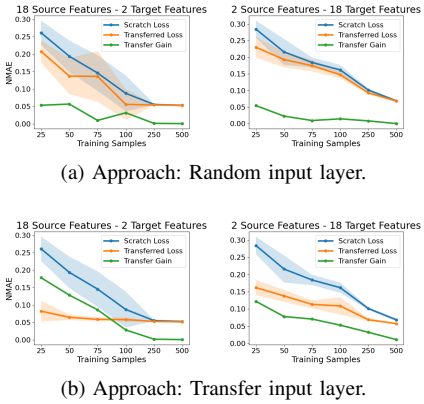
Fig. 6. NMAE for (a-b) homogeneous and (c-f) heterogeneous transfer learning in a shallow-4 network using the Random input layer approach.

Similarly, in the target domain, the samples are also split. The training set size is varied between 25 and 500 samples, and it is used to train the transferred model $M_{S \to T}$, using one of the proposed transfer approaches, and the scratch model M_T to enable calculation of transfer gain. Both models are trained and tested on the same samples, being the test size much bigger than the number of samples employed for the training in the target domain.

A. Homogeneous vs. heterogeneous transfer learning

In Figure 6, belonging to Scenario 1 and 2 in Table III, we report NMAE for different number of training samples where features in the source and target domains are either the same (homogeneous: Figure 6a and 6b) or different (heterogeneous: Figure 6c to 6f). In these results, the neural-network model is of type shallow-4, the trace is *K2P* (see Table II), and the prediction task is different in the source (*RAvg*) and target (*WAvg*) domains (i.e., $T_S \neq T_T$). *Random input layer* is the approach used for the heterogeneous transfer-learning cases. Each plot shows the transfer gain, transferred model loss, and scratch model loss. The scratch model (M_T) refers to a model that is rebuilt from scratch, that is a randomly initialized model trained with the available samples N_t in the target domain.

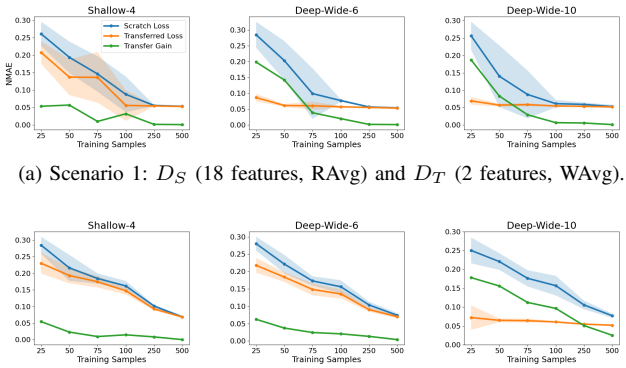
The results show that in most cases both homogeneous and heterogeneous transfer learning lead to a positive transfer gain. However, a small negative transfer gain can be seen in a few cases for low number of target samples, and when both domains contain only 2 features. As expected, the gain is reduced with increasing the number of samples in the target domain, which corroborates our previous results [5]. In the case of heterogeneous transfer learning, the transfer gain is larger when transferring from a model with a larger feature space (e.g., Figure 6c and 6e) compared to transferring from a smaller model (e.g., Figure 6d and 6f). This is due to the fact that more information can be obtained from the source domain, which can then be transferred to the target domain.



(a) Approach: Random input layer.

(b) Approach: Transfer input layer.

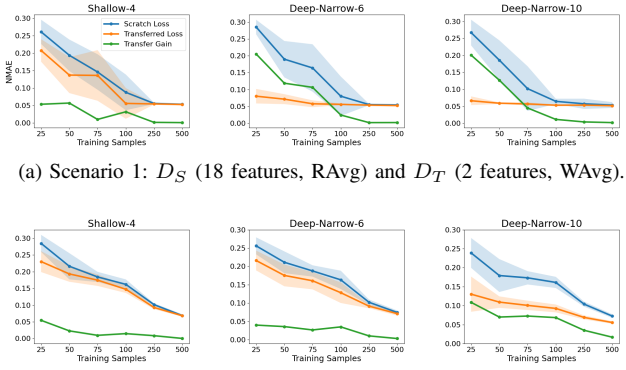
Fig. 7. NMAE for the two approaches using a shallow-4 neural network.



(a) Scenario 1: D_S (18 features, RAvg) and D_T (2 features, WAvg).

(b) Scenario 2: D_S (2 features, RAvg) and D_T (18 features, WAvg).

Fig. 8. NMAE for shallow-4 and deep-wide networks using the Random input layer approach.



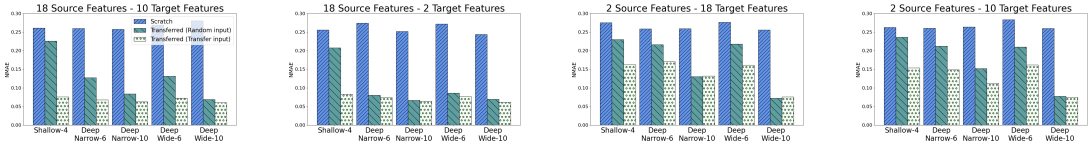
(a) Scenario 1: D_S (18 features, RAvg) and D_T (2 features, WAvg).

(b) Scenario 2: D_S (2 features, RAvg) and D_T (18 features, WAvg).

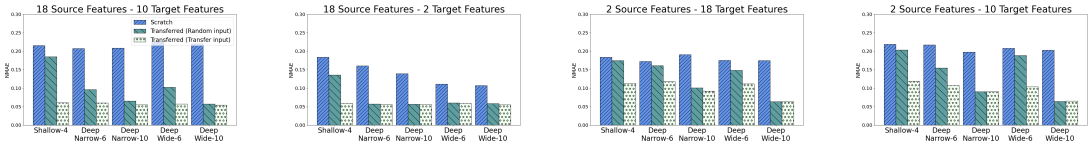
Fig. 9. NMAE for shallow-4 and deep-narrow networks using the Random input layer approach.

B. Random vs. transferred input layer strategies

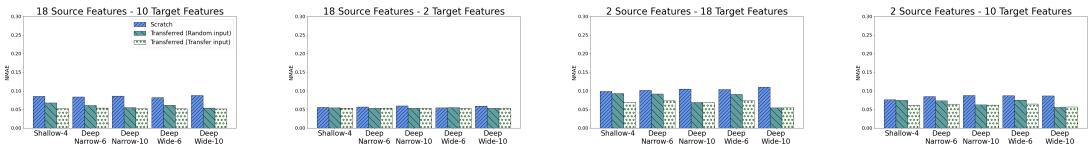
In Figure 7, belonging to Scenario 1 and 2 in Table III, we study the performance of the two heterogeneous transfer approaches discussed in Section IV. All the other configurations are similar to the previous experiments. In each plot, we calculate the model loss for the scratch and the transferred model, and report the transfer gain. As can be seen in this figure, larger transfer gain is obtained when using the



(a) 25 training samples

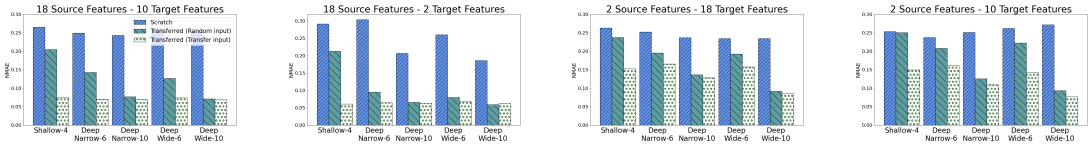


(b) 75 training samples

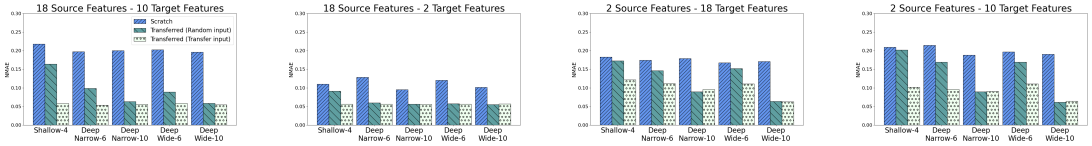


(c) 250 training samples

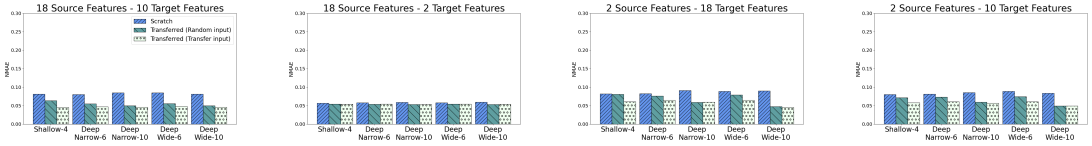
Fig. 10. NMAE comparison between the scratch model and the two proposed transfer approaches when the number of samples in the target domain is: (a) 25 (b) 75, and (c) 250. The experiments correspond to Scenario 1 and 2 with trace K2P. In the source domain (D_S), the number of features is 18 or 2 and the learning task is T_S = RAvg. In the target domain (D_T), the number of features is 18, 10, or 2 and the learning task is T_T = WAvg.



(a) 25 training samples



(b) 75 training samples



(c) 250 training samples

Fig. 11. NMAE comparison between the scratch model and the two proposed transfer approaches, similarly to Figure 10 but in this case using the K2F traces, corresponding to Scenarios 3 and 4.

transfer input layer strategy (Figure 7b) than when using the *random input layer* strategy (Figure 7a), specially for a lower number of samples. This observation is true for both cases of transferring from a smaller or lager feature set. The superiority of the *transfer input layer* strategy shows the importance of transferring the weights of the first layer whenever possible.

C. Shallow vs. deep neural networks

In Figure 8, corresponding to Scenarios 1 and 2 in Table III, we study the effectiveness of heterogeneous transfer learning using shallow and deep-wide neural network architectures, as stated in Table I. Each scenario is evaluated for three different configurations: shallow-4, deep-wide-6, and deep-wide-10. The source and target models have the same architecture. In addition, we fix the transfer strategy to *random input layer* in this set of experiments to be able to solely study the impact of the network architecture depth on the transfer gain.

Figure 8a shows the case where the number of features is reduced to 2 in the target domain, from 18 in the source domain, with varying prediction tasks. Figure 8b covers the opposite case. As can be observed, transfer gain increases with the neural-network depth, and a larger gain is obtained when

having a lower number of samples. The depth increment also helps to greatly reduce the variance of the transferred model in all cases. More interestingly, transfer gain in deep-wide-10 is still significant when transferring the model from only 2 features (Figure 8b). It is necessary to highlight that here, the number of features employed in the source domain highly influences the results obtained in deeper networks. In other words, the model that was trained in the source domain with more features (Figure 8a) has lower variance and reaches lower loss values using fewer training samples, as compared to the model trained with fewer features (Figure 8b).

By having 10 layers, the transferred model loss reaches its minimum even if only few samples are available in the target domain (e.g., 25). It implies that adding more layers does not help in improving the transfer gain any further.

D. Narrow vs. wide neural networks

Although the deep-wide neural network options offer several advantages in heterogeneous transfer learning, it comes at the cost of a large number of parameters, and thus a longer training time. To address this issue, in Figure 9 corresponding to Scenarios 1 and 2 in Table III, we investigate the impact of solely increasing the network depth while keeping the number of parameters in the same range as the shallow-4 network. Table I reports the average training time of the shallow, deep-wide, and deep-narrow neural networks over all cases in Figure 8 and 9. As can be seen in this table, the training time of the deep-narrow neural networks is close to that of the shallow one, while the training time in these networks is shorter than the deep-wide alternatives. Contrastingly, as shown in Figure 9, no significant changes in transfer gain have been observed as compared to the deep-wide alternatives (Figure 8).

E. Evaluation over multiple traces and approaches

In this section, we provide two comprehensive examples shown in Figures 10 and 11, experimenting over K2P and K2F traces, respectively. Figure 10 refers to Scenario 1 and 2 in Table III while Figure 11 belongs to Scenario 3 and 4. The figures compare different neural-network design alternatives proposed and discussed in this paper. NMAE is reported for all 5 neural network architectures over different number of samples ($N = [25, 75, 250]$) in the target domain. The two transferred strategies (i.e., *random input layer* and *transfer input layer*) are also compared against the scratch loss. The number of features in the source domain is selected to be 18 or 2 whereas it is 18, 10, and 2 in the target domain.

As can be seen in Figure 10, the impact of the *transfer input layer* strategy is more significant in shallow-4 network. The reason is that the first layer constitutes a large proportion of the total number of parameters. Thereby, a significant amount of knowledge will be lost when re-initializing the weights of the first layer as in the *random input layer* strategy. On the other hand, deeper networks would trap the knowledge in the middle and last layers of the neural network, which could be successfully transferred to the target domain. Thereby, we conclude that although a shallow neural network may work well in the source domain, it may not be the case in a heterogeneous scenario when the learned model is transferred to the target domain (to train model $M_{S\rightarrow T}$), especially when the *random input layer* strategy is applied. On the other hand, it is interesting to observe that the impact of the neural-network architecture choices is more limited when employing the *transfer input layer* strategy.

We also observe that, in most cases, wider neural network choices lead to lower NMAE. However, the advantage is limited compared to narrow alternatives which offer shorter training time due to a smaller number of parameters. Thereby, deep-narrow architectures would be better choices in terms of offering low NMAE while still having a short training time.

Another important observation is that the transfer gain is larger when transferring from a source domain with more features (e.g., 18 in the source and 2 in the target domain). However, interestingly, positive gain is also obtained in the opposite case (e.g., 2 in the source and 18 in the target domain).

Last but not the least, the transfer gain shrinks as the number of samples in the target domain increases, which we have already seen in multiple earlier examples.

We repeated the experiment on the K2F trace (Figure 11) and similar observations have been reached, clearly showing the applicability and generality of heterogeneous transfer learning in service metric predictions.

VII. Conclusion

In this paper, we proposed and evaluated two approaches for heterogeneous transfer learning, named *Random input layer* and *Transfer input layer*, for improved prediction of service performance in telecom networks. The approaches enable transfer of neural-network models, and thus knowledge, in-between environments with highly dissimilar feature sets.

The approaches were evaluated in multiple scenarios using realistic data sets obtained from a testbed running two different network services under varying load, namely a Video-on-Demand and Key-Value Store service. The obtained results provide empirical evidence showing a positive transfer gain in a majority of cases. For example, the knowledge embedded in a neural-network model, trained with 2 features in the source domain, can successfully be transferred to a target domain with 18 input features, and vise versa.

Further, a comprehensive evaluation of the two approaches show that the *Transfer input layer* often is more capable, especially with low availability of target-domain samples, and smaller model architectures. Choosing an appropriate neural-network depth is also crucial for transfer gain and shorter training time. Deeper networks are embedding more knowledge which results in a higher transfer gain in all scenarios.

Acknowledgment

This research has been supported by the Swedish Governmental Agency for Innovation Systems, VINNOVA, through projects ITEA3 AutoDC and Celtic ANIARA, and Swedish Foundation for Strategic Research, SSF Strategic Mobility.

REFERENCES

[1] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 1–99, 2018.

[2] J. Du, C. Jiang, J. Wang, Y. Ren, and M. Debbah, "Machine learning for 6g wireless networks: Carrying forward enhanced bandwidth, massive access, and ultrareliable/low-latency service," *IEEE Vehicular Technology Magazine*, vol. 15, no. 4, pp. 122–134, 2020.

[3] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[4] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019.

[5] F. Moradi, R. Stadler, and A. Johnsson, "Performance prediction in dynamic clouds using transfer learning," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 242–250.

[6] H. Larsson, J. Taghia, F. Moradi, and A. Johnsson, "Towards source selection in transfer learning for cloud performance prediction," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 599–603.

[7] ——, "Source selection in transfer learning for improved service performance predictions," in *2021 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2021.

[8] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *arXiv preprint arXiv:1411.1792*, 2014.

[9] L. Mou, Z. Meng, R. Yan, G. Li, Y. Xu, L. Zhang, and Z. Jin, "How transferable are neural networks in nlp applications?" *arXiv preprint arXiv:1603.06111*, 2016.

[10] Y. Shehu and R. Harper, "Towards improved fault localization using transfer learning and language modeling," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2020.

[11] R. Krishna, V. Nair, P. Jamshidi, and T. Menzies, "Whence to learn? transferring knowledge in configurable systems using beetle," *IEEE Transactions on Software Engineering*, 2020.

[12] R. Kumar, A. Mankodi, A. Bhatt, B. Chaudhury, and A. Amrutiya, "Cross-platform performance prediction with transfer learning using machine learning," in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2020, pp. 1–7.

[13] O. Day and T. M. Khoshgoftaar, "A survey on heterogeneous transfer learning," *Journal of Big Data*, vol. 4, no. 1, pp. 1–42, 2017.

[14] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International conference on artificial neural networks*. Springer, 2018, pp. 270–279.

[15] X. Shi, Q. Liu, W. Fan, S. Y. Philip, and R. Zhu, "Transfer learning on heterogenous feature spaces via spectral transformation," in *2010 IEEE international conference on data mining*. IEEE, 2010, pp. 1049–1054.

[16] J. Zhou, S. Pan, I. Tsang, and Y. Yan, "Hybrid heterogeneous transfer learning through deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, 2014.

[17] L. Duan, D. Xu, and I. Tsang, "Learning with augmented features for heterogeneous domain adaptation," *arXiv preprint arXiv:1206.4660*, 2012.

[18] S. Sukhija, N. C. Krishnan, and G. Singh, "Supervised heterogeneous domain adaptation via random forests," in *IJCAI*, 2016, pp. 2039–2045.

[19] W.-Y. Chen, T.-M. H. Hsu, Y.-H. H. Tsai, Y.-C. F. Wang, and M.-S. Chen, "Transfer neural trees for heterogeneous domain adaptation," in *European Conference on Computer Vision*. Springer, 2016, pp. 399–414.

[20] Z. Xia, L. Wang, W. Qu, J. Zhou, and Y. Gu, "Neural network based deep transfer learning for cross-domain dependency parsing," in *International Conference on Artificial Intelligence and Security*. Springer, 2020, pp. 549–558.

[21] T. Karb, N. Kühl, R. Hirt, and V. Glivici-Cotruta, "A network-based transfer learning approach to improve sales forecasting of new products," *arXiv preprint arXiv:2005.06978*, 2020.

[22] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1, no. 10.

[23] Z. Wang, Z. Dai, B. Póczos, and J. Carbonell, "Characterizing and avoiding negative transfer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 293–11 302.

[24] F. S. Samani, "Data traces for efficient learning on high-dimensional operational data," https://github.com/foroughsh/KTH-traces, 2021.

[25] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler, "A service-agnostic method for predicting service metrics in real time," *International Journal of Network Management*, vol. 28, no. 2, p. e1991, 2018.

[26] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. Long, "Managing flash crowds on the internet," in *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.* IEEE, 2003, pp. 246–249.

[27] "Sar, 2016. [online]. available: http://linux.die.net/man/1/sar."

[28] J. Ahmed, T. Josefsson, A. Johnsson, C. Flinta, F. Moradi, R. Pasquini, and R. Stadler, "Automated diagnostic of virtualized service performance degradation," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–9.

[29] "Keras, 2018. [online]. available: https://keras.io/."

[30] "Tensorflow, 2018. [online]. available: https://github.com/tensorflow/tensorflow."

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.