# Semi-supervised Sequence Learning

**Andrew M. Dai**
Google Inc.
adai@google.com

**Quoc V. Le**
Google Inc.
qvl@google.com

## Abstract

We present two approaches that use unlabeled data to improve sequence learning with recurrent networks. The first approach is to predict what comes next in a sequence, which is a conventional language model in natural language processing. The second approach is to use a sequence autoencoder, which reads the input sequence into a vector and predicts the input sequence again. These two algorithms can be used as a "pretraining" step for a later supervised sequence learning algorithm. In other words, the parameters obtained from the unsupervised step can be used as a starting point for other supervised training models. In our experiments, we find that long short term memory recurrent networks after being pretrained with the two approaches are more stable and generalize better. With pretraining, we are able to train long short term memory recurrent networks up to a few hundred timesteps, thereby achieving strong performance in many text classification tasks, such as IMDB, DBpedia and 20 Newsgroups.

## 1 Introduction

Recurrent neural networks (RNNs) are powerful tools for modeling sequential data, yet training them by back-propagation through time [36, 26] can be difficult [8]. For that reason, RNNs have rarely been used for natural language processing tasks such as text classification despite their powers in representing sequential structures.

On a number of document classification tasks, we find that it is possible to train Long Short-Term Memory recurrent networks (LSTM RNNs) [9] to achieve good performance with careful tuning of hyperparameters. Furthermore, a simple pretraining step can significantly stabilize the training of LSTMs. For example, we can use a next step prediction model, i.e., a *recurrent language model* in NLP, as an unsupervised method. Another method is to use a *sequence autoencoder,* which uses a RNN to read a long input sequence into a single vector. This vector will then be used to reconstruct the original sequence. The weights obtained from these two pretraining methods can then be used as an initialization for standard LSTM RNNs to improve training and generalization.

In our experiments on document classification with 20 Newsgroups [16] and DBpedia [19], and sentiment analysis with IMDB [21] and Rotten Tomatoes [25], LSTMs pretrained by recurrent language models or sequence autoencoders are usually better than LSTMs initialized randomly. This pretraining helps LSTMs reach or surpass previous baselines on these datasets without additional data.

Another important result from our experiments is that using more unlabeled data from related tasks in the pretraining can improve the generalization of a subsequent supervised model. For example, using unlabeled data from Amazon reviews to pretrain the sequence autoencoders can improve classification accuracy on Rotten Tomatoes from 79.7% to 83.3%, an equivalence of adding substantially more labeled data. This evidence supports the thesis that it is possible to use unsupervised learning with more unlabeled data to improve supervised learning. With sequence autoencoders, and outside unlabeled data, LSTMs are able to match or surpass previously reported results.

We believe our semi-supervised approach (as also argued by [1]) has some advantages over other unsupervised sequence learning methods, e.g., Paragraph Vectors [18], because it can allow for easy fine-tuning. Our semi-supervised learning approach is related to Skip-Thought vectors [13], with two differences. The first difference is that Skip-Thought is a harder objective, because it predicts adjacent sentences. The second is that Skip-Thought is a pure unsupervised learning algorithm, without fine-tuning.

## 2 Sequence autoencoders and recurrent language models

Our approach to sequence autoencoding is inspired by the work in sequence to sequence learning (also known as *seq2seq*) by Sutskever et al. [31], which has been successfully used for machine translation [20, 10], text parsing [32], image captioning [34], video analysis [30], speech recognition [4, 3] and conversational modeling [27, 33]. Key to their approach is the use of a recurrent network as an encoder to read in an input sequence into a hidden state, which is the input to a decoder recurrent network that predicts the output sequence.

The sequence autoencoder is similar to the above concept, except that it is an unsupervised learning model. The objective is to reconstruct the input sequence itself. That means we replace the output sequence in the *seq2seq* framework with the input sequence. In our sequence autoencoders, the weights for the decoder network and the encoder network are the same (see Figure 1).
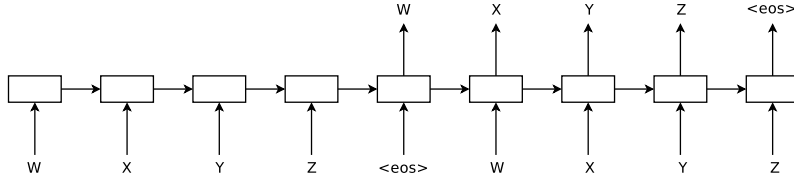


Figure 1: The sequence autoencoder for the sequence "WXYZ". The sequence autoencoder uses a recurrent network to read the input sequence in to the hidden state, which can then be used to reconstruct the original sequence.

We find that the weights obtained from the sequence autoencoder can be used as an initialization of another supervised network, one which tries to classify the sequence. We hypothesize that this is because the network can already memorize the input sequence. This reason, and the fact that the gradients have shortcuts, are our hypothesis of why the sequence autoencoder is a good and stable approach in initializing recurrent networks.

A significant property of the sequence autoencoder is that it is unsupervised, and thus can be trained with large quantities of unlabeled data to improve its quality. Our result is that additional unlabeled data can improve the generalization ability of recurrent networks. This is especially useful for tasks that have limited labeled data.

We also find that recurrent language models [23] can be used as a pretraining method for LSTMs. This is equivalent to removing the encoder part of the sequence autoencoder in Figure 1. Our experimental results show that this approach works better than LSTMs with random initialization.

## 3 Overview of methods

In our experiments, we use LSTM recurrent networks [9] because they are generally better than RNNs. Our LSTM implementation is standard and has input, forget, and output gates [6, 7]. We compare basic LSTMs against LSTMs initialized with the sequence autoencoder method. When LSTMs are initialized with a sequence autoencoder, the methods are called SA-LSTMs in our experiments. When LSTMs are initialized with a language model, the method is called LM-LSTMs.

In most of our experiments our output layer predicts the document label from the LSTM output at the last timestep. We also experiment with the approach of putting the label at every timestep and linearly increasing the weights of the prediction objectives from 0 to 1 [24]. This way we can inject gradients to earlier steps in the recurrent networks. We call this approach *linear label gain*.

Lastly, we also experiment with the method of jointly training the supervised learning task with the sequence autoencoder and call this method *joint training*.

## 4 Experiments

In our experiments with LSTMs, we follow the basic recipes as described in [31] by clipping the cell outputs and gradients. The benchmarks of focus are text understanding tasks, with all datasets being publicly available. The tasks are sentiment analysis (IMDB and Rotten Tomatoes) and text classification (20 Newsgroups and DBpedia). Commonly used methods on these datasets, such as bag-of-words or n-grams, typically ignore long-range ordering information (e.g., modifiers and their objects may be separated by many unrelated words); so one would expect recurrent methods which preserve ordering information to perform well. Nevertheless, due to the difficulty in optimizing these networks, recurrent models are not the method of choice for document classification.

In our experiments with the sequence autoencoder, we train it to reproduce the full document after reading all the input words. In other words, we do not perform any truncation or windowing. We add an end of sentence marker to the end of each input sequence and train the network to start reproducing the sequence after that marker. To speed up performance and reduce GPU memory usage, we perform truncated backpropagation up to 400 timesteps from the end of the sequence. We preprocess the text so that punctuation is treated as separate tokens and we ignore any non-English characters and words in the DBpedia text. We also remove words that only appear once in each dataset and do not perform any term weighting or stemming.

After training the recurrent language model or the sequence autoencoder for roughly 500K steps with a batch size of 128, we use both the word embedding parameters and the LSTM weights to initialize the LSTM for the supervised task. We then train on that task while fine tuning both the embedding parameters and the weights and use early stopping when the validation error starts to increase. We choose the dropout parameters based on a validation set.

Using SA-LSTMs, we are able to match or surpass reported results for all datasets. It is important to emphasize that previous best results come from various different methods. So it is significant that one method achieves strong results for all datasets, presumably because such a method can be used as a general model for any similar task. A summary of results in the experiments are shown in Table 1. More details of the experiments are as follows.

Table 1: A summary of the error rates of SA-LSTMs and previous best reported results.

| Dataset | SA-LSTM | Previous best result |
|---|---|---|
| IMDB | 7.24% | 7.42% |
| Rotten Tomatoes | 16.7% | 18.5% |
| 20 Newsgroups | 15.6% | 17.1% |
| DBpedia | 1.19% | 1.74% |

### 4.1 Sentiment analysis experiments with IMDB

In this first set of experiments, we benchmark our methods on the IMDB movie sentiment dataset, proposed by Maas et al. [21].[1] There are 25,000 labeled and 50,000 unlabeled documents in the training set and 25,000 in the test set. We use 15% of the labeled training documents as a validation set. The average length of each document is 241 words and the maximum length of a document is 2,526 words. The previous baselines are bag-of-words, ConvNets [12] or Paragraph Vectors [18].

Since the documents are long, one might expect that it is difficult for recurrent networks to learn. We however find that with tuning, it is possible to train LSTM recurrent networks to fit the training set. For example, if we set the size of hidden state to be 512 units and truncate the backprop to be 400, an LSTM can do fairly well. With random embedding dimension dropout [37] and random word dropout (not published previously), we are able to reach performance of around 86.5% accuracy in the test set, which is approximately 5% worse than most baselines.

---

[1] http://ai.Stanford.edu/amaas/data/sentiment/index.html

Fundamentally, the main problem with this approach is that it is unstable: if we were to increase the number of hidden units or to increase the number of backprop steps, the training breaks down very quickly: the objective function explodes even with careful tuning of the gradient clipping. This is because LSTMs are sensitive to the hyperparameters for long documents. In contrast, we find that the SA-LSTM works better and is more stable. If we use the sequence autoencoders, changing the size of the hidden state or the number of backprop steps hardly affects the training of LSTMs. This is important because the models become more practical to train.

Using sequence autoencoders, we overcome the optimization instability in LSTMs in such a way that it is fast and easy to achieve perfect classification on the training set. To avoid overfitting, we again use input dimension dropout, with the dropout rate chosen on a validation set. We find that dropping out 80% of the input embedding dimensions works well for this dataset. The results of our experiments are shown in Table 2 together with previous baselines. We also add an additional baseline where we initialize a LSTM with word2vec embeddings on the training set.

Table 2: Performance of models on the IMDB sentiment classification task.

| Model | Test error rate |
| --- | --- |
| LSTM with tuning and dropout | 13.50% |
| LSTM initialized with word2vec embeddings | 10.00% |
| LM-LSTM (see Section 2) | 7.64% |
| SA-LSTM (see Figure 1) | 7.24% |
| SA-LSTM with linear gain (see Section 3) | 9.17% |
| SA-LSTM with joint training (see Section 3) | 14.70% |
| Full+Unlabeled+BoW [21] | 11.11% |
| WRRBM + BoW (bnc) [21] | 10.77% |
| NBSVM-bi (Naïve Bayes SVM with bigrams) [35] | 8.78% |
| seq2-bow$n$-CNN (ConvNet with dynamic pooling) [11] | 7.67% |
| Paragraph Vectors [18] | 7.42% |

The results confirm that SA-LSTM with input embedding dropout can be as good as previous best results on this dataset. In contrast, LSTMs without sequence autoencoders have trouble in optimizing the objective because of long range dependencies in the documents.

Using language modeling (LM-LSTM) as an initialization works well, achieving 8.98%, but less well compared to the SA-LSTM. This is perhaps because language modeling is a short-term objective, so that the hidden state only captures the ability to predict the next few words.

In the above table, we use 1,024 units for memory cells, 512 units for the input embedding layer in the LM-LSTM and SA-LSTM. We also use a hidden layer 512 units with dropout of 50% between the last hidden state and the classifier. We continue to use these settings in the following experiments except with 30 units in the final hidden layer.

In Table 3, we present some examples from the IMDB dataset that are correctly classified by SA-LSTM but not by a bigram NBSVM model. These examples often have long-term dependencies or have sarcasm that is difficult to find by solely looking at short phrases.

## 4.2 Sentiment analysis experiments with Rotten Tomatoes and the positive effects of additional unlabeled data

The success on the IMDB dataset convinces us to test our methods on another sentiment analysis task to see if similar gains can be obtained. The benchmark of focus in this experiment is the Rotten Tomatoes dataset [25].[2] The dataset has 10,662 documents, which are randomly split into 80% for training, 10% for validation and 10% for test. The average length of each document is 22 words and the maximum length is 52 words. Thus compared to IMDB, this dataset is smaller both in terms of the number of documents and the number of words per document.

---

[2]http://www.cs.cornell.edu/people/pabo/movie-review-data/

Table 3: IMDB sentiment classification examples that are correctly classified by SA-LSTM and incorrectly by NBSVM-bi.

| Text | Sentiment |
|---|---|
| This film is not at all as bad as some people on here are saying. I think it has got a decent horror plot and the acting seem normal to me. People are way over-exagerating what was wrong with this. It is simply classic horror, the type without a plot that we have to think about forever and forever. We can just sit back, relax, and be scared. | Positive |
| Looking for a REAL super bad movie? If you wanna have great fun, don't hesitate and check this one! Ferrigno is incredibly bad but is also the best of this mediocrity. | Negative |
| A professional production with quality actors that simply never touched the heart or the funny bone no matter how hard it tried. The quality cast, stark setting and excellent cinemetography made you hope for Fargo or High Plains Drifter but sorry, the soup had no seasoning...or meat for that matter. A 3 (of 10) for effort. | Negative |
| The screen-play is very bad, but there are some action sequences that i really liked. I think the image is good, better than other romanian movies. I liked also how the actors did their jobs. | Negative |

Our first observation is that it is easier to train LSTMs on this dataset than on the IMDB dataset and the gaps between LSTMs, LM-LSTMs and SA-LSTMs are smaller than before. This is because movie reviews in Rotten Tomatoes are sentences whereas reviews in IMDB are paragraphs.

As this dataset is small, our methods tend to severely overfit the training set. Combining SA-LSTMs with 95% input embedding and 50% word dropout improves generalization and allows the model to achieve 20.3% test set error. Further tuning the hyperparameters on the validation set yields 19.3% test error.

To better the performance, we add unlabeled data from the IMDB dataset in the previous experiment and Amazon movie reviews [22] to the autoencoder training stage.[3] We also run a control experiment where we use the pretrained word vectors trained by word2vec from Google News.

Table 4: Performance of models on the Rotten Tomatoes sentiment classification task.

| Model | Test error rate |
|---|---|
| LSTM with tuning and dropout | 20.3% |
| LSTM with linear gain | 21.1% |
| LM-LSTM | 21.7% |
| SA-LSTM | 20.3% |
| LSTM with word vectors from word2vec Google News | 20.5% |
| SA-LSTM with unlabeled data from IMDB | 18.6% |
| SA-LSTM with unlabeled data from Amazon reviews | 16.7% |
| MV-RNN [28] | 21.0% |
| NBSVM-bi [35] | 20.6% |
| CNN-rand [12] | 23.5% |
| CNN-non-static (ConvNet with vectors from word2vec Google News) [12] | 18.5% |

The results for this set of experiments are shown in Table 4. Our observation is that if we use the word vectors from word2vec, there is only a small gain of 0.5%. This is perhaps because the recurrent weights play an important role in our model and are not initialized properly in this experiment. However, if we use IMDB to pretrain the sequence autoencoders, the error decreases from 20.5% to 18.6%, nearly a 2% gain in accuracy; if we use Amazon reviews, a larger unlabeled dataset (7.9

---

[3]The dataset is available at http://snap.stanford.edu/data/web-Amazon.html, which has 34 million general product reviews, but we only use 7.9 million movie reviews in our experiments.

million movie reviews), to pretrain the sequence autoencoders, the error goes down to 16.7% which is another 2% gain in accuracy.

This brings us to the question of how well this method of using unlabeled data fares compared to adding more labeled data. As argued by Socher et al. [29], a reason of why the methods are not perfect yet is the lack of labeled training data, they proposed to use more labeled data by labeling an addition of 215,154 phrases created by the Stanford Parser. The use of more labeled data allowed their method to achieve around 15% error in the test set, an improvement of approximately 5% over older methods with less labeled data.

We compare our method to their reported results [29] on sentence-level classification. As our method does not have access to valuable labeled data, one might expect that our method is severely disadvantaged and should not perform on the same level. However, with unlabeled data and sequence autoencoders, we are able to obtain 16.7%, ranking second amongst many other methods that have access to a much larger corpus of labeled data. The fact that unlabeled data can compensate for the lack of labeled data is very significant as unlabeled data are much cheaper than labeled data. The results are shown in Table 5.

Table 5: More unlabeled data vs. more labeled data. Performance of SA-LSTM with additional unlabeled data and previous models with additional labeled data on the Rotten Tomatoes task.

| Model | Test error rate |
|---|---|
| LSTM initialized with word2vec embeddings trained on Amazon reviews | 23.3% |
| SA-LSTM with unlabeled data from Amazon reviews | 16.7% |
| NB [29] | 18.2% |
| SVM [29] | 20.6% |
| BiNB [29] | 16.9% |
| VecAvg [29] | 19.9% |
| RNN [29] | 17.6% |
| MV-RNN [29] | 17.1% |
| RNTN [29] | 14.6% |

### 4.3 Text classification experiments with 20 newsgroups

The experiments so far have been done on datasets where the number of tokens in a document is relatively small, a few hundred words. Our question becomes whether it is possible to use SA-LSTMs for tasks that have a substantial number of words. For that purpose, we carry out the next experiments on the 20 newsgroups dataset [16].[4] There are 11,293 documents in the training set and 7,528 in the test set. We use 15% of the training documents as a validation set. Each document is an email with an average length of 267 words and a maximum length of 11,925 words. Attachments, PGP keys, duplicates and empty messages are removed. As the newsgroup documents are long, it was previously considered improbable for recurrent networks to learn anything from the dataset. The best methods are often simple bag-of-words.

We repeat the same experiments with LSTMs and SA-LSTMs on this dataset. Similar to observations made in previous experiments, SA-LSTMs are generally more stable to train than LSTMs. To improve generalization of the models, we again use input embedding dropout and word dropout chosen on the validation set. With 70% input embedding dropout and 75% word dropout, SA-LSTM achieves 15.6% test set error which is much better than previous classifiers in this dataset. Results are shown in Table 6.

### 4.4 Character-level document classification experiments with DBpedia

In this set of experiments, we turn our attention to another challenging task of categorizing Wikipedia pages by reading character-by-character inputs. The dataset of attention is the DBpedia

---

[4]http://qwone.com/~jason/20Newsgroups/

Table 6: Performance of models on the 20 newsgroups classification task.

| Model | Test error rate |
|---|---|
| LSTM | 18.0% |
| LSTM with linear gain | 71.6% |
| LM-LSTM | 15.3% |
| SA-LSTM | 15.6% |
| Hybrid Class RBM [17] | 23.8% |
| RBM-MLP [5] | 20.5% |
| SVM + Bag-of-words [2] | 17.1% |
| Naïve Bayes [2] | 19.0% |

dataset [19], which was also used to benchmark convolutional neural nets in Zhang and LeCun [38]. DBpedia had no duplication or tainting issues from the outset so we compare experimental results on this dataset. DBpedia is a crowd-sourced effort to extract information from Wikipedia and categorize it into an ontology.

For this experiment, we follow the same procedure suggested in Zhang and LeCun [38]. The task is to classify DBpedia abstracts into one of 14 categories after reading the character-by-character input. The dataset is split into 560,000 training examples and 70,000 test examples. A DBpedia document has an average of 300 characters while the maximum length of all documents is 13,467 characters. As this dataset is large, overfitting is not an issue and thus we do not perform any dropout on the input or recurrent layers. For this dataset, we use a two-layered LSTM, each layer has 512 hidden units and and the input embedding has 128 units.

Table 7: Performance of models on the DBpedia character level classification task.

| Model | Test error rate |
|---|---|
| LSTM | 13.64% |
| LSTM with linear gain | 1.32% |
| LM-LSTM | 1.50% |
| SA-LSTM | 2.34% |
| SA-LSTM with linear gain | 1.23% |
| SA-LSTM with 3 layers and linear gain | 1.19% |
| SA-LSTM (word-level) | 1.40% |
| Bag-of-words | 3.57% |
| Small ConvNet | 1.98% |
| Large ConvNet | 1.73% |

In this dataset, we find that the linear label gain as described in Section 3 is an effective mechanism to inject gradients to earlier steps in LSTMs. This linear gain method works well and achieves 1.32% test set error, which is better than SA-LSTM. Combining SA-LSTM and the linear gain method achieves 1.19% test set error, a significant improvement from the results of convolutional networks as shown in Table 7.

### 4.5 Object classification experiments with CIFAR-10

In these experiments, we attempt to see if our pre-training methods extend to non-textual data. To do this, we train a LSTM on the CIFAR-10 image dataset, consisting of 60,000 32x32 colour images divided into 10 classes. The input at each timestep of the LSTM is an entire row of pixels and we predict the class of the image after reading the final row. We use the same method as in [15] to perform data augmentation. We also trained a LSTM to do next row prediction given the current row (we denote this as LM-LSTM) and a LSTM to autoencode the image by rows (SA-LSTM). The loss function during unsupervised learning is the Euclidean L2 distance between the predicted and

the target row. We then fine-tune these on the classification task and present the classification results in Table 8. While we do not achieve the results attained by state of the art convolutional networks, our 2-layer pretrained LM-LSTM is able to exceed the results of the baseline convolutional DBN model [14] despite not using any convolutions and outperforms the non pre-trained LSTM.

Table 8: Performance on the CIFAR-10 object classification task with 50,000 train images.

| Model (400 units) | Test error rate |
|---|---|
| 1-layer LSTM | 25.0% |
| 1-layer LM-LSTM | 23.1% |
| 1-layer SA-LSTM | 25.1% |
| 2-layer LSTM | 26.0% |
| 2-layer LM-LSTM | 18.0% |
| 2-layer SA-LSTM | 25.1% |
| Convolution DBNs [14] | 21.1% |

## 5   Discussion

In this paper, we showed that it is possible to use LSTM recurrent networks for NLP tasks such as document classification. Further, we demonstrated that a language model or a sequence autoencoder can help stabilize the learning in LSTM recurrent networks. On five benchmarks that we tried, LSTMs can reach or surpass the performance levels of all previous baselines.

## References

[1] R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *J. Mach. Learn. Res.*, 6:1817–1853, December 2005.

[2] A. Cardoso-Cachopo. Datasets for single-label text categorization. `http://web.ist.utl.pt/acardoso/datasets/`, 2015. [Online; accessed 25-May-2015].

[3] William Chan, Navdeep Jaitly, Quoc V Le, and Oriol Vinyals. Listen, attend and spell. *arXiv preprint arXiv:1508.01211*, 2015.

[4] J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio. End-to-end continuous speech recognition using attention-based recurrent nn: First results. *arXiv preprint arXiv:1412.1602*, 2014.

[5] Y. Dauphin and Y. Bengio. Stochastic ratio matching of RBMs for sparse high-dimensional inputs. In *NIPS*, 2013.

[6] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 2000.

[7] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. In *ICML*, 2015.

[8] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*, 2001.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[10] S. Jean, K. Cho, R. Memisevic, and Y. Bengio. On using very large target vocabulary for neural machine translation. In *ICML*, 2014.

[11] R. Johnson and T. Zhang. Effective use of word order for text categorization with convolutional neural networks. In *NAACL*, 2014.

[12] Y. Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[13] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler. Skip-thought vectors. In *NIPS*, 2015.

[14] A. Krizhevsky. Convolutional deep belief networks on CIFAR-10. Technical report, University of Toronto, 2010.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[16] K. Lang. Newsweeder: Learning to filter netnews. In *ICML*, 1995.

[17] H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio. Learning algorithms for the classification restricted boltzmann machine. *JMLR*, 2012.

[18] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. In *ICML*, 2014.

[19] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, et al. DBpedia – a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 2014.

[20] T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.

[21] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *ACL*, 2011.

[22] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *RecSys*, pages 165–172. ACM, 2013.

[23] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

[24] J. Y. H. Ng, M. J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, 2015.

[25] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL*, 2005.

[26] D. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 1986.

[27] L. Shang, Z. Lu, and H. Li. Neural responding machine for short-text conversation. In *EMNLP*, 2015.

[28] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP*, 2012.

[29] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

[30] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using LSTMs. In *ICML*, 2015.

[31] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[32] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. In *NIPS*, 2015.

[33] O. Vinyals and Q. V. Le. A neural conversational model. In *ICML Deep Learning Workshop*, 2015.

[34] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *CVPR*, 2014.

[35] S. I. Wang and C. D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *ACL*, 2012.

[36] P. J. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard, 1974.

[37] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[38] X. Zhang and Y. LeCun. Character-level convolutional networks for text classification. In *NIPS*, 2015.