TENSOR RING NETS ADAPTED DEEP MULTI-TASK LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent deep multi-task learning (MTL) has been shown to be quite successful in alleviating data scarcity of some task by utilizing domain-specific knowledge from related tasks. In this work, we propose a novel knowledge sharing mechanism for linking task-specific models, namely tensor ring multi-task learning (TRMTL). TRMTL models each task with one separate DNN and encodes DNN's parameters with a sequence of latent tensor cores. Meanwhile, the parameter sharing scheme is carried out among the subsets of latent tensor cores of multiple tasks in a distributed manner. Our model has a highly compact representation and is efficient in transferring the task-invariant knowledge, while being super flexible in learning the task-specific features. TRMTL is a general framework that readily subsumes other tensor factorization based deep MTL methods. TRMTL also allows each individual task to have its own distinct input and output feature dimensionality of each layer. Experiments on a variety of datasets demonstrate our model is capable of significantly improving each single task's performance, particularly favorable in scenarios where some of the tasks have insufficient data.

1 INTRODUCTION

Multi-task learning (MTL) (Caruana, 1997; Maurer et al., 2016) is an approach for boosting the overall performance in each individual task by learning multiple related tasks simultaneously. The underlying assumption of MTL is that if multiple tasks are related, one can extract some domain-specific knowledge preferable by other tasks. In the deep learning context, requiring sufficiently flexible models to jointly fit data of multiple tasks can be seen as adding an inductive bias to the models, which could be beneficial to learn more favorable feature representations. Recently, deep MTL has gained much popularity and been successfully explored in an abroad range of applications, such as computer vision (Zhang et al., 2014; Misra et al., 2016; Trottier et al., 2017), natural language processing (Luong et al., 2015; Liu et al., 2017), speech recognition (Wu et al., 2015; Huang et al., 2015) and so on. One key challenge in MTL is how to determine a better information sharing mechanism across related tasks. Regarding deep multi-task networks, this is equivalent to designing a better sharing pattern for network parameters, such as weight matrices and convolutional kernels.

Typically, the commonly used deep MTL models achieve this goal by either hard or soft parameter sharing (Ruder, 2017). Hard parameter sharing usually partitions the architecture into the shared layers at the bottom (for all tasks) and the task-specific layers at the top (one branch per task). The shared layers extract useful task-independent (or task-invariant) features that are relevant to all tasks. Hard sharing is robust to the risk of over-fitting (Baxter, 1997) and has been witnessed its effectiveness in several recent deep MTL literatures (Zhang et al., 2014; Yin & Liu, 2017). However, such kind of architecture can be harmful when learning high-level task-specific features, since it focuses only on learning common low-level features for all tasks. Moreover, these common features may be polluted by some noxious tasks, leading to negative-transfer of the low-level features (Yosinski et al., 2014). Rather than explicitly constructing the shared layers, soft parameter sharing separately learns one DNN for each task with its own set of parameters, and the individual DNNs are implicitly connected by imposing some constraint or regularization scheme on the aligned weight parameters. The deep MTL models of this kind include using ℓ_2 norm regularization (Duong et al., 2015) and trace norm regularization (Yang & Hospedales, 2016). Some other work, such as deep multilinear relation network (MRN) (Long & Wang, 2015; Long et al., 2017), employed tensor normal priors

only to regularize the task-specific layers (i.e., fully connected layers) yet shared the lower layers in a standard hard fashion.

On the other hand, DNNs normally contain millions of trainable parameters and require extremely large memory to store them. The situation becomes even worse when training multiple tasks jointly at the same time. All of the aforementioned deep MTL models suffer from this circumstances especially when limited training data is available. To address the parameter bottleneck in a single DNN, Novikov et al. (2015) introduced a tensor train (TT) (Oseledets, 2011) factorization based approach to significantly compress the weights in fully connected (FC) layers. Following this line of research, Yang and his colleagues (Yang & Hospedales, 2017) applied tensor factorization methods to stack and decompose the corresponding layer-wise weights from multiple tasks, leading to a compact deep MTL model with promising performance.

Although being compact, their method, named deep multi-task representation learning (DMTRL) (Yang & Hospedales, 2017), turns out to be rather restricted in selecting the sharing patterns since DMTRL attaches too much importance on the common knowledge. To be specific, DMTRL shares almost all the parts of weights as a common set of basis for multiple tasks, leaving the remaining small portions of weights to encode the task-specific information. Such sharing pattern is kept the same across all layers, which is vulnerable to negative-transfer of the features. Moreover, DMTRL requires the corresponding layer-wise weights of all tasks to have the same dimensionality. This restriction makes little sense for the case of loosely-related tasks, since the individual features might be quite different and the sizes of layer-wise features may vary a lot from task to task.

In this work, we propose a new general knowledge sharing mechanism for connecting task-specific models. We refer to our method as tensor ring multi-task learning (TRMTL), which can be viewed as a mixed strategy of soft and hard parameter sharing. Similar to soft sharing scheme, we provide one individual DNN per task and compactly encode each layer's parameters using tensor ring (TR) representation (Zhao et al., 2016) via a sequence of latent tensor cores. In the meantime, like hard sharing scheme, a subset of these latent cores are tied across multiple related tasks.

Our main contribution is that we introduce a generalized, super flexible and highly compact latent-subspace based deep MTL model, which could effectively mitigate the dilemma of both negative-transfer in lower layers and under-transfer in higher layers. In contrast to the non-latent-subspace based methods (i.e., MRN), TRMTL needs a lot fewer parameters and especially useful in tasks with small sample size. Comparing to latent-subspace based method, we show DMTRL is a special case that is derived from our TRMTL when several strict restrictions are satisfied. By contrast, TRMTL offers each task a great deal of flexibility to better represent its own task-specific features. Additionally, TRMTL is free from the restraint of equal-sized weight in DMTRL. We achieve the state-of-the-art performance on a variety of datasets and validate that each individual task can gains much benefit from our proposed architecture.

2 PRELIMINARIES

2.1 TENSOR NOTATIONS

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. Matrices are 2nd-order tensors that are denoted by boldface capital letters as $\mathbf{W}$. We denote vectors using boldface lower-case letters as $\mathbf{w}$.

2.2 TENSOR RING DECOMPOSITION

A tensor ring decomposition (TR) (Zhao et al., 2016) decomposes a tensor $\mathcal{W}$ into a sequence 3rd-order latent tensors that are multiplied circularly. Mathematically, $\mathcal{W}$ can be expressed in an element-wise form given by

$$\begin{aligned} \mathcal{W}_{i_1,i_2,\ldots,i_D} &= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_D=1}^{R_D} \mathcal{G}^{(1)}_{r_1,i_1,r_2} \mathcal{G}^{(2)}_{r_2,i_2,r_3} \cdots \mathcal{G}^{(D)}_{r_D,i_D,r_{D+1}} \\ &= \text{Trace}\{\mathbf{G}^{(1)}[i_1]\mathbf{G}^{(2)}[i_2] \cdots \mathbf{G}^{(D)}[i_D]\}, \end{aligned} \tag{1}$$

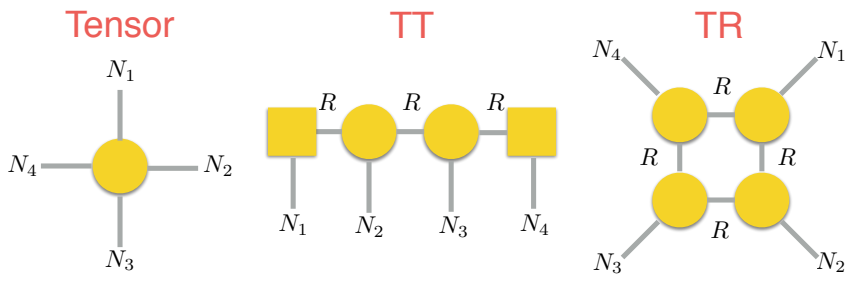


Figure 1: The diagrams of a 4th-order tensor and its TT-format and TR-format. The first and last cores in TT-format are the matrices as the border rank is one (shown in square shape).

where 'Trace' stands for the trace operation. $\mathcal{G}^{(k)} \in \mathbb{R}^{R_k \times N_k \times R_{k+1}}$ denotes the kth latent tensor (or latent core), while $\mathbf{G}^{(k)}[i_k] \in \mathbb{R}^{R_k \times R_{k+1}}, 1 \leq i_k \leq N_k$, corresponds to the i_kth lateral slice matrix of the latent core $\mathcal{G}^{(k)}$. The TR-format requires the two adjacent latent cores, such as $\mathcal{G}^{(k)}$ and $\mathcal{G}^{(k+1)}$, are 'linked' by a common dimension of size R_{k+1}. In particular, $\mathcal{G}^{(D)}$ is connected back to $\mathcal{G}^{(1)}$ by satisfying the condition $R_{D+1} = R_1$. The collection of $[R_1, R_2, ..., R_D]$ is defined as TR-rank. Under the 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 $\prod_{k=1}^{D} N_k$. An example of TR-format is illustrated in Figure 1. Compared with TT (Oseledets, 2011), TR generalizes TT by relaxing the border rank condition (the border rank is one in TT). TR is more flexible than TT because TT-ranks can be equally distributed in the cores, whereas TT-ranks have a relatively fixed pattern. More importantly, TR-ranks are usually smaller than TT-ranks, leading to a much more compact model than TT. Also, TR enjoys the property of circular dimensional permutation invariance which TT is not (Zhao et al., 2016).

3 Methodology

In general, our tensor ring multi-task learning (TRMTL) learns one DNN per task by representing the original weight of each layer with a tensor ring layer (TRL), i.e., utilizing a sequence of TR-cores. Then, a subset of TR-cores are tied across multiple tasks to encode the task-independent knowledge, while the rest TR-cores of each task are treated as private cores for task-specific knowledge.

3.1 Tensor ring layer

We start the section by describing the tensor ring layer (TRL), which lays a groundwork for our TR based deep MTL approach. Following the 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 $k \in \{1, ..., D\}$. Applying TR representation in equation 1 to $\mathcal{W}$, we reach the TR-matrix format as

$$\begin{aligned} \mathbf{W}(i, j) &= \mathcal{W}((\phi_1(i), \psi_1(j)), ..., (\phi_D(i), \psi_D(j))) \\ &= \text{Trace}\{\mathbf{G}^{(1)}[(\phi_1(i), \psi_1(j))]\mathbf{G}^{(2)}[(\phi_2(i), \psi_2(j))] \cdots \mathbf{G}^{(D)}[(\phi_D(i), \psi_D(j))]\}. \end{aligned} \tag{2}$$

Now each network layer is parameterized only by a collection of TR-cores instead of the original weights. The training can be conducted by applying the standard stochastic gradient descent based methods to these TR-cores. Notice that TRL is very similar to the recently proposed TR based weight compression (Wang et al., 2018) for neural network, except we adopt slightly different 4th-order latent cores in TR-matrix. As for CNN, one can easily extend TR to a convolutional kernel

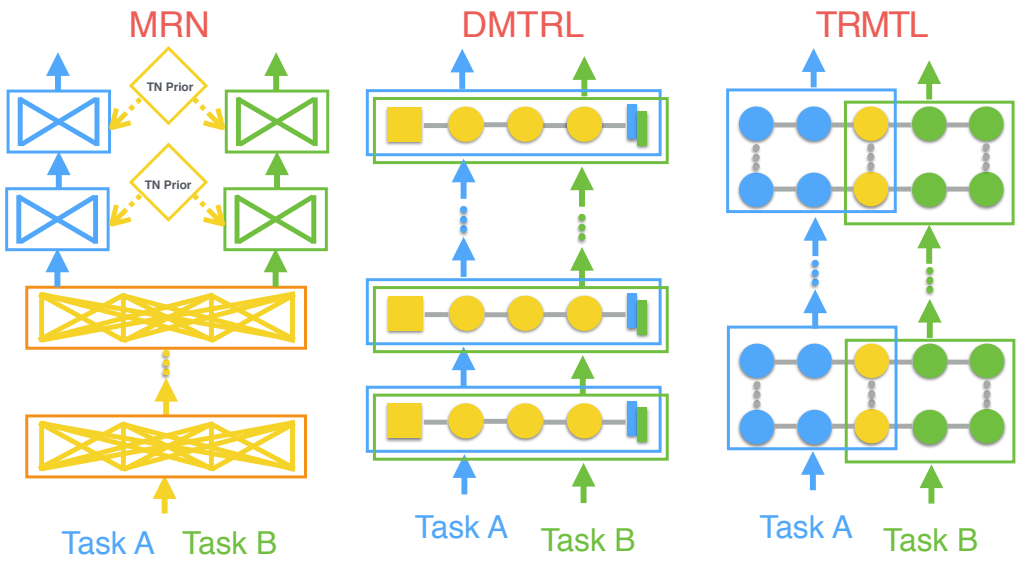


Figure 2: The overall sharing mechanism of MRN (left), DMTRL (middle, for the case of CNN) and our TRMTL (right) w.r.t. two tasks. The shared portion is depicted in yellow. MRN: original weights are totally shared at the lower layers and the relatedness between tasks at the top layers is modeled by tensor normal priors. DMTRL: all weights are equal-sized and are stacked and decomposed into TT-format. Almost all the cores are used as the shared nodes except the last TT-core (matrix). The portion of sharing is the same for all the layers. TRMTL: 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 portion of sharing can be different from layer to layer.

$\mathcal{K} \in \mathbb{R}^{H \times W \times U \times V}$, where $H \times W$ is the spatial size and U and V correspond to the input and output features maps. Since the spatial size of kernel is usually small, we only compress kernel along the modes of input and output feature maps. To this end, we converts the 4th-order kernel $\mathcal{K}$ into TR-format by means of TR-matrix as

$$\mathcal{K}(h, w, u, v) = \text{Trace}\{\mathbf{G}^{(0)}[(h, w)]\mathbf{G}^{(1)}[(\phi_1(u), \psi_1(v))] \cdots \mathbf{G}^{(D)}[(\phi_D(u), \psi_D(v))]\}. \quad (3)$$

3.2 Tensor Ring Multi-Task Learning

Under this TR-core representation, 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 TRL, 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. The following equation 4 demonstrates just one special case of our model where the very first c TR-cores are selected to share.

$$\mathcal{Y}_t(v_1, ..., v_{D_t}) = \sum_{u_1=1}^{U_1} \cdots \sum_{u_{D_t}=1}^{U_{D_t}} \mathcal{H}_t(u_1, ..., u_{D_t})$$
$$\text{Trace}\{\mathbf{G}^{(1)}[(u_1, v_1)] \cdots \mathbf{G}^{(c)}[(u_c, v_c)]\mathbf{G}_t^{(c+1)}[(u_{c+1}, v_{c+1})] \cdots \mathbf{G}_t^{(D_t)}[(u_{D_t}, v_{D_t})]\}. \quad (4)$$

In our model, the common TR-cores $\{\mathbf{G}^{(1)}, ..., \mathbf{G}^{(c)}\}$ correspond to any subset of c elements, and can be arbitrarily chosen from the set of all TR-cores, leaving the rest cores as task-specific TR-cores. Pay close attention that our TRMTL neither restricts on which TR-cores to share, nor restricts the shared cores to be in an consecutive order. Finally, we reshape tensor $\mathcal{Y}_t$ back into a vector output $\mathbf{y}_t \in \mathbb{R}^{V_t}$. The proposed sharing scheme is illustrated in Figure 2. 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 4, TRMTL represents each weight element in weight matrix as function of a sequence product of the lateral slice matrices of the corresponding shared TR-cores and private TR-cores. Intuitively, this strategy suggests the value of each weight element is partially determined by some task-invariant latent factors, and meanwhile, also partially affected by some task-individual 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.

4 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 vector weights $\{\mathbf{w}_t \in \mathbb{R}^M\}_{t=1}^T$ of T tasks to have the same size, so that these weights could be stacked up into a matrix $\mathbf{W} \in \mathbb{R}^{M\times T}$. 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 task-independent latent basis vectors, whereas each column vector of $\mathbf{S} \in \mathbb{R}^{K\times T}$ is task-specific and contains the combination coefficients for these common latent bases. Just recently, (Yang & Hospedales, 2017) extended this matrix based MTL to its high-order counterpart DMTRL by making use of tensor factorization. Likewise, DMTRL first puts equal-sized 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 $\mathcal{K} \in \mathbb{R}^{H\times W\times U\times V\times T}$. Then, DMTRL factorizes $\mathcal{W}$ (or $\mathcal{K}$ for CNN) via TT decomposition, yielding 3 TT-cores (or 5 TT-cores for CNN) (Yang & Hospedales, 2017). Analogously, the first 2 TT-cores (or the first 4 TT-cores for CNN) play exactly the same role as $\mathbf{L}$ in sense that it comprises 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 DMTRL scheme is shown in the middle in Figure 1.

In contrast, TRMTL generalizes DMTRL from a variety of aspects: TRMTL adapts TR latent-subspace representation, which is superior to TT-format as stated in the previous section; TRMTL parameterizes one DNN for each task individually, thus not demanding the layer-wise weights to be equal-sized and stacked into one big tensor; TRMTL allows for any sharing pattern for distinct layer among latent cores; TRMTL generalizes to allow the layer-wise weight to be represented by a relatively lager number of latent cores, leading to a more compact model. In essence, DMTRL turns out to be just a very special case of our TRMTL. This only happens if we first degenerate TRMTL to TT representation (border ranks have to be 1), and then require that all the first 2 TT-cores (or 4 TT-cores for CNN, or D-1 TT-cores for Dth-order weight tensor) are shared, except the last TT-core for the private knowledge. And the model is further limited to keep only this sharing pattern across all hidden layers. Sharing almost all portion of parameters may be risky to negative-transfer. As an effect, the common latent cores will dominate in DMTRL, which greatly suppresses its capability in expressing the task-specific variations in transferable features. It's also worth mentioning that Wang et al. (2018) only applied TR-format for compressing weights in a single deep net, whereas ours focuses on incorporating TR-format into the deep MTL context. Two methods have different goals and deal with different applications.

Apart from the latent-subspace based methods, Long et al. (2017) lately proposed a deep multi-linear relationship network (MRN) which incorporates tensor normal distribution as priors over the parameter tensors of the task-specific layers. MRN jointly learns the transferable features as well as multilinear relationship among tasks, with the objective to alleviate both under-transfer and negative-transfer of the knowledge. However, like most existing deep MTL methods (Zhang et al.,

Samples A vs B	STL		MRN		Tucker		DMTRL		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 1: Performance comparison of STL, MRN, DMTRL-Tucker, DMTRL and our TRMTL on MNIST with TR-ranks $R = 10$. Task A (B) corresponds to odd-digit classes (even-digit classes).

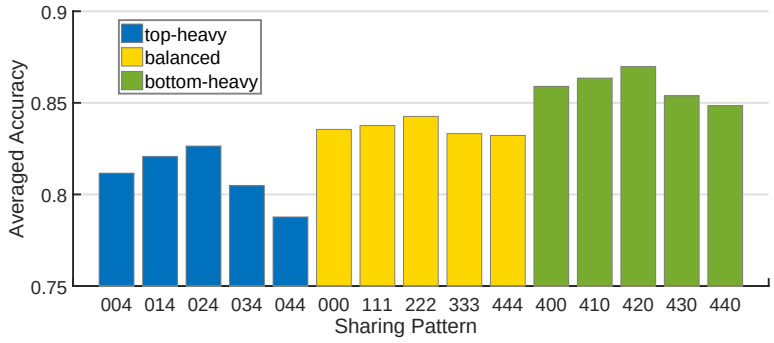


Figure 3: The averaged accuracy of two tasks involved with 50 samples in the cases of '1800 vs 50' and '50 vs 1800'. The patterns in 'bottom-heavy' category mean more parameters are shared at the bottom layers than the top layers, while 'top-heavy' patterns indicate the opposite style. The pattern '024' means 0, 2 and 4 TR-cores are shared from lower to higher layers, respectively.

2014; Ouyang et al., 2014; Chu et al., 2015), MRN follows the architecture where all the lower layers are shared, which may harm the transferability if tasks are loosely 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)) become computationally prohibitive for large scale networks. Compared with above mentioned non-latent-subspace based approaches, such as (Zhang et al., 2014; Ouyang et al., 2014; Chu et al., 2015; Long et al., 2017), TRMTL is extremely compact and hence needs much fewer parameters, which is obviously advantageous in tasks with small sample size.

5 Experimental Results

As for the baseline, we separately train one model for each task with no parameter sharing involved. We refer to this as single task learning (STL). We also compare our TRMTL with MRN (Long et al., 2017), DMTRL and its variant DMTRL-Tucker (Yang & Hospedales, 2017). To be fair, all the methods are adopted with same network architecture, i.e., same number of layers. We repeat the experiments five times and record the average classification accuracy. The model selection is conducted by cross-validation on training set. The price to pay for the great flexibility of our TRMTL is the tuning of the portions of the weight sharing along the different layers. For this purpose, we tensorize the layer-wise weight into a Dth-order tensor, whose D modes have roughly the same dimensionality, such that the resulting D TR-cores are approximately equal if we assume the same TR-ranks. Thus, we can measure the faction of sharing by the number of TR-cores, which is needed to tune via cross validation. However, the search space of this hyper-parameter grows rapidly as number of the layers increase. In practice, we can mitigate this issue a lot by following a useful guidance that this number tends to decrease as the layers increase. We employ the similar trick introduced in (Yang & Hospedales, 2017) to specify the TR-ranks as well as the number of TR-cores, which turns out not difficult to tune.

5.1 MNIST dataset

We first validate our model on the MNIST benchmark LeCun et al. (1998). In this test, we partition 10 digit classes into the odd digits and even digits. One task is to classify the odd digits and the other one is to classify the even digits. We use MLP architecture with three tensorized hidden layers. Under TRMTL, each of these layers is encoded using 4 TR-cores, with output modes of $\{6, 6, 6, 6\}$, $\{6, 6, 6, 6\}$, and $\{4, 4, 4, 4\}$, respectively. We test different sharing patterns and report the ones with the best accuracies. As we can see from Table 1 that TRMTL performs consistently better than other methods, especially in scenarios where small samples are involved. In order to see how sharing styles affect our performance, we examine various patterns from three representative categories. In Figure 3, we gauge the transferability between tasks with unbalanced training samples by the averaged accuracy on the small-sample tasks. Overall, the 'bottom-heavy' patterns tend to achieve better results than those from the other categories, which is in conformity with fact that

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

Table 2: 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 C. TR-ranks $R = 10$ for TRMTL.

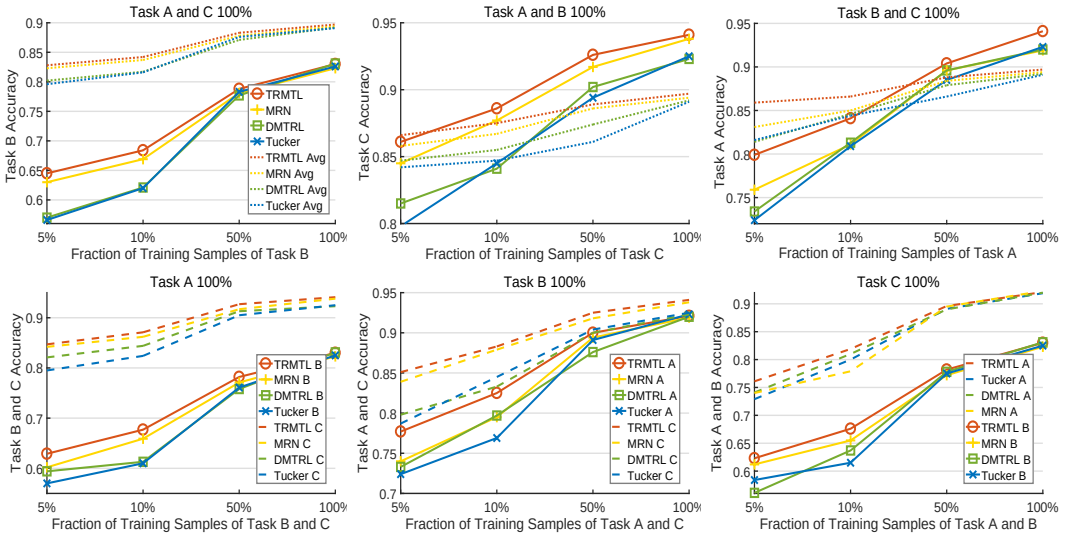


Figure 4: Performance comparison of MRN, DMTRL-Tucker, DMTRL and our TRMTL-4431 on CIFAR-10 with different fractions of training data. Top row: 100% data for two of the three tasks, and show the accuracy for the other one task (in solid lines) as well as the averaged accuracy of all three tasks (in dotted 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).

deep features evolve from general to specific along the network (Yosinski et al., 2014). For instance, the pattern '420' makes a lot sense and obviously outperforms the pattern '044' which overlaps all weights at the top layers but shares nothing at the bottom layer.

5.2 CIFAR-10 DATASET

In this section, we conduct experiments on CIFAR-10 dataset (Krizhevsky & Hinton, 2009), which contains a total number of $60,000$ colour images of size 32×32 from 10 object classes. For our test, we divide 10 classes into 3 groups, the first of which consists of 4 classes that are related to non-animals (task A), the second group contains 4 animal classes including 'cat', 'dog', 'deer' and 'horse' (task B), while the remaining 2 classes are categorized into the third group (task C). We adopt the following architecture: $(3 \times 64\, C3) - (64 \times 128\, C3) - (128 \times 256\, C3) - (256 \times 512\, C3) - (8192 \times 1024\, FC) - (1024 \times 512\, FC) - (512 \times 10\, FC)$, where $C3$ stands for a 3×3 convolutional

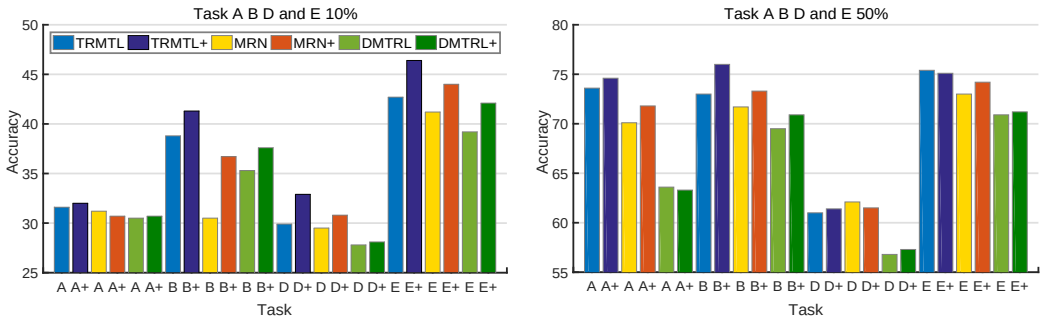


Figure 5: The results of accuracy changes of tasks A, B, D and E, when the faction of the data for training for task C is increased from 10% to 90%. '+' corresponds to the results after the samples augmentation of task C. Left (Right): 10% (50%) data for training for task A, B, D and E.

layer. We employ TRL on the last two CNN layers and first two FC layers, in which the most of the parameters concentrate, yielding 4 TR-cores per layer. Now we are interested in the effectiveness of different models when transferring the useful knowledge from data-abundant task to data-scarcity task. To this end, we begin with the test cases where all of task have insufficient training samples, e.g., '5% vs 5% vs 5%'. After that, we compare the precision improvement of the individual task(s) when the other task(s) is (are) equipped with the whole training data. Table 2 records 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 the 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, the performance of TRMTL-4431 is superior to that of TRMTL-4444, indicating sharing all nodes like '4444' is not a desirable style. Figure 4 also demonstrates 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.

5.3 Omniglot Dataset

Finally, the tests are carried out on the Omniglot (Lake et al., 2015) dataset for the application of recognizing handwritten letters in multiple languages. The dataset consists of handwritten letters from 50 alphabets, with a total number of 1623 unique characters. We divide the whole alphabets into five groups (tasks A to E), each of which links to the alphabets from 10 different languages. We test a more challenging case, where only 1 task (task C) has sufficient samples while the samples of the other 4 tasks (task A, B, D and E) are limited. Figure 5 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 cause the accuracy to increase for most of the time. Particularly, the gap of the accuracy enhancement is more obvious for the case of 10% data. Please refer to the appendix for the architecture and more experimental results.

6 Conclusion

In this paper, we have introduced a novel knowledge sharing mechanism for connecting task-specific models in deep MTL, namely TRMTL. The proposed approach models each task separately in the form of TR representation using a sequence latent cores. Next, TRMTL shares the common knowledge by ting any subset of layer-wise TR cores among all tasks, leaving the rest TR cores for private knowledge. TRMTL is highly compact yet super flexible to learn both task-specific and task-invariant features. TRMTL is empirically verified on various datasets and achieves the state-of-the-art results in helping the individual tasks to improve their overall performance.

REFERENCES

Jonathan Baxter. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning*, 28(1):7–39, 1997.

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.

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.

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.

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.

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.

Ludovic Trottier, Philippe Giguère, and Brahim Chaib-draa. Multi-task learning by deep collaboration and application in facial landmark detection. *arXiv preprint arXiv:1711.00111*, 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.

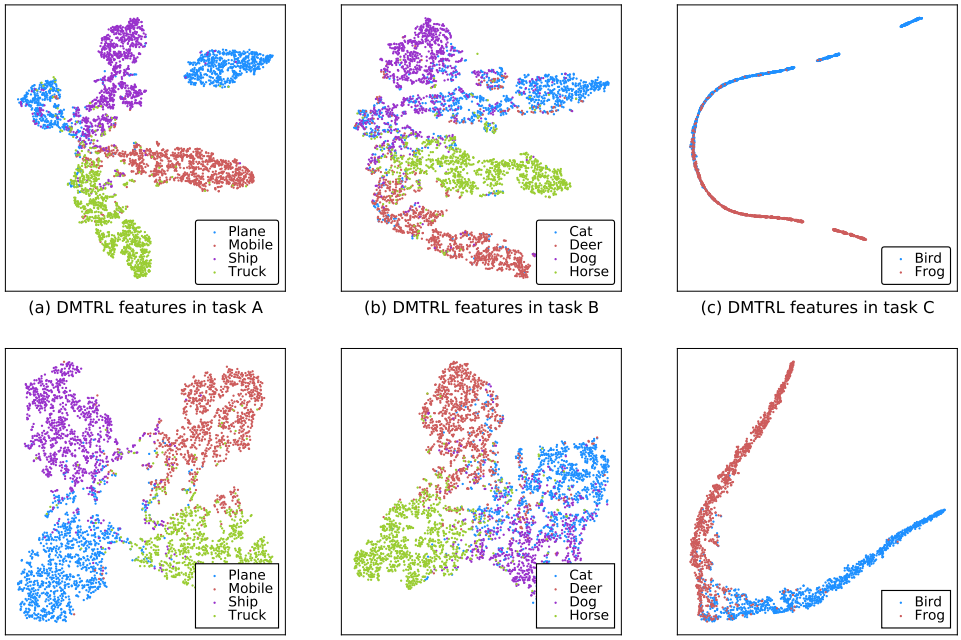
Yongxin Yang and Timothy M Hospedales. Trace norm regularised deep multi-task learning. *arXiv preprint arXiv:1606.04038*, 2016.

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

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pp. 3320–3328, 2014.

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.



(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 6: Features visualization of 2000 CIFAR-10 images. Tasks 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.

6.1 CIFAR VISUALIZATION

It is also interesting to get an idea on what our model has learned via the visualization of the high level features. Figure 6 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 downstream classification tasks.

6.2 OMNIGLOT EXPERIMENT

For this dataset, we adopt a similar architecture as in the previous experiment for CNN as

$$(1 \times 8\ C3) - (8 \times 16\ C3) - (16 \times 32\ C3) - (23,328 \times 256\ FC) - (256 \times 50\ FC),$$

where the last two convolution layers and first fully connected layer are represented using TRL 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 $\{18, 12, 12, 9\}$, $\{4, 4, 4, 4\}$. The best sharing pattern of our model is '432', which is selected by CV. Table 3 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.

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 3: Performance comparison of STL, MRN, DMTRL and our TRMTL on Omniglot with different fractions of training samples.