GENERALIZING DEEP MULTI-TASK LEARNING WITH HETEROGENEOUS STRUCTURED NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Many real applications show a great deal of interest in learning multiple tasks from different data sources/modalities with unbalanced samples and dimensions. Unfortunately, existing cutting-edge deep multi-task learning (MTL) approaches cannot be directly applied to these settings, due to either heterogeneous input dimensions or the heterogeneity in the optimal network architectures of different tasks. It is thus demanding to develop knowledge-sharing mechanism to handle the intrinsic discrepancies among network architectures across tasks. To this end, we propose a flexible knowledge-sharing framework for jointly learning multiple tasks from distinct data sources/modalities. The proposed framework allows each task to own its task (data)-specific network design, via utilizing a compact tensor representation, while the sharing is achieved through the partially shared latent cores. By providing more elaborate sharing control with latent cores, our framework is effective in transferring task-invariant knowledge, yet also being efficient in learning task-specific features. Experiments on both single and multiple data sources/modalities settings display the promising results of the proposed method, especially favourable in insufficient data scenarios.

1 INTRODUCTION

Multi-task learning (MTL) (Caruana, 1997; Maurer et al., 2016) is an approach for boosting the overall performance of each individual task by learning multiple related tasks simultaneously. In the deep learning setting, jointly fitting sufficiently flexible deep neural networks (DNNs) to data of multiple tasks can be seen as adding an inductive bias to the deep models, which can facilitate the learning of feature representations that are preferable by all tasks. Recently, the deep MTL has been successfully explored in a broad range of applications, such as computer vision (Zhang et al., 2014; Misra et al., 2016), natural language processing (Luong et al., 2015; Liu et al., 2017), speech recognition (Wu et al., 2015; Huang et al., 2015) and so on.

Nevertheless, one key challenge in deep MTL remains largely unaddressed, that is, almost all existing deep MTL approaches (Yang & Hospedales, 2017; Long et al., 2017) restrict themselves only to the setting of multi-label learning (or multi-output regression) (Zhang & Yang, 2017). In other words, different tasks must be fed with input data from the same source (or domain). Such requirement, however, seriously limits the applicability of those models to a more realistic scenario of deep MTL, where the tasks involve distinct data sources (domains) with unbalanced sample sizes or dimensions.

More specifically, tasks from some domains with abundant samples or small input dimensions are relatively easy to handle, whereas tasks from other domains are quite challenging due to the insufficient training data and large dimensionality. For instance, classifying hand-written digits (MNIST dataset (LeCun et al., 1998)) is somewhat similar to the recognition of hand-drawn characters (Omniglot dataset (Lake et al., 2015)). The Omniglot task is much harder than the MNIST task, as each character in Omniglot has only 20 training samples, while the input dimensionality is about 15 times larger than MNIST digit. As another example, predicting binary attributes (i.e., 'young', 'bald', 'receding hairline') from human face images (CelebA dataset (Liu et al., 2015)) ought to be related to the age group classification using human photos taken in the wild (Adience dataset (Eidinger et al., 2014)). The Adience task turns out to be the more difficult one since the wild images are not preprocessed and 7.6 times fewer than CelebA samples. Hence, it makes good sense to jointly learn these

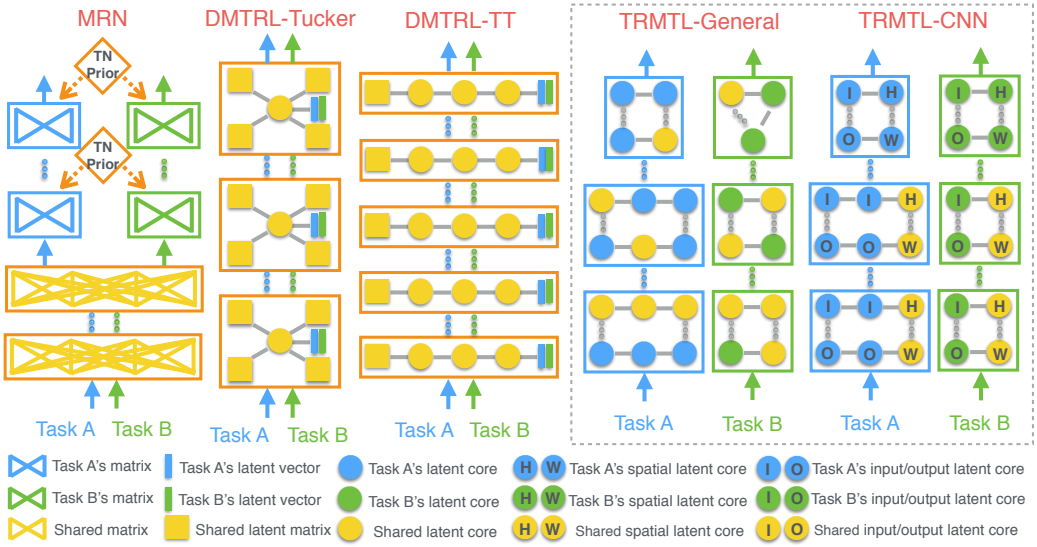


Figure 1: The overall sharing mechanisms of multilinear relationship network (MRN), two variants of deep multi-task representation learning (DMTRL) for CNN setting and our TRMTL (general setting and CNN setting) w.r.t. two tasks. The shared portion is depicted in yellow. MRN: original weights are totally shared at the lower layers and the relatedness between tasks at the top layers is modelled by tensor normal priors. DMTRL (TT or Tucker): all layer-wise weights must be equal-shape so as to be stacked and decomposed into factors. For each task, almost all the factors are shard at each layer except the very last 1D vector. Such pattern of sharing is identical at all layers. TRMTL (General): layer-wise weights are separately encoded into TR-formats for different tasks, and a subset of latent cores are selected to be tied across two tasks. The portions of sharing can be different from layer to layer. TRMTL (CNN): spatial cores (height and width cores) in the tensorized convolutional kernel are shared, while cores of input/output channels of the kernel are task-specific.

tasks to extract better feature representations, especially for the hard tasks, which could be achieved through transferring domain-specific knowledge from easy tasks.

Unfortunately, existing cutting-edge deep MTL models are only suited for the multi-label learning where different tasks share the same training inputs (i.e., $\mathbf{X}^i = \mathbf{X}^j$ for $i \neq j$, where $\mathbf{X}^i$ denotes the input for task $\mathcal{T}_i$), and thus cannot be directly applied to above learning scenarios. This is due to those models fail to provide knowledge-sharing mechanisms that can cope with the intrinsic discrepancies among network architectures across tasks. Such discrepancies either arise from the heterogeneous dimensions of input data or from the heterogeneous designs of layer-wise structures.

Conventionally, knowledge-sharing mechanisms of deep MTL can be hard or soft parameter sharing (Ruder, 2017). Hard sharing models (Zhang et al., 2014; Yin & Liu, 2017) share all parameters at the lower layers but with no parameters being shared at the upper layers across tasks. Soft sharing models (Duong et al., 2015; Yang & Hospedales, 2016; Long & Wang, 2015), on the other hand, learn one DNN per task with its own set of parameters, and the tasks are implicitly connected through imposing regularization terms on the aligned weights. The common issue with above mechanisms is that, for the sharing part, the network architectures of all tasks are strictly required to be identical. It turns out that some of the tasks have to compromise on a sub-optimal network architecture, which may lead to the deterioration in the overall performance. Ideally, at all potentially shared layers, each task should be capable of encoding both task-specific and task-independent portions of variation.

To overcome this limitation, we propose a latent-subspace knowledge-sharing mechanism that allows to associate each task with distinct source (domain) of data. By utilizing tensor representation, different portions of parameters can be shared via latent cores as common knowledge at distinct layers, so that each task can better convey its private knowledge. In this work, we realize our proposed framework via tensor ring (TR) format (Zhao et al., 2016) and refer it as tensor ring multi-task learning (TRMTL), as shown in Figure 1.

Our main contributions are twofold: (1) we offer a new distributed knowledge-sharing mechanism that can address the discrepancies of network architectures among tasks. Compared to existing deep MTL models that are only for multi-label learning, the joint learning of tasks from multi-datasets (multi-domains) with heterogeneous architectures becomes feasible. (2) we provide a TR-based implementation of the proposed framework, which further enhances the performance of deep MTL models in terms of both compactness and expressive power.

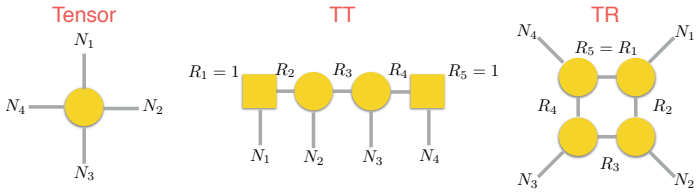


Figure 2: The diagrams of a 4th-order tensor in TT and TR-format.

2 Tensor Preliminaries

High-order tensors (Kolda & Bader, 2009) are referred to as multi-way arrays of real numbers. Let $\mathcal{W} \in \mathbb{R}^{N_1 \times \cdots \times N_D}$ be a Dth-order tensor in calligraphy letter, where D is called mode or way. Some original work have successfully applied tensor decompositions to applications such as imaging analysis and computer vision (Vasilescu & Terzopoulos, 2002; 2003). As a special case of tensor networks, the recent TR decomposition (Zhao et al., 2016) decomposes a tensor $\mathcal{W}$ into a sequence of 3rd-order latent cores that are multiplied circularly. An example of TR-format is illustrated in Figure 2. In TR-format, any two adjacent latent cores are 'linked' by a common dimension of size R_{k+1}, $k \in \{1, ..., D\}$. In particular, the last core is connected back to the first core by satisfying the border rank condition $R_{D+1} = R_1$. The collection of $[R_1, R_2, ..., R_D]$ is defined as TR-rank. Under TR-format, merely $\sum_{k=1}^{D} N_k R_k R_{k+1}$ parameters are needed to represent the original tensor $\mathcal{W}$ of size $\prod_{k=1}^{D} N_k$. Compared with tensor train (TT) format (Oseledets, 2011), TR generalizes TT by relaxing the border rank condition. (Zhao et al., 2016) concludes that TR is more flexible than TT w.r.t. low-rank approximation. The authors observe the pattern of ranks distribution on cores tend to be fixed in TT. In TT, the ranks of middle cores are often much larger than those of the side cores, while TR-ranks has no such drawbacks and can be equally distributed on cores. The authors find that, under the same approximation accuracy, the *overall ranks* in TR are usually much smaller than those in TT, which makes TR a more compact model than TT.

3 Methodology

Our general framework learns one DNN per task by representing the original weight of each layer with a tensor representation layer, i.e., utilizing a sequence of latent cores. Then, a subset of cores are tied across multiple tasks to encode the task-independent knowledge, while the rest cores of each task are treated as private cores for task-specific knowledge.

3.1 Tensor Representation Layer Implemented via Tensor Ring Format

We start the section by describing the tensor representation layer, which lays a groundwork for our deep MTL approach. Our TR-based implementation is called tensor ring representation layer (TRRL). Following the spirit of TT-matrix (Novikov et al., 2015) representation, TR is able to represent a large matrix more compactly via TR-matrix format. Specifically, let $\mathbf{W}$ be a matrix of size $M \times N$ with $M = \prod_{k=1}^{D} M_k, N = \prod_{k=1}^{D} N_k$, which can be reshaped into a Dth-order tensor $\mathcal{W} \in \mathbb{R}^{M_1 N_1 \times M_2 N_2 \cdots \times M_D N_D}$ via bijective mappings $\phi(\cdot)$ and $\psi(\cdot)$. The map $\phi(i) = (\phi_1(i), ..., \phi_D(i))$ transforms the row index $i \in \{1, ..., M\}$ into a D-dimensional vector index $(\phi_1(i), ..., \phi_D(i))$ with $\phi_k(i) \in \{1, ..., M_k\}$; similarly, the map $\psi(\cdot)$ converts the column index $j \in \{1, ..., N\}$ also into a D-dimensional vector index $(\psi_1(j), ..., \psi_D(j))$ where $\psi_k(j) \in \{1, ..., N_k\}$. In this way, one can establish a one-to-one correspondence between a matrix element $\mathbf{W}(i, j)$ and a tensor element $\mathcal{W}((\phi_1(i), \psi_1(j)), ..., (\phi_D(i), \psi_D(j)))$ using the compound index $(\phi_k(\cdot), \psi_k(\cdot))$ for mode $k \in \{1, ..., D\}$. We formulate the TR-matrix format as

$$
\begin{aligned}
\mathbf{W}(i,j) &= \mathcal{W}((\phi_1(i), \psi_1(j)), ..., (\phi_D(i), \psi_D(j))) \\
&= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_D=1}^{R_D} \mathcal{G}^{(1)}_{r_1, (\phi_1(i), \psi_1(j)), r_2} \mathcal{G}^{(2)}_{r_2, (\phi_2(i), \psi_2(j)), r_3} \cdots \mathcal{G}^{(D)}_{r_D, (\phi_D(i), \psi_D(j)), r_1} \\
&= \sum_{r_1=1}^{R_1} \mathbf{g}^{(1)\mathsf{T}}_{r_1}[(\phi_1(i), \psi_1(j))] \mathbf{G}^{(2)}[(\phi_2(i), \psi_2(j))] \cdots \mathbf{g}^{(D)}_{r_1}[(\phi_D(i), \psi_D(j))] \\
&= \mathrm{Tr}\{\mathbf{G}^{(1)}[(\phi_1(i), \psi_1(j))] \mathbf{G}^{(2)}[(\phi_2(i), \psi_2(j))] \cdots \mathbf{G}^{(D)}[(\phi_D(i), \psi_D(j))]\},
\end{aligned}
\tag{1}
$$

where 'Tr' is the trace operation. $\mathcal{G}^{(k)}$ denotes the kth latent core, while $\mathbf{G}^{(k)}[(\phi_k(i), \psi_k(j))] \in \mathbb{R}^{R_k \times R_{k+1}}$ corresponds to the $(\phi_k(i), \psi_k(j))$th slice matrix of $\mathcal{G}^{(k)}$. $\mathbf{g}_{r_1}^{(1)\mathsf{T}}[(\phi_1(i), \psi_1(j))]$ represents the r_1th row vector of the $\mathbf{G}^{(1)}[(\phi_1(i), \psi_1(j))]$ and $\mathbf{g}_{r_1}^{(D)}[(\phi_D(i), \psi_D(j))]$ is the r_1th column vector of $\mathbf{G}^{(D)}[(\phi_D(i), \psi_D(j))]$. Notice that the third line in equation 1 implies TRRL is more powerful than TT layer in terms of model expressivity, as TRRL can in fact be written as a sum of R_1 TT layers. In the deep MTL context, the benefits of tensorization in our TRRL are twofold: a sparser, more compact tensor network format for each task and a potentially finer sharing granularity across the tasks.

3.2 The Proposed Knowledge-Sharing Framework

3.2.1 The general formulation

Our sharing strategy is to partition each layer's parameters into task-independent TR-cores as well as task-specific TR-cores. More specifically, for some hidden layer of an individual task $t \in \{1, ..., T\}$, we begin with reformulating the layer's weights $\mathbf{W}_t \in \mathbb{R}^{U_t \times V_t}$ in terms of TR-cores by means of TRRL, where $U_t = \prod_{k=1}^{D_t} U_t^k, V_t = \prod_{k=1}^{D_t} V_t^k$. We thereafter reshape a layer's input $\mathbf{h}_t \in \mathbb{R}^{U_t}$ into a D_tth-order tensor $\mathcal{H}_t \in \mathbb{R}^{U_t^1 \times \cdots \times U_t^{D_t}}$. Next, the layer's input tensor $\mathcal{H}_t$ can be transformed into layer's output tensor $\mathcal{Y}_t \in \mathbb{R}^{V_t^1 \times \cdots \times V_t^{D_t}}$ via $\mathcal{W}_t$ in TR-format. Finally, we have

$$\mathcal{Y}_t(v_1, ..., v_{D_t}) = \sum_{u_1=1}^{U_1} \cdots \sum_{u_{D_t}=1}^{U_{D_t}} \mathcal{H}_t(u_1, ..., u_{D_t}) \mathrm{Tr}\{\mathbf{G}_{com}^{(1)}[(u_1, v_1)] \cdots \mathbf{G}_t^{(p)}[(u_p, v_p)] \cdots \mathbf{G}_{com}^{(q)}[(u_q, v_q)] \cdots \mathbf{G}_t^{(r)}[(u_r, v_r)] \cdots \mathbf{G}_{com}^{(D_t)}[(u_{D_t}, v_{D_t})]\}, \quad (2)$$

where the common TR-cores subset $\{\mathbf{G}_{com}^{(\cdot)}\}$ has C elements which can be arbitrarily chosen from the set of all D_t cores, leaving the rest cores $\{\mathbf{G}_t^{(\cdot)}\}$ as task-specific TR-cores. Pay attention that our TRMTL neither restricts on which cores to share, nor restricts the shared cores to be in a consecutive order. Finally, we reshape tensor $\mathcal{Y}_t$ back into a vector output $\mathbf{y}_t \in \mathbb{R}^{V_t}$. Note that the portion of sharing, which is mainly measured by C, can be set to different values from layer to layer. According to equation 2, TRMTL represents each weight element in weight matrix as function of a sequence product of the slice matrices of the corresponding shared cores and private cores. Intuitively, this strategy suggests the value of each weight element is partially determined by some common latent factors, and meanwhile, also partially affected by some private latent factors. Thus, our sharing is carried out in an distributed fashion. This is more efficient than conventional sharing strategies in which each weight element is either 100% shared or 100% not shared.

Although we describe our general framework in terms of TR format, it is straightforward to implement our framework with other tensor network representations, such as Tucker (Tucker, 1966), TT (Novikov et al., 2015), projected entangled pair states (PEPS) and entanglement renormalization ansatz (MERA) (Cichocki et al., 2016), as long as each layer-wise weight matrix is tensorized and decomposed into a sequences latent cores.

3.2.2 Special case for CNN setting

Our model can be easily extended to convolutional kernel $\mathcal{K} \in \mathbb{R}^{H \times W \times I \times O}$, where $H \times W$ is the spatial sizes and I and O are the input and output channels. Note that here TRRL is similar to TR based weight compression Wang et al. (2018), but we use a different 4th-order latent cores in TR-matrix. As one special case of our general framework (TRMTL-CNN), we just share the spatial cores (height cores and width cores) in the tensorized kernel (via TRRL), while cores corresponding to input/output channels may differ from task to task:

$$\mathcal{K}_t(h, w, i, o) = \mathrm{Tr}\{\mathbf{G}_{com}^{(1)}[(h_1, w_1)] \cdots \mathbf{G}_{com}^{(C)}[(h_C, w_C)] \mathbf{G}_t^{(1)}[(i_1, o_1)] \cdots \mathbf{G}_t^{(D_t)}[(i_{D_t}, o_{D_t})]\}, \quad (3)$$

where C is typically 1 for small-sized spatial dimensions. Thus, there is no need to specify how many and which cores to share for TRMTL-CNN.

Table 1: Performance and model complexity comparison on MNIST dataset.

Samples 1800 vs 1800	STL		MRN		DMTRL-Tucker		DMTRL-TT		Ours-410		Ours-420	
	A	B	A	B	A	B	A	B	A	B	A	B
Accuracy	96.8	96.9	96.4	96.6	95.2	96.2	96.2	96.7	**97.5**	**97.7**	97.4	97.6
Number of parameters	6,060K		3,096K		1,194K		1,522K		16K		**13K**	

4 EXPERIMENTAL RESULTS

4.1 EXPERIMENTAL SETTINGS

We compare our TRMTL with single task learning (STL), MRN (Long et al., 2017), two variants of DMTRL (Yang & Hospedales, 2017). We repeat the experiments 5 times and record the average accuracy. The detailed settings and more experimental results are in the supplementary material.

4.1.1 TENSOR REPRESENTATION

Before the sharing, we first tensorize the layer-wise weight into a Dth-order tensor, whose D modes have roughly the same dimensionality, such that the cores have approximately equal sizes if we assume the same TR-ranks. In this manner, we may measure the fraction of knowledge sharing by the number of shared cores. D is empirically set to be from 4 to 6 in most of our tests. For simplicity, we assume TR-ranks to be identical for all TR-cores across layers for each task. We choose TR-ranks by cross validation on a range of values among 5, 10 and 15. Note that there is no tensorization step in DMTRL in (Yang & Hospedales, 2017), and DMTRL selects TT-ranks via tensor decomposition according to some specified threshold.

4.1.2 SHARED CORES SELECTION

Our general TRMTL is highly flexible as we impose no restrictions on which cores to be shared and where to share across tasks. In practical implementation, we may need to trade-off some model flexibility for the ease of sharing-pattern selection by introducing some useful prior knowledge about the domain. For instance, many vision tasks tend to share more cores at the lower layers than upper layers. There are various strategies on how to select the shard cores w.r.t. both the location and the number. Authors of (Zhao et al., 2017) discover that distinct cores control an image at different scales of resolution. The authors demonstrate this by decomposing a tensorized 2D image and then adding noise to one specific core at a time. They show the core in the first location controls small-scale patches while the core in the last location influences on large-scale partitions. Motivated by this, under the general formulation 3.2.1, we preferentially share the features from the detailed scale to the coarse scale, which means we follow a natural left-to-right order in location to select different C number of cores at distinct layers. C is needed to tune via cross validation. In practice, we apply a greedy search on C layer by layer to effectively reduce the searching space. Another practical option is to prune the searching space by following the very useful guidance that C tends to decrease as the layers increase. For certain CNN based architectures, we adopt the special case TRMTL-CNN. Since the cores produced by tensorized convolutional kernel have their specific roles, we just share the cores that are associated to the spatial dimensions (height and width cores), leaving input/output cores being task-specific. In our tests, C is just 1 due to the small spatial kernels, thus eliminating the need of the tuning of this hyper-parameter.

4.2 DEEP MTL WITH SINGLE DOMAIN SOURCE

We begin our test with data from single domain source to validate the basic properties of our model.

4.2.1 EFFECT OF SHARING PATTERNS

Our first validation test is conducted on MNIST, where the `task A` is to classify the odd digits and the `task B` is to classify the even ones. To see how sharing styles and hyper-parameter C can affect the performance, we examine various patterns from three representative categories, as shown in Figure 3. For instance, the patterns in 'bottom-heavy' category mean more parameters are shared at the bottom layers than the top layers, while 'top-heavy' indicates the opposite style. The validation is conducted on MNIST using multi-layer perceptron (MLP) with three tensorized hidden layers, each of which is encoded using 4 TR-cores. The pattern '014', for example, means the C are 0, 1 and 4 from lower to higher layers, respectively. We gauge the transferability between tasks with unbalanced training samples by the averaged accuracy on the small-sample tasks. Clearly, the

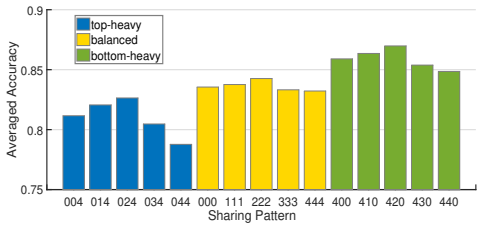


Figure 3: Validation test w.r.t sharing patterns and 'C'. The averaged accuracy of two tasks involved with 50 samples. The training samples for `task A` vs `task B` are '1800 vs 50' and '50 vs 1800'.

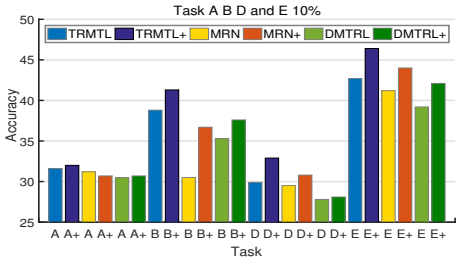


Figure 4: Results of accuracy changes of `task A`, `B`, `D` and `E`, when the faction of the data for training for `task C` is increased from 10% to 90%. '+' corresponds to the results after the samples augmentation of `task C`. 10% data for training for `task A`, `B`, `D` and `E`. The best pattern of TRMTL is '432'.

'bottom-heavy' patterns achieve significantly better results than those from the other two categories. The pattern '420' is reasonable and obviously outperforms the pattern '044' in Figure 3, since '044' overlaps all weights at the top layers but shares nothing at the bottom layer. Within each category, TRMTL is robust to small perturbation of C for pattern selection, both '410' and '420' obtain similarly good performance.

4.2.2 Effect of model complexity

We also examine the complexity of the compared models on MNIST. In Table 1, STL and MRN have enormous $6,060$K and $3,096$K parameters, since they share weights in the original space. DMTRL-Tucker and TT with pre-train trick (Yang & Hospedales, 2017) are parameterized by $1,194$K and $1,522$K parameters. In contrast, TRMTL achieves the best accuracies while the numbers of parameters are significantly down to 16K and 13K. The huge reduction is due to the tensorization and the resulting more sparser TRRL with overall lower ranks.

4.2.3 Effect of unbalanced sample sizes

Our next validation is carried out on Omniglot dataset (Krizhevsky & Hinton, 2009) to verify efficacy of knowledge transfer from data-abundance tasks to data-scarcity ones within one source of data domain. Omniglot data Lake et al. (2015) consists of $1,623$ unique characters from 50 alphabets with resolution of 105×105. We divide the whole alphabets into 5 tasks (`task A` to `task E`), each of which links to the alphabets from 10 languages. We now test a more challenging case, where only 1 task (`task C`) has sufficient samples while the samples of the other 4 tasks are limited. Figure 4 demonstrates the amount of the accuracy changes for each task, both with and without the aid of the data-rich task. We observe our TRMTL is able to make the most of the useful knowledge from `task C` and significantly boosts the accuracies of all other tasks.

4.2.4 Validation on recurrent neural networks

In our last validation, we like to explore whether the proposed sharing mechanism also works for recurrent neural networks. Hence, we test on UFC11 dataset (Liu et al., 2009) that contains $1,651$ Youtube video clips from 11 actions, which are converted to the resolution of $120 \times 180 \times 3$. We assign 5 actions ('basketball', 'biking', 'diving', 'golf swinging' and 'horse back riding') to the `task A` and leave the rest 6 actions ('soccer juggling', 'swinging', 'tennis swinging', 'trampoline jumping', 'volleyball spiking' and 'walking') as the `task B`. The RNN is implemented using one-layer long short-term memory (LSTM) with input length of 190. The weights corresponding to the input video are tensorized and encoded into 4 TR-cores, whose input and output dimensions are $[8, 20, 20, 18]$ and $[4, 4, 4, 4]$, respectively. The TR-rank is set to $[2, 2, 2, 2, 2]$. Only one layer of cores need to be shared and they are shared in a left-to-right order. The recognition precisions w.r.t. the number of shared cores are recorded in Table 2. We find that sharing TR-cores between tasks via our TRMTL significantly improves the performance comparing to no sharing case, and sharing all 4 TR-cores achieves the best results for this RNN situation.

4.3 Deep MTL with Multiple Domain Sources

In this section, we show the key advantage of our method in handling multiple tasks defined on distinct data domains, where the optimal network architectures of the tasks could be different.

Table 2: Validation of TRMTL for RNN on UFC11.

UFC11	Number of Shared Cores				
	0	1	2	3	4
Task A (5 actions)	58.5	55.4	55.5	61.2	**63.1**
Task B (6 actions)	48.0	54.9	56.6	56.0	**58.3**

Table 3: Specification of network architecture of TRMTL for the Omiglot-MNIST datasets.

Layer		Omniglot (105×105)	MNIST (28×28)
FC1	in modes	[7, 7, 5, 5, 3, 3]	[7, 7, 4, 4]
	out modes	[7, 7, 5, 5, 3, 3]	[7, 7, 5, 5]
FC2	in modes	[7, 7, 5, 5, 3, 3]	[7, 7, 5, 5]
	out modes	[7, 7, 5, 5, 3, 3]	[7, 7, 5, 5]
FC3	in modes	[7, 7, 5, 5, 3, 3]	[7, 7, 5, 5]
	out modes	[2, 2, 2, 2, 2, 2]	[2, 2, 2, 2]
FC4	in modes	[64]	[16]
	out modes	[10]	[10]

4.3.1 Results on heterogeneous MLPs

We first verify on Omniglot and MNIST combination, where `task A` is to classify hand-drawn characters from first 10 alphabets, while `task B` is to recognize 10 hand-written digits. `Task A` is much harder than `task B`, as each character in `task A` has a very fewer training samples (only 20 per character). Table 3 shows the architecture specification of TRMTL using 4 layers MLP, we can see `task A` and `task B` possess their respective layer-wise network structures, while different portions of cores could be partially shared across layers. In contrast, to apply DMTRL, one has to first convert the heterogeneous inputs into equal-sized features using *one hidden layer with totally unshared weights*, so that the weights in following layers with same shape can be stacked up. In Table 4, TRMTL obtains similar results to its competitors for the easier MNIST task, while both TRMTL-200 and 211 significantly outperform STL and DMTRL by a large margin for the more difficult Omniglot task. The poor performance of DMTRL due to its architecture's not being able to share any feature at the bottom hidden layer but has to share almost all the features at upper layers.

Table 4: Performance comparison of STL, DMTRL and our TRMTL on Omniglot-MNIST datasets.

Omniglot A vs MNIST B	STL		DMTRL-Tucker		DMTRL-TT		TRMTL-200		TRMTL-211	
	A	B	A	B	A	B	A	B	A	B
50% vs 100%	55.0	98.1	47.3	**98.5**	50.0	98.4	58.4	98.3	**59.7**	98.3
70% vs 100%	60.5	98.1	46.1	98.3	50.9	**98.6**	62.9	98.3	**65.4**	98.3
100% vs 100%	63.3	98.1	50.7	**98.5**	52.3	**98.5**	66.8	98.3	**67.5**	98.3

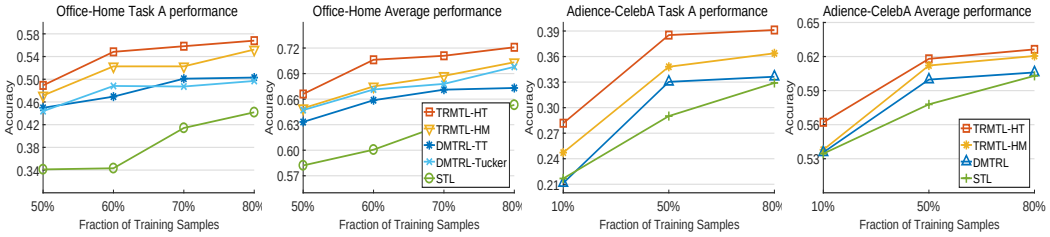


Figure 5: Left two figures: Performance comparison on Home-Office datasets with different fractions of training data. Right two figures: Performance comparison on Adience-CelebA datasets. In all figures, TRMTL-HT and TRMTL-HM correspond to the heterogeneous architectures and homogeneous architectures, respectively.

We also conduct experiments on the challenging Office-Home dataset (Venkateswara et al., 2017) to evaluate the effectiveness of TRMTL in handling data from distinct domains. The dataset contains over 10,000 images collected from different domains including *Art*, *Clipart*, *Product* and *Real World*, which forms `task A` to `task D`, respectively. Each task is assigned to recognize 65 object categories presenting in offices or homes. The image styles and the levels of difficulty vary from task to task, e.g., images from *Product* (`task C`) have empty backgrounds while images of *Art* (`task A`) have complicated backgrounds. We train three FC layers on the features extracted from images of each task using pre-trained VGG-16 (Simonyan & Zisserman, 2014). In Figure 5, our TRMTL variants consistently outperform other competitors by a large margin, i.e., over 5% in accuracy for the toughest `task A` when 80% samples are available. The noticeable improvements are mainly credited to our sharing mechanism, which effectively shares the common signature of object identity across tasks regardless of their individual image styles. For TRMTL, we observe TRMTL-HT exceeds TRMTL-HM by at least 2% in the averaged accuracy and by 1% in the hardest `task A`, showing the efficacy of employing non-identical architectures on sharing high-level features.

4.3.2 Results on heterogeneous CNNs

To further illustrate the merit of sharing knowledge using heterogeneous architectures, we next apply our TRMTL directly to the raw images via CNNs. We test on two large-scale human face datasets:

Table 5: Heterogeneous (left) and homogeneous (right) network architectures for Adience-CelebA datasets.

Layer	Adience (227×227×3) in/out modes	Adience (227×227×3) win/stride	CelebA (218×178×3) in/out modes	CelebA (218×178×3) win/stride
Conv1	[3] [6, 4, 4]	[7, 7] / 4	[3] [4, 4, 4]	[7, 7] / 2
Conv2	[6, 4, 4] [4, 4, 4, 4]	[5, 5] / 1	[4, 4, 4] [4, 4, 4, 2]	[5, 5] / 2
Conv3	[4, 4, 4, 4] [4, 4, 4, 4]	[3, 3] / 2	[4, 4, 4, 2] [4, 4, 4, 4]	[3, 3] / 2
FC1	[8, 8, 4, 4, 4] [4, 4, 4, 4, 2]		[164864] [40]	
FC2	[4, 4, 4, 4, 2] [4, 4, 4, 4, 2]			
FC3	[512] [8]			

Layer	Adience (227×227×3) in/out modes	Adience (227×227×3) win/stride	CelebA (218×178×3) in/out modes	CelebA (218×178×3) win/stride
Conv1	[3] [4, 4, 4]	[7, 7] / 2	[3] [4, 4, 4]	[7, 7] / 2
Conv2	[4, 4, 4] [4, 4, 4, 2]	[5, 5] / 2	[4, 4, 4] [4, 4, 4, 2]	[5, 5] / 2
Conv3	[4, 4, 4, 2] [4, 4, 4, 4]	[3, 3] / 2	[4, 4, 4, 2] [4, 4, 4, 4]	[3, 3] / 2
FC1	[8, 8, 4, 4, 4] [4, 4, 4, 4, 2]		[164864] [40]	
FC2	[4, 4, 4, 4, 2] [4, 4, 4, 4, 2]			
FC3	[512] [8]			

Adience Eidinger et al. (2014) and CelebA Liu et al. (2015). Adience dataset contains $26,580$ unfiltered 227×227 face photos taken in the wild with variation in appearance, pose, lighting and etc; CelebA has a total number of $202,599$ preprocessed face images of resolution 218×178. For this test, the `task A` assigned to Adience data is to predict the label of age group that a person belongs to (8 classes), and we associate the `task B` to CelebA data to classify 40 binary facial attributes. Note that `task A` is much harder than `task B`, as the number of samples in Adience (with face images in the wild) is about 7.6 times fewer than that of CelebA (with cropped and aligned face images). Since Adience bears similarity to CelebA, we are interested to see whether the performance of the tough task (`task A`) can be enhanced by jointly learning on two domains of data. The heterogeneous architectures of TRMTL-CNN are shown in Table 5, in which we adopt the special case of TRMTL by sharing the spatial cores (e.g., [7, 7], [5, 5] and [3, 3]) in convolutional kernel yet preserving the differences w.r.t. input and output channel cores.

We focus on comparing the heterogeneous case with the homogeneous case in Table 5 where the shared structures are identical. As expected, in Figure 5, our TRMTL significantly outperforms other methods on the hard `task A`. In the meantime, TRMTL obtains the best averaged accuracies on two tasks in nearly all cases, indicating the data-scarcity `task A` has little harmful impact on the data-abundant `task B`. For our TRMTL, we also observe TRMTL-HM exhibits worse accuracies than TRMTL-HT, which implies that a comprise on an identical CNN design for all tasks, such as input/output channel core and stride size, lead to deteriorated overall performance. The test also shows the effectiveness of TRMTL in sharing low-level features with heterogeneous architectures.

5 Discussion

Our general TRMTL framework relies on the manual selection of shared cores, i.e., one need to specify the number of shared cores C at each layer if we choose to share the cores in a left-to-right order across tasks. Although we can employ some efficient heuristics, the search space of this hyper-parameter may grow rapidly as number of the layers increase. Besides the greedy search, a more sophisticated and possible option is to automatically select sharable core pairs that have the highest similarity. We may consider two cores as a candidate pair if the same perturbation of the two cores induces similar changes in the errors of respective tasks. In this way, one can adaptively select most similar cores from tasks according to a certain threshold, leaving the rest as private cores.

We should also point out that tensorization operation plays a key role in our proposed sharing mechanism. Due to the tensorization, the cores can be shared in a much finer granularity via our TRMTL framework. Furthermore, tensorizing weight matrix into high-order weight tensor yields more compact tensor network format (with much lower overall ranks), and thus a higher compression ratio for parameters. In contrast, DMTRL tends to produce a lot more parameters without tensorization.

6 Conclusion

In this work, we have extended the conventional deep MTL to a broader paradigm where multiple tasks may involve more than one source data domain. To resolve the issues caused by the discrepancies among different tasks' network structures, we have introduced a novel knowledge sharing framework for deep MTL, by partially sharing latent cores via tensor network format. Our method is empirically verified on various learning settings and achieves the state-of-the-art results in helping tasks to improve their overall performance.

REFERENCES

Rich Caruana. Multitask learning. *Machine Learning*, 1997.

Xiao Chu, Wanli Ouyang, Wei Yang, and Xiaogang Wang. Multi-task recurrent neural network for immediacy prediction. In *International Conference on Computer Vision*, pp. 3352–3360, 2015.

Andrzej Cichocki, Namgil Lee, Ivan V Oseledets, A-H Phan, Qibin Zhao, and D Mandic. Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: Perspectives and challenges part 1. *arXiv preprint arXiv:1609.00893*, 2016.

Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Annual Meeting of the Association for Computational Linguistics and International Joint Conference on Natural Language Processing*, volume 2, pp. 845–850, 2015.

Eran Eidinger, Roee Enbar, and Tal Hassner. Age and gender estimation of unfiltered faces. *IEEE Transactions on Information Forensics and Security*, 9(12):2170–2179, 2014.

Zhen Huang, Jinyu Li, Sabato Marco Siniscalchi, I-Fan Chen, Ji Wu, and Chin-Hui Lee. Rapid adaptation for deep neural networks through multi-task learning. In *Conference of the International Speech Communication Association*, 2015.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3): 455–500, 2009.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Machine learning*, 2009.

Abhishek Kumar and Hal Daume III. Learning task grouping and overlap in multi-task learning. *arXiv preprint arXiv:1206.6417*, 2012.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Yann LeCun, Corinna Cortes, and Christopher JC Burges. The MNIST database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

Jingen Liu, Jiebo Luo, and Mubarak Shah. Recognizing realistic actions from videos. 2009.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Adversarial multi-task learning for text classification. In *Association for Computational Linguistics*, volume 1, pp. 1–10, 2017.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *International Conference on Computer Vision*, 2015.

Mingsheng Long and Jianmin Wang. Learning multiple tasks with deep relationship networks. *arXiv preprint arXiv:1506.02117*, 2015.

Mingsheng Long, Zhangjie Cao, Jianmin Wang, and S Yu Philip. Learning multiple tasks with multilinear relationship networks. In *Advances in Neural Information Processing Systems*, pp. 1594–1603, 2017.

Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

Andreas Maurer, Massimiliano Pontil, and Bernardino Romera-Paredes. The benefit of multitask representation learning. *The Journal of Machine Learning Research*, 17(1):2853–2884, 2016.

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Conference on Computer Vision and Pattern Recognition*, pp. 3994–4003. IEEE, 2016.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Wanli Ouyang, Xiao Chu, and Xiaogang Wang. Multi-source deep learning for human pose estimation. In *Conference on Computer Vision and Pattern Recognition*, pp. 2329–2336, 2014.

Bernardino Romera-Paredes, Hane Aung, Nadia Bianchi-Berthouze, and Massimiliano Pontil. Multilinear multitask learning. In *International Conference on Machine Learning*, pp. 1444–1452, 2013.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3): 279–311, 1966.

M Alex O Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision*, pp. 447–460. Springer, 2002.

M Alex O Vasilescu and Demetri Terzopoulos. Multilinear subspace analysis of image ensembles. In *null*, pp. 93. IEEE, 2003.

Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *(IEEE) Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Wenqi Wang, Yifan Sun, Brian Eriksson, Wenlin Wang, and Vaneet Aggarwal. Wide compression: tensor ring nets. In *Conference on Computer Vision and Pattern Recognition*, pp. 13–31. IEEE, 2018.

Kishan Wimalawarne, Masashi Sugiyama, and Ryota Tomioka. Multitask learning meets tensor factorization : task imputation via convex optimization. In *Advances in Neural Information Processing Systems*, pp. 2825–2833, 2014.

Zhizheng Wu, Cassia Valentini-Botinhao, Oliver Watts, and Simon King. Deep neural networks employing multi-task learning and stacked bottleneck features for speech synthesis. In *International Conference on Acoustics, Speech and Signal Processing*, pp. 4460–4464. IEEE, 2015.

Yongxin Yang and Timothy Hospedales. Deep multi-task representation learning: a tensor factorisation approach. In *International Conference on Learning Representations*, 2017.

Yongxin Yang and Timothy M Hospedales. Trace norm regularised deep multi-task learning. *arXiv preprint arXiv:1606.04038*, 2016.

Xi Yin and Xiaoming Liu. Multi-task convolutional neural network for pose-invariant face recognition. *arXiv preprint arXiv:1702.04710*, 2017.

Yu Zhang and Qiang Yang. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European Conference on Computer Vision*, pp. 94–108. Springer, 2014.

Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535*, 2016.

Qibin Zhao, Masashi Sugiyama, Longhao Yuan, and Andrzej Cichocki. Learning efficient tensor representations with ring structure networks. *Workshop of International Conference on Learning Representations*, 2017.

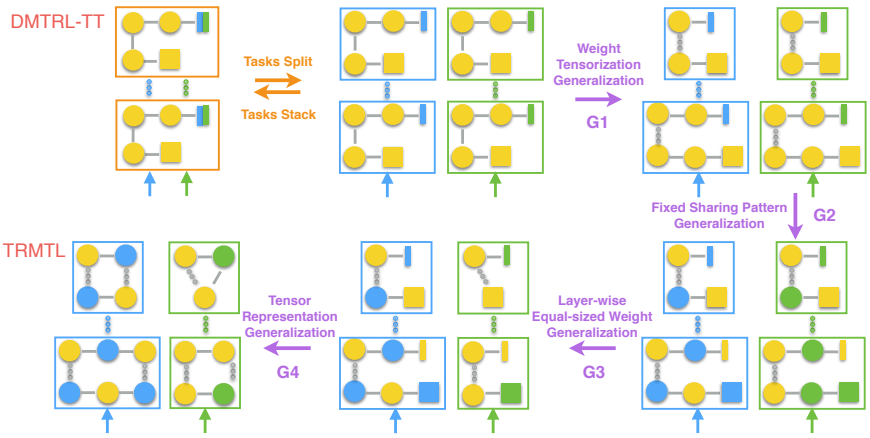


Figure 6: The demonstration of four types generalizations from DMTRL-TT to our TRMTL.

A Related Work

The classical matrix factorization based MTL (Kumar & Daume III, 2012; Romera-Paredes et al., 2013; Wimalawarne et al., 2014) requires the dimensionality of weight vectors $\{\mathbf{w}_t \in \mathbb{R}^M\}_{t=1}^T$ of T tasks to be equal-sized, so that these weights could be stacked up into one weight matrix $\mathbf{W} \in \mathbb{R}^{M\times T}$. The work (Kumar & Daume III, 2012) assumes $\mathbf{W}$ to be low-rank and factorizes it as $\mathbf{W} = \mathbf{LS}$. Here, $\mathbf{L} \in \mathbb{R}^{M\times K}$ consists of K task-independent latent basis vectors, whereas each column vector of $\mathbf{S} \in \mathbb{R}^{K\times T}$ is task-specific and contains the mixing coefficients of these common latent bases. Yang & Hospedales (2017) extended this to its tensorial counterpart deep multi-task representation learning (DMTRL) by making use of tensor factorization. Likewise, DMTRL starts by putting the equal-shaped weight matrices $\{\mathbf{W}_t \in \mathbb{R}^{M\times N}\}_{t=1}^T$ side by side along the 'task' mode to form a 3rd-order weight tensor $\mathcal{W} \in \mathbb{R}^{M\times N\times T}$. In the case of CNN, this weight tensor corresponds to a 5th-order filter tensor $\mathcal{K} \in \mathbb{R}^{H\times W\times U\times V\times T}$. DMTRL then factorizes $\mathcal{W}$ (or $\mathcal{K}$), for instance via TT-format, into 3 TT-cores (or 5 TT-cores for $\mathcal{K}$) Yang & Hospedales (2017). Analogously, the first 2 TT-cores (or the first 4 TT-cores) play exactly the same role as $\mathbf{L}$ for the common knowledge; the very last TT-core is in fact a matrix (similar to $\mathbf{S}$), with each column representing the task-specific information.

The fundamental difference between our TRMTL and DMTRL is that ours can tailor heterogeneous network structures to various tasks. In contrast, DMTRL is not flexible enough to deal with such variations with tasks. Specifically, our TRMTL differs widely with DMTRL and generalizes DMTRL from a variety of aspects. In order to reach TRMTL from DMTRL-TT, one needs to take *four major types of generalizations* (G1-G4), as shown in Figure 6. Firstly (in G1), TRMTL tensorizes the weight into a higher-order weight tensor before factorizing it. By doing so, the weight can be embedded into more latent cores than that of just 3 cores (or 5 cores) in DMTRL, which yields a more compact model and makes the sharing at a finer granularity feasible. Secondly (in G2), DMTRL stringently requires that the first D-1 cores (D is weight tensor's order) must be all shared at every hidden layer, only the last vector is kept for private knowledge. By contrast, TRMTL allows for any sharing pattern at distinct layer. Thirdly (in G3), there is no need for layer-wise weights to be equal-sized and stacked into one big tensor as in TRMTL, each task may have its individual input domains. Finally (in G4), TRMTL further generalizes TT to TR-format. For each task in DMTRL, the first core must be a matrix and the last core must be a vector (with both border rank and outer mode size being 1). Notice that our TRMTL also conceptually subsumes DMTRL-Tucker in terms of the first three aspects of generalizations (G1-G3). It is also worth mentioning that (Wang et al., 2018) only applies TR-format for weight compression in a single deep net, whereas ours incorporates a more general tensor network framework into the deep MTL context.

The authors of (Long et al., 2017) lately proposed multilinear relationship network (MRN) which incorporates tensor normal priors over the parameter tensors of the task-specific layers. However, like methods (Zhang et al., 2014; Ouyang et al., 2014; Chu et al., 2015), MRN follows the architecture where all the lower layers are shared, which is also not tailored for the extended MTL paradigm, and may harm the transferability if tasks are not that tightly correlated. In addition, the relatedness of tasks is captured by the covariance structures over features, classes and tasks. Constantly updating these covariance matrices (via SVD in (Long et al., 2017)) becomes computationally prohibitive for large scale networks. Compared to these *non-latent-subspace* methods, TRMTL is highly compact and needs much fewer parameters, which is obviously advantageous in tasks with small sample size.

Table 6: Specification of network architecture and factorized TRRL representation on MNIST dataset.

MNIST		Task A (Odd)	Task B (Even)
FC1	input modes output modes	[7, 7, 4, 4] [6, 6, 6, 6]	[7, 7, 4, 4] [6, 6, 6, 6]
FC2	input modes output modes	[6, 6, 6, 6] [6, 6, 6, 6]	[6, 6, 6, 6] [6, 6, 6, 6]
FC3	input modes output modes	[6, 6, 6, 6] [4, 4, 4, 4]	[6, 6, 6, 6] [4, 4, 4, 4]
FC4	input modes output modes	[256] [10]	[256] [10]

Table 7: Performance comparison of STL, MRN, DMTRL and our TRMTL on MNIST dataset.

Samples A vs B	STL		MRN		Tucker		DMTRL-TT		Ours-410		Ours-420	
	A	B	A	B	A	B	A	B	A	B	A	B
1800 vs 1800	96.8	96.9	96.4	96.6	95.2	96.2	96.2	96.7	**97.5**	**97.7**	97.4	97.6
1800 vs 100	96.8	88.1	96.5	88.6	95.2	85.5	96.1	86.3	**97.6**	**90.2**	97.5	89.9
100 vs 1800	88.0	96.9	89.3	96.5	85.4	96.6	87.1	96.6	90.1	97.5	**90.3**	**97.6**
100 vs 100	88.0	88.1	88.2	88.4	84.3	84.8	86.8	86.0	88.7	**89.6**	**89.2**	89.5

Table 8: Specification of network architecture and factorized TRRL representation on Omniglot dataset.

Omniglot	Kernel/Weight		Task A		Task B	
Conv1	input modes output modes	window size	[1] [8]	[3, 3]	[1] [8]	[3, 3]
Conv2	input modes output modes	window size	[2, 2, 2] [4, 2, 2]	[3, 3]	[2, 2, 2] [4, 2, 2]	[3, 3]
Conv3	input modes output modes	window size	[2, 2, 2, 2] [4, 2, 2, 2]	[3, 3]	[2, 2, 2, 2] [4, 2, 2, 2]	[3, 3]
FC1	input modes output modes		[18, 12, 12, 9] [4, 4, 4, 4]		[18, 12, 12, 9] [4, 4, 4, 4]	
FC2	input modes output modes		[256] [10]		[256] [10]	

B Detailed Settings and More Experimental Results

B.1 Deep MTL with Single Domain Source

B.1.1 Effect of sharing patterns

The detailed specification of network architecture and factorized TRRL representation of the experiments on MNIST dataset are recorded in Table 6. In Table 7, our TRMTL achieves the best results and is robust to small perturbation of C for pattern selection, since both '410' and '420' patterns obtain similarly good performance.

B.1.2 Effect of unbalanced sample sizes

For the Omniglot Dataset, we adopt a similar architecture as in the previous experiment for CNN as $(1 \times 8\,\text{C3}) - (8 \times 16\,\text{C3}) - (16 \times 32\,\text{C3}) - (23,328 \times 256\,\text{FC}) - (256 \times 50\,\text{FC})$, where the last two convolution layers and first fully connected layer are represented using TRRL with the input/output feature modes of TR-cores being $\{2, 2, 2\}$, $\{4, 2, 2\}$, and $\{2, 2, 2, 2\}$, $\{4, 4, 2, 2\}$, and $\{18, 12, 12, 9\}$, $\{4, 4, 4, 4\}$. Table 8 displays the details of network specification.

The best sharing pattern of our model is '432'. Figure 7 demonstrates the amount of the accuracy changes for each task (for the case of 50% training data), both with and without the aid of the data-rich task. Table 9 summarizes the performance of the compared methods when the distinct fractions of data are used as training data. Our TRMTL obtains the best overall performance in both data-rich and data-scarcity situations.

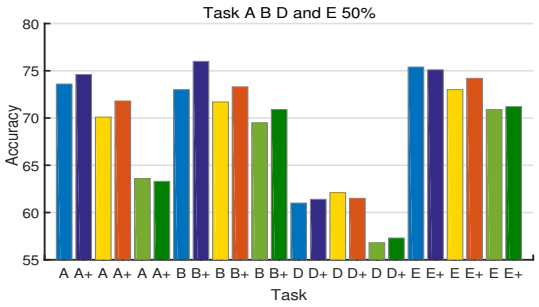


Figure 7: The results of accuracy changes of `task A`, `B`, `D` and `E`, when the faction of the data for training for `task C` is increased from 10% to 90%. '+' corresponds to the results after the samples augmentation of `task C`. 50% data for training for `task A`, `B`, `D` and `E`. The best sharing pattern of TRMTL is '432'.

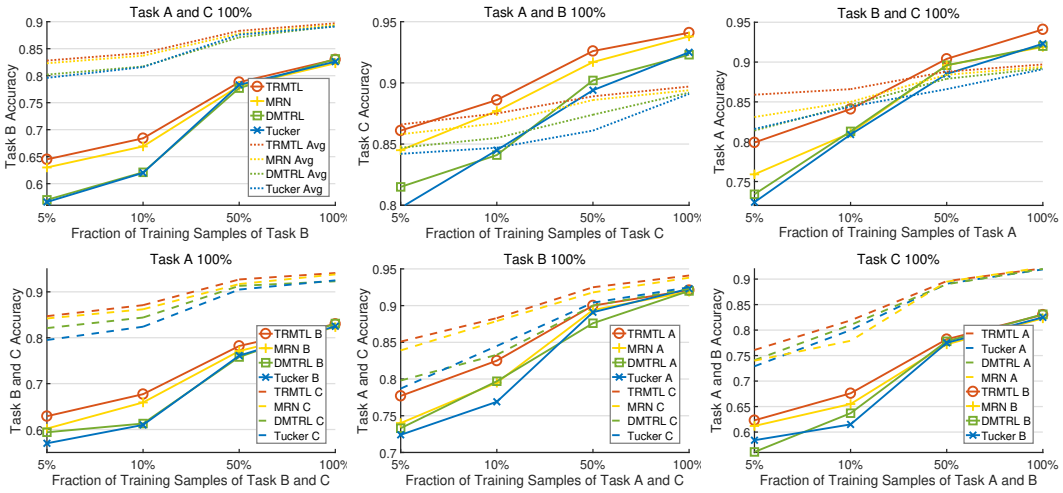


Figure 8: Performance comparison of MRN, DMTRL-Tucker, DMTRL-TT and our TRMTL-4431 on CIFAR-10 with different fractions of training data. Top row: 100% data for two of the three tasks, and show the accuracy for the other one task (in solid lines) as well as the averaged accuracy of all three tasks (in dotted lines). Bottom row: 100% data for one of the three tasks, and show the accuracies for the other two tasks (in dashed and solid lines).

In this section, we also conduct more experiments on CIFAR-10 dataset. We assign 10 classes into 3 tasks, in which `task A` relates to non-animals; `task B` comprises 4 animal classes including 'cat', 'dog', 'deer' and 'horse'; `task C` contains the remaining 2 classes. We like to verify the performance of different models in transferring the useful knowledge from data-abundant task to data-scarcity task within one source of data domain. To this end, we first test on CIFAR dataset with settings where each task may have insufficient training samples like 5%, 10% or 50%. For this test, we adopt the following architecture: $(3 \times 64\,\text{C3}) - (64 \times 128\,\text{C3}) - (128 \times 256\,\text{C3}) - (256 \times 512\,\text{C3}) - (8192 \times 1024\,\text{FC}) - (1024 \times 512\,\text{FC}) - (512 \times 10\,\text{FC})$, where C3 stands for a 3×3 convolutional layer. We employ the general form of TRL on the last two CNN layers and first two FC layers where the most of the parameters concentrate, yielding 4 TR-cores per layer.

Figure 8 illustrates how the accuracies of one task (two tasks) vary with sample fractions, given the remaining two tasks (one task) get access to the full data. We observe the trends in which the accuracies of our model exceed the other competitors by a relatively large margin (shown in solid lines), in the cases of limited training samples, e.g., 5% or 10%. In the mean time, the advantage of our TRMTL is still significant in terms of the averaged accuracies of three tasks (shown in dash lines), which implies the data-scarcity task has little bad influence on the data-abundant tasks.

Table 10 reports the results of our two best patterns ('4431' and '4421'), as well as the one with 'bad' pattern '4444'. Clearly, TRMTL ('4431' and '4421') outperforms other methods in nearly all the cases. As for `task A` , for instance, the precision of TRMTL-4431 is increased by 1.7% when the data of `task C` becomes 100%. Even more, such enhancement further grows up to 5.5% in the situation that both `task B` and `C`'s training samples are fully available. This is in contrast to MRN whose precision improvements are merely 0.4% and 3.0% in the corresponding scenarios. Again,

Table 9: Performance comparison of STL, MRN, DMTRL and our TRMTL on Omniglot dataset.

Samples A vs B vs C vs D vs E	Task	STL	MRN	DMTRL	Tucker	Ours-432
10% vs 10% vs 10% vs 10% vs 10%	A	30.4	31.2	30.5	28.9	**31.6**
	B	32.4	35.4	35.3	32.9	**38.9**
	C	47.5	47.8	44.1	47.9	**48.2**
	D	29.2	29.5	27.8	28.4	**29.9**
	E	40.5	41.2	38.7	**43.0**	42.7
	Average	36.0	37.0	35.3	36.2	**38.3**
50% vs 50% vs 50% vs 50% vs 50%	A	61.1	70.1	63.6	59.0	**73.6**
	B	66.4	71.7	69.5	67.3	**73.0**
	C	73.1	77.8	75.3	70.9	**80.5**
	D	55.8	**62.1**	56.8	55.8	61.0
	E	68.8	73.0	70.9	71.0	**75.4**
	Average	65.0	70.9	67.2	64.8	**72.7**
90% vs 90% vs 90% vs 90% vs 90%	A	72.2	78.6	74.0	75.5	**80.5**
	B	75.4	**80.7**	77.9	76.4	79.5
	C	82.7	86.5	81.7	82.5	**88.7**
	D	60.5	69.7	65.3	62.7	**72.2**
	E	74.9	**82.1**	76.7	75.4	80.7
	Average	73.1	79.5	75.1	74.5	**80.3**

Table 10: Performance comparison of STL, MRN, DMTRL and our TRMTL on CIFAR-10 with unbalanced training samples, e.g., '5% vs 5% vs 5%' means 5% of training samples are available for the respective `task A`, `task B` and `task A`. TR-ranks $R = 10$ for TRMTL.

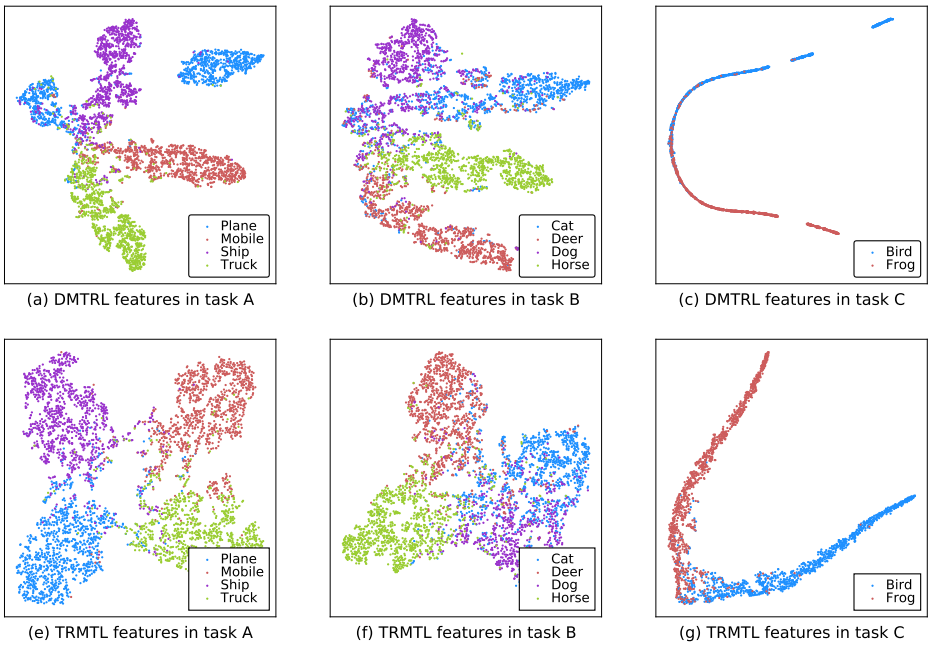
Samples A vs B vs C	Task	STL	MRN	DMTRL	Ours-4444	Ours-4431	Ours-4421
100% vs 100% vs 100%	A	91.4	91.8	**92.2**	90.6	92.1	92.2
	B	80.9	82.3	82.3	81.6	**83.0**	82.6
	C	91.8	93.9	92.3	93.4	**94.1**	93.9
	Average	88.0	89.3	88.9	88.5	**89.8**	89.6
5% vs 5% vs 5%	A	72.7	72.9	73.7	72.4	74.4	**74.7**
	B	57.0	60.3	55.5	59.0	61.3	**61.5**
	C	80.6	82.7	79.5	82.7	82.9	**84.4**
	Average	70.1	72.0	69.6	71.4	72.9	**73.5**
5% vs 5% vs 100%	A	72.7	73.3	74.2	73.6	76.1	**76.4**
	B	57.0	61.2	56.3	60.2	62.3	**63.0**
	C	91.8	92.1	91.5	91.8	**93.1**	93.0
	Average	73.8	75.5	74.0	75.2	77.2	**77.4**
5% vs 100% vs 100%	A	72.7	75.9	74.3	76.9	**79.9**	79.8
	B	80.9	80.2	79.7	79.5	**81.2**	.81.1
	C	91.8	93.3	92.1	92.7	**93.9**	**93.9**
	Average	81.8	83.1	82.0	83.0	**85.0**	84.9

the performance of TRMTL-4431 is superior to that of TRMTL-4444, indicating sharing all nodes like '4444' is not a desirable style.

It is also interesting to get an idea on what our model has learned via the visualization of the high level features. Figure 9 illustrates the task-specific features of our TRMTL (and DMTRL) using t-SNE for the dimensionality reduction. We can see a clear pattern of the clustered features produced by our model that are separated for different classes, which could be more beneficial the down-stream classification tasks.

B.1.3 Effect of inputs with heterogeneous dimensions

In this section, we show the advantage of our method in handling tasks with heterogeneous inputs within single source of data domain. For this test, the tasks are assigned to input images with different spatial sizes or distinct channels (i.e. RGB or grayscale) on CIFAR-10 dataset. In order to apply DMTRL, one has to first convert the heterogeneous inputs into equal-sized features using one hidden layer with totally unshared weights, so that the weights in following layers can be stacked up and factorized. To better show the influence of heterogeneous inputs on the competitors, we adopt MLP with 4 hidden layers. The architectures for the heterogenous spatial sizes case and distinct channels case are shown in Table 11 and 12, respectively. For a good pattern of our TRMTL, such



(a) DMTRL features in task A (b) DMTRL features in task B (c) DMTRL features in task C

(e) TRMTL features in task A (f) TRMTL features in task B (g) TRMTL features in task C

Figure 9: Features visualization of 2000 CIFAR-10 images. `Task A`, `B` and `C` correspond to three categories in CIFAR-10, i.e., non-animals, animals with bird and frog excluded, bird and frog. Top row: DMTRL features. Bottom row: our features.

Table 11: Specification of network architecture and factorized TRRL representation on heterogenous inputs with distinct spatial sizes for CIFAR-10.

Heterogenous Spatial Sizes		Task A (16×32)	Task B (16×16)	Task C (32×32)
FC1	input modes	[4, 4, 4, 4, 3, 2]	[4, 4, 4, 4, 3, 1]	[4, 4, 4, 4, 3, 4]
	output modes	[4, 4, 4, 4, 3, 8]	[4, 4, 4, 4, 3, 8]	[4, 4, 4, 4, 3, 8]
FC2	input modes	[8, 8, 6, 4, 4]	[8, 8, 6, 4, 4]	[8, 8, 6, 4, 4]
	output modes	[8, 8, 6, 4, 8]	[8, 8, 6, 4, 8]	[8, 8, 6, 4, 8]
FC3	input modes	[8, 8, 6, 8, 4]	[8, 8, 6, 8, 4]	[8, 8, 6, 8, 4]
	output modes	[8, 8, 6, 8, 8]	[8, 8, 6, 8, 8]	[8, 8, 6, 8, 8]
FC4	input modes	[8, 8, 6, 8, 8]	[8, 8, 6, 8, 8]	[8, 8, 6, 8, 8]
	output modes	[3, 3, 3, 3, 3]	[3, 3, 3, 3, 3]	[3, 3, 3, 3, 3]
FC5	input modes	[243]	[243]	[243]
	output modes	[10]	[10]	[10]

as '5410', the first hidden layer of each task is encoded into 6 TR-cores, 5 of which can be shared. As recorded in Table 13, DMTRL based methods behave significantly worse than our TRMTL. The poor performance of DMTRL is induced by fact that lowest features from related tasks cannot be shared at all because of the heterogeneous input dimensionality.

B.2 Deep MTL with Multiple Domain Sources

B.2.1 Results on heterogeneous MLPs

For the Office-Home experiment, Table 14 shows the details of network specification.

Table 12: Specification of network architecture and factorized TRRL representation on heterogenous inputs with distinct channels (RGB and grayscale image) for CIFAR-10.

RGB and Grayscale		Task A (RGB)	Task B (Gray)	Task C (Gray)
FC1	input modes	[4, 4, 4, 4, 4, 3]	[4, 4, 4, 4, 4, 1]	[4, 4, 4, 4, 4, 1]
	output modes	[4, 4, 4, 4, 4, 6]	[4, 4, 4, 4, 4, 6]	[4, 4, 4, 4, 4, 6]
FC2	input modes	[8, 8, 6, 4, 4]	[8, 8, 6, 4, 4]	[8, 8, 6, 4, 4]
	output modes	[8, 8, 6, 4, 8]	[8, 8, 6, 4, 8]	[8, 8, 6, 4, 8]
FC3	input modes	[8, 8, 6, 8, 4]	[8, 8, 6, 8, 4]	[8, 8, 6, 8, 4]
	output modes	[8, 8, 6, 8, 8]	[8, 8, 6, 8, 8]	[8, 8, 6, 8, 8]
FC4	input modes	[8, 8, 6, 8, 8]	[8, 8, 6, 8, 8]	[8, 8, 6, 8, 8]
	output modes	[3, 3, 3, 3, 3]	[3, 3, 3, 3, 3]	[3, 3, 3, 3, 3]
FC5	input modes	[243]	[243]	[243]
	output modes	[10]	[10]	[10]

Table 13: The results of heterogenous input dimensionality on CIFAR-10. Top: each task associates with RGB or grayscale image. Bottom: each task has input images of different spatial sizes.

Model	RGB A	Gray B	Gray C	Gray A	RGB B	Gray C	Gray A	Gray B	RGB C
STL	74.0	56.8	77.3	68.4	62.3	77.3	68.4	56.8	83.2
DMTRL-Tucker	72.8	55.2	76.6	66.6	61.6	77.2	66.3	55.4	82.6
DMTRL-TT	73.1	54.1	77.2	66.2	61.5	77.4	66.7	54.8	81.7
TRMTL-5410	**79.4**	**59.3**	**82.9**	**73.5**	**64.9**	**83.5**	**74.4**	**59.4**	**88.9**

Model	16×16 A	32×32 B	16×32 C	32×32 A	16×32 B	16×16 C	16×32 A	16×16 B	32×32 C
STL	73.9	62.3	82.2	74.0	62.0	83.1	74.3	**62.4**	82.2
DMTRL-Tucker	72.9	60.9	82.1	73.1	61.2	82.5	72.6	61.2	82.6
DMTRL-TT	72.3	61.9	82.5	73.1	62.2	82.2	73.4	61.5	82.8
TRMTL-5410	**74.8**	**63.2**	**86.8**	**74.9**	**62.8**	**86.6**	**75.2**	**62.4**	**86.7**

Table 14: Specifications of heterogenous FC network architectures in the Office-Home experiment. Input images are processed by convolutional part of pre-trained VGG-16 before feeding to TRMTL.

Layer		Art task	Clipart task	Product task	Real World task	Homogenous network
FC1	in modes	[8, 8, 8, 7, 7]				
	out modes	[4, 4, 4, 5, 5]				
FC2	in modes	[4, 4, 4, 5, 5]	[4, 4, 4, 5, 5]	[4, 4, 4, 5, 5]	[4, 4, 4, 5, 5]	[4, 4, 4, 5, 5]
	out modes	[4, 4, 4, 5, 5]	[3, 4, 4, 5, 5]	[2, 4, 4, 5, 5]	[1, 4, 4, 5, 5]	[1, 4, 4, 5, 5]
FC3	in modes	[1600]	[1200]	[800]	[400]	[400]
	out modes	[65]	[65]	[65]	[65]	[65]