Paper 15 | Universal Language Model Fine-tuning for Text Classification by Jeremy Howard and Sebastian Ruder – published in May 2018

Abstract

1. 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
2. The proposed method significantly outperforms the state-of-the-art on six test classification tasks, reducing the error buy 18-24% on the majority of datasets

Introduction

1. Research in NLP focused mostly on transductive transfer. For inductive transfer, fine-tuning pre-trained word embeddings, 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
2. Language models overfit to small datasets and suffered catastrophic forgetting when fine-tuned with a classifier
3. The contributions of this paper involve:
    a. Proposed ULMFiT, a method that can be used to achieve computer vision-like transfer learning for any task for NLP
    b. Proposed discriminative fine-tuning, slanted triangular learning rates, and gradual unfreezing
    c. Shown that the proposed method enables extremely sample-efficient transfer learning

Universal Language Model Fine-tuning

1. Proposed ULMFiT, which pretrains a language model (LM) on a large general-domain corpus and fine-tunes it on the target task using novel techniques
2. This method is universal because
    a. Works across tasks varying in document size, number, and label type
    b. Uses a single architecture and training process
    c. Requires no custom feature engineering or preprocessing
    d. Does not require additional in-domain documents or labels
3. In the experiments, the authors use the state-of-the-art language model AWD-LSTM, a regular LSTM with various tuned dropout hyperparameters

4. ULMFiT consists of the following steps:

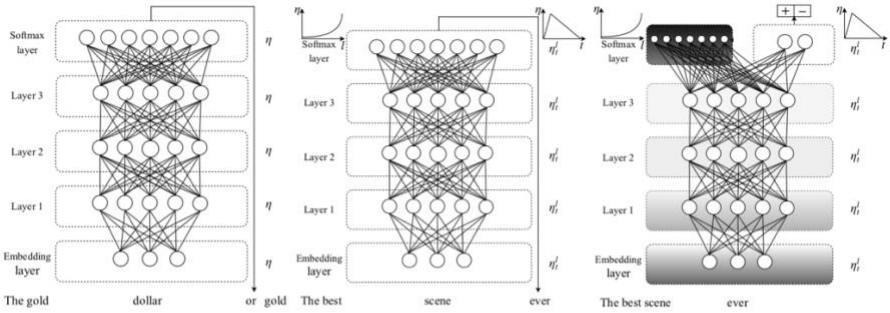

Figure 1: ULMFiT consists of three stages: a) The LM is trained on a general-domain corpus to capture general features of the language in different layers. b) The full LM is fine-tuned on target task data using discriminative fine-tuning ('*Discr*') and slanted triangular learning rates (STLR) to learn 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).

   a. General-domain LM pretraining
      - The corpus being used should be large and capture general properties of language
      - The authors pretrain the language model on Wikitext-103 consisting of 28595 pre-processed Wikipedia articles and 103 million words
      - Pretraining is most beneficial for tasks with small datasets and enables generalisation even with 100 labelled examples
   b. Target task LM fine-tuning
      - The data of the target task will likely to come from a different distribution
      - Therefore, we need to fine-tune the LM on data of the target task
      - 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
      - Proposed following fine-tuning for LM:
        - Discriminative fine-tuning
          i. Allows us to tune each layer with different learning rates
          ii. Therefore, for each layer, we will have a different set of parameters as well as different learning rates
          iii. Empirically, the authors found that it worked well to first choose the learning rate of the last layer and using it as the learning rate for lower layers
        - Slanted triangular learning rates (STLR)
          i. 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
          ii. Using the same learning rate throughout training is not the best way!
          iii. STLR consists of first linearly increases the learning rate and then linearly decays it according to an update schedule:

$$cut = \lfloor T \cdot cut\_frac \rfloor$$

$$p = \begin{cases} t/cut, & \text{if } t < cut \\ 1 - \frac{t-cut}{cut \cdot (1/cut\_frac - 1)}, & \text{otherwise} \end{cases} \quad (3)$$

$$\eta_t = \eta_{max} \cdot \frac{1 + p \cdot (ratio - 1)}{ratio}$$

T : number of training iterations
cut_frac : fraction of iterations we increase the LR
cut : iteration where we switch from increasing to decreasing LR
p : fraction of the number of iterations we have increased or will decrease the LR
ratio : how much smaller the lowest LR is from the max LR
n_t : the learning rate at iteration t

c. Target task classifier fine-tuning
    - Augment the pretrained LM with two additional linear blocks. The first linear layer takes as the input the pooled last hidden layer states
    - Concat pooling
        - As input documents can consists of hundreds of words, information may get lost if we only consider the last hidden state of the model
        - Therefore, we concatenate the hidden state at the last time step with both the max-pooled and the mean-pooled representation of the hidden states over as many time steps as fit in GPU memory
    - Gradual unfreezing
        - If we fine-tune all layers at once, we risk catastrophic forgetting
        - Propose gradually unfreezing the model, starting from the last layer as this contains the least general knowledge
            i. Unfreeze the last layer and fine-tune all unfrozen layers for one epoch
            ii. Unfreeze the next lower frozen layer and repeat, until we fine-tune all layers until convergence at the last iteration
    - Backpropagation through time for large documents
        - Divide the document into fixed-length batches of size b
        - At the beginning of each batch, the model is initialised with the final state of the previous batch
        - Keep track of the hidden states for mean and max-pooling
        - Gradients are back-propagated to the batches whose hidden states contributed to the final prediction
    - Bidirectional language model
        - For all the experiments, the authors pretrain both a forward and a backward LM
        - Fine-tune a classifier for each LM independently and average the classifier predictions

Results

	Model	Test		Model	Test
IMDb	CoVe (McCann et al., 2017)	8.2	TREC-6	CoVe (McCann et al., 2017)	4.2
	oh-LSTM (Johnson and Zhang, 2016)	5.9		TBCNN (Mou et al., 2015)	4.0
	Virtual (Miyato et al., 2016)	5.9		LSTM-CNN (Zhou et al., 2016)	3.9
	ULMFiT (ours)	**4.6**		ULMFiT (ours)	**3.6**

Table 2: Test error rates (%) on two text classification datasets used by McCann et al. (2017).

	AG	DBpedia	Yelp-bi	Yelp-full
Char-level CNN (Zhang et al., 2015)	9.51	1.55	4.88	37.95
CNN (Johnson and Zhang, 2016)	6.57	0.84	2.90	32.39
DPCNN (Johnson and Zhang, 2017)	6.87	0.88	2.64	30.58
ULMFiT (ours)	**5.01**	**0.80**	**2.16**	**29.98**

Table 3: Test error rates (%) on text classification datasets used by Johnson and Zhang (2017).

1. The method outperforms CoVe, a state-of-the-art transfer learning, on both datasets
2. Despite pretraining on significantly less data, we consistently outperform
3. In-depth analysis
    a. Low-shot learning
        - Supervised and unsupervised ULMFiT both outperform the model that trained from scratch with 10x and 20x more data, demonstrating the benefit of general-domain LM pretraining
    b. Impact of pretraining

Pretraining	IMDb	TREC-6	AG
Without pretraining	5.63	10.67	5.52
With pretraining	**5.00**	**5.69**	**5.38**

Table 4: Validation error rates for ULMFiT with and without pretraining.

        - Pretraining is most useful for small and medium-sized datasets, which are most common in commercial applications
    c. Impact of LM quality

LM	IMDb	TREC-6	AG
Vanilla LM	5.98	7.41	5.76
AWD-LSTM LM	**5.00**	**5.69**	**5.38**

Table 5: Validation error rates for ULMFiT with a vanilla LM and the AWD-LSTM LM.

        - On the smaller TREC-6, a vanilla LM without dropout runs the risk of overfitting, which decreases performance
        - The other datasets show that a vanilla LM still achieves good results

d. Impact of LM fine-tuning

LM fine-tuning	IMDb	TREC-6	AG
No LM fine-tuning	6.99	6.38	6.09
Full	5.86	6.54	5.61
Full + discr	5.55	6.36	5.47
Full + discr + stlr	**5.00**	**5.69**	**5.38**

Table 6: Validation error rates for ULMFiT with different variations of LM fine-tuning.

- Fine-tuning the LM is most beneficial for larger datasets
- The fine-tuning implemented improve the performance across all three datasets and are necessary on the smaller TREC-6, where regular fine-tuning is not beneficial

e. Impact of classifier fine-tuning

Classifier fine-tuning	IMDb	TREC-6	AG
From scratch	9.93	13.36	6.81
Full	6.87	6.86	5.81
Full + discr	5.57	6.21	5.62
Last	6.49	16.09	8.38
Chain-thaw	5.39	6.71	5.90
Freez	6.37	6.86	5.81
Freez + discr	5.39	5.86	6.04
Freez + stlr	5.04	6.02	5.35
Freez + cos	5.70	6.38	**5.29**
Freez + discr + stlr	**5.00**	**5.69**	5.38

Table 7: Validation error rates for ULMFiT with different methods to fine-tune the classifier.

- Fine-tuning the classifier significantly improves over training from scratch, particularly on small dataset
- 'Last' severely underfits and is never able to lower the training error to 0
- The full ULMFiT (bottom row) shows the best performance across 2 of the datasets and a competitive result on AG. It is the only method that shows excellent performance across the board

Future directions

1. Language model fine-tuning will be particularly useful in the following:
    a. NLP for non-English languages, where training data for supervised pretraining tasks is scarce
    b. New NLP tasks where non-state-of-the-art architecture exists
    c. Tasks with limited amounts of labelled data
2. Improve language model pretraining and fine-tuning and make them more scalable
3. Apply the method to novel tasks and models such as entailment or question answering
4. More studies are required to better understand what knowledge a pretrained language model captures, how these changes during fine-tuning, and what information different tasks require