Improving LSTM Document Classifiers by Optimizing Word Embedding Techniques

Kevin Liu, William Sweeny, Justin Tran
Department of Computer Science
Princeton University
{kl16, wsweeny, jctran}@princeton.edu

Abstract

This project aims to modify the LSTM$_{reg}$ architecture defined by (Adhikari et al., 2019). We attempt to improve upon the original architecture by introducing ELMo word embeddings (Peters et al., 2018) and a different schema for implementing dropout to augment the original model's word-level embedding dropout. We assess the impact of these word embedding techniques by running our model against a document classification task using the AAPD and Reuters datasets. As a baseline, we compare our results to the unmodified architecture, and analyze the differences in classification F_1 score between our modified architecture and the original.

1 Introduction

The task our model seeks to optimize for is text classification. Specifically, Adhikari et al. claim that simple, well-trained and well-parameterized models can be competitive with more sophisticated models, even without features like attention (Adhikari et al., 2019). Instead, techniques like word-level embedding dropout are explored to help optimize attentionless LSTM models. We intend to evaluate this claim by sticking with their basic model as a foundation, and making small, intentional changes to determine whether modifications to word embeddings can still lead to improvements in classification tasks. Additionally, these modifications to word embeddings are intended to align with the original author's goals of maintaining architectural simplicity.

2 Prior Work

Attention, hierarchical structure, and sequence generation have become more popular in recent literature, but this inevitably increases model complexity (Adhikari et al., 2019). For example, Wang et al. showed that attention-based LSTMs are great at obtaining superior results on document classification (Wang et al., 2016), but such increases in accuracy come at the cost of increased complexity. This runs contrary to the goal of decreasing complexity while retaining accuracy.

Adhikari et al. lament a widespread belief that current advances in ML are exaggerated by the lack of empirical rigor. There have been case studies where more advanced models struggle with simple baselines (Sculley et al., 2018). Notably, Melis et al. found that a properly regularized, standard LSTM model outperforms other recent and more complex models (Melis et al., 2017).

It's worth further exploring regularization as there have been numerous studies on word-level embedding dropout to prevent overfitting. Variants of dropout schemes in RNNs have been shown to yield improvements (Gal and Ghahramani, 2016) and word-level embedding dropout has been shown to be effective in LSTMs at achieving state-of-the-art results (Merity et al., 2017).

Lastly, incorporating ELMo (Peters et al., 2018) has brought about drastic improvements in results across a large variety of models by simply pluggin them into existing models.

3 Methodology

For our project, we will retrofit an existing LSTM architecture with additional layers, such as regularization and dropout layers. We intend to test several related architectures, each with a small, isolated change to its layers, and then evaluate the performance of each. As referenced in the future works section of Adhikari et al., we intend to incorporate ELMo word embeddings into this

framework and empirically evaluate its effects.

The authors of the original paper perform a type of dropout they call "embedding dropout" in which they drop out entire word embeddings, effectively removing some of the words during each iteration. We will refer to it as "word-level embedding dropout". One such change will be modifying the existing word-level embedding dropout used in the LSTM architecture. We will be replacing this with a form of "partial" dropout that doesn't drop out entire word embeddings, but instead drops out portions of *each* embedding vector. Word-level embedding dropout is used to mitigate the effects that a few selected strong words may have on influencing the model; the goal is to avoid scenarios where the model becomes overly reliant on certain words and then fails to incorporate the nuances of other words. We hypothesize that partial embedding dropout will offer an alternative approach that instead focuses on regularizing sub-word level information, thereby preventing the model from being overly attached to certain specific features of embeddings.

By making several small isolated changes, we hope to have fine-grained insight into which changes were beneficial or harmful to the accuracy of the model as a whole. The baseline architecture we will be modifying is LSTM$_{reg}$, as described in Adhikari et al. (Adhikari et al., 2019) The model consists of a bidirectional LSTM feeding into a concatenated set of forward and backward hidden features, and then aggregated via max-pooling over time.

4 Evaluation and Datasets

We will be using two datasets: the AAPD academic paper dataset and Reuters article tagging dataset.[1] Both the AAPD and Reuters datasets are multi-labeled datasets for classification. The AAPD academic paper dataset consists of abstracts and up to 54 corresponding subjects the academic paper pertains to. The target is to predict corresponding subjects of an academic paper according to the content of the abstract. The Reuters dataset consists of news articles that have been tagged with the trading commodity topics discussed in each article. Articles covering multiple topics may have multiple labels.

[1]All datasets located at following link: https://git.uwaterloo.ca/jimmylin/hedwig-data/tree/master/datasets.

In each case, to evaluate our models, we will compute the average accuracy and F_1 scores of each of our modified models on each dataset, and compare these scores to the unmodified baseline model. The baseline model evaluates the AAPD and Reuters datasets with an F_1 score.

We will also compare the training time of each model to see which features most dramatically affect the speed of computation, and assess each model's architectural complexity on a qualitative scale.

5 Baseline Results

We began by setting up training the baseline LSTM$_{reg}$ model according to the parameters outlined in the original paper (Adhikari et al., 2019). This was done to ensure our model was training correctly on the given datasets with the stated parameters and that their experiment was reproducible. Note that Reuters and AAPD are multi-labeled datasets for classification and they are available and feasibly-sized datasets to train our models on. Also note that the Yelp 2014 and IMDB datasets included in the original paper were not evaluated by us.

The Yelp 2014 dataset could not be found in their repository and the IMDB dataset is not trained on due to its large size and extremely long training time in comparison to the other datasets.

	Val. F_1
Original Paper Results	89.1
Replicated Results	88.5

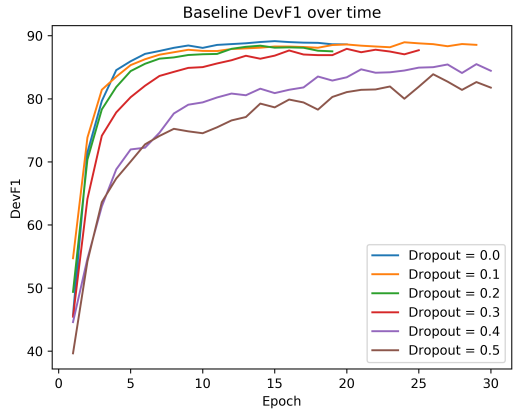
Table 1: Reuters Dataset F_1 Scores with LSTM$_{reg}$

	Val. F_1
Original Paper Results	73.1
Replicated Results	72.8

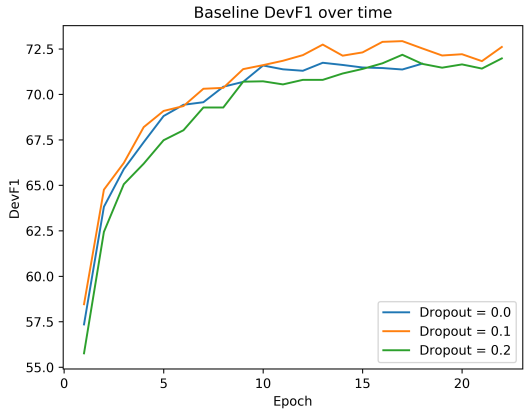
Table 2: AAPD Dataset F_1 Scores with LSTM$_{reg}$

Observe that the replicated results for both datasets receive similar F_1 scores to those found in the original paper. With these results, we are ready to move forward with tweaking the LSTM$_{reg}$ model.

We also investigated the importance of word-level embedding dropout which is a form of dropout defined by the authors as a method by which they perform "dropout on entire word embeddings, effectively removing some of the words



(a) Figure 1: Baseline dev F_1 scores for the Reuters dataset



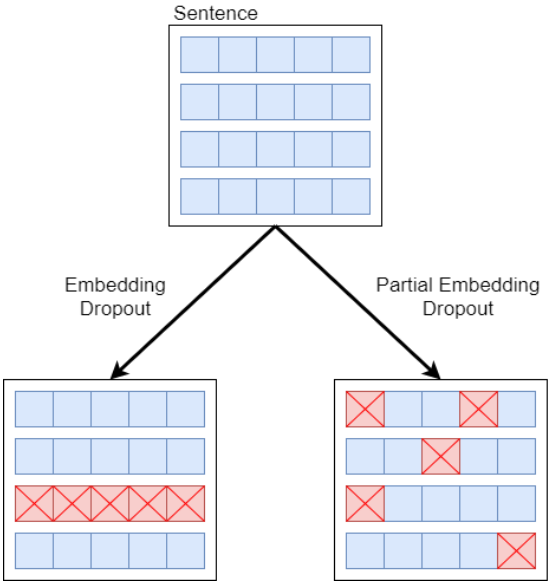
(a) Figure 2: Baseline dev F_1 scores for the AAPD dataset

at each training iteration". Take a word-level embedding dropout probability of 0.1: A given word embedding vector has a 10 percent chance of being dropped entirelyn . By varying the rates of dropout in the model, one can observe the effectiveness of utilizing embedding dropout in document classification.

We then tested varying the word-level embedding dropout rate of the LSTM_{reg} model from an interval of $[0, 0.5]$. We trained the model on a Google Cloud Compute n1-highmem-8 cluster running an NVIDIA K80 GPU.

We can see that even small dropout rates often lead to a decrease in the F_1 score of the model in the smaller Reuters dataset. On the larger AAPD dataset, there are extremely modest gains to F_1 for very small word-level embedding dropout rates, but these gains vanish as soon as the dropout rate increases.

Especially on larger datasets, word-level embedding dropout shows some potential for improved training performance, but the specific scheme of word-level embedding dropout used in LSTM_{reg} leaves much to be desired. These results motivate the investigation of other dropout schemes, which may offer similar or improved F_1 scores compared to models using only word-level embedding dropout, or using no dropout at all. In particular, we will investigate the effects of using a "partial embedding dropout", which we hope will allow our model to achieve competitive performance on both the smaller Reuters and larger AAPD datasets.



(a) Figure 3: Word-level Embedding Dropout versus *Partial* Embedding Dropout

Our proposed partial embedding dropout is a different form of dropout from the stated "word-level embedding dropout" used in the original paper. Rather than applying dropout at the word level, partial embedding dropout instead applies dropouts within the embedding vector itself. If the dropout level were set to 0.1, we would be dropping 10 percent of all features in each embedding rather than dropping ten percent of the words. See Figure **??** for a visual representation of this.

6 Experimental Setup

6.1 Training

In order to best understand the effect of our changes on overall model performance, we structured our experiment as an ablation study, and only altered one aspect of the model at a time.

We first configured our LSTM_{reg} model to run using ELMo word embeddings. This was achieved using an open-source ELMo implementation pro-

vided by AllenNLP[2], which generates length 1024 contextualized embeddings for each word in an input sentence. These contextualized embeddings are concatenated to the original length 300 word2vec embeddings, resulting in a length 1324 representation of each word.

Introducing ELMo embeddings into the model proved quite resource-intensive and increased training times dramatically, with per-epoch training times jumping from around two minutes to upwards of ten minutes, for a four-to-sixfold increase in total training time. While it is likely that the process of computing ELMo representations for each sentence requires additional time and computational resources, we do not believe this was the primary cause of increased train times. Instead, we conjecture the largest factor affecting training time was the increase in the size of embeddings used for each word; with embeddings being substantially larger in size, there is a significant amount more data to process, and correspondingly high training times are to be expected. It is probable that introducing ELMo embeddings with lower dimensionality would result in substantially faster training times, but it is unclear what effect this might have on the model's classification accuracy or F_1 score – these questions are left to future research.

The process of introducing ELMo word embeddings to the LSTM$_{reg}$ model was complicated by limitations of the underlying libraries. Both PyTorch in general and AllenNLP in particular rely upon NVIDIA's cuDNN library[3] in order to enable hardware acceleration of training using a GPU. Unfortunately, there is a known issue in the cuDNN library whereby large matrix inputs (such as those presented in the ELMo word embeddings) sometimes fail in situations where clients request a large batch size.[4] The issue is unfixed in cuDNN's code as of the time of writing, so we were forced to implement a workaround. As a result, on training runs where ELMo embeddings were used, we were forced to reduce the batch size from 32 (as was standard during all baseline trainings) to 16. It is possible that this reduction in batch size could have negatively impacted accuracy or F_1 scores.

[2]AllenNLP: https://github.com/allenai/allennlp/blob/master/tutorials/how_to/elmo.md

[3]cuDNN: https://developer.nvidia.com/cudnn

[4]cuDNN GitHub Issue: https://github.com/pytorch/pytorch/issues/4107#issuecomment-350906473

After introducing ELMo embeddings to the model, we moved on to the second major alteration to the model, and ran separate trials in which the default word-level embedding dropout was replaced by our modified partial embedding dropout scheme. Note that because this experiment is designed in the style of an ablation study, this modified dropout scheme was tested only in conjunction with the default word2vec embeddings, and not the modified ELMo word embeddings.

6.2 Hyperparameters

Unless otherwise specified, we used the default hyperparameters of the LSTM$_{reg}$ architecture for each dataset.

Specifically, we used a batch size of 32 (reduced to 16 for ELMo), an embedding dimension of 300 (increased to 1324 for ELMo), and a single hidden layer with dimension 512. The node dropout rate was set to 0.5, and the embedding dropout rate was varyied according to the methodology discussed in Section 7. On the Reuters dataset we used a learning rate of 0.01, and on the AAPD dataset we used a learning rate of 0.001. The model was trained for a maximum of 30 epochs, using early stopping with a patience of 5.

As in the original LSTM$_{reg}$ paper, all random seeds were set to 3435 for reproducibility.

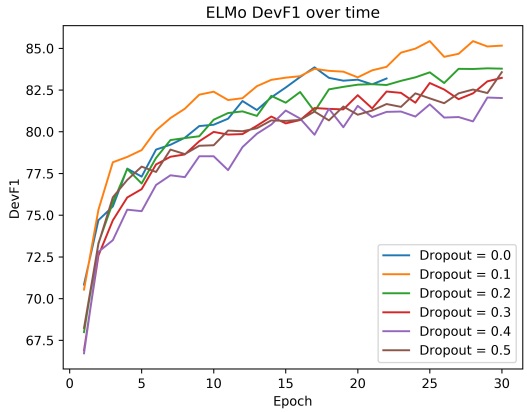
7 Results and Discussion

7.1 ELMo Embeddings

	Dropout Rate				
	0.1	**0.2**	**0.3**	**0.4**	**0.5**
F_1 Score	**85.45**	83.82	83.23	83.55	83.55

Table 3: Reuters Dataset Dev F_1 Scores with LSTM$_{reg}$ Model using ELMo Embeddings and Word-Level Embedding Dropout

Somewhat surprisingly, the addition of ELMo to word embeddings did not result in the significant gains to performance that were expected (but perhaps there simply weren't enough epochs realize such gains).



(a) Figure 4: ELMo Embeddings Reuters Dev F_1 scores

The version of the model trained using ELMo embeddings was only able to achieve a peak F_1 score of 85.45 on the Reuters dataset, compared to the peak F_1 of 89.1 achieved by the baseline model. The reasons behind this apparent lack of performance improvement are unclear. It is possible that the increased size of embeddings causes the ELMo-based model to train more slowly, and that given additional training time results would eventually surpass the baseline. Training beyond the 30 epochs specified could have led to greater F_1 scores. This is supported by Figures 3 and 4, which show that models using ELMo tend to plateau more gradually than those without. It is also possible that the ELMo results suffered due to the decrease in batch size necessitated by the cuDNN bug discussed in Section 6.

It appeared that a lower word-level embedding droprate led to greater F_1 scores for the model when using ELMo. As seen in Table 3, a droprate of 0.1 led to the highest F_1 score of 85.45 when compared to all other tested dropout rates. Figure 4 also supports this statement as a dropout rate of 0.1 consistently yields a higher F_1 score than all other dropout rates through most phases of training. The next closest F_1 score was only 83.82 when the dropout rate was 0.2. Dropout rates greater than 0.5 may have led to better models because the size of the word embeddings became so large when using ELMo. Though even higher dropout rates could have led to even higher F_1 scores, it appears that lower dropout rates when used with word-level embedding dropout yielded the most positive results.

7.2 Partial Embedding Dropout

Our experiments replacing Adhikari et al.'s word-level embedding dropout with our own form of partial embedding dropout yielded more promising results.

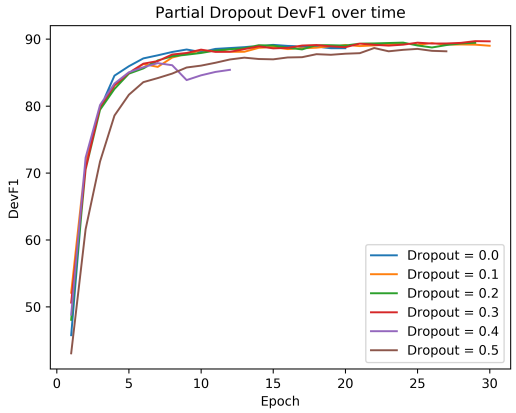
	Dropout Rate				
	0.1	**0.2**	**0.3**	**0.4**	**0.5**
F_1 Score	89.41	89.48	**89.70**	86.41	88.67

Table 4: Reuters Dataset Dev F_1 Scores with $LSTM_{reg}$ Model and Partial Embedding Dropout

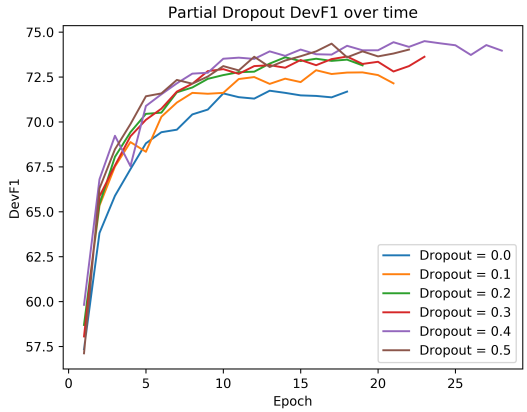
	Dropout Rate				
	0.1	**0.2**	**0.3**	**0.4**	**0.5**
F_1 Score	72.88	73.60	73.64	**74.50**	74.36

Table 5: AAPD Dataset Dev F_1 Scores with $LSTM_{reg}$ Model and Partial Embedding Dropout

The F_1 Scores from Table 3 indicate that certain dropout rates with partial embedding dropout perform on par with *or better* than the baseline results in the original paper. Notably, the model trained on the Reuters dataset with dropout rate of 0.3 resulted in a higher F_1 score than the average F_1 score provided by the baseline model with their word-level embedding dropout.



(a) Figure 5: Partial Embedding Dropout Reuters Dev F_1 scores



(a) Figure 6: Partial Embedding Dropout AAPD Dev F_1 scores

It is important to note that the original authors trained all neural models on five different seeds and presented their F_1 score results with a standard deviation. Given the limited time, funding, and GPU hardware available to us, we were only able to train our models with a single seed and thus cannot provide a standard deviation for our F_1 Scores.

The AAPD results turned out more impressive than the results on the Reuters dataset. The highest F_1 score of 74.50 was obtained with a dropout rate of 0.4. Compared to the baseline result of 73.1, 74.50 represents a small but sizable improvement. Notably, this result falls above the standard deviation of the baseline result, so we are confident that our partial embedding dropout scheme has some potential. However, we should approach these results with some caution. Since we lacked the resources to run multiple trials at the peak dropout, it is not clear whether our training run was particularly lucky or unlucky. Given additional funding, we could conduct further trials in an attempt to determine the exact size of the improvements made by our partial dropout.

Our approach is also beneficial in terms of its effects on the model's qualitative complexity. Our partial embedding dropout scheme makes only minor modifications to the conceptual framework of similar dropout schemes already in common use today. This conceptual similarity, combined with our scheme's ease of implementation and intuitive motivation give our scheme a high degree of pedagogical potential.

In terms of training time, our model is also competitive with the baseline. Analysis of the two dropout implementations reveals that our partial embedding dropout scheme uses a comparable number of matrix operations to the baseline word-level embedding dropout, and this is supported by empirical observations of the training time. There was no substantial difference between the running times of the partial dropout and word-level dropout schemes, with both yielding training times in the neighborhood of two minutes per epoch.

In summary, our partial embedding dropout scheme yielded promising results, matching or improving upon the F_1 scores of the word-level embedding dropout approach. Moreover, our dropout scheme is favorable in terms of training time and model complexity, since neither was substantially compromised in order to achieve our observed performance gains.

8 Conclusions and Future Work

Ultimately, our partial embedding dropout scheme was able to achieve modest success. On the other hand, the introduction of contextualized ELMo word embeddings was disappointing, increasing train times substantially while offering diminished performance. Though we were able to achieve F_1 scores above the baseline using partial embedding dropout, further research is required in order to compute the F_1 standard distribution and determine the precise effect size of our improved embedding dropout scheme.

A potential extension of this work is to complete running ELMo on the AAPD dataset (as we could not finish training these models given technical and time constraints). Though it is likely that it would lead to lower F_1 scores when compared to the baseline results, it is still a good idea to confirm the results on multiple datasets. With that being said, training our models on all datasets mentioned by the authors (AAPD, Reuters, IMDB, and Yelp 2014) with multiple seeds would also lead us towards this goal of consistency with the original paper.

In addition, we could seek to balance the size of the word2vec embeddings with the ELMo embeddings. Currently, we concatenate a length 300 word2vec embedding to a length 1024 ELMo embedding to create our length 1324 final word embedding. This is embedding is dominated by information from ELMo and tweaks to this could lead to better results. Reducing ELMo to a length 512 embedding would help this issue along with increasing the dimensions of the word2vec embed-

ding.

Another new type of dropout could also be developed by combining the original word-level embedding dropout with our partial embedding dropout. With this new form of dropout, a fraction of embeddings get dropped (as in word-level embedding dropout) but you only drop another fraction of the embedding and not the entire word (as in partial embedding dropout). Hopefully, this solution would combine the benefits of word-level and partial embedding dropout, allowing for the model to achieve improved generality at both a word-level and an embedding-level.

9 Contributions

We all contributed equally to this project by working together (quite literally , in the same room) so it's difficult to clearly delineate who did what. We each worked on different aspects of the code to train the models and collect the data. Each of us also contributed equally to the poster, and also to the writing of this report.
Kevin Liu: contributed equally
William Sweeny: contributed equally
Justin Tran: contributed equally

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.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1019–1027. Curran Associates, Inc.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2017. On the state of the art of evaluation in neural language models.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing lstm language models.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

D. Sculley, Jasper Snoek, Alex Wiltschko, and Ali Rahimi. 2018. Winner's curse? on pace, progress, and empirical rigor.

Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. 2016. Attention-based LSTM for aspect-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, Austin, Texas. Association for Computational Linguistics.