Better Fine-Tuning via Instance Weighting for Text Classification

Zhi Wang,[1*] **Wei Bi,**[2†] **Yan Wang,**[2] **Xiaojiang Liu**[2]
[1]Department of Control and Systems Engineering, Nanjing University, Nanjing, China
[2]Tencent AI Lab, Shenzhen, China
heyuanmingong@gmail.com, {victoriabi, brandenwang, kieranliu}@tencent.com

Abstract

Transfer learning for deep neural networks has achieved great success in many text classification applications. A simple yet effective transfer learning method is to fine-tune the pre-trained model parameters. Previous fine-tuning works mainly focus on the pre-training stage and investigate how to pre-train a set of parameters that can help the target task most. In this paper, we propose an Instance Weighting based Fine-tuning (IW-Fit) method, which revises the fine-tuning stage to improve the final performance on the target domain. IW-Fit adjusts instance weights at each fine-tuning epoch dynamically to accomplish two goals: 1) identify and learn the specific knowledge of the target domain effectively; 2) well preserve the shared knowledge between the source and the target domains. The designed instance weighting metrics used in IW-Fit are model-agnostic, which are easy to implement for general DNN-based classifiers. Experimental results show that IW-Fit can consistently improve the classification accuracy on the target domain.

Introduction

Text classification is one of the fundamental tasks in Natural Language Processing (NLP) with many important applications such as sentiment analysis (Landeiro and Culotta, 2016; Xu et al., 2017; Zhang, Huang, and Zhao, 2018), information extraction (Angeli, Premkumar, and Manning, 2015), and topic labeling (Wang and Manning, 2012). Deep Neural Network (DNN) based classifiers have achieved state-of-the-art performance on many text classification tasks (Cao et al., 2017; Yang, Salakhutdinov, and Cohen, 2017; Zhang, Barzilay, and Jaakkola, 2017). However, two limitations generally exist when these methods are used in practice: a) Classifiers are often trained from scratch; b) A large annotated dataset is often required (Lu et al., 2014; Howard and Ruder, 2018). To overcome these limitations, transfer learning is usually applied by transferring knowledge learned on a large source domain to help the classification task on the target domain.

In general, there are mainly two categories of approaches for transfer learning on DNN-based text classifiers. The first category is to transfer the source domain corpora (i.e., out-domain data), which detects instances that are close to the target domain (i.e., pseudo in-domain data), then trains a model using the augmented set including the in-domain and the pseudo in-domain data (Wang et al., 2017a,b). Such approaches involve the search of the pseudo in-domain data from a potentially very large source domain and repeatedly re-train the model using the augmented data, thus the computational efficiency may be problematic. Moreover, the scoring function to select the pseudo in-domain data often needs to be designed for each task specifically. Thus, a more popular class of approaches is to transfer the model parameters, which pre-trains a model using the source corpus and then fine-tune the parameters on the target domain (Kim et al., 2015; McCann et al., 2017). It enables a fast and efficient adaptation to the target domain without repeatedly accessing or processing the large source dataset. Considering these superiorities, we study the fine-tuning method in this paper.

Most previous literature on fine-tuning a pre-trained model focuses on how to pre-train a set of parameters that can help the target task most. For example, Pennington, Socher, and Manning (2014) pre-trained the popular GloVe by leveraging the statistical information on the nonzero elements in a word-word co-occurrence matrix. Howard and Ruder (2018) pre-trained a language model using the long short-term memory network and used it to initialize the first layer of the target classifier. Based on a set of pre-trained parameters, existing methods directly apply standard optimization algorithms such as SGD or Adam (Sutskever et al., 2013; Kingma and Ba, 2014) for fine-tuning on the target domain. [1] However, these optimization algorithms are originally designed for randomly initialized DNN models.

In this work, we propose an Instance Weighting based Fine-tuning (IW-Fit) method, which revises the fine-tuning stage to improve the classification accuracy on the target domain when a pre-trained model from the source domain is given. The idea of our method is illustrated in Figure 1. At each fine-tuning epoch, we will assign dynamic weights to target instances to accomplish two goals: 1) identify and learn the specific knowledge of the target domain effec-

[*]Work done when Zhi Wang was interning at Tencent AI Lab.
[†]Corresponding author

[1]Some work fixed part of/all the pre-trained parameters and only updated other model parameters to prevent overfitting (Johnson and Zhang, 2017; Felbo et al., 2017), which can be seen as a special case of the fine-tuning methods.

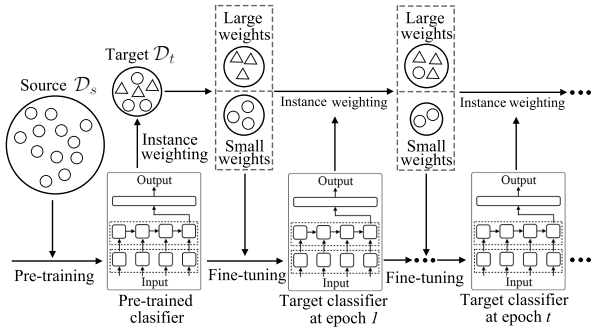


Figure 1: Illustration of the proposed Instance Weighting based Fine-tuning (IW-Fit). The weights are dynamically adjusted based on the current classifier.

tively; 2) well preserve the shared knowledge between the source and the target domains. Intuitively, IW-Fit puts more training effort on instances that either contain more target knowledge or help to preserve the shared knowledge, according to their relationship with the current classifier. To accomplish this idea, we propose to calculate the instance weights using two metrics: *the prediction loss* and *the variance of historical prediction losses*. A good property of the two designed metrics is that they do not depend on the specific design of the DNN-based classifier, thus can be easily applied to any network structure.

We extensively evaluate IW-Fit on two text classification tasks under various settings: *Amazon* to *Yelp* and *Yelp-2015* to *Yelp-2016*/*Yelp-2017*. Our experimental results show that IW-Fit can consistently improve the classification accuracy on the target domain.

In summary, our contributions are threefold:

1. We propose Instance Weighting based Fine-tuning (IW-Fit), a method applied on the fine-tuning stage given a set of pre-trained parameters.
2. We design two model-agnostic metrics to calculate the weights used in IW-Fit and their mixing variants.
3. We perform extensive experiments to verify that IW-Fit can consistently improve the classification accuracy on the target domain over several baselines.

Related Work

With respect to the contribution of this paper, we discuss two threads of related work in this section: DNN-based fine-tuning methods and methods that involve the idea of instance weighting in NLP tasks.

DNN-based fine-tuning methods transfer the model parameters from a pre-trained model to the target domain for fine-tuning. In many state-of-the-art models for various NLP tasks, the popular pre-trained word embeddings from the Skip-grams model (Mikolov et al., 2013) or the GloVe (Pennington, Socher, and Manning, 2014) were usually utilized as an initialization. Johnson and Zhang (2017) improved the pre-trained region embedding by extending ShallowCNN with unsupervised training on the extra unlabeled data. Felbo et al. (2017) sequentially unfroze and fine-tuned a single layer at a time to enhance a regularizing effect at the expense of extra computation. Howard and Ruder (2018) performed a two-step approach by fine-tuning the pre-trained language model first and then the entire classifier on the target domain. Similarly, Chung, Lee, and Glass (2018) pre-trained a complex question-answering (QA) model on a large source domain and transferred all model parameters to the target domain for fine-tuning.

As stated in the introduction, the above methods focus on the pre-training stage. In our work, we try to improve the fine-tuning stage with the idea of instance weighting, which can be combined with all these methods complimentarily.

Another thread of work is about using the idea of instance weighting in various NLP tasks. Previous approaches assigned each instance/domain a weight relying on designed rules or statistical methods. Jiang and Zhai (2007) iteratively pruned instances across domains for better domain adaptation under a semi-supervised learning setting. Neural machine translation domain adaptation methods utilized a pre-trained language model (Wang et al., 2017b), or the sentence embedding from a machine translation model (Wang et al., 2017a), to score the out-domain data used for jointly training with the in-domain data. Data selection methods (Ruder and Plank, 2017; Ruder, Ghaffari, and Breslin, 2017) optimized an instance weighting function of many specific domain similarity features to select relevant training instances that are useful for transfer learning. The idea could be traced back to curriculum learning (Bengio et al., 2009; Jiang et al., 2015), which required a manually pre-defined ranking to weigh training instances for speeding up learning.

In this paper, we investigate the idea of instance weighting used in the DNN-based fine-tuning methods. Different from the previous instance weighting methods, we assign different weights to the target data only, and the fine-tuning method is trained on the target data without repeatedly accessing or processing any source data.

Instance Weighting based Fine-tuning

In this section, we first formulate the problem of the DNN-based fine-tuning method. Then, the framework of the proposed Instance Weighting based Fine-tuning (IW-Fit) method is presented, following by the designed weighting metrics used in IW-Fit explained in detail. Finally, we present two mixing variants of the two weighting metrics.

Problem Formulation

We are given a static source task $\mathcal{T}_s$ and a target task $\mathcal{T}_t$ with $\mathcal{T}_s \neq \mathcal{T}_t$. Our goal is to improve the classification accuracy in $\mathcal{T}_t$. Specifically, we have a source dataset $\mathcal{D}_s$, and a target dataset $\mathcal{D}_t = \{(\mathbf{x}, \mathbf{y})\}$. Here, $\mathbf{x}$ denotes the input of an instance and $\mathbf{y} \in \{0, 1\}^m$ is the corresponding one-hot label vector over the m classes. Let $g_{\boldsymbol{\theta}}(\cdot)$ denote the discriminative function of a DNN parameterized by $\boldsymbol{\theta}$. In text classification, the loss function, f, is generally set as the cross-entropy loss over m classes:

$$f(\mathbf{y}, g_{\boldsymbol{\theta}}(\mathbf{x})) = -\mathbf{y}^T \log p(\mathbf{y}|\mathbf{x}), \quad (1)$$

where $p(\mathbf{y}|\mathbf{x})$ is the prediction probability vector output by the softmax layer of the DNN. Given an observed dataset $\mathcal{D}$,

Algorithm 1: Instance Weighting based Fine-tuning (IW-Fit) for text classification

Input: Datasets $\mathcal{D}_s$, $\mathcal{D}_t$; Learning rate α; Mini-batch size k.
Output: Optimal parameters $\boldsymbol{\theta}^*$ for $\mathcal{D}_t$.

```
Pre-train the classifier on D_s to obtain θ_s.
Initialize the classifier g_θ(x) on D_t: θ ← θ_s.
while Not converged do
    while Epoch not ended do
        Fetch a mini-batch {(x_i, y_i)}_1^k from D_t uniformly at random.
        Calculate and normalize the instance weight w_i by metrics introduced in (4) and (5), or their mixing variants in Section 3.4.
        Update the network in (2).
    end
end
```

we usually train the DNN with the following objective:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} f(\mathbf{y}, g_{\boldsymbol{\theta}}(\mathbf{x})). \tag{2}$$

In our problem setting, we aim at obtaining the optimal parameters $\boldsymbol{\theta}^*$ that can achieve a high classification accuracy on the target domain $\mathcal{D}_t$.

In fine-tuning methods, we first train a model on the source domain $\mathcal{D}_s$. Next, we can apply existing methods (Howard and Ruder, 2018; Chung, Lee, and Glass, 2018; Felbo et al., 2017) to select a set of/all pre-trained model parameters as the initialization for the target model. Then, the target model starts to update itself using the target dataset $\mathcal{D}_t$ until it is converged. By using standard optimization algorithms such as SGD (Sutskever et al., 2013) or Adam (Kingma and Ba, 2014), we update the model parameters by treating all target instances with equal importance.

Framework and Two Weighting Metrics

Our proposed IW-Fit method is presented in Algorithm 1. On the fine-tuning stage, we assign a weight w_i to each instance in one mini-batch according to metrics introduced later, and update the network by a weighted sum of their individual gradients as:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \sum_{i=1}^{k} w_i \cdot \nabla_{\boldsymbol{\theta}} f(\mathbf{y}_i, g_{\boldsymbol{\theta}}(\mathbf{x}_i)), \tag{3}$$

where α is the learning rate, and k is the mini-batch size.

Next, we will introduce our two metrics to calculate the weight w and their mixing variants in detail. Recall that the designed weighting metrics in IW-Fit should accomplish two goals: 1) identify and learn the specific knowledge of the target domain effectively; 2) well preserve the shared knowledge between the source and the target domains.

At the early fine-tuning epochs, parameters are close to those obtained from the pre-trained classifier that has not adapted to the target domain yet. An instance with a large prediction loss means that it is not fitted well by the pre-trained classifier, and it is likely to contain more target knowledge. Thus, if we assign large weights to these instances, the model will put more emphasis on learning the specific knowledge on the target domain. This motivates us to use *the prediction loss* as our first metric.

After a number of fine-tuning epochs, the classifier is likely to capture more target-specific knowledge, and some target-specific instances tend to have small prediction losses. To enable the algorithm to consistently distinguish these instances, we propose another metric - *the variance of historical prediction losses*. For instances with target-specific knowledge, it generally starts with a large prediction loss. If it is now with a small gradient, we can still identify it through calculating the variance of its historical prediction losses along the fine-tuning epochs.

On the other hand, for instances similar to the source domain, it usually starts with a small prediction loss at the early epochs. If such instances receive large losses at the later fine-tuning epochs, it is probable that our fine-tuned classifier is over-fitted by instances with the target-specific knowledge. In this case, these instances produce a large prediction loss as well as a large variance of prediction losses. Assigning them large weights can balance our model to accomplish the second goal. Next, we introduce how to calculate the two metrics in detail.

Prediction loss

The prediction loss in (1) can be used as the first metric to assign the instance weight:

$$w = \frac{1}{\tau}(-\mathbf{y}^T \log p(\mathbf{y}|\mathbf{x}) + \epsilon), \tag{4}$$

where ϵ is a smoothness constant for preventing the weight of an instance with a small loss being zero, and τ is a normalization constant making the average of weights from a mini-batch equal to 1. When ϵ gets larger, all w's will be close to 1, and this metric reduces to uniform weighting.

Variance of historical prediction losses

At epoch t of our algorithm, assume that $h^{t-1} = [f^1, \ldots, f^{t-1}]$ is the vector containing the historical terms of the prediction loss $f = -\mathbf{y}^T \log p(\mathbf{y}|\mathbf{x})$. Our second metric can be computed based on the variance of prediction losses:

$$w = \frac{1}{\tau}(std(h^{t-1}) + \epsilon), \tag{5}$$

where $std(h^{t-1})$ is the estimated standard derivation plus its confidence interval in h^{t-1}:

$$std(h^{t-1}) = \sqrt{\varsigma(h^{t-1}) + \frac{\varsigma^2(h^{t-1})}{|h^{t-1}| - 1}},$$

where $\varsigma^2(h^{t-1})$ is the estimated variance of prediction losses, and $|h^{t-1}|$ is the number of stored prediction losses.

Mixing Variants of Weighting Metrics

As stated in the above framework, the prediction loss is more effective at the early fine-tuning epochs. On the

contrary, the variance of historical prediction losses should be used at some latter epochs. Thus, we further present two mixing variants to combine these two weighting metrics.

Hard-mixing
We adopt the prediction loss at the first η burn-in epochs and then switch to use the variance of prediction losses as:

$$w = \frac{1}{\tau}[\mathbf{I}_{t\leq\eta} \cdot (-\mathbf{y}^T \log p(\mathbf{y}|\mathbf{x})) + \mathbf{I}_{t>\eta} \cdot std(h^{t-1}) + \epsilon], \quad (6)$$

where the indicator function $\mathbf{I}_A$ equals 1 when A is true and 0 otherwise.

Soft-mixing
In the hard-mixing variant, the hyperparameter η needs to be carefully adjusted, and the two metrics are mutually exclusive at each fine-tuning epoch. To explore a potentially more flexible combination of the two metrics, we propose to use a soft-mixing variant as:

$$w = \frac{1}{\tau}[\beta \cdot (-\mathbf{y}^T \log p(\mathbf{y}|\mathbf{x})) + (1-\beta) \cdot std(h^{t-1}) + \epsilon], \quad (7)$$

where β is a balancing ratio that decreases linearly from 1 to 0 at fine-tuning epochs. Experimental results show that IW-Fit with the soft-mixing weighting metric generally performs the best among all compared fine-tuned methods.

Convergence analysis

Following Reddi et al. (2016), we give a convergence analysis in Theorem 1 to show that IW-Fit stabilizes and converges to a stationary point, and the convergence rate is $O(1/\sqrt{T})$. The proof can be found in Appendix A.

Theorem 1 *Suppose the loss function of IW-Fit in (1) $f \in \mathcal{F}_n$, where $\mathcal{F}_n$ is the class of finite-sum Lipschitz smooth functions, has σ-bounded gradients, and the instance weight w is clipped to be bounded by $[\underline{w}, \overline{w}]$. Let $\alpha_t = \alpha = c/\sqrt{T}$ where $c = \sqrt{\frac{2(f(\boldsymbol{\theta}^0)-f(\boldsymbol{\theta}^*))}{L\sigma^2\underline{w}\overline{w}}}$, and $\boldsymbol{\theta}^*$ is an optimal solution. Then, the iterates of IW-Fit satisfy:*

$$\min_{0\leq t\leq T-1} \mathbb{E}[||\nabla f(\boldsymbol{\theta}^t)||^2] \leq \sqrt{\frac{2(f(\boldsymbol{\theta}^0) - f(\boldsymbol{\theta}^*))L\overline{w}}{T\underline{w}}}\sigma.$$

Experiments

In this section, we first introduce the two transfer learning tasks conducted in our experiments, following by compared baselines and training configurations. Next, the empirical results of IW-Fit with different configurations are reported. We want to empirically answer the following questions:

1. Can IW-Fit improve the classification accuracy on the target domain?
2. Why can IW-Fit boost the fine-tuning performance on the target domain?
3. How robust is IW-Fit?

We present our first set of experimental results to answer the first question. Testing accuracies of all methods show the advantage of IW-Fit over the baselines. Further, we divide the target instances into two types according to their differences from the source domain and observe that instances differing more from the source domain can benefit more. Finally, we demonstrate the robustness of IW-Fit regarding various aspects in our last experiment section.

Tasks

Our classification task is to use the review content to predict the number of stars ranging from 1 to 5 that the user has given (Johnson and Zhang, 2017; Tang, Qin, and Liu, 2015; Yang et al., 2016). To simulate the transfer learning setting and verify the effectiveness of IW-Fit, we conduct two tasks:

- ***Amazon* to *Yelp***: We conduct this task for cross-domain transfer. The source dataset from 2015 *Amazon* reviews (Zhang, Zhao, and LeCun, 2015; Conneau et al., 2016) is mainly about products of online shopping. The target dataset of *Yelp* reviews is obtained from the latest 2018 Yelp Dataset Challenge, which is mainly about the daily life such as restaurants or home services.
- ***Yelp-2015* to *Yelp-2016* and *Yelp-2015* to *Yelp-2017***: The topics on the web change frequently and the reviews can be easily outdated. From the results shown in later sections, transfer learning is beneficial for this cross-time transfer task. From *Yelp*, we select out reviews in the year 2015/2016/2017 as the subsets *Yelp-2015*/*2016*/*2017*. We use *Yelp-2015* as the source dataset, and *Yelp-2016*/*2017* as the target datasets, respectively. No overlapped data exist among *Yelp-2015*, *2016* and *2017*.

Dataset construction: In each set of experiments, we randomly select $100,000$ instances from the source domain to construct the source dataset. Correspondingly, $10,000$ training instances and $1,000$ testing instances are randomly sampled from each target domain to construct the target dataset. The number of instances in each category is the same for all datasets on both the source and the target domains. We also try various data sizes for both domains in our last experiment section. To demonstrate the performance stability of the compared methods, we randomly sample three sets of source/target datasets for each task and run three sets of experiments for each of our experimental settings.

Baselines

Since we aim at improving the transfer learning performance on the fine-tuning stage, our focus is to compare with the fine-tuning baseline in this paper. Additionally, several other baselines are compared in our experiment:

- **SrcOnly**: It trains the model only on the source data, i.e., $\mathcal{D} = \mathcal{D}_s$. The obtained model is usually not generalized well on the target domain due to different data distributions between $\mathcal{D}_s$ and $\mathcal{D}_t$.
- **TgtOnly**: The model is trained on the target data with $\mathcal{D} = \mathcal{D}_t$. However, we often have limited training data on the target domain.
- **All**: Both datasets are adopted to train the model, i.e., $\mathcal{D} = \{\mathcal{D}_s, \mathcal{D}_t\}$. This method also suffers from the data distribution gap between $\{\mathcal{D}_s, \mathcal{D}_t\}$ and $\mathcal{D}_t$.

Method		*Amazon* to *Yelp*
Baselines	TgtOnly	50.22 ± 0.11
	SrcOnly	46.30 ± 0.06
	All	50.94 ± 0.08
Fine-tuned	Uniform	52.46 ± 0.06
	Gradient	52.80 ± 0.09
	Variance	53.20 ± 0.07
	Hard-mixing	53.44 ± 0.08
	Soft-mixing	$\mathbf{53.54 \pm 0.07}$

Table 1: Testing accuracies from *Amazon* to *Yelp*.

Method		*Yelp-2016*	*Yelp-2017*
Baselines	TgtOnly	50.23 ± 0.11	51.14 ± 0.10
	SrcOnly	56.43 ± 0.10	54.06 ± 0.11
	All	56.97 ± 0.11	54.73 ± 0.10
Fine-tuned	Uniform	57.18 ± 0.09	55.08 ± 0.08
	Gradient	57.82 ± 0.10	55.08 ± 0.08
	Variance	57.60 ± 0.08	55.66 ± 0.10
	Hard-mixing	58.13 ± 0.09	55.78 ± 0.10
	Soft-mixing	$\mathbf{58.30 \pm 0.09}$	$\mathbf{55.90 \pm 0.10}$

Table 2: Testing accuracies from *Yelp-2015* to *Yelp-2016* / *Yelp-2017*.

Training Configurations

Since we are interested in validating the usefulness of IW-Fit on the fine-tuning stage, we use the same DNN structures, hyperparameters, and datasets for pre-training and fine-tuning for all compared methods for a fair comparison.

We adopt the recurrent neural network with long short-term memory (LSTM) as the text encoder due to its superior performance in various text classification problems (Liu et al., 2016; Liu, Qiu, and Huang, 2017; Lin et al., 2017; Yang, Salakhutdinov, and Cohen, 2017). The encoder is followed by a fully-connected (FC) layer with ReLU activation and a Softmax layer that outputs the final prediction probabilities. The hyperparameters are set as: *vocabulary size* = $30,000$, *embedding size* = 100, *LSTM hidden size* = 200, *FC layer size* = 200, *dropout rate* = 0.2, *L2 regularization* = $1e-4$, *batch size* = 32, *fine-tuning epochs* = 50, *burn-in epochs* = 10, *learning rate* = $1e-3$. We also change to use a CNN-based text encoder and vary different parameter settings in section Robustness Study.

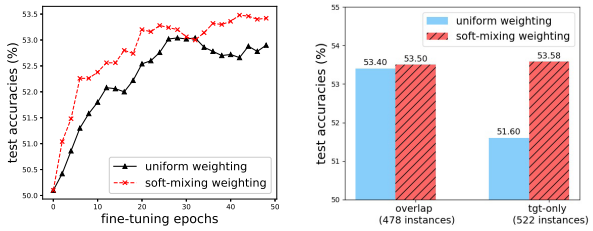
Due to the randomness of DNNs, we repeat five runs over each training dataset and report the mean and the standard error regarding the testing accuracy on the target domain.

Results of All Compared Methods

Due to the limit of space, we show one set of results for each task. The other two sets of results for all the tasks are provided in the Appendices.

Table 1 reports the classification accuracy results on the task from *Amazon* to *Yelp*. [2] Table 2 reports the results on the task from *Yelp-2015* to *Yelp-2016*/*Yelp-2017*. It can be observed that all fine-tuned methods perform better than the baselines. IW-Fit achieves superior performance compared to the fine-tuned method with uniform weighting. Specifically, IW-Fit with the soft-mixing weighting metric performs the best with an increased accuracy of 1.08% on the task *Amazon* to *Yelp*, 1.12% on task *Yelp-2015* to *Yelp-2016*, and 0.82% on the task *Yelp-2015* to *Yelp-2017*. We attribute this advantage to the flexible combination of the two metrics, which enables the algorithm to emphasize domain-specific instances while balancing the model to preserve the shared knowledge. In the other two sets of experiments in Appendix B, the accuracies increase by 0.80% and 0.65% on the task *Amazon* to *Yelp*. In appendix C, the accuracies increase by 0.76% and 0.64% on the task *Yelp-2015* to *Yelp-2016*, and 0.70% and 1.22% on the task *Yelp-2015* to *Yelp-2017*. As can be seen, IW-Fit can consistently improve the classification accuracy on the target domain.

[2]For all reported results, bold numbers are significantly better than the second best results with p-values smaller than 0.01.



(a) (b)

Figure 2: Testing accuracies of the fine-tuned methods with uniform and soft-mixing weighting: (a) at the fine-tuning epochs; (b) on two types of target instances.

In Figure 2-(a), we further plot the testing accuracies of IW-Fit with soft-mixing weighting metric and the fine-tuned method with uniform weighting along the fine-tuning epochs on the task *Amazon* to *Yelp*. We can observe that with the use of IW-Fit, the testing accuracy on the target domain consistently increases during the entire fine-tuning process.

Results and Analysis of IW-Fit

To understand why IW-Fit can help fine-tuning, we first analyze what types of target instances can benefit from our method. We extract the top 1000 frequent tokens in *Amazon* and *Yelp* domains, respectively. The two sets of tokens have an overlap of 575 tokens. Then, we divide the 1000 testing instances in *Yelp* domain into two types: 1) *tgt-only* instances that do not contain any frequent *Amazon* tokens; 2) *overlap* instances that contain tokens which appear in both domains. *Tgt-only* instances should be considered more different from the source domain than *overlap* instances.

Figure 2-(b) shows the testing accuracies on the two types of target instances. The test accuracy on the 478 *overlap* instances increases slightly by 0.10%, which shows that IW-Fit well preserves the shared knowledge between the source and the target domains. The performance on the other 522 *tgt-only* instances is improved obviously with an increased accuracy of 1.98%. It can be observed that a better boost can

Metric	Weight	Example
Prediction loss (Epoch 1)	0.53 (low)	(Star 5) My mirrors turned out *perfect*, I highly *recommend* ABC Glasses!
		(Star 1) *Worst* customer service ever.
	1.44 (high)	(Star 5) So many options for you to include that are **vegetarian friendly**.
		(Star 1) **Shrimp tempura** was **greasy**.
Variance of prediction losses (Epoch 20)	0.62 (low)	(Star 5) *Great* experience. *Nice* staff.
		(Star 1) *Terrible* service, I will never go back and do not recommend at all.
	1.37 (high)	(Star 5) Everything is **delish**! Brownie **dessert** in made when you ordered it.
		(Star 1) **Overpriced** and so **salty**. My tongue was tingling due to the crazy amount of MSG in the broth.

Table 3: Training examples with different weights at Epochs 1 and 20 using IW-Fit. *Italic* words commonly appear on both domains, while **bold** words appear more frequently on the target than the source domain.

be achieved on instances that differ more from the source domain. This indicates that the advantage of IW-Fit is due to capturing more specific knowledge of the target domain while well preserving the shared knowledge.

Table 3 gives some training instances with different weights on the task *Amazon* to *Yelp* using IW-Fit. The weights are computed according to *the prediction loss* at Epoch 1 and according to *the variance of historical prediction losses* at Epoch 20. We can observe that by both metrics, instances with high weights tend to contain more domain-specific words, which appear more frequently on the target than the source domain. Correspondingly, instances with low weights are likely to contain domain-independent words that commonly appear on both domains. This result also illustrates the effectiveness of the proposed metrics for assigning instance weights used in IW-Fit.

Robustness Study

In this section, we conduct various experiments to demonstrate the robustness of IW-Fit regarding five aspects including: 1) using CNN as the text encoder; 2) varying different parameters for fine-tuning; 3) varying the target data size for fine-tuning; 4) varying the source data size for pre-training; 5) varying the training mini-batch size. Here we show the results of Uniform and Soft-mixing for better readability. Full results of all compared methods can be found in the Appendices.

Using CNN as the text encoder

Since our method and the proposed metrics do not depend on the specific design of the DNN-based classifiers, we investigate the performance of IW-Fit under different DNN structures by changing to use Text-CNN (Kim, 2014) as our text encoder. Table 4 shows the testing accuracies of Soft-mixing and Uniform. It can be observed that the proposed model-agnostic method still achieves statistically significant improvement on the fine-tuning stage under the CNN architecture.

Varying the sets of model parameters for fine-tuning

In Problem Formulation, we have discussed that previous fine-tuning methods study to transfer different sets of parameters for fine-tuning. For the LSTM-based text classifier used in our experiments, the model parameters include word embeddings, parameters in LSTM and those in FC layers. To test the robustness of our method regarding the transferred model parameters, we keep some model parameters fixed and only fine-tune the other parameters.

Tables 5 and 8 report the results when the encoder including word embedding and LSTM is fixed (i.e., only fine-tune parameters in FC), when only the word embedding is fixed (i.e., fine-tune parameters in LSTM and FC) and when no parameters are fixed (i.e., fine-tune all parameters). We can see that in these three fine-tuning settings, Soft-Mixing always performs better than Uniform. Additionally, both methods yield the best performance when fine-tuning all the model parameters. We conjecture that this is due to the appropriate model structure for both the source and the target datasets so that fine-tuning all the model parameters results in the best performance.

Varying the fine-tuning data size.

Moreover, the performance of the fine-tuned methods should be verified on datasets with different sizes (Chung, Lee, and Glass, 2018; Howard and Ruder, 2018). This experiment is conducted to study the relationship between the amount of target data and IW-Fit. We vary the training data size of the target dataset in the fine-tuning process. The results are shown in Tables 6 and 9. As expected, the performance of Uniform and Soft-mixing is improved when the number of training data used for fine-tuning increases. Soft-mixing achieves the best performance for most settings, and it obtains a consistent improvement in testing accuracy by 0.38% to 1.12% compared to Uniform.

Varying the pre-training data size.

We also vary the size of the source dataset used for pre-training to analyze the relationship between the amount of source data and IW-Fit. The results are shown in Tables 7 and 10. Usually, the performance of the fine-tuned methods can be boosted as the source data size increases. It can be inferred that model pre-trained on a larger source domain can provide a better initialization of parameters for the target domain. Consistent with the above observations, Soft-mixing can stably improve the classification accuracy by $0.50\% \sim 1.30\%$ on the target domain.

Varying the training mini-batch size

Generally, small batch sizes tend to produce convergence in fewer epochs while large ones offer more data-parallelism and better computational efficiency (Keskar et al., 2016). To investigate the robustness of IW-Fit regarding the training batch size, we vary the batch size from 16 to 64 on the task *Amazon* to *Yelp*. Table 11 reports the testing accuracies of Soft-mixing and Uniform. It shows that the proposed method achieves better classification accuracy regarding different mini-batch sizes during fine-tuning.

Conclusion and Future Work

In this paper, we proposed the IW-Fit method, which incorporates the idea of instance weighting into the fine-tuning

	Amazon to *Yelp*	*Yelp-2015* to *Yelp-2016*	*Yelp-2015* to *Yelp-2017*
Uniform	47.94 ± 0.06	55.44 ± 0.08	53.26 ± 0.04
Soft-mixing	$\mathbf{48.56 \pm 0.07}$	$\mathbf{56.00 \pm 0.06}$	$\mathbf{53.80 \pm 0.03}$

Table 4: Testing accuracies of Uniform and Soft-mixing implemented with the CNN architecture.

Fixed parameters	*Yelp2015* to *Yelp-2016*			*Yelp2015* to *Yelp-2017*		
	encoder	embedding	None	encoder	embedding	None
Uniform	56.96 ± 0.02	56.78 ± 0.02	57.18 ± 0.09	53.46 ± 0.04	54.93 ± 0.06	55.08 ± 0.08
Soft-mixing	$\mathbf{57.68 \pm 0.03}$	$\mathbf{57.60 \pm 0.04}$	$\mathbf{58.30 \pm 0.09}$	$\mathbf{54.84 \pm 0.04}$	$\mathbf{55.48 \pm 0.07}$	$\mathbf{55.90 \pm 0.10}$

Table 5: Testing accuracies of varying fine-tuning parameters from *Yelp-2015* to *Yelp-2016* / *Yelp-2017*.

Percentage of target set	*Yelp2015* to *Yelp-2016*			*Yelp2015* to *Yelp-2017*		
	25%	50%	75%	25%	50%	75%
Uniform	56.58 ± 0.03	56.98 ± 0.02	57.15 ± 0.05	54.40 ± 0.03	54.52 ± 0.06	54.90 ± 0.04
Soft-mixing	$\mathbf{57.28 \pm 0.06}$	$\mathbf{57.92 \pm 0.06}$	$\mathbf{58.20 \pm 0.10}$	$\mathbf{54.95 \pm 0.08}$	55.42 ± 0.10	$\mathbf{55.28 \pm 0.03}$

Table 6: Testing accuracies of varying the target fine-tuning data size from *Yelp-2015* to *Yelp-2016* / *Yelp-2017*.

Percentage of source set	*Yelp2015* to *Yelp-2016*			*Yelp2015* to *Yelp-2017*		
	25%	50%	75%	25%	50%	75%
Uniform	54.58 ± 0.10	55.77 ± 0.07	56.26 ± 0.05	53.63 ± 0.05	55.00 ± 0.10	55.05 ± 0.06
Soft-mixing	$\mathbf{55.23 \pm 0.09}$	$\mathbf{56.37 \pm 0.07}$	$\mathbf{57.56 \pm 0.06}$	$\mathbf{54.83 \pm 0.07}$	$\mathbf{55.73 \pm 0.10}$	$\mathbf{55.55 \pm 0.03}$

Table 7: Testing accuracies of varying the source pre-training data size from *Yelp-2015* to *Yelp-2016* / *Yelp-2017*.

Fixed parameters	encoder	embedding	None
Uniform	47.98 ± 0.04	51.00 ± 0.05	52.46 ± 0.06
Soft-mixing	$\mathbf{48.23 \pm 0.03}$	$\mathbf{51.53 \pm 0.10}$	$\mathbf{53.54 \pm 0.07}$

Table 8: Testing accuracies by varying fine-tuning parameters from *Amazon* to *Yelp*.

Percentage of target set	25%	50%	75%
Uniform	49.72 ± 0.09	50.90 ± 0.10	51.68 ± 0.10
Soft-mixing	$\mathbf{50.46 \pm 0.09}$	$\mathbf{51.50 \pm 0.09}$	$\mathbf{52.36 \pm 0.10}$

Table 9: Testing accuracies by varying the target fine-tuning data size from *Amazon* to *Yelp*.

Percentage of source set	25%	50%	75%
Uniform	50.30 ± 0.09	51.15 ± 0.08	52.20 ± 0.07
Soft-mixing	$\mathbf{51.20 \pm 0.09}$	$\mathbf{51.78 \pm 0.08}$	$\mathbf{52.70 \pm 0.10}$

Table 10: Testing accuracies of varying the source pre-training data size from *Amazon* to *Yelp*.

stage to improve the text classification accuracy on the target domain. The proposed method enabled the algorithm

Batch size	16	32	64
Uniform	51.78 ± 0.05	52.46 ± 0.06	51.56 ± 0.07
Soft-mixing	$\mathbf{52.30 \pm 0.04}$	$\mathbf{53.54 \pm 0.06}$	$\mathbf{52.12 \pm 0.05}$

Table 11: Testing accuracies of Uniform and Soft-mixing regarding the mini-batch size on the task *Amazon* to *Yelp*.

to emphasize the specific knowledge on the target domain, while well preserving the shared knowledge between the source and the target domains. Rather than relying on specific rules or statistical approaches, the weighting metrics are model-agnostic and easy to implement for general DNN-based structures. We performed extensive experiments on two kinds of transfer learning tasks to verify the advantage of IW-Fit. Experimental results showed the effectiveness of the weighting metrics, and that IW-Fit consistently boosted the classification accuracy on the fine-tuning stage. In the future, we will try to extend our idea for applications on more NLP tasks such as sequence labeling and text generation.

References

Angeli, G.; Premkumar, M. J. J.; and Manning, C. D. 2015. Leveraging linguistic structure for open domain information extraction. In *ACL*, volume 1, 344–354.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J.

2009. Curriculum learning. In *ICML*, 41–48.

Cao, Z.; Li, W.; Li, S.; and Wei, F. 2017. Improving multi-document summarization via text classification. In *AAAI*, 3053–3059.

Chung, Y.-A.; Lee, H.-Y.; and Glass, J. 2018. Supervised and unsupervised transfer learning for question answering. In *NAACL HLT*, volume 1, 1585–1594.

Conneau, A.; Schwenk, H.; Barrault, L.; and Lecun, Y. 2016. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*.

Felbo, B.; Mislove, A.; Søgaard, A.; Rahwan, I.; and Lehmann, S. 2017. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. In *EMNLP*, 1615–1625.

Howard, J., and Ruder, S. 2018. Universal language model fine-tuning for text classification. In *ACL*, volume 1, 328–339.

Jiang, J., and Zhai, C. 2007. Instance weighting for domain adaptation in NLP. In *ACL*, 264–271.

Jiang, L.; Meng, D.; Zhao, Q.; Shan, S.; and Hauptmann, A. G. 2015. Self-paced curriculum learning. In *AAAI*, volume 2, 2694–2700.

Johnson, R., and Zhang, T. 2017. Deep pyramid convolutional neural networks for text categorization. In *ACL*, volume 1, 562–570.

Keskar, N. S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; and Tang, P. T. P. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR*.

Kim, Y.-B.; Stratos, K.; Sarikaya, R.; and Jeong, M. 2015. New transfer learning techniques for disparate label sets. In *ACL*, volume 1, 473–482.

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

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. In *ICLR*.

Landeiro, V., and Culotta, A. 2016. Robust text classification in the presence of confounding bias. In *AAAI*, 186–193.

Lin, Z.; Feng, M.; Santos, C. N. d.; Yu, M.; Xiang, B.; Zhou, B.; and Bengio, Y. 2017. A structured self-attentive sentence embedding. In *ICLR*.

Liu, P.; Qiu, X.; Chen, J.; and Huang, X. 2016. Deep fusion lstms for text semantic matching. In *ACL*, volume 1, 1034–1043.

Liu, P.; Qiu, X.; and Huang, X. 2017. Adversarial multi-task learning for text classification. In *ACL*, volume 1, 1–10.

Lu, Z.; Zhu, Y.; Pan, S. J.; Xiang, E. W.; Wang, Y.; and Yang, Q. 2014. Source free transfer learning for text classification. In *AAAI*, 122–128.

McCann, B.; Bradbury, J.; Xiong, C.; and Socher, R. 2017. Learned in translation: Contextualized word vectors. In *NIPS*, 6294–6305.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*, 3111–3119.

Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global vectors for word representation. In *EMNLP*, 1532–1543.

Reddi, S. J.; Hefny, A.; Sra, S.; Poczos, B.; and Smola, A. 2016. Stochastic variance reduction for nonconvex optimization. In *ICML*, 314–323.

Ruder, S., and Plank, B. 2017. Learning to select data for transfer learning with bayesian optimization. In *EMNLP*, 372–382.

Ruder, S.; Ghaffari, P.; and Breslin, J. G. 2017. Data selection strategies for multi-domain sentiment analysis. *arXiv preprint arXiv:1702.02426*.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In *ICML*, 1139–1147.

Tang, D.; Qin, B.; and Liu, T. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *EMNLP*, 1422–1432.

Wang, S., and Manning, C. D. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *ACL*, 90–94.

Wang, R.; Finch, A.; Utiyama, M.; and Sumita, E. 2017a. Sentence embedding for neural machine translation domain adaptation. In *ACL*, volume 2, 560–566.

Wang, R.; Utiyama, M.; Liu, L.; Chen, K.; and Sumita, E. 2017b. Instance weighting for neural machine translation domain adaptation. In *EMNLP*, 1482–1488.

Xu, W.; Sun, H.; Deng, C.; and Tan, Y. 2017. Variational autoencoder for semi-supervised text classification. In *AAAI*, 3358–3364.

Yang, Z.; Yang, D.; Dyer, C.; He, X.; Smola, A.; and Hovy, E. 2016. Hierarchical attention networks for document classification. In *NAACL HLT*, 1480–1489.

Yang, Z.; Salakhutdinov, R.; and Cohen, W. W. 2017. Transfer learning for sequence tagging with hierarchical recurrent networks. In *ICLR*.

Zhang, Y.; Barzilay, R.; and Jaakkola, T. 2017. Aspect-augmented adversarial networks for domain adaptation. *Transactions of the Association of Computational Linguistics* 5(1):515–528.

Zhang, T.; Huang, M.; and Zhao, L. 2018. Learning structured representation for text classification via reinforcement learning. In *AAAI*, 6053–6060.

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level convolutional networks for text classification. In *NIPS*, 649–657.