Transfer Learning in Stacked LSTM's for Text Generation

Alexander Fischer
University of Massachusetts, Amherst
afischer@umass.edu

Abstract

Recurrent neural networks, like all machine learning models, require a large amount of training data in order to be effective. One way to mitigate this problem is with transfer learning. We propose performing transfer learning on stacked LSTM's using text data. We demonstrate the effectiveness of this technique by training a stacked LSTM on a large corpus of text data (we used books from project Gutenberg), then transferring the network weights to be trained on a very small corpus of text data(we used song lyrics from the Beatles). Our network learned from the transferred network outperformed the best LSTM we could train on only the small text corpus, establishing the viability of transfer learning in this fashion.

1. Introduction

Consider the task of training a recurrent neural network to generate text of a certain type, *e.g*. text matching the style of a particular author. This task requires a large amount of training data, so it is generally infeasible unless there exists a large corpus of such text. For example, one can train an RNN to output text similar to, say, Shakespeare's writings, as Shakespeare was a prolific writer and there exists a large corpus of his work, but one cannot train an effective RNN to output text similar to, say, lyrics of a specific musician, as one musician generally cannot write enough songs to properly train an RNN.

Humans have no problem understanding the style of some text given only a small sample of said text. Why can they learn so much faster than RNN's? Because RNN's have to learn the entire structure of the English language from their training data, whereas humans know the structure of the English language before thinking about any textual styles. Humans *transfer* their knowledge of the low-level structure of English—the spelling, grammar, et cetera—to the task of recognizing and generating text of a certain style, whereas RNN's do not.

We can improve the performance of RNN's on small training datasets by first training them on larger, more general datasets—say, the large amounts of English text from any source—then train the resulting RNN to mimic a specific source of text for which there exists little training data by only training a subset of the RNN's weights on the more specific data. This will allow the RNN to have a high enough model capacity to understand and generate text that follows the complex rules of the English language, while preventing it from overfitting the small amount of more specific training data.

This is analogous to transfer learning tasks in computer vision. It is common to the high-level features of a high-capacity neural network trained on a large dataset such as ImageNet as inputs to a small neural network trained on a smaller dataset [6] [7]. However, this is more more complicated with recurrent neural networks because it is not immediately obvious which activations correspond to lower-level features which will be invariant across different text sources (such as spelling, grammar, et cetera) and which activations correspond to higher-level features (such as word choice, subject matter, and tone), unlike with CNN's for computer vision in which the later layers correspond to higher-level features.

Previous work [5] has adapted a pre-trained speech recognition RNN to work with a specific speaker by training a small subset of the RNN on the speaker's voice; however, to the best of our knowledge, this technique has not been attempted for text generation. Such work designed the RNN to have a specific subset of weights that is speaker dependent and thus will be trained post-tranfser.

1.1. Stacked LSTM's

We will use stacked long short term memory (LSTM) recurrent neural networks [2] [1] for our text-generation networks. Stacked LSTM's consist of several distinct LSTM layers composed with each other. At every timestep, the input data is fed into the first LSTM, then the output of that LSTM acts as the input of the next input LSTM, and more generally the output of each LSTM acts as the input of each successive LSTM. The output of the final LSTM is the output of the entire network at any given timestep.

We will perform transfer learning on stacked LSTM's by

training the whole network on a large dataset, then training a subset of the layers on a smaller dataset. To my knowledge, this training methodology has never been used with LSTM's.

2. Problem Statement

We have a small text dataset, and we want to train an LSTM to generate text based on that dataset; however, the size of the dataset limits the capacity of a model we can train before overfitting. Instead, we pre-train a higher-capacity stacked LSTM on a large text dataset that is similar to our goal text, then train a subset of the LSTM's weights on our goal text.

2.1. Datasets

We used three data sources for this project. The first, primary dataset is all song lyrics from the Beatles. This dataset is small compared to other datasets used to train recurrent neural networks. Combined, it contains 181 songs for a total of 146,000 characters. Since we used one third of the data each for training, validation, and test data, the training data for this dataset was tiny compared to most datasets used to train neural networks. This is the dataset on which we will focus our attempts to build an accurate network for text generation.

We used two secondary datasets that we used to pre-train networks which we then transferred to train on our primary dataset. The first secondary dataset was a collection [3] of lyrics from 55,000 songs from Kaggle. Its much larger size allowed us to train a much higher capacity network on it. This text is quite similar to Beatles lyrics, so we expected transfer learning to work quite well between this dataset and the Beatles dataset.

The second secondary dataset was a collection of books from Project Gutenberg [4], an organization that focuses on digitizing works of literature in the public domain. Because this dataset is so massive, we sampled several books from the dataset and pre-trained a network on those. Like the Kaggle dataset, the size of this dataset allowed us to train a much higher capacity network on it. This dataset is only somewhat similar to Beatles lyrics, as the public-domain books are from several decades prior and feature very different text style, so we were unsure if transferring a network trained on this dataset to the Beatles task would work.

We wish to train an RNN to generate text that mimics some pre-existing text for which we have only a small corpus of, too small to fully train an RNN. We must train an RNN to generate English text using a large corpus of pre-existing English text, then further train the resulting RNN on the specific text to generate text in the desired style.

The principal challenge is to determine how to train a pre-trained RNN to mimic a certain textual style. There is no obvious notion of which activation in an RNN correspond to high-level features associated with textual style, and we will have to determine that in order to effectively train a limited subset of the RNN.

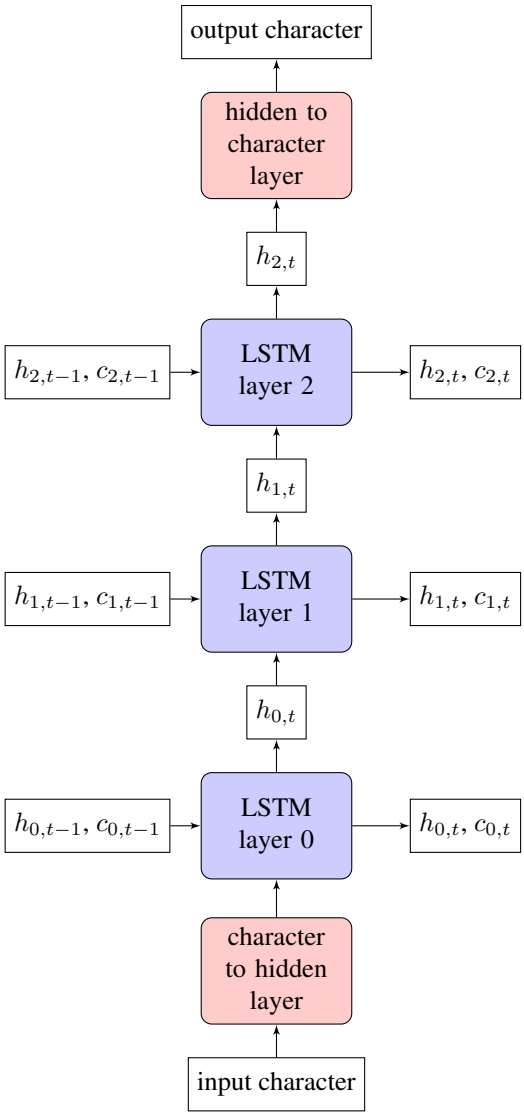

Figure 1. Our 3-layer stacked LSTM architecture. $h_{i,t}$ and $c_{i,t}$ refer to the hidden state and memory state, respectively, of level i of the stack at timestep t. Blue boxes represent individual LSTM layers, and red boxes represent fully connected linear layers.

3. Technical Approach

TODO

To train the pre-trained network on the smaller, more specific text, we will train a subset of the weights of the pre-trained LSTM that produce activations that correspond to higher-level features of text. We will have to do some investigation in order to determine which weights to train. This

investigation will take the form of feeding the pre-trained LSTM text with various low-level and high-level features modified in specific ways and seeing which memory cells are correlated with various statistical features of the input text.

For example, we will feed the pre-trained LSTM text with a variety of sentiments, from happy to sad, and see which memory units and hidden-layer activations have the highest correlation with the sentiment of the input text. These memory units and hidden layer activations should correspond to high-level features, meaning the weights that affect them should be trained on the smaller, more specific corpus.

We will also feed the pre-trained LSTM text, then subsequently that same text but with low-level features of the text changed, such as spelling errors and slightly different wording. We will see which memory units and hidden layer activations change in response to these changes, and which ones do not change. These memory units and hidden layer activations should correspond to low-level features, meaning the weights that affect them should be held constant after pre-training.

Ideally, these procedures should yield an intuitive heuristic as to which weights affect high-level and low-level features of text, rather than a hacked-together list of weights which seem to do what we want. For example, the later layers in convolution neural networks for computer vision correspond to higher-level features; we would like to discover a similar guiding principle for LSTM's. Such a guiding principle would allow us to train a much more principled, complete set of weights on our smaller, more specific corpus. It would also allow easier generalization of our results to different RNN architectures and to problems beyond English text, rather than having to repeat the aforementioned weight investigation procedures every time we wish to try a different pre-trained network.

To evaluate this approach, we will train a pre-trained LSTM on smaller and smaller text datasets and see how small the dataset can be before the LSTM overfits it. This should be much smaller than the smallest datasets that an LSTM can be trained on from scratch without overfitting.

We will use both objective and subjective metrics to measure the performance of our LSTM's. We will use the standard test-set accuracy of our LSTM's trained from pre-trained complex LSTM's and compare them to the test-set accuracy of LSTM's trained from scratch on our training data. Also, for a subjective performance metric, we will subjectively compare the text generated by our LSTM's and trained-from-scratch LSTM's. It should be obvious upon reading both which network better learns the structure of the small training data by seeing which generated text seems to be a better generalization of the training data.

Layers retrained	**Minimum validation loss**	**Test loss on minimum validation loss network**
1, 2	0.962	1.100
0, 2	0.954	1.090
0, 1	0.964	1.101
0	1.001	1.133
1	0.983	1.123
2	0.979	1.116

Table 1. Results for transferring a 3-layer stacked LSTM network trained on Project Gutenberg books to song lyrics from The Beatles. We used cross-entropy as our loss function. The best network we could train on only the Beatles dataset resulted in a cross-entropy loss of 1.04.

Layers retrained	**Minimum validation loss**	**Test loss on minimum validation loss network**
1, 2	0.817	0.949
0, 2	0.816	0.950
0, 1	0.818	0.946
0	0.816	0.949
1	0.816	0.950
2	0.813	0.946

Table 2. Results for transferring a 3-layer stacked LSTM network trained on general song lyrics to song lyrics from The Beatles. We used cross-entropy as our loss function. The best network we could train on only the Beatles dataset resulted in a cross-entropy loss of 1.04.

4. Experimental Results

As expected, networks that were learned starting from transferred weights trained on larger datasets performed better on the small Beatles dataset. The results of our experiments can be see in tables 1 and 2.

5. Further Work

6. Conclusion

We demonstrated the effectiveness of a novel approach to transfer learning with LSTM's for text generation. We used this technique to train a network that predicted and generated text on a small dataset (song lyrics from The Beatles) better than we could do by training a network only on said small dataset.

References

[1] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[2] S. Hochreiter, Killough, and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[3] S. Kuznetsov. 55000+ song lyrics. `https://www.kaggle.com/mousehead/songlyrics/home`.

[4] S. Lahiri. Complexity of Word Collocation Networks: A Preliminary Structural Analysis. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 96–105, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.

[5] Y. Miao and F. Metze. On speaker adaptation of long short-term memory recurrent neural networks. In *Interspeech*, 2015.

[6] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. In *CVPR*, 2014.

[7] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014.