Universal Language Model Fine-tuning for Text Classification

Jeremy Howard[*]
fast.ai
University of San Francisco
j@fast.ai

Sebastian Ruder[*]
Insight Centre, NUI Galway
Aylien Ltd., Dublin
sebastian@ruder.io

Abstract

Inductive transfer learning has greatly impacted computer vision, but existing approaches in NLP still require task-specific modifications and training from scratch. We propose Universal Language Model Fine-tuning (ULMFiT), an effective transfer learning method that can be applied to any task in NLP, and introduce techniques that are key for fine-tuning a language model. Our method significantly outperforms the state-of-the-art on six text classification tasks, reducing the error by 18-24% on the majority of datasets. Furthermore, with only 100 labeled examples, it matches the performance of training from scratch on $100\times$ more data. We open-source our pretrained models and code[1].

1 Introduction

Inductive transfer learning has had a large impact on computer vision (CV). Applied CV models (including object detection, classification, and segmentation) are rarely trained from scratch, but instead are fine-tuned from models that have been pretrained on ImageNet, MS-COCO, and other datasets (Sharif Razavian et al., 2014; Long et al., 2015a; He et al., 2016; Huang et al., 2017).

Text classification is a category of Natural Language Processing (NLP) tasks with real-world applications such as spam, fraud, and bot detection (Jindal and Liu, 2007; Ngai et al., 2011; Chu et al., 2012), emergency response (Caragea et al., 2011), and commercial document classification, such as for legal discovery (Roitblat et al., 2010).

[1]http://nlp.fast.ai/ulmfit.

[*]Equal contribution. Jeremy focused on the algorithm development and implementation, Sebastian focused on the experiments and writing.

While Deep Learning models have achieved state-of-the-art on many NLP tasks, these models are trained from scratch, requiring large datasets, and days to converge. Research in NLP focused mostly on *transductive* transfer (Blitzer et al., 2007). For *inductive* transfer, fine-tuning pretrained word embeddings (Mikolov et al., 2013), a simple transfer technique that only targets a model's first layer, has had a large impact in practice and is used in most state-of-the-art models. Recent approaches that concatenate embeddings derived from other tasks with the input at different layers (Peters et al., 2017; McCann et al., 2017; Peters et al., 2018) still train the main task model from scratch and treat pretrained embeddings as fixed parameters, limiting their usefulness.

In light of the benefits of pretraining (Erhan et al., 2010), we should be able to do better than *randomly initializing* the remaining parameters of our models. However, inductive transfer via fine-tuning has been unsuccessful for NLP (Mou et al., 2016). Dai and Le (2015) first proposed fine-tuning a language model (LM) but require millions of in-domain documents to achieve good performance, which severely limits its applicability.

We show that not the idea of LM fine-tuning but our lack of knowledge of how to train them effectively has been hindering wider adoption. LMs overfit to small datasets and suffered catastrophic forgetting when fine-tuned with a classifier. Compared to CV, NLP models are typically more shallow and thus require different fine-tuning methods.

We propose a new method, Universal Language Model Fine-tuning (ULMFiT) that addresses these issues and enables robust inductive transfer learning for any NLP task, akin to fine-tuning ImageNet models: The same 3-layer LSTM architecture—with the same hyperparameters and no additions other than tuned dropout hyperparameters—outperforms highly engineered models and transfer learning approaches on six widely studied text classification tasks. On IMDb, with 100 labeled examples, ULMFiT matches the performance of training from scratch with $10\times$ and—given 50k unlabeled examples—with $100\times$ more data.

Contributions Our contributions are the following: 1) We propose Universal Language Model Fine-tuning (ULMFiT), a method that can be used to achieve CV-like transfer learning for any task for NLP. 2) We propose *discriminative fine-tuning*, *slanted triangular learning rates*, and *gradual unfreezing*, novel techniques to retain previous knowledge and avoid catastrophic forgetting during fine-tuning. 3) We significantly outperform the state-of-the-art on six representative text classification datasets, with an error reduction of 18-24% on the majority of datasets. 4) We show that our method enables extremely sample-efficient transfer learning and perform an extensive ablation analysis. 5) We make the pretrained models and our code available to enable wider adoption.

2 Related work

Transfer learning in CV Features in deep neural networks in CV have been observed to transition from *general* to task-*specific* from the first to the last layer (Yosinski et al., 2014). For this reason, most work in CV focuses on transferring the first layers of the model (Long et al., 2015b). Sharif Razavian et al. (2014) achieve state-of-the-art results using features of an ImageNet model as input to a simple classifier. In recent years, this approach has been superseded by fine-tuning either the last (Donahue et al., 2014) or several of the last layers of a pretrained model and leaving the remaining layers frozen (Long et al., 2015a).

Hypercolumns In NLP, only recently have methods been proposed that go beyond transferring word embeddings. The prevailing approach is to pretrain embeddings that capture additional context via other tasks. Embeddings at different levels are then used as features, concatenated either with the word embeddings or with the inputs at intermediate layers. This method is known as hypercolumns (Hariharan et al., 2015) in CV[2] and is used by Peters et al. (2017), Peters et al. (2018), Wieting and Gimpel (2017), Conneau et al. (2017), and McCann et al. (2017) who use language modeling, paraphrasing, entailment, and Machine Translation (MT) respectively for pretraining. Specifically, Peters et al. (2018) require engineered custom architectures, while we show state-of-the-art performance with the same basic architecture across a range of tasks. In CV, hypercolumns have been nearly entirely superseded by end-to-end fine-tuning (Long et al., 2015a).

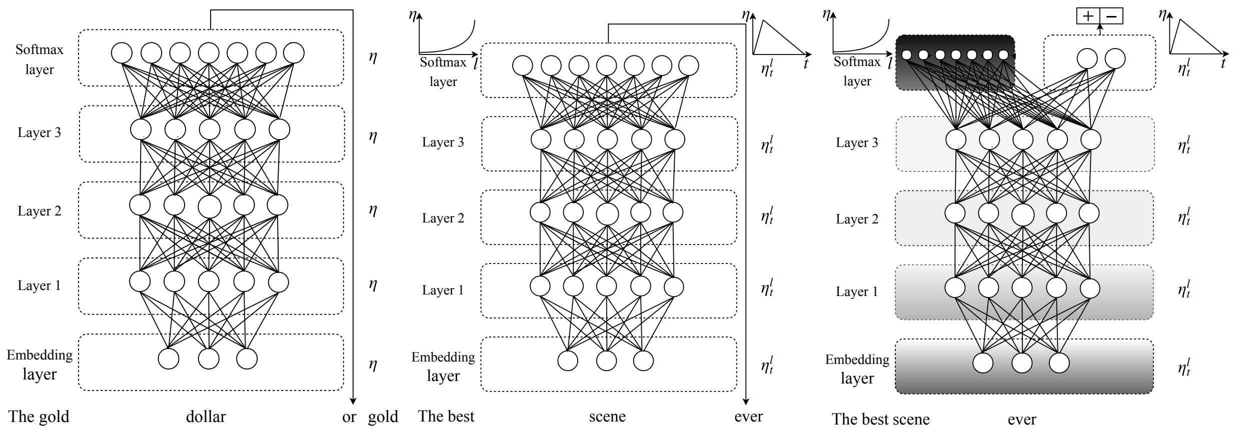
Multi-task learning A related direction is multi-task learning (MTL) (Caruana, 1993). This is the approach taken by Rei (2017) and Liu et al. (2018) who add a language modeling objective to the model that is trained jointly with the main task model. MTL requires the tasks to be trained from scratch every time, which makes it inefficient and often requires careful weighting of the task-specific objective functions (Chen et al., 2017).

Fine-tuning Fine-tuning has been used successfully to transfer between similar tasks, e.g. in QA (Min et al., 2017), for distantly supervised sentiment analysis (Severyn and Moschitti, 2015), or MT domains (Sennrich et al., 2015) but has been shown to fail between unrelated ones (Mou et al., 2016). Dai and Le (2015) also fine-tune a language model, but overfit with 10k labeled examples and require millions of in-domain documents for good performance. In contrast, ULMFiT leverages general-domain pretraining and novel fine-tuning techniques to prevent overfitting even with only 100 labeled examples and achieves state-of-the-art results also on small datasets.

3 Universal Language Model Fine-tuning

We are interested in the most general *inductive* transfer learning setting for NLP (Pan and Yang, 2010): Given a static source task $\mathcal{T}_S$ and *any* target task $\mathcal{T}_T$ with $\mathcal{T}_S \neq \mathcal{T}_T$, we would like to improve performance on $\mathcal{T}_T$. Language modeling can be seen as the ideal source task and a counterpart of ImageNet for NLP: It captures many facets of language relevant for downstream tasks, such as long-term dependencies (Linzen et al., 2016), hierarchical relations (Gulordava et al., 2018), and sentiment (Radford et al., 2017). In contrast to tasks like MT (McCann et al., 2017) and entailment (Conneau et al., 2017), it provides data in near-unlimited quantities for most domains and languages. Additionally, a pretrained LM can be easily adapted to the idiosyncrasies of a target

[2]A hypercolumn at a pixel in CV is the vector of activations of all CNN units above that pixel. In analogy, a hypercolumn for a word or sentence in NLP is the concatenation of embeddings at different layers in a pretrained model.



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

Figure 1: ULMFiT consists of three stages: a) The LM is trained on a general-domain corpus to capture general features of the language in different layers. b) The full LM is fine-tuned on target task data using discriminative fine-tuning (‘*Discr*’) and slanted triangular learning rates (STLR) to learn task-specific features. c) The classifier is fine-tuned on the target task using gradual unfreezing, ‘*Discr*’, and STLR to preserve low-level representations and adapt high-level ones (shaded: unfreezing stages; black: frozen).

task, which we show significantly improves performance (see Section 5). Moreover, language modeling already is a key component of existing tasks such as MT and dialogue modeling. Formally, language modeling induces a hypothesis space $\mathcal{H}$ that should be useful for many other NLP tasks (Vapnik and Kotz, 1982; Baxter, 2000).

We propose Universal Language Model Fine-tuning (ULMFiT), which pretrains a language model (LM) on a large general-domain corpus and fine-tunes it on the target task using novel techniques. The method is *universal* in the sense that it meets these practical criteria: 1) It works across tasks varying in document size, number, and label type; 2) it uses a single architecture and training process; 3) it requires no custom feature engineering or preprocessing; and 4) it does not require additional in-domain documents or labels.

In our experiments, we use the state-of-the-art language model AWD-LSTM (Merity et al., 2017a), a regular LSTM (with no attention, short-cut connections, or other sophisticated additions) with various tuned dropout hyperparameters. Analogous to CV, we expect that downstream performance can be improved by using higher-performance language models in the future.

ULMFiT consists of the following steps, which we show in Figure 1: a) General-domain LM pretraining (§3.1); b) target task LM fine-tuning (§3.2); and c) target task classifier fine-tuning (§3.3). We discuss these in the following sections.

3.1 General-domain LM pretraining

An ImageNet-like corpus for language should be large and capture general properties of language. We pretrain the language model on Wikitext-103 (Merity et al., 2017b) consisting of 28,595 preprocessed Wikipedia articles and 103 million words. Pretraining is most beneficial for tasks with small datasets and enables generalization even with 100 labeled examples. We leave the exploration of more diverse pretraining corpora to future work, but expect that they would boost performance. While this stage is the most expensive, it only needs to be performed once and improves performance and convergence of downstream models.

3.2 Target task LM fine-tuning

No matter how diverse the general-domain data used for pretraining is, the data of the target task will likely come from a different distribution. We thus fine-tune the LM on data of the target task. Given a pretrained general-domain LM, this stage converges faster as it only needs to adapt to the idiosyncrasies of the target data, and it allows us to train a robust LM even for small datasets. We propose *discriminative fine-tuning* and *slanted triangular learning rates* for fine-tuning the LM, which we introduce in the following.

Discriminative fine-tuning As different layers capture *different types of information* (Yosinski et al., 2014), they should be fine-tuned to *different extents*. To this end, we propose a novel fine-tuning method, *discriminative fine-tuning*[3].

Instead of using the same learning rate for *all* layers of the model, discriminative fine-tuning allows us to tune *each* layer with different learning rates. For context, the regular stochastic gradient descent (SGD) update of a model's parameters θ at time step t looks like the following (Ruder, 2016):

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta) \tag{1}$$

where η is the learning rate and $\nabla_\theta J(\theta)$ is the gradient with regard to the model's objective function. For discriminative fine-tuning, we split the parameters θ into $\{\theta^1, \ldots, \theta^L\}$ where θ^l contains the parameters of the model at the l-th layer and L is the number of layers of the model. Similarly, we obtain $\{\eta^1, \ldots, \eta^L\}$ where η^l is the learning rate of the l-th layer.

The SGD update with discriminative fine-tuning is then the following:

$$\theta_t^l = \theta_{t-1}^l - \eta^l \cdot \nabla_{\theta^l} J(\theta) \tag{2}$$

We empirically found it to work well to first choose the learning rate η^L of the last layer by fine-tuning only the last layer and using $\eta^{l-1} = \eta^l/2.6$ as the learning rate for lower layers.

Slanted triangular learning rates For adapting its parameters to task-specific features, we would like the model to quickly converge to a suitable region of the parameter space in the beginning of training and then refine its parameters. Using the same learning rate (LR) or an annealed learning rate throughout training is not the best way to achieve this behaviour. Instead, we propose *slanted triangular learning rates* (STLR), which first linearly increases the learning rate and then linearly decays it according to the following update schedule, which can be seen in Figure 2:

$$\begin{aligned} cut &= \lfloor T \cdot cut\_frac \rfloor \\ p &= \begin{cases} t/cut, & \text{if } t < cut \\ 1 - \frac{t-cut}{cut \cdot (1/cut\_frac - 1)}, & \text{otherwise} \end{cases} \\ \eta_t &= \eta_{max} \cdot \frac{1 + p \cdot (ratio - 1)}{ratio} \end{aligned} \tag{3}$$

where T is the number of training iterations[4], cut_frac is the fraction of iterations we increase the LR, cut is the iteration when we switch from increasing to decreasing the LR, p is the fraction of the number of iterations we have increased or will decrease the LR respectively, $ratio$ specifies how much smaller the lowest LR is from the maximum LR η_{max}, and η_t is the learning rate at iteration t. We generally use $cut\_frac = 0.1$, $ratio = 32$ and $\eta_{max} = 0.01$.

STLR modifies triangular learning rates (Smith, 2017) with a short increase and a long decay period, which we found key for good performance.[5] In Section 5, we compare against aggressive cosine annealing, a similar schedule that has recently been used to achieve state-of-the-art performance in CV (Loshchilov and Hutter, 2017).[6]

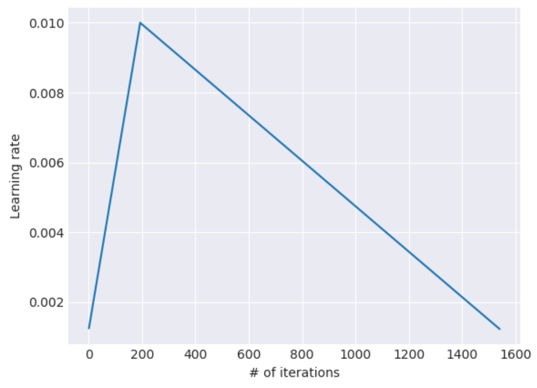


Figure 2: The slanted triangular learning rate schedule used for ULMFiT as a function of the number of training iterations.

3.3 Target task classifier fine-tuning

Finally, for fine-tuning the classifier, we augment the pretrained language model with two additional linear blocks. Following standard practice for CV classifiers, each block uses batch normalization (Ioffe and Szegedy, 2015) and dropout, with ReLU activations for the intermediate layer and a softmax activation that outputs a probability distribution over target classes at the last layer. Note that the parameters in these task-specific classifier layers are the only ones that are learned from scratch. The first linear layer takes as the input the pooled last hidden layer states.

Concat pooling The signal in text classification tasks is often contained in a few words, which may

[3] An unrelated method of the same name exists for deep Boltzmann machines (Salakhutdinov and Hinton, 2009).

[4] In other words, the number of epochs times the number of updates per epoch.

[5] We also credit personal communication with the author.

[6] While Loshchilov and Hutter (2017) use multiple annealing cycles, we generally found one cycle to work best.