Multi-Task Label Embedding for Text Classification

Honglun Zhang[1], **Liqiang Xiao**[1], **Wenqing Chen**[1], **Yongkun Wang**[2], **Yaohui Jin**[1]
[1]State Key Lab of Advanced Optical Communication System and Network
MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
[2]Network and Information Center, Shanghai Jiao Tong University
{jinyh}@sjtu.edu.cn

Abstract

Multi-task learning in text classification leverages implicit correlations among related tasks to extract common features and yield performance gains. However, a large body of previous work treats labels of each task as independent and meaningless one-hot vectors, which cause a loss of potential label information. In this paper, we propose **Multi-Task Label Embedding** to convert labels in text classification into semantic vectors, thereby turning the original tasks into vector matching tasks. Our model utilizes semantic correlations among tasks and makes it convenient to scale or transfer when new tasks are involved. Extensive experiments on five benchmark datasets for text classification show that our model can effectively improve the performances of related tasks with semantic representations of labels and additional information from each other.

1 Introduction

Text classification is a common *Natural Language Processing* (NLP) issue that tries to infer the most appropriate label for a given sentence or document, for example, sentiment analysis, topic classification and so on. With the developments and prosperities of *Deep Learning* (Bengio et al., 2013), many neural network based models have been exploited by a large body of literature and achieved inspiring performance gains on various text classification tasks. These models are robust at feature engineering and can represent word sequences as fixed-length vectors with rich semantic information, which are notably ideal for subsequent NLP tasks.

Due to numerous parameters to train, neural network based models rely heavily on adequate amounts of annotated corpora, which cannot always be met as constructions of large-scale high-quality labeled datasets are extremely time-consuming and labor-intensive. *Multi-Task Learning* (**MTL**) solves this problem by jointly training multiple related tasks and leveraging potential correlations among them to increase corpora size implicitly, extract common features and yield performance gains. Inspired by (Caruana, 1997), there is a large body of research dedicated to MTL with neural network based models (Collobert and Weston, 2008; Liu et al., 2015b, 2016a,b; Zhang et al., 2017). These models usually contain a pre-trained lookup layer that map words into dense, low-dimension and real-value vectors with semantic implications, namely known as *Word Embedding* (Mikolov et al., 2013b), and utilize some lower layers to capture common features that are further fed to follow-up task-specific layers. However, many existing models may have at least one of the following three disadvantages:

- Lack of **Label Information**. Labels of each task are represented by independent and meaningless one-hot vectors, for example, *positive* and *negative* in sentiment analysis encoded as $[1, 0]$ and $[0, 1]$, which may cause a loss of potential label information.

- Inability to **Scale**. Network structures are elaborately designed to model various correlations for MTL, but many of them are structurally fixed and some can only deal with interactions between two tasks, namely pairwise interactions. When new tasks are involved, the network structures have to be modified and all networks have to be trained again.

- Inability to **Transfer**. Human beings can handle a completely new task without anymore efforts after learning with several related tasks, which can be called the ability

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 4545–4553
Brussels, Belgium, October 31 - November 4, 2018.

of *Transfer Learning* (Ling et al., 2008). As discussed above, the network structures of many previous models are fixed and there are no layers specially designed for unannotated new tasks.

In this paper, we proposed **Multi-Task Label Embedding** (**MTLE**) to map labels of each task into semantic vectors, similar to how Word Embedding deals with word sequences, thereby turning the original tasks into vector matching tasks. The idea of embedding label is not new and is primarily designed to improve general neural text classification models (Bengio et al., 2010; Norouzi et al., 2013; Ma et al., 2016), but here we mainly focus on integrating label embedding to enhance MTL. MTLE utilizes semantic correlations among tasks and effectively solves the problems of scaling and transferring when new tasks are involved.

We conduct extensive experiments on five benchmark datasets for text classification. Compared to learning separately, jointly learning multiple related tasks based on MTLE demonstrates significant performance gains for each task.

Our contributions are four-fold:

- Our model efficiently leverages label information of each task by mapping labels into dense, low-dimension and real-value vectors with semantic implications.
- It is particularly convenient for our model to scale for new tasks. The network structures need no modifications and only samples from the new tasks require training.
- After training on several related tasks, our model can also naturally transfer to deal with new tasks without anymore training, while still achieving appreciable performances.
- We consider different scenarios of MTL and demonstrate strong results on several benchmark datasets for text classification. Our model outperforms most of the state-of-the-art baselines.

2 Problem Statements

2.1 Single-Task Learning

For a text classification task, the **Input** is a word sequence $x = \{x_1, x_2, ..., x_T\}$, where T is the sequence length and we need to output the most appropriate **Label** from the set $\{y_1, y_2, ..., y_C\}$, or their one-hot representation $\{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_C\}$, where C is the number of classes. In the supervised case, we have the **Annotation**, that is, the corresponding ground truth label $\tilde{\mathbf{y}}$ for each input, while in the unsupervised case, we only know the label set but lack the specific annotations.

A pre-trained lookup layer is used to get the embedding vector $\mathbf{x}_t \in \mathbb{R}^d$ for each word x_t. A text classification model f is trained to produce the predicted distribution $\hat{\mathbf{y}}$ for each $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\}$.

$$f(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T) = \hat{\mathbf{y}} \tag{1}$$

and the learning objective is to minimize the overall cross-entropy on the training set.

$$l = -\sum_{i=1}^{N}\sum_{j=1}^{C} \tilde{y}_{ij} \log \hat{y}_{ij} \tag{2}$$

where N denotes the number of training samples.

2.2 Multi-Task Learning

Given K supervised text classification tasks, $T_1, T_2, ..., T_K$, a multi-task learning model F is trained to transform each $\mathbf{x}^{(k)}$ from T_k into multiple predicted distributions $\{\hat{\mathbf{y}}^{(1)}, ..., \hat{\mathbf{y}}^{(K)}\}$.

$$F(\mathbf{x}_1^{(k)}, ..., \mathbf{x}_T^{(k)}) = \{\hat{\mathbf{y}}^{(1)}, ..., \hat{\mathbf{y}}^{(K)}\} \tag{3}$$

where only $\hat{\mathbf{y}}^{(k)}$ is used for loss computation. The overall training loss is a weighted combination of costs for each task.

$$L = -\sum_{k=1}^{K} \lambda_k \sum_{i=1}^{N_k}\sum_{j=1}^{C_k} \tilde{y}_{ij}^{(k)} \log \hat{y}_{ij}^{(k)} \tag{4}$$

where λ_k, N_k and C_k denote the weight, the number of training samples and the number of classes for each task T_k.

2.3 Ability to Scale

When new tasks are involved, for example, given K' more supervised tasks $T_{K+1}, T_{K+2}, ..., T_{K+K'}$, the MTL model F, which has been trained on the old K tasks, should be able to scale with few structural modifications.

$$F(\mathbf{x}_1^{(k)}, ..., \mathbf{x}_T^{(k)}) = \{\hat{\mathbf{y}}^{(1)}, ..., \hat{\mathbf{y}}^{(K+K')}\} \tag{5}$$

where $\mathbf{x}^{(k)}$ denotes samples from not only the old tasks but also the new ones.

There are mainly two scaling methods for F to deal with the new tasks. We can continue training F and further tune the model parameters based on samples from the new tasks, which we define as **Hot Update**, or re-train F again based on training samples from all tasks, which is defined as **Cold Update**.

2.4 Ability to Transfer

Given K'' more completely unannotated new tasks $T^{\circ}_{K+1}, T^{\circ}_{K+2}, ..., T^{\circ}_{K+K''}$, F should be able to transfer what it learned from the old tasks, which is defined as **Zero Update**, as no more training samples are available and the model parameters are not updated anymore.

For each $\mathbf{x}^{(k)}$ from the K'' tasks, F should produce the corresponding predicted distributions for the new tasks, even if the annotations of these new tasks are not provided at all.

$$F(\mathbf{x}_1^{(k)}, ..., \mathbf{x}_T^{(k)}) = \{\hat{\mathbf{y}}^{(K+1)}, ..., \hat{\mathbf{y}}^{(K+K'')}\} \quad (6)$$

which requires that F should be able to understand the meanings of each label from the new tasks and infer the most appropriate one.

3 Methodology

In text classification tasks, labels can be single-word or double-word, for example, *positive* and *negative* in binary sentiment classification, *very positive*, *positive*, *neutral*, *negative* and *very negative* in 5-category sentiment classification, but there are few labels that contain three words or more. Inspired by Word Embedding, we propose **Multi-Task Label Embedding** (**MTLE**) to convert labels of each task of MTL into dense, low-dimension and real-value vectors with semantic implications, thereby disclosing potential intra-task and inter-task correlations from both texts and labels.

3.1 Architecture

Figure 1 illustrates the general idea of MTLE, which mainly consists of three parts, the **Input Encoder**, the **Label Encoder** and the **Matcher**.

In the Input Encoder, each input sequence $x^{(k)}$ from T_k is transformed into embedding representation $\mathbf{x}^{(k)} = \{\mathbf{x}_1^{(k)}, \mathbf{x}_2^{(k)}, ..., \mathbf{x}_T^{(k)}\}$ by the Embedding Layer (E_I). The Learning Layer (L_I) is applied to recurrently comprehend $\mathbf{x}^{(k)}$ and generate a fixed-length vector $\mathbf{X}^{(k)}$, which can be regarded as an overall representation of the original input sequence $x^{(k)}$.

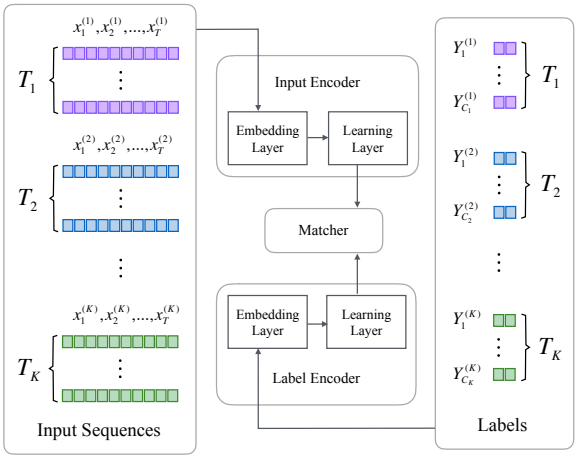

Figure 1: General Architecture of MTLE

In the Label Encoder, there are C_k labels in T_k, where each $y_j^{(k)} (1 \leq j \leq C_k)$ consists of one word or two words, and is mapped into the vector representation $\mathbf{y}_j^{(k)}$ by the Embedding Layer (E_L). The Learning Layer (L_L) absorbs $\mathbf{y}_j^{(k)}$ to generate a fixed-length vector $\mathbf{Y}_j^{(k)}$, which can be utilized as an overall semantic representation of the original label $y_j^{(k)}$.

In order to classify a sample $x^{(k)}$ from T_k, the Matcher obtains the corresponding $\mathbf{X}^{(k)}$ from the Input Encoder, all $\mathbf{Y}_j^{(k)} (1 \leq j \leq C_k)$ from the Label Encoder, and conducts vector matching to select the most appropriate class label.

In MTLE, E_I and E_L obtain understandings of words from texts and labels respectively, while L_I and L_L achieve representation abilities of vector sequences. All these four modules are learned within and shared among tasks.

3.2 Implementation Details

We can explore different embedding methods and neural networks to achieve E_I, E_L and L_I, L_L respectively, but here we choose to implement them in an easier way and spend more efforts to investigate the effectiveness of MTLE.

The implementation details of MTLE are illustrated in Figure 2. L_I and L_L are both fully-connection layers with the weights $\mathbf{W}_I$ and $\mathbf{W}_L$ of $|V| \times d$, where $|V|$ denotes the vocabulary size and d is the embedding size. We can get a pre-trained lookup table based on open domain corpora to initialize $\mathbf{W}_I, \mathbf{W}_L$ and further tune them during training.

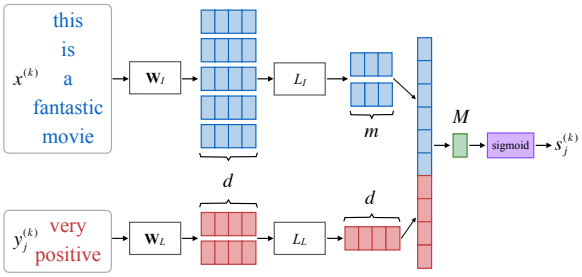

Figure 2: Implementation Details of MTLE

L_I and L_L should be trainable models that can transform a vector sequence of arbitrary length into a fixed-length vector, which can be implemented by a *Bi-directional Long Short-Term Memory Network* (**BiLSTM**) that can recurrently process vector sequences and learn long-term dependencies.

While there are numerous variants of the standard LSTM (Hochreiter and Schmidhuber, 1997), here we follow the implementation of (Graves, 2013). At each time step t, states of the LSTM can be fully described by five vectors in $\mathbb{R}^m$, an input gate $\mathbf{i}_t$, a forget gate $\mathbf{f}_t$, an output gate $\mathbf{o}_t$, the hidden state $\mathbf{h}_t$ and the memory cell $\mathbf{c}_t$, which adhere to the following transition equations.

$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{V}_i\mathbf{c}_{t-1} + \mathbf{b}_i) \quad (7)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{V}_f\mathbf{c}_{t-1} + \mathbf{b}_f) \quad (8)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{V}_o\mathbf{c}_{t-1} + \mathbf{b}_o) \quad (9)$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}_c\mathbf{h}_{t-1}) \quad (10)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (11)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (12)$$

where $\mathbf{x}_t$ is the current input, σ denotes logistic sigmoid function and $\odot$ denotes element-wise multiplication.

A BiLSTM consists of two LSTM layers that process the input sequences in original and reversed orders, and its output is the concatenation of hidden states from the forward and backward LSTM at each time step.

$$\mathbf{h}_t = \overrightarrow{\mathbf{h}}_t \oplus \overleftarrow{\mathbf{h}}_t \quad (13)$$

where $\oplus$ denotes vector concatenation.

We apply the above equations to implement L_I with hidden size m. However, it is inappropriate to apply a BiLSTM for L_L, as most labels contain only one or two words. Instead, L_L accepts the embedding vectors of each word from a label and calculate the average.

For an input sample $x^{(k)}$ and all C_k labels $y_j^{(k)}$ from T_k, the corresponding $\mathbf{X}^{(k)} \in \mathbb{R}^{2m}$ and $\mathbf{Y}_j^{(k)} \in \mathbb{R}^d$ are calculated as follows.

$$\mathbf{X}^{(k)} = L_I(\mathbf{W}_I(x^{(k)})) \quad (14)$$

$$\mathbf{Y}_j^{(k)} = L_L(\mathbf{W}_L(y_j^{(k)})) \quad (15)$$

In this paper, we mainly focus on the idea and effects of MTLE, so rather than exploring some useful mechanisms like gating, external memory or attention for stronger abilities of sequence learning, we choose the vanilla BiLSTM for quick implementations and spend most of our efforts on investigating the effectiveness of MTLE.

We concatenate $\mathbf{X}^{(k)}, \mathbf{Y}_j^{(k)}$ and apply another fully-connection layer with only one neuron, denoted by M, to implement the Matcher, which accepts outputs from L_I and L_L to produce a score of matching. Given the matching scores of each label, we refer to the idea of cross-entropy and calculate the loss function for a sample $x^{(k)}$ from T_k as follows.

$$s_j^{(k)} = \sigma(M(\mathbf{X}^{(k)} \oplus \mathbf{Y}_j^{(k)})) \quad (16)$$

$$\hat{y}_j^{(k)} = \frac{\exp(s_j^{(k)})}{\sum_{c=1}^{C_k} \exp(s_c^{(k)})} \quad (17)$$

$$l^{(k)} = -\sum_{j=1}^{C_k} \tilde{y}_j^{(k)} \log \hat{y}_j^{(k)} \quad (18)$$

The overall training objective is to minimize the weighted linear combination of costs for each task.

$$L = -\sum_{k=1}^{K} \lambda_k \sum_{i=1}^{N_k} l_i^{(k)} \quad (19)$$

MTLE provides an easy and intuitive way to realize MTL, where input texts and class labels from different tasks are jointly learned and compactly fused. During training, E_I and E_L learn better understanding of word semantics for different tasks, L_I and L_L obtain stronger abilities of sequence representation, while M produces more accurate scores for vector matching.

3.3 Scaling and Transferring

When new tasks are involved, it is particularly convenient for MTLE to scale or transfer as the network structure needs no modifications. For input samples $x^{(k)}$ and class labels $y_j^{(k)}$ from the new tasks, we can apply E_I, E_L, L_I, L_L to get

their vector representations $\mathbf{X}^{(k)}$ and $\mathbf{Y}_j^{(k)}$, calculate the matching scores and find the most appropriate label.

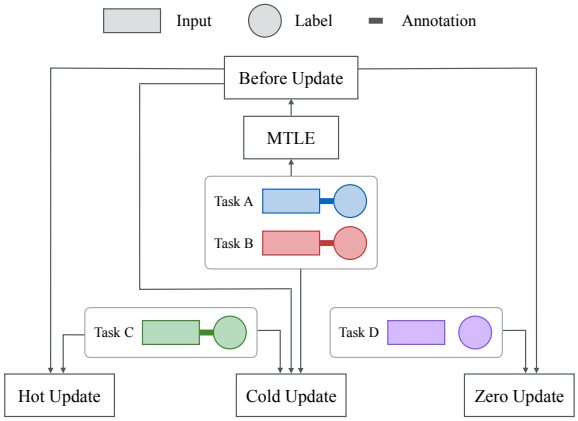

Figure 3: Three different updating methods

If the new tasks are annotated, we can apply Hot Update or Cold Update for scaling and better tune the parameters. If the new tasks are completely unannotated, we can use Zero Update for transferring and produce reasonable predictions.

The differences among Hot Update, Cold Update and Zero Update are illustrated in Figure 3, where **Before Update** denotes the state of MTLE trained on the old tasks before the new tasks are introduced. We will further compare these updating methods in the Experiment Section.

4 Experiment

In this section, we design extensive experiments with multi-task learning based on five benchmark datasets for text classification. We investigate the empirical performances of MTLE and compare them with existing state-of-the-art baselines.

4.1 Datasets

As Table 1 shows, we select five benchmark datasets for text classification, which are commonly used in a large body of research on MTL. We design three experiment scenarios to evaluate the performances of MTLE. Larger text classification datasets (Zhang et al., 2015) are not chosen as there are already enough samples to train and MTL may increase computation workloads. We use the accuracy for evaluations as samples from these datasets are mostly well balanced.

- **Multi-Cardinality** Movie review datasets with different average lengths and class numbers, including **SST-1** (Socher et al., 2013), **SST-2**, **IMDB** (Maas et al., 2011).
- **Multi-Domain** Product review datasets on different domains from *Multi-Domain Sentiment Dataset* (Blitzer et al., 2007).
- **Multi-Objective** Text classification datasets with different objectives, including **IMDB**, **RN** (Apté et al., 1994), **QC** (Li and Roth, 2002).

4.2 Hyperparameters and Training

Training of MTLE is conducted through back propagation with batch gradient descent (Amari, 1993). We obtain a pre-trained lookup table by applying *Word2Vec* (Mikolov et al., 2013a) on the Google News corpus, which contains more than 100B words with a vocabulary size of about 3M. During each epoch, we randomly divide training samples from different tasks into batches of fixed size. For each iteration, we randomly select one task and choose an untrained batch, calculate the gradient and update the parameters accordingly.

Parameters of the neural layers are randomly initialized with the Xavier initializer (Glorot and Bengio, 2010). We apply 10-fold cross-validation and different combinations of hyperparameters are investigated, of which the best one is described in Table 2.

4.3 Results of MTLE vs. Single Task

In Table 3, we compare the performances of MTLE with single-task learning, where only a BiLSTM layer is applied.

MTLE achieves significant performance gains with label information and additional correlations from related tasks. Multi-Domain, Multi-Cardinality and Multi-Objective benefit from MTLE with average improvements of 5.5%, 2.7% and 1.2%, as they contain increasingly weaker relevance among tasks. The result of IMDB in Multi-Cardinality is slightly better than that in Multi-Objective (91.3 against 90.9), as SST-1 and SST-2 share more semantically useful information with IMDB than RN and QC.

4.4 Abilities to Scale and Transfer

In order to investigate the abilities of MTLE to scale and transfer, we use $A + B \rightarrow C$ to denote the case where MTLE is trained on task A and B, while C is the newly involved one. We design three cases based on different scenarios and

Table 1: Five benchmark text classification datasets

Dataset	Description	Type	Average Length	Class	Objective
SST	Movie reviews in Stanford Sentiment Treebank including **SST-1** and **SST-2**	Sentence	19 / 19	5 / 2	Sentiment
IMDB	Internet Movie Database	Document	279	2	Sentiment
MDSD	Product reviews on books, DVDs, electronics and kitchen (**BDEK**)	Document	176 / 189 / 115 / 97	2	Sentiment
RN	Reuters Newswire topics classification	Document	146	46	Topics
QC	Question Classification	Sentence	10	6	Question Types

Table 2: Hyperparameter settings

Embedding size	$d = 300$
Hidden layer size of LSTM	$m = 100$
Batch size	$\delta = 32$
Initial learning rate	$\eta = 0.1$
Regularization weight	$\lambda = 10^{-5}$

compare the influences of Hot Update, Cold Update, Zero Update on each task.

- **Case 1** SST-1 + SST-2 → IMDB.
- **Case 2** B + D + E → K.
- **Case 3** RN + QC → IMDB.

where in Zero Update, we ignore the training set of C and just evaluate our model on the testing set.

As Table 4 shows, Before Update denotes the model trained on the old tasks before the new tasks are involved, so only evaluations on the old tasks are conducted.

Cold Update re-trains the model of Before Update with both the old tasks and the new tasks, thus achieving similar performances with the last line in Table 3. Different from Cold Update, Hot Update resumes training only on the new tasks, requires much less training time, while still obtains competitive results for all tasks. The new tasks like IMDB and Kitchen benefit more from Hot Update than the old tasks, as the parameters are further tuned according to annotations from these new tasks.

Zero Update provides inspiring possibilities for completely unannotated tasks. There are no more annotations for additional training from the new tasks, so we just apply the model of Before Update for evaluations on the testing sets of the new tasks. Zero Update achieves competitive performances in Case 1 (90.9 for IMDB) and Case 2 (86.7 for Kitchen), as tasks from these two cases all belong to sentiment datasets of different cardinalities or domains that contain rich semantic correlations with each other. However, the result for IMDB in Case 3 is only 74.2, as sentiment shares less relevance with topic and question type, thus leading to poor transferring performances.

4.5 Multi-Task or Label Embedding

MTLE mainly employs two mechanisms, label embedding and multi-task learning, so both implicit information from labels and potential correlations from other tasks make differences. In this section, we conduct experiments to explore the contributions of label embedding and multi-task learning respectively.

We choose the four tasks from Multi-Domain scenario and train MTLE on each task separately, so their performances are only influenced by label embedding. Then we re-train MTLE from scratch for every two tasks, every three tasks from them and record the performances of each task in different cases, where both label embedding and multi-task learning matter.

The results are illustrated in Figure 4. The first three graphs show the results of MTLE trained on every one, every two and every three tasks. In the first graph, the four tasks are trained separately and achieve improvements of 3.0%, 3.1%, 3.3%, 2.3% compared to the single task in Table 3. As more tasks are added step by step, MTLE produces increasing performance gains for each task and achieves an average improvement of 5.5% when all the four tasks are trained together. So it can

Table 3: Results of MTLE on different scenarios

Model	Multi-Cardinality			Multi-Domain				Multi-Objective			AvgΔ
	SST-1	SST-2	IMDB	B	D	E	K	IMDB	RN	QC	
Single Task	46.2	86.1	88.9	78.3	79.8	81.5	82.3	89.0	84.2	92.7	-
MTLE	**49.8**	**88.4**	**91.3**	**84.5**	**85.2**	**87.3**	**86.9**	**90.9**	**85.5**	**93.2**	+3.4

Table 4: Results of Hot Update, Cold Update and Zero Update in different cases

Model	Case 1			Case 2				Case 3		
	SST-1	SST-2	IMDB	B	D	E	K	RN	QC	IMDB
Before Update	48.6	87.6	-	83.7	84.5	85.9	-	84.8	**93.4**	-
Cold Update	**49.8**	**88.5**	91.2	**84.4**	**85.2**	**87.2**	86.9	**85.5**	93.2	91.0
Hot Update	49.6	88.1	**91.4**	84.2	84.9	87.0	**87.1**	85.2	92.9	**91.1**
Zero Update	-	-	90.9	-	-	-	86.7	-	-	74.2

be concluded that both information from labels as well as correlations from other tasks account for considerable parts of contributions.

In the last graph, diagonal cells denote improvements of every one task, while off-diagonal cells denote average improvements of every two tasks, so an off-diagonal cell of darker color indicates stronger correlation between the two tasks. An interesting finding is that Books is more related with DVDs and Electronics is more relevant to Kitchen. A possible reason may be that Books and DVDs are products targeted for reading or watching, while customers care more about appearances and functionalities when talking about Electronics and Kitchen.

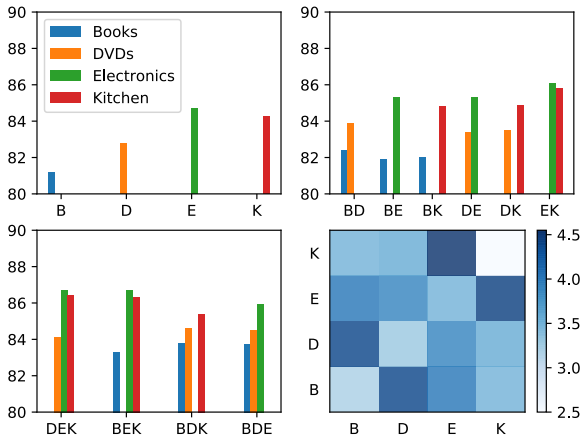

Figure 4: Performance gains of each task in different cases

4.6 Comparisons with State-of-the-art Models

We compare MTLE against the following models.

- **NBOW** Neural Bag-of-Words that sums up embedding vectors of all words and applies a non-linearity followed by a softmax layer.
- **PV** Paragraph Vectors followed by logistic regression (Le and Mikolov, 2014).
- **CNN** Convolutional Neural Networks for Sentence Classification (Kim, 2014).
- **MT-CNN** Multi-Task learning with Convolutional Neural Networks (Collobert and Weston, 2008) where lookup tables are partially shared.
- **MT-DNN** Multi-Task learning with Deep Neural Networks (Liu et al., 2015b) that utilizes bag-of-word representations and a hidden shared layer.
- **MT-RNN** Multi-Task learning with Recurrent Neural Networks with a shared-layer (Liu et al., 2016b).
- **DSM** Deep multi-task learning with Shared Memory (Liu et al., 2016a) where a external memory and a reading/writing mechanism are introduced.
- **GRNN** Gated Recursive Neural Network for sentence modeling and text classification (Chen et al., 2015).
- **Tree-LSTM** A generalization of LSTMs to tree-structured network topologies (Tai et al., 2015).

As Table 5 shows, MTLE achieves competitive or better performances on most tasks except

Table 5: Comparisons of MTLE against state-of-the-art models

Model	SST-1	SST-2	IMDB	Books	DVDs	Electronics	Kitchen	QC
NBOW	42.4	80.5	83.6	-	-	-	-	88.2
PV	44.6	82.7	**91.7**	-	-	-	-	91.8
CNN	48.0	88.1	-	-	-	-	-	93.6
MT-CNN	-	-	-	80.2	81.0	83.4	83.0	-
MT-DNN	-	-	-	79.7	80.5	82.5	82.8	-
MT-RNN	49.6	87.9	91.3	-	-	-	-	-
DSM	49.5	87.8	91.2	82.8	83.0	85.5	84.0	-
GRNN	47.5	85.5	-	-	-	-	-	**93.8**
Tree-LSTM	**50.6**	86.9	-	-	-	-	-	-
MTLE	49.8	**88.4**	91.3	**84.5**	**85.2**	**87.3**	**86.9**	93.2

for QC, as it contains less correlations with other tasks. Tree-LSTM outperforms our model on SST-1 (50.6 against 49.8), but it requires an external parser to get the sentence topological structure and utilizes treebank annotations. PV slightly surpasses MTLE on IMDB (91.7 against 91.3), as sentences from IMDB are much longer than SST and MDSD, which require stronger abilities of long-term dependency learning.

In this paper, we mainly focus the idea and effects of integrating label embedding to enhance multi-task learning, so we apply the BiLSTM to realize L_I, which can be further implemented by other more powerful sequence learning models (Liu et al., 2015a; Chen et al., 2015; Tai et al., 2015) and produce better performances. Explorations of other embedding layers and learning layers may be appreciated, but due to limited pages we choose to research these contents in future work.

5 Related Work

There is a large body of literature related to multi-task learning with neural networks in NLP (Collobert and Weston, 2008; Liu et al., 2015b, 2016a,b; Zhang et al., 2017).

(Collobert and Weston, 2008) use a shared lookup layer for common features, followed by task-specific layers for several traditional NLP tasks including part-of-speech tagging and semantic parsing. They use a fix-size window to solve the problem of variable-length input sequences, which can be better addressed by RNN.

(Liu et al., 2015b, 2016a,b; Zhang et al., 2017) all investigate MTL for text classification. (Liu et al., 2015b) apply bag-of-word representation, but information on word order is lost. (Liu et al., 2016a) introduce an external memory for information sharing with a reading/writing mechanism for communications. (Liu et al., 2016b) propose three different models for MTL with RNN and (Zhang et al., 2017) constructs a generalized architecture for RNN based MTL. However, models of these papers ignore essential information of labels and mostly can only address pairwise interactions between two tasks. Their network structures are also fixed, thereby failing to scale or transfer when new tasks are involved.

Different from the above work, MTLE maps labels of text classification tasks into semantic vectors and provide a more intuitive way to realize MTL with the abilities to scale and transfer. Input sequences from three or more tasks are jointly learned together with their labels, benefitting from each other and obtaining better sequence representations.

6 Conclusion and Future Work

In this paper, we propose Multi-Task Label Embedding to map labels of text classification tasks into semantic vectors. MTLE utilizes semantic correlations among tasks and effectively solves the problems of scaling and transferring when new tasks are involved. We explore three different scenarios of MTL and MTLE can improve performances of most tasks with semantic representations of labels and additional information from others in all scenarios.

In future work, we would like to explore other learning layers and generalize MTLE to address other NLP tasks, for example, sequence labeling and sequence-to-sequence learning.

References

Shun-ichi Amari. 1993. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(3):185–196.

Chidanand Apté, Fred Damerau, and Sholom M. Weiss. 1994. Automated Learning of Decision Rules for Text Categorization. *ACM Trans. Inf. Syst.*, 12(3):233–251.

Samy Bengio, Jason Weston, and David Grangier. 2010. Label embedding trees for large multi-class tasks. In *NIPS*, pages 163–171.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. 2013. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828.

John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification. In *ACL*.

Rich Caruana. 1997. Multitask Learning. *Machine Learning*, 28(1):41–75.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Shiyu Wu, and Xuanjing Huang. 2015. Sentence Modeling with Gated Recursive Neural Network. In *EMNLP*, pages 793–798.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*, pages 160–167.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256.

Alex Graves. 2013. Generating Sequences With Recurrent Neural Networks. *CoRR*, abs/1308.0850.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*, pages 1746–1751.

Quoc V. Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *ICML*, pages 1188–1196.

Xin Li and Dan Roth. 2002. Learning Question Classifiers. In *COLING*.

Xiao Ling, Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. 2008. Spectral domain-transfer learning. In *ACM SIGKDD*, pages 488–496.

Pengfei Liu, Xipeng Qiu, Xinchi Chen, Shiyu Wu, and Xuanjing Huang. 2015a. Multi-Timescale Long Short-Term Memory Neural Network for Modelling Sentences and Documents. In *EMNLP*, pages 2326–2335.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016a. Deep Multi-Task Learning with Shared Memory for Text Classification. In *EMNLP*, pages 118–127.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016b. Recurrent Neural Network for Text Classification with Multi-Task Learning. In *IJCAI*, pages 2873–2879.

Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. 2015b. Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval. In *NAACL HLT*, pages 912–921.

Yukun Ma, Erik Cambria, and Sa Gao. 2016. Label embedding for zero-shot fine-grained named entity typing. In *COLING*, pages 171–180.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *NAACL HLT*, pages 142–150. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013b. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*, pages 3111–3119.

Mohammad Norouzi, Tomas Mikolov, Samy Bengio, Yoram Singer, Jonathon Shlens, Andrea Frome, Greg Corrado, and Jeffrey Dean. 2013. Zero-shot learning by convex combination of semantic embeddings. *CoRR*, abs/1312.5650.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *EMNLP*, pages 1631–1642, Stroudsburg, PA. Association for Computational Linguistics.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *ACL*, pages 1556–1566.

Honglun Zhang, Liqiang Xiao, Yongkun Wang, and Yaohui Jin. 2017. A generalized recurrent neural architecture for text classification with multi-task learning. In *IJCAI-17*, pages 3385–3391.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*, pages 649–657.