The merits of Universal Language Model Fine-tuning for Small Datasets – a case with Dutch book reviews

Benjamin van der Burgh
Leiden Institute of Advanced Computer Science
Leiden University
b.van.der.burgh@liacs.leidenuniv.nl

Suzan Verberne
Leiden Institute of Advanced Computer Science
Leiden University
s.verberne@liacs.leidenuniv.nl

Abstract

We evaluated the effectiveness of using language models, that were pre-trained in one domain, as the basis for a classification model in another domain: Dutch book reviews. Pre-trained language models have opened up new possibilities for classification tasks with limited labelled data, because representation can be learned in an unsupervised fashion. In our experiments we have studied the effects of training set size (100–1600 items) on the prediction accuracy of a ULMFiT classifier, based on a language models that we pre-trained on the Dutch Wikipedia. We also compared ULMFiT to Support Vector Machines, which is traditionally considered suitable for small collections. We found that ULMFiT outperforms SVM for all training set sizes and that satisfactory results (~90%) can be achieved using training sets that can be manually annotated within a few hours. We deliver both our new benchmark collection of Dutch book reviews for sentiment classification as well as the pre-trained Dutch language model to the community.

1 Introduction

Typically, results for supervised learning increase with larger training set sizes. However, many real-world text classification tasks rely on relatively small data, especially for applications in specific domains. Often, a large, *unlabelled* text collection is available to use, but *labelled* examples require human annotation. This is expensive and time-consuming. Since deep and complex neural architectures often require a large amount of labeled data, it has been difficult to significantly beat the traditional models – such as Support Vector Machines – with neural models (Adhikari et al., 2019).

In 2018, a breakthrough was reached with the use of pre-trained neural language models and transfer learning (Howard and Ruder, 2018; Peters et al., 2018; Devlin et al., 2018; Liu et al., 2019). Transfer learning no longer requires models to be trained from scratch but allows researchers and developers to reuse features from models that were trained on different, much larger text collections (e.g. Wikipedia). For this pre-training, no explicit labels are needed; instead, the models are trained to perform straightforward language modelling tasks, i.e. predicting words in the text.

Even though the models are trained on these seemingly trivial predictive tasks, transfer learning with these models is highly effective: the pre-trained language models can be fine-tuned to perform classification tasks with a relatively small amount of labelled task-specific data. Thus, pre-trained language models can alleviate the small labelled data size for domain-specific data sets.

In their 2018 paper, Howard and Ruder show the success of transfer learning with Universal Language Model Fine-tuning (ULMFiT) for six text classification tasks. They also demonstrate that the model has a relatively small loss in accuracy when reducing the number of training examples to as few as 100 (Howard and Ruder, 2018).

In this paper we further address the use of ULMFiT for small training set sizes. We consider the case of data from a new domain, where we have a large amount of unlabelled data, but limited labelled data. Given the vast number of network parameters and the limited number of training instances (100 to 1600), we expect to quickly overfit on the training data if all parameters are optimized using the small labelled data, often referred to as *catastrophic forgetting*. Alternatively, we ‘freeze’ the parameters of the language model, which means that we fix all network parameters except for the parameters of the final layer. In doing so, we limit the ability of the model to adapt to the target domain, but in return avoid the problem of catastrophic forgetting, since the language model parameters are untouched.

In this paper, we evaluate ULMFiT with a pre-trained language model and fixed hyperparameters for the representation layers. We only tune the drop-out multiplier and learning rate for the linear layers.

For our experiments on Dutch texts, we created a new data collection consisting of Dutch-language book reviews. We fine-tune a general pre-trained Wikipedia model on the reviews collection. We then take various-sized labelled portions of the book review data to (a) investigate the effect of training set size, and (b) compare the accuracy of ULMFiT to the accuracy of Support Vector Machines (SVM).

The contributions of this paper compared to previous work are: (1) We deliver a new benchmark dataset for sentiment classification in Dutch; (2) We deliver pre-trained ULMFiT models for Dutch language; (3) We show the merit of pre-trained language models for small labeled datasets, compared to traditional classification models.

2 Data

Data set We released the 110k Dutch Book Reviews Dataset (110kDBRD).[1] This dataset contains book reviews along with associated binary sentiment polarity labels. It is inspired by the Large Movie Review Dataset (Maas et al., 2011) and intended as a benchmark for sentiment classification in Dutch. We scraped 110 thousand book reviews from the website Hebban.[2] These reviews each consist of a text and a score from 1 to 5, which we converted to categorical labels (1 and 2: negative; 3: neutral; 4 and 5: positive).

Data split For our experiments, we split the 110k documents as follows: 20k documents are used in the classifier training and evaluation (positive and negative classes balanced). Of those 20k, we reserve 5k documents as a held-out test set that cannot be consulted during training nor language model pre-training. The remaining 15k documents are used for training the classifier.

Data sampling for training For the experiments on dataset size we use the following training set sizes $m = \{100, 200, 400, 800, 1600\}$. Each experiment is trained 10 times to investigate model stability. These 10 subsamples are chosen randomly out of the complete 15k training set (not balanced). Note that the same test set is used for all experiments to make the results directly comparable.

[1] https://benjaminvdb.github.io/110kDBRD/, also including the scripts used to scrape the data from the review website.

[2] https://www.hebban.nl

3 Language model training

3.1 General-domain language model pre-training

In order to learn text representations we use the AWD-LSTM language modelling architecture originally used by ULMFiT and implemented in the Fast.ai Python library. This library also constructs a supervised dataset by randomly masking out words in a text. We use a unidirectional language model, i.e., the target word is predicted using words that precede it. Similarly to Howard and Ruder (2018), we have chosen Wikipedia for language modelling, because it provides a large, freely available corpus of high quality. Moreover, we found that the pre-processing scripts for the English version of Wikipedia could be re-used for the Dutch.

We used a recent dump of Wikipedia and converted it to raw text, which was then split on white spaces into tokens. After that, we replaced all numbers with the same placeholder token, such that the specific value is ignored, but the fact that a number occurred can be used in the model. The 60k most frequent tokens were included in the vocabulary V and the remaining out-of-vocabulary words were replaced with a special 'unknown' token. An embedding layer of size 400 was used to learn a dense token representation, followed by 3 LSTM layers with 1150 hidden units each to form the encoder.

This is followed by a classification module that maps each representation to a score $0 \leq s_t \leq 1$, for each token $t \in V$ where $\sum s_t = 1$ and can as such be interpreted as a probability distribution over the vocabulary. We used the reference implementation of ULMFiT in the Fast.ai Python library. The entire Wikipedia dataset was split into 92M tokens for training and 185k for both testing and validating the language model. A slanted triangular learning rate (Howard and Ruder, 2018) with a learning rate of $5 * 10^{-3}$ was used for 20 epochs.

3.2 Target task language model fine-tuning

After training the language model on Wikipedia, we continued training on data from our target domain, i.e., the 110k Dutch Book Review Dataset. The preprocessing was done similarly to the preprocessing on Wikipedia, but the vocabulary of the previous step was reused. We used all data except for a 5k holdout set (105k reviews) to fine-tune network parameters using the same slanted triangular learning rates. However, this time we first trained the parameters of the classification module to convert the pre-trained features into predictions for the new target dataset. After that all network parameters were trained for 10 epochs.

3.3 Target task classifier fine-tuning

The goal is to predict the sentiment polarity (positive or negative) given a review text. Therefore, the training dataset is constructed such that the dependent variable represents a sentiment polarity instead of a token from the vocabulary. The encoder of the language model is kept, such that a dense representation can be constructed given an input text, and the classification module is replaced to adjust for the new target classes.

4 Experiments

4.1 Preprocessing

We applied the default text processing implemented in the Fast.ai Python library by splitting on whitespace and padding texts within a batch to the same length. The amount of required padding characters was reduced by grouping texts of similar length together, while adding some randomness during training to avoid showing the network with the same batches in each epoch.

4.2 Hyperparameters

We optimized hyperparameters for each training set size and for each fold using HpBandster.[3] A one-cycle policy, as outlined in (Smith, 2018), was used, which requires a lower and upper bound for the momentum, describing its adaptive curve during a single epoch. This resulted in five optimized hyperparameters: learning rate, momentum lower and upper, dropout and batch size. In the objective function, we optimized for binary cross-entropy loss.

[3]`https://github.com/automl/HpBandSter`

4.3 Baselines

We compared our classification models to Linear Support Vector Machines (SVM) because it is a commonly used and well performing classifier for small text collections. We used the implementation of LinearSVC in scikit-learn.[4] LinearSVC has one hyperparameter, C, which we optimized using HpBandster on the range of values from 10^{-4} to 10^{4} with squared hinge loss as optimization function (default for LinearSVC in scikit-learn). For feature extraction we used the CountVectorizer and TF-IDF transformer in scikit-learn. TF-IDF weights were trained on the same 105k documents on which the ULMFiT model was fine-tuned.

For comparison we also trained two models, one SVM and one ULMFiT model, with manually tuned hyperparameters on all available book reviews in the training set (15k). These models achieved 93.84% (ULMFiT) and 89.16% (SVM).

5 Results

5.1 Effect of training set size

The results of the experiments described in Section 4 can be found in Figure 1 (left). A few observations can be made from this plot. Firstly, for the ULMFiT model, the accuracy on the test set improves with each increase in the training dataset size, as can be expected.

Secondly, both models behave rather unstable for smaller training datasets, as can be seen by the large deviation from the mean and the outliers: different random subsamples give deviant results for the smaller training set sizes. Since the pre-trained model is based on data from a different domain, it can be expected that more than 100 instances are needed to accommodate for the new domain.

5.2 Comparison to SVM

Figure 1 compares the prediction accuracies for ULMFiT and SVM. We had expected the SVM model to perform better for smaller training set sizes, but it is outperformed by ULMFiT for each size. Also, the ULMFiT models show smaller deviations between random subsamples than the SVM models.

We also found that the prediction accuracy of the SVM model using all 15,000 training items

[4]`https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`

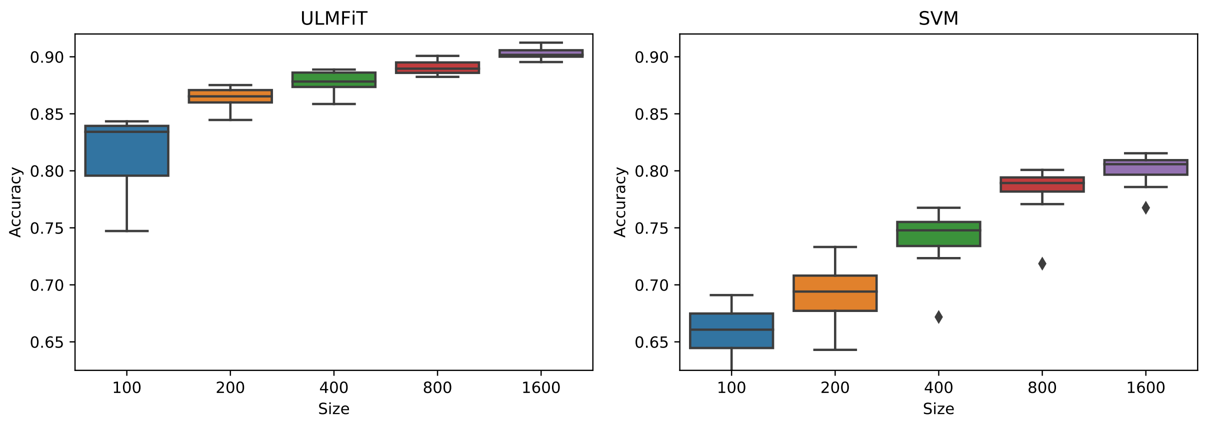

Figure 1: Results for ULMFiT (a) and SVM (b) in terms of accuracy on the test set with varying training set sizes. The boxes represent the deviation among the random subsample per training set size.

(89.16%) is surpassed by the ULMFiT model when using only 1600 training instances: all 10 random subsamples for ULMFiT reach an accuracy of at least 89.54% (the left purple box in Figure 1). This could mean that the pre-trained model captures many of the required characteristics of Dutch such that they can be largely used without modifications.

6 Conclusions

Pre-trained language models have opened up possibilities for classification tasks with limited labelled data. In our experiments we have studied the effects of training set size on the prediction accuracy of a ULMFiT classifier based on pre-trained language models for Dutch. In order to make a fair comparison, we have used state-of-the-art optimization methods to optimize the hyperparameters of each model.

Our results confirm what had been stated in (Howard and Ruder, 2018), but had not been verified for Dutch or in as much detail. For this particular dataset and depending on the requirements of the model, satisfactory results might be achieved using training sets that can be manually annotated within a few hours.

Moreover, a large part of modeling effort lies in the training of a language model on an – in this case – generic corpus, which can be reused for other domains. While the prediction accuracy could be improved by optimizing all network parameters on a large dataset, we have shown that training only the weights of the final layer outperforms our SVM models by a large margin.

ULMFiT uses a relatively simple architecture that can be trained on moderately powerful GPUs. This fact, combined with the general availability of unlabeled data and the ability to share language models, suggests that these methods could be applied in domains where manual labelling has traditionally been too expensive. Further research should be conducted to compare how differences between the source and target datasets affect the prediction accuracy and whether more powerful network architectures can also be used.

References

Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. 2019. Rethinking complex neural network architectures for document classification. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4046–4051, Minneapolis, Minnesota. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Nelson F Liu, Matt Gardner, Yonatan Belinkov, Matthew Peters, and Noah A Smith. 2019. Linguistic knowledge and transferability of contextual representations. *arXiv preprint arXiv:1903.08855*.

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 *Proceedings of the 49th Annual Meeting of* *the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Leslie N. Smith. 2018. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820.