Deep Neural SCOTUS: A Comparative Analysis of Language-Modeling Techniques for US Supreme Court Opinions

Andrew A. Miller-Smith[‡]
MS, Statistics
Stanford University
aams@stanford.edu
Mentor: Xiaoxue Zang

Abstract

Language modeling plays a key role in computational linguistics, featuring prominently in many natural-language processing applications. While many studies have examined word-based representations of text, various groups have begun to experiment with character representations as well. In this paper I present a comparative study of two language-modeling approaches on a corpus of United States Supreme Court opinions: a standard Long Short-Term Memory model (LSTM) with word-embeddings and the same model enhanced with a character-level representation of the sentence derived from a Convolutional Neural Network (CNN). While both models struggled on large datasets–perhaps an indication of non-identically and non-independently distributed data–the combined model failed to achieve the same loss and perplexity as the word-based model on any inputs, achieving a test-set score of 2468.71 in comparison with the LSTM's 258.60. While these difficulties raise questions about the data and implementations used in this experiment, the results suggest the technique used to incorporate character-level information into the word model obfuscated signals as opposed to enhancing them.

1 Introduction

Since the advent of neural-network-based techniques for natural language processing, the field has relied heavily on the word vectors proposed by Tomas Mikolov et. al in their seminal paper, *Distributed Representations of Words and Phrases and their Compositionality*.[5]. These vectors allow for a flexible and distributed representation of tokens and creates an n-dimensional space in which to explore mathematical relations. Despite these achievements, such formatting implicitly discretizes meaning at the word level, ignoring the roles of character-level constructs. This creates difficulty when dealing with unknown words, a serious challenge when analyzing casual, idiosyncratic, or misspelled texts.

To address this issue, various groups have explored the possibility of character encodings. These vectors mimic the structure of word vectors, representing the distribution of neighboring characters surrounding a given character. A growing field of research involves the combination of word- and character-level features in models to offer the best of both worlds. In this paper I present a comparative study of two language models for United States Supreme Court Opinions, the former a standard word-based model and the latter the standard model enhanced with character-level information.

[*] LinkedIn: https://www.linkedin.com/in/andrewmillersmith1/
[†] GitLab: https://gitlab.com/Aams1000

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

2 Related work

Early successes using character-based models emerged in the mid 2010s. Yoon Kim et. al. in 2015[3] demonstrated the potential of using character embeddings in language models, employing a combination of a CNN, highway layers, and an LSTM to predict output at the word level. This approach matched state-of-the-art numbers on the English Penn Treebank while using sixty-percent fewer parameters than the competing word-based models. Luong and Manning [4] applied character embeddings to machine translation in 2016, combining a standard word-based LSTM approach with a supplementary character-based prediction mechanism for unknown words. This hybrid model achieved new state-of-the-art BLEU scores for multiple languages, including a ten-plus-point jump for English to Czech translation.

In addition to language modeling and translation, groups such as Liang et. al. [1] have used character embeddings in tasks such as relation classification. In their 2017 paper, the group combined character-level encodings from a CNN with word-level encodings from a bidirectional GRU to achieve state-of-the-art results on multiple data sets. These results suggest combining word-level and character-level information brings value to non-vocabulary classification tasks as well.

In this paper I draw particular inspiration from Kim et. al.'s CNN-, max-pooling-, and highway-layer-themed approach to character encoding as well as Liang et. al.'s idea of combining encodings in classification tasks.

3 Approach

In order to design a fair study, I attempted to keep the word-exclusive and the combined models roughly equivalent in terms of complexity. This proved challenging, as the CNN architectures used in character-based models frequently employ a multi-stage pipeline of convolutions and max pooling layers to achieve their results.

In the following descriptions, let e_{word} and e_{char} denote the size of the word and character embeddings respectively and let $|V_{word}|$ denote the size of the vocabulary. Both model uses pre-trained word vectors from the Stanford University GloVe dataset,[6] while the combined model trains its own character embeddings.

3.1 Word-based LSTM

The word-based model features a bidirectional LSTM followed by a highway layer and a linear projection. Given an input tensor $x \in \mathbb{R}^{(e_{word})}$, the LSTM passes each word through a shared linear projection, maintaining a hidden state $h_t \in \mathbb{R}^{e_{word}}$ and cell state $c_t \in \mathbb{R}^{e_{word}}$ for each time step t. The information from the internal states propagates as each new word enters, allowing for the sharing of prior knowledge.

The bidirectional LSTM outputs the final hidden states of both the forward and the backward pass. The algorithm concatenates these to form a vector $h_{cat} \in \mathbb{R}^{2e_{word}}$, passes them through a dropout layer, then feeds them through a highway layer into a vector $v \in \mathbb{R}^{2e_{word}}$. The highway layer computes the following vectors

$$
\begin{aligned}
v_{proj} &= W_{proj}v + b_{proj}) \in \mathbb{R}^{2e_{word}} \\
v_{gate} &= \sigma(W_{gate}v + b_{gate}) \in \mathbb{R}^{2e_{word}} \\
v_{highway} &= v_{proj} \odot v_{gate} + (1 - v_{gate}) \odot v \in \mathbb{R}^{2e_{word}}
\end{aligned}
$$

This residual connection allows the algorithm to optionally take advantage of the layer.

As a final step, this vector $v \in \mathbb{R}^{2e_{word}}$ passes through a linear projection into the dimension of the output space, producing the vector $v_{out} \in \mathbb{R}^{|V_{word}|}$. The model applies the softmax function to compute scores and compares them with the target words via the cross entropy loss.

3.2 Combined model

Whereas the word-based LSTM involves only one core model (followed by linear projections), the combined model uses two encoding mechanisms, one for words and one for characters.

3.2.1 Character encoder

Given an input tensor $x \in \mathbb{R}^{(w, e_{char})}$ (where w is the window size), the model passes the input through a one-dimensional convolutional layer to produce an output tensor v_{conv}. The convolution result moves through a leaky RELU activation before a final max pooling layer to produce the tensor $v_{pool} \in \mathbb{R}^{e_{char}}$.

$$\begin{aligned} v_{conv} &= \text{Conv1D}(x) \\ v_{pool} &= \text{MaxPool}(\text{LeakyRELU}(v_{conv})) \in \mathbb{R}^{e_{char}} \end{aligned}$$

After the pooling, the resulting tensor passes through a highway layer as described in the prior section, except this time with a LeakyRELU activation.

$$\begin{aligned} v_{proj} &= \text{LeakyRELU}(W_{proj} v_{pool} + b_{proj}) \in \mathbb{R}^{e_{char}} \\ v_{gate} &= \sigma(W_{gate} v_{pool} + b_{gate}) \in \mathbb{R}^{e_{char}} \\ v_{out} &= v_{proj} \odot v_{gate} + (1 - v_{gate}) \odot v_{pool} \in \mathbb{R}^{e_{char}} \end{aligned}$$

The encoder sends the highway output v_{out} through a dropout layer to produce the encoded vector $v_{char} \in \mathbb{R}^{e_{char}}$.

$$v_{char} = \text{Dropout}(v_{out}) \in \mathbb{R}^{e_{char}}$$

3.2.2 Word encoder

The combined model follows a similar architecture to that of the word-exclusive model, first passing the input window through a bidirectional LSTM. The resulting hidden states $h_1, h_2 \in \mathbb{R}^{e_{word}}$ from the forward and backward passes combine to form the vector $v_{word-cat} \in \mathbb{R}^{2e_{word}}$. This concatenated vector passes as-is into the decoding mechanism.

3.2.3 Decoder

Whereas initial designs for the combined-model decoder featured more sophisticated techniques frequently seen in character-exclusive models, I simplified the mechanism to preserve relative parity between the models. The decoder receives input vectors $v_{char} \in \mathbb{R}^{e_{char}}$ and $v_{word-cat} \in \mathbb{R}^{2e_{word}}$. Before combining the vectors, the model projects the word encoding through a linear layer into output $v_{word} \in \mathbb{R}^{e_{word}}$ (this packages the word information into a vector of comparable length to v_{char}).

The model concatenates v_{word} and v_{char} and projects them linearly into $\mathbb{R}^{e_{word}}$ before passing the result through a highway layer without activation. The highway output travels through a final linear projection into the vector $v_{out} \in \mathbb{R}^{|V_{word}|}$. This vector contains the model's predictions over the corpus, which the model compares against the target values via the cross-entropy loss.

4 Experiments

4.1 Data

Kaggle hosts a data set of approximately thirty-five-thousand United States Supreme Court (SCOTUS) rulings [2] between 1789 and 2017. The files include opinions of all types, such as majority, dissenting,

and concurring, and feature work from ninety-six justices. Entries include various meta data detailing the author, date filed, vote breakdown, and more. The samples are scraped from various legal websites, and while they offer faithful representations of the opinions, they require moderate Regex processing to standardize formatting and address errant unicode characters.

SCOTUS decisions present numerous idiosyncrasies as compared to most formal texts. The modern opinions frequently employ atypical or archaic diction and formatting, and older writings burst with anachronisms and misspellings. These opinions also include unusual footnotes and citations of other court cases. An example of a modern opinion comes from the *Hollingsworth v. Perry* (2013) excerpt below.

```
All the California Supreme Court's decision stands for is that, so far
as California is concerned, petitioners may ``assert legal arguments in
defense of the state's interest in the validity of the initiative
measure'' in federal court. 628 F. 3d 1191, 1193. That interest is by
definition a generalized one, and it is precisely because proponents assert
such an interest that they lack standing under this Court's prece-
dents.
```

Given the first sentence above, the language modeling algorithms would receive inputs and targets as follows.

Sample inputs and targets	
Input	Target
All the California Supreme Court	’
the California Supreme Court ’	s
California Supreme Court ’ s	decision
Supreme Court ’ s decision	stands
...	...

Table 1: Sample inputs and targets

4.2 Evaluation method

For an evaluation metric, I used the standard language-model technique of perplexity. Given a window of length t and a corpus of size T, one calculates the perplexity of a model's predictions as follows.

$$PPL = \prod_{t=1}^{T} \left(\frac{1}{P(x^{t+1}|x_1, \ldots, x_t)} \right)^{\frac{1}{T}}$$

Note that this is equivalent to the exponentiated cross entropy loss.

4.3 Experiment details

As mentioned in the Approach section, I kept hyperparameters constant across all models for the sake of a standardized comparison. I used a window size $w = 5$, a learning rate $\alpha = 0.01$, a dropout rate $p_{drop} = 0.3$, a batch size $b = 64$, a word embedding size $e_{word} = 200$, a character embedding size $e_{char} = 256$, a word vocabulary of size $|V_{word}| = 400001$, and a character vocabulary of size $|V_{char}| = 201$.

The sizes of the vocabularies correspond to the number of words and characters in the corpus plus values for unknown tokens, padding tokens, start tokens, and end tokens. I selected the embedding lengths in an attempt to balance memory and computation costs with expressiveness. The other values came at the example or recommendation of the Stanford University CS224N course staff. Both models trained with a training set of size 5,000,000, a validation set of size 40,000, and a test set of size 40,000.

4.4 Results

The word-exclusive model proved adept on toy datasets, achieving nearly one-hundred percent accuracy and perplexity scores of one on all inputs. As I scaled the size of the training sets, the model continued to overfit the training data, typically beginning with a loss between 5.00 and 6.00 and a perplexity of between 250 and 300 and improving to a little above 3.00 and 20 respectively.

The word-based model encountered difficulty with larger training sets. Though it performed well on data sets with up to 300,000 inputs, it struggled when training on a set of 500,000 examples, maintaining initial losses and perplexities yet failing to improve over dozens of epochs. Despite the constant training-set performance, validation perplexity progressively declined, beginning with perplexities around 550 and growing super-linearly to more than 100,000 within a few epochs.

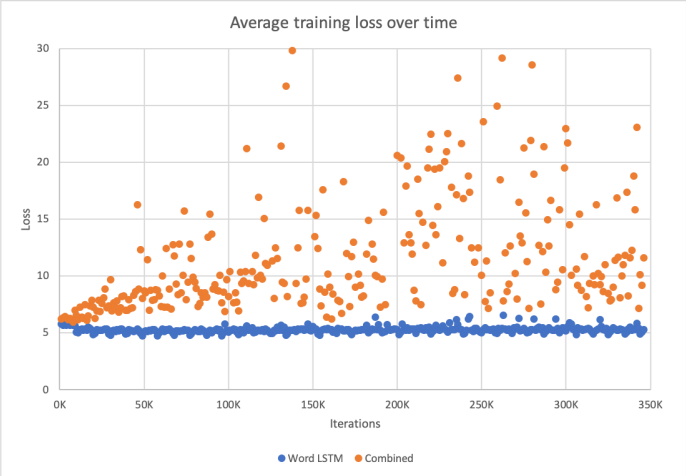

Figure 1: Training loss over time

The word-exclusive model exhibited similar behavior when exercised on the full, 5,000,000 example training set, failing to improve training accuracy yet steadily performing worse on the validation set.

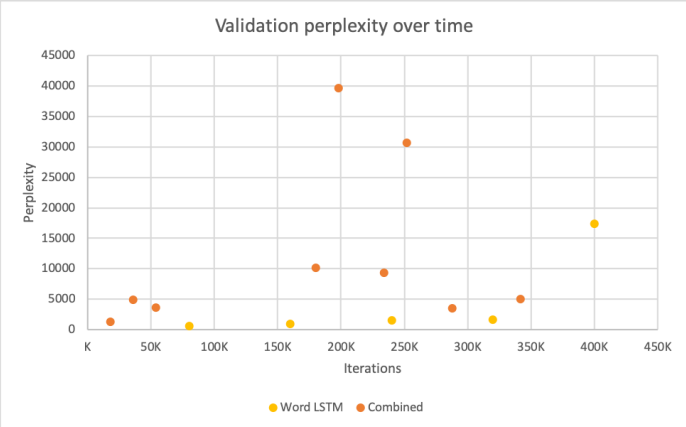

Figure 2: Validation perplexity over time

The combined model exhibited worse performance, consistently failing to match the word-exclusive model. Even on toy data sets, it proved unable to fit to perfect accuracy. Adjusting the model in one of the following ways allowed it to reach such accuracy:

- Skipping the linear layer mapping the LSTM output from $\mathbb{R}^{2e_{word}}$ to $\mathbb{R}^{e_{word}}$ and instead concatenating it with the character encoding directly
- Removing the highway layer before making predictions

The amended model required approximately ten times as many epochs as the word-exclusive model to achieve the same results. Extensive investigation of the highway layer and linear layer revealed no perceptible bugs, and given these components' presence in the word-exclusive model and multiple CS224N assignments (both in terms of design and code), they do not seem broken. On the test set the word-exclusive model and the combined model scored perplexities of 258.60 and 2468.71 respectively.

Test set perplexity	
Word LSTM	258.60
Combined	2468.71

The combined model also proved susceptible to wild swings in accuracy–one batch could produce a loss around 10.00 or 12.00 only to be followed by a loss of 20.00 or 30.00. Note that given the high loss, the perplexity for some batches achieved numerical errors, so I omit those spikes in the graphs.

4.5 Analysis

The models' performance raises multiple questions. Regarding the word-exclusive model, the degradation in learning capacity at 500,000 examples dropped off precipitously as opposed to in a gradual manner, suggesting an issue with the data or the model itself. It also seems strange that the training performance did not continue to decline when increasing the number of training examples from 500,000 to 5,000,000.

While the model failed to improve on the training set over time, it's accuracy on the validation set declined. A drop in validation accuracy often suggests overfitting, yet the absence of high scores on the training set remains difficult to explain. It seems the gradient updates failed to change accuracy on the training set while simultaneously damaging the model's ability to generalize.

As for the combined model, its failure to perform suggests the incorporation of the character-encoding might have interfered with the word-exclusive model. Projecting the two hidden states from the word LSTM into a single vector in before concatenating them with the character encoding appears to have hampered the model's predictive ability. Note that the resulting vector contained 256 entries from the character-encoding yet only 200 (not 400) from the word-encoding, which might explain the apparent loss of information.

The improvements when removing highway layer present a more challenging question, namely why a layer with an optional residual connection would impede learning. As mentioned earlier, extensive review of the code in question has revealed no errors, which corroborates the same component's ostensibly issue-free appearance in the word-exclusive model and in other CS224N assignments. While it remains possible that the data prior to the projection offered more value than the data following it, I do not have a compelling answer and leave this question open.

5 Conclusions and future work

The use of character-level information in this experiment failed to improve the performance of the word-based character model. The technique used in the combined model seems to have clashed with the word LSTM, suggesting a more strategic combination of word and character representations might be preferable to the methods used in this paper.

As for future plans, I would like to further investigate the models' difficulty learning on corpora larger than 500,000 words. While such a striking phenomenon might stem from issues with implementation or design, it might also have to do with the data sets themselves. As mentioned in the analysis section, the assumption of identically and independently distributed opinions might not hold across two-and-a-half centuries of Supreme Court justices, writing styles, and spellings.

In addition to refining the two existing models, I would like to apply a character-exclusive approach to this task by designing a system to predict the following character instead of the following word. Due to time and budget constraints, I did not include such an approach in this project. I believe a character model could offer improved flexibility and results as compared to the systems tested in this paper.

References

[1] Weiran Xu Dongyun Liang and Yinge Zhao. Combining word-level and character-level representations for relation classification of informal text. In *Proceedings of the 2nd Workshop on Representation Learning for NLP (August 2017)*, 2017.

[2] Garrett Fiddler. Scotus opinions: Full text and metadata of all opinions written by scotus justices through 2017.

[3] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2741–2749. AAAI Press, 2016.

[4] Minh-Thang Luong and Christopher D. Manning. Achieving open vocabulary neural machine translation with hybrid word-character models. *CoRR*, abs/1604.00788, 2016.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.