Global Pre-Trained Word Vectors As Universal Unsupervised Feature Space

Ilia Zaitsev and Paul Tero
St.Petersburg, Russia and Brighton, UK
{ilia,paul}@emotions.tech

Abstract

We describe a straightforward approach to training standard GRU/LSTM based text classifiers using global pre-trained word vectors as universal unsupervised features across multiple tasks. The approach shows zero-shot domain adaptation and state-of-the art results on various well-known datasets. We provide provable and reproducible empirical evidence that unsupervised pre-trained word vectors significantly improve test accuracy, are less computationally expensive, allow training deep architectures on small datasets and opens doors to the future of transfer learning.

1 Introduction

In the last decade deep learning has gained significant attention within the field of Natural Language Processing. There have been compelling results in Neural Machine Translation (Sutskever et al., 2014), Neural Language Modelling (Mikolov, 2012) and image captioning (Karpathy, 2016). However the effectiveness of deep learning algorithms in classic text classification tasks is still debatable. Much simpler and exponentially faster shallow algorithms are on par with deep learning classifiers (Joulin, 2016).

In this paper, we propose a simple approach which benefits from deep learning in standard supervised settings. We show that very important tasks like Sentiment Analysis and Topic Modelling can achieve state-of-the-art accuracy using deep learning even on relatively small datasets. We also show that that deep models trained on one dataset generalise much better than shallow models for datasets with different distributions from different domains. For example, we show how to train a sentiment analysis polarity model on a product reviews dataset that shows high accuracy on movie and business reviews.

The idea is to use static pre-trained word vectors as universal features across multiple tasks. This essentially divides the training of the RNN into two distinct steps: first unsupervisedly train the embedding layer, and then train the network. This is not necessarily a new idea, but it shows excellent results on standard classification tasks:

Datasets	**VDCNN** (Conneau et al, 2016)	**Our RNN with Static vectors**
Yelp Polarity	95.7	**96.6**
Amazon Polarity	95.7	**96.0**
Yelp Full	64.7	**66.0**
Amazon Full	63.0	**63.3**
Yahoo Answers	73.4	**74.7**

Table 1: Sentiment Analysis Benchmarks

Additionally, we have yet to see it applied to domain adaptation: the model trained on Amazon dataset and tested against the IMDB, Yelp and Amazon test sets, shows test accuracy of 93%, 94.7% and 96% respectively.

Technically similar ideas (Kim, 2014) have been circulating for a few years but unfortunately unsupervised pre-trained vectors are still viewed as nothing more than just a small trick to boost accuracy. Despite this common view our work shows that global pre-trained word vectors have far-reaching implications for the success of training natural language based deep learning architectures.

Our motivation is the following: there's no doubt that the amount of unsupervised data in natural language is much larger than in any possible supervised datasets. Therefore, by obtaining word vectors from datasets larger than any task-specific datasets, we start operating in the global feature space that is the only habitat for natural language.

Any further fine-tuning of global vectors breaks the natural vector space and creates limited artificial one that could show a bit better results on a test set but will always lead to a lower accuracy on extrinsic evaluation.

The structure of this paper is the following: we will first present a quick background of the history of word vectors and embeddings. We then describe our network architecture and a series of experiments to show that it is indeed the vectors which are making the difference. Then we discuss future directions this work could take.

2 Background

2.1 Distributed Representation

In the 1980s, researchers began to train Neural Networks to learn the distributed representation of concepts (Hinton, 1986). The words of natural language can be viewed as concepts with shared features and their representations can be learned by a neural network as real-valued vectors based on the patterns from a training activity.

2.2 Word Embeddings

By the early 2000s (Bengio et al., 2003), researchers started training Feed Forward Neural Networks Language Models to directly model probability distributions over the next word in a sequence (with no task-specific labels), and training word vectors as models parameters. After training, the vectors showed certain properties. For example, similar words like "dog" and "cat" would be relatively close to each other in vector space. A bit later, it was realised that these vectors are useful by themselves, apart from the task they were trained on (Collobert and Weston, 2008).

By 2010 Recurrent Neural Network Language Models (RNNLMs) were showing good results (Mikolov, 2012) but were difficult and time-consuming to train, and so generally used only small vocabularies. Input to these RNNLMs was usually a 1-of-N vector representing a single word such as (0,0,1,0...0,0) for "and", where N was the total number of words in the vocabulary. The vocabulary size N was typically around 20,000 words. The first layer of the RNN connected each of these 20,000 inputs to a recurrent hidden layer. If the hidden layer had H=200 nodes, then there were NxH = 20,000x200 = 4 million connecting weights. These could be viewed as 20,000 vectors of 200 dimensions, one vector per word. This layer has become known as an embedding layer as words are embedded in a vector space.

2.3 Large Scale Language Models

A series of innovations led from RNNLMs to the simpler network structure of word2vec (Mikolov et al., 2013), which can train vector representations of every word in very big vocabularies within hours. These vectors showed even more impressive properties, such as the ability to do simple vector sums such as: queen - woman + man = king.

There is another way to train such vectors, which does not involve embedding words in a vector space. This starts with an NxN matrix of word co-occurrences, applies smoothing techniques, and then reduces the number of dimensions to KxN via techniques like Singular Value Decomposition. As with RNNs, this began as a means to an end, to solve a specific task, but it was realised that the vectors themselves are valuable. Stanford's GloVe algorithm (Pennington et al., 2014) is an example of this approach. In fact it has been shown that vectors created by these two approaches perform similarly (Levy and Goldberg, 2014).

In both cases, producing really good vectors requires a lot of data, on the order of hundreds of billions of words. In this case, "really good vectors" means vectors that mathematically represent real structures in language. For example, not only should "movie" and "film" be very close to each other in vector space, but operations like *watched – watch = observed – observe* should hold. To create such good vectors requires vast preprocessed data sources, such as the Google News Corpus or the Common Crawl database. This in turn requires access to the data and the ability to process it. Therefore, it is easier to download someone else's vectors than create them yourself and many projects use the Google word2vec vectors created in 2013 or the Stanford Common Crawl vectors from 2014.

2.4 Recurrent Neural Network for Text Classification

Recurrent Neural Networks are one of the successful examples of powerful deep learning architectures. In supervised settings an RNN can be viewed as a task specific encoder of word se-

quences of arbitrary length. It takes a variable-length string of text as input and outputs a fixed-length vector, which is then fed to a softmax layer for classification.

Early versions of RNNs (Elman, 1990) were difficult to train because of the exploding gradients and learning long-term dependencies issues. However, nowadays, LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014) types of RNNs are pretty stable, patched with gradient clipping (Mikolov, 2012).

Curiously, at present RNNs are viewed as much too powerful tools for simple classification tasks (Joulin, 2016). Modern LSTM based architectures have so many trainable parameters that they simply overfit most existing datasets. In these networks, the embedding layer is usually the largest layer, and is not regularised at all in many implementations. So eventually, it leads to overfitting. There are many different strategies to regularise the embedding layer (Peng et al., 2014) including dropout (Gal and Ghahramani, 2015) but none of them are considered as standard practice.

This issue raises a question about the practical applications of RNNs and other deep learning architectures in solving standard NLP tasks like sentiment analysis and topic modelling.

3 Pre-Trained Word Vectors

It has been shown in the context of computer vision that unsupervised pre-training possibly acts as a powerful regulariser and introduces a useful prior to the training of supervised classifiers which is much better then random initialisation (Erhan et al., 2010). By analogy, we believe that unsupervised pre-training of the embedding layer acts in a similar manner.

Furthermore, we found that usually there is no point breaking unsupervised pre-trained vector spaces by supervised fine-tuning of the RNN's embedding layer. An explanation for this finding possibly comes directly from the core idea of distributed representation of words. The words of natural language are ambiguous by nature and the context (the meaning of whole phrases or sentences) must be used for word disambiguation (Walts and Pollack, 1985). In the deep learning paradigm the primary role of algorithms is to exploit the structure in the input distribution to discover good representations with task-specific higher-level learned features defined in terms of ambiguous lower-level features (Bengio, 2012). So the word vectors as available universal building blocks should stay as they are (as ambiguous lower-level features) and only higher-level layers should be trained to perform task-specific disambiguation.

4 Deep Neural Network Architecture for Text Classification

Therefore we propose an RNN architecture using pre-trained word vectors in the embedding layer, freezing that embedding layer, and then adding further small trainable GRU layers (encoder), with sigmoid or softmax to perform classification on the output.

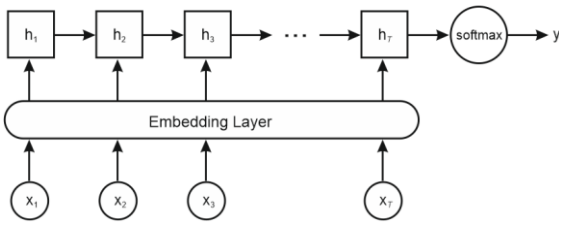

Figure 1: Deep Neural Network Architecture for Text Classification
x – input (a text sequence); h_T – last time-step (a fixed-length vector representation of variable-length input).

We take the vectors trained by word2vec or GloVe on very large corpora and plug them back into the embedding layer of a task-specific RNN, and then leave them alone. We allow the rest of the network learn the task at hand based on a set of static vectors. Those static/frozen vectors already contain a large amount of semantic structure which can be used as-is by recurrent layers in the network. So we are essentially dividing the task of training an RNN into two parts. First pre-train an unsupervised global vector space. Then train a task-specific recurrent network.

We find that this approach can achieve state-of-the-art results on classic sentiment analysis data set benchmarks.

To verify these results we performed many experiments – varying various aspects of our network architecture (the vectors, type of embedding, number of layers, type of unit, dropout, etc) and measuring the results. The next section describes those experiments in detail and presents the results.

5 Experimental Evaluation

To verify our results, we ran a series of experiments. In all we tested over 100 different neural network models. All the networks contained an initial embedding layer for the word vectors, connected to between 1 and 5 hidden layers, and a final output layer.

Each network takes as input a text sequence of 10-1000 words, such as a movie or product review, and assesses to which one of the target classes they belong to. In all we ran four experiments testing different (1) network structures, (2) vectors, (3) data sets and (4) domain adaptation.

All of our experiments were run on a high powered Linux computer with a Titan X 12Gb graphics card. We used the Keras library for Python, running on top of Theano.

5.1 Experiment 1 – Network Structures

This was not the first experiment we ran (that was the “datasets”) but it is most logical to present this one first. In this experiment, we use the same pre-trained vectors over many different network structures. For the vectors, we used Stanford's 300 dimension GloVe vectors for 2.1 million words trained on 842 billion words of Common Crawl data. The vocabulary size is N=2,100,000 and the number of dimensions corresponds to H=300.

We tried 16 different network structures: all combinations of 1 or 3 layers, GRUs or LSTMs, 16 or 256 nodes per layer, and with a dropout of 0.2 or 0.5 between layers. The final layer is a binary classifier with a binary cross entropy loss function.

For each of these 16 structures, we ran four tests. First, as a control, we initialised the vector space with uniformly distributed random numbers and did not allow the vectors to be updated (we call this "static random"). Secondly, we used initially random vectors and allowed them to be trained ("trained random"). This is the approach typically followed in most papers, where vectors are trained jointly with other parameters as part of a specific task. Thirdly we used the pre-trained GloVe vectors and did not allow them to be trained ("static vectors"). Finally, we allowed pre-trained vectors to be updated ("trained vectors"). This is often referred to as "fine tuning".

Layers	Type	Units	Dropout	Static random	Trained random	Static vectors	Trained vectors
1	GRU	64	0.20	64.2	78.9	**89.8**	80.5
1	GRU	64	0.50	58.4	78.6	**85.3**	78.2
1	GRU	256	0.20	54.9	79.8	**90.0**	80.5
1	GRU	256	0.50	50.5	80.7	**89.8**	80.8
1	LSTM	64	0.20	58.8	82.4	**88.2**	80.9
1	LSTM	64	0.50	50.0	82.2	**85.0**	81.5
1	LSTM	256	0.20	49.9	80.6	**89.4**	81.1
1	LSTM	256	0.50	49.9	83.3	**87.3**	83.2
3	GRU	64	0.20	69.1	83.7	**90.6**	83.7
3	GRU	64	0.50	65.6	83.4	**88.7**	83.6
3	GRU	256	0.20	83.6	82.1	**90.1**	83.4
3	GRU	256	0.50	49.9	83.5	**90.9**	84.5
3	LSTM	64	0.20	50.0	82.2	**87.4**	82.9
3	LSTM	64	0.50	49.8	50.0	50.0	49.9
3	LSTM	256	0.20	49.9	83.2	**89.3**	83.4
3	LSTM	256	0.50	49.9	**58.9**	49.9	50.0

Table 2: Experiment 1 – Network Structures
16 different network structures on IMDB

For data we used the standard International Movie Database (IMDB) of movie reviews labelled as either positive or negative. The data set consists of 20000 reviews for training, 5000 for validation and 25000 for testing. We allowed each network to run for 50 epochs over the training data, with a maximum sequence length of 300. Each run took between 20 minutes and 7 hours, with the longer times for the trainable vectors.

We tried a couple of the tests multiple times and found that results can be up to 0.5% different over different runs. We present accuracy results to one decimal place.

The Table 2 shows 16 different network structures, with varying layers, unit types, numbers of units and drop-out. Each row shows four percentage results for the four vector/embedding types.

We can see that results for static vectors are consistently higher in almost all cases.

"Static random" vectors means that the vector space provides no language structure, and so the task of semantic analysis must be learned wholly by the 1 or 3 hidden layers of the network. In this case, more layers and nodes perform better, with a 3 layer GRU of 256 units each scoring 83.6%. In that network there are approximately 3 layers x 256 units x 256 units x 9 parameters per unit = 1.8m parameters (the first layer actually has 300 x 256).

If we allow those random vectors to be trained (the traditional approach to embeddings) then most of the network structures can achieve 79% to 83%. With an initial layer of 256 units, there are an extra 2.1m x 300 x 256 parameters to train, about 160 million of them. This enables the network to learn much more. Leaving it running for more than 50 epochs does not improve results much.

Pre-trained "static vectors" performed the best of all. Multilayer GRU networks are particularly effective, achieving nearly 91%. In fact, the network achieves this after only 20 epochs, so training is much quicker too.

Pre-trained non-static vectors, where the network is allowed to "fine-tune" the vectors performed about the same as random trained vectors. We can see that these networks do fairly well after just 1 epoch, a bit better after 5 or 6 epochs and then start degrading in performance. This is a clear sign of overfitting.

We didn't intend to test it, but the results also show that GRUs outperform LSTMs over the same number of epochs and it seems that GRUs also handle drop-out better than LSTMs. The highest result above is 90.9% for a 3 layer GRU network with 256 nodes per layer and a dropout of 0.5.

5.2 Experiment 2 – Vectors

Our second experiment analysed two different network structures, trained over 17 different vector sets of varying dimensions. We used all the vectors provided by the GloVe project – 4 sizes of vectors trained on 6 billion words from Common Crawl, one trained on 42 billion and one on 840

Vectors	**Vocabulary**	**Dims**	**1x128xLSTM**		**3x64xGRU**	
			Static	**Trainable**	**Static**	**Trainable**
crawl6B	400000	50	81.0	82.5	84.9	**86.5**
crawl6B	400000	100	85.9	81.2	**87.9**	85.0
crawl6B	400000	200	86.7	80.7	**89.3**	83.9
crawl6B	400000	300	88.0	80.4	**89.1**	83.1
crawl42B	1917494	300	89.5	81.4	**90.1**	84.7
crawl840B	2196017	300	88.8	81.9	**90.6**	84.7
twitter	1193514	25	73.0	84.0	82.5	**87.9**
twitter	1193514	50	83.6	81.8	**86.0**	85.9
twitter	1193514	100	84.4	81.4	**86.9**	85.3
twitter	1193514	200	87.0	80.6	**89.1**	84.9
random	2196017	100	50.0	**80.9**	62.0	77.4
random	2196017	200	50.0	**80.6**	69.4	78.3
random	2196017	300	49.9	80.5	73.9	**81.0**
IMDB training	26783	300	88.6	83.3	**90.1**	82.8
IMDB all	41215	300	88.8	86.9	**91.3**	83.4
News+Wiki	832528	300	88.3	86.9	**89.3**	88.7
Google	3000000	300	85.7		**89.4**	

Table 3: Experiment 2 – Vectors
17 different vector sets of varying dimensions on IMDB

Model	IMDB	Yelp	Amazon	Yelp 5	Amazon 5	Yahoo Answers
char-CNN (Zhang and LeCun, 2015)		94.7	94.5	62.0	59.5	71.2
VDCNN (Conneau et al., 2016)		95.7	95.7	64.7	63.0	73.4
fastText with bigrams (Joulin et al., 2016)	88.3*	95.7	94.6	63.9	60.2	72.3
Ours (RNN with Static vectors)	**91.6**	**96.6**	**96**	**66**	**63.3**	**74.7**

Table 4: Experiment 3 – Datasets
Sentiment Analysis and Topic Classification Benchmarks

** We trained fastText on IMDB ourselves.*

billion, along with 4 different sizes trained on Twitter data.

In addition, we included 3 initially random vectors sets (which corresponds to the traditional embeddings approach labelled "trained random" in the previous experiment). Plus, we used the word2vec algorithm to train our own vectors on just the 20,000 reviews in the IMDB training set (which took about 15 seconds), and further vectors on all 50,000 IMDB movie reviews (which took 40 seconds). We also trained our own vectors using an updated version of the shell script provided by word2vec to download and train on about 8 billion words of news and Wikipedia data. Finally we include the Google News vectors, 3 million vectors of words and phrases trained on 100 billion words from the Google News Corpus (Mikolov et al., 2013). For Google our preprocessing included combining common word sequences into phrases as found in the Google News vocabulary.

We tried two different network structures – a single layer LSTM network with 128 nodes, and a 3 layer GRU network with 64 nodes each. The former because most academic papers feature LSTM networks. The latter because we found GRU worked particularly well in our settings.

For each vector space and network structure, we trained a static and trainable model, making 68 tests in total. For all tests, we used a dropout of 0.3, a maximum sequence length of 300 words, and trained for 50 epochs over the IMDB training set.

Again we can see (Table 3) that static vectors beat trainable vectors in most cases. Only for the initially random vectors and those with very low dimensions (the 50 dimension Common Crawl 6 billion words, and the 25 dimension Twitter vectors) does training the vectors improve results.

One of the most interesting results is for the IMDB vectors, vectors we created ourselves using word2vec on the 20000 training reviews. This result is the most direct parallel to training initially random vectors (the random 300 trainable results) because it has only ever seen the training data (whereas all the other vector spaces have seen other data as well). Compare the score of 90% for doing a 2 step process (training vectors, then training the network) with the 81% for training the vectors and RNNLM together from an initially random vector space. Not only 9% better but nearly 3 times faster too – 1:22 hours versus 3:50 hours for 50 epochs.

Note that there are no results for trainable Google vectors. The Google News vectors include short phrases and so have the biggest vocabulary size of 3 million. We were unable to train these networks because they required too much memory on the graphics cards.

5.3 Experiment 3 – Datasets

We wanted to ensure our results weren't just a peculiarity of using a small data set, and we wanted to see if we could match state-of-the-art results. We referred to the table of results from the Facebook AI Research Lab's paper Bag of Tricks for Efficient Text Classification (Joulin et al., 2016). They compared their new fastText algorithm with a variety of algorithms on several different data sets.

For these tests we used Stanford's GloVe vectors and Google News vectors. Our network architecture for almost all cases consisted of small 5 layers of 64 GRU units. We began by trying 1 layer with many more nodes but our training machine ran out of video memory. With several layers of fewer nodes, the model could fit in memory, and our tests above showed that fewer nodes spread over several layers with dropout could perform very well.

We ran our architecture over five different English data sets from the paper above: 560000 Yelp business reviews with positive/negative sentiment classification, 3.6 million Amazon product reviews with positive/negative, Yelp business reviews with 1-5 stars, Amazon product reviews with 1-5 stars, and a non SA task of classifying Yahoo answers by topic. We also include tests on the IMDB database. These did not appear in the paper above, so we could only compare to fastText, which we downloaded and compiled and ran ourselves.

For the 5 star and 10 topic data sets, we implemented softmax as the output layer of our network rather than binary classification.

The Table 4 shows the results. Again with the caveat that we use an external data source (the vectors).

We were able to show the state-of-the-art results in all cases.

5.4 Experiment 4 – Domain Adaptation

For our final experiment, we wanted to see how well our models generalised, how well they could adapt to different domains. So we tried each of the models trained above on the different test sets. For example, we tried the IMDB-trained model on the Amazon test set.

For comparison, we trained Facebook's fastText algorithm. We chose this one because it was the easiest to download and get running. It is also very fast to train, on the order of seconds or minutes, as compared to hours for our RNN training, which was very impressive. Also fastText has the built-in ability to work with n-grams (sequences of multiple words). We tried up to four-grams and show the results where it led to an improvement.

The Table 5 shows models trained on the IMDB, Yelp and Amazon data sets and tested against the IMDB, Yelp and Amazon test sets.

This table shows that static vectors generalise very well. The score of 93.0% for the Amazon-trained model, tested on the IMDB test set is an especially good result. .

6 Future Work

We have run extensive tests on the simple text classification task to show that static pre-trained word vectors can improve RNN performance substantially. This work can be logically extended to other text related tasks.

Trained on IMDB	IMDB	Yelp	Amazon
fastText	88.3	79.5	80.2
fastText bigrams	87.8	79.5	80.1
Static vectors RNN	**91.6**	**87**	**87.8**
Trained on Yelp	IMDB	Yelp	Amazon
fastText	83.0	93.8	86.2
fastText bigram	85.1	95.7	89.1
fastText trigram	85.4	95.8	89.4
Static vectors RNN	**88.6**	**96.6**	**91.9**
Trained on Amazon	IMDB	Yelp	Amazon
fastText	88.1	87.0	91.6
fastText bigram	90.9	89.0	94.2
Static vectors RNN	**93**	**94.7**	**96**

Table 5: Experiment 4 – Domain Adaptation

Our best results above were achieved with the vectors from the largest vector spaces – those trained on the most number of words. However both the GloVe 840 billion vectors and the Google News vectors are getting old (2014 and 2013 respectively). New words and terms have entered common usage since then.

We feel there is a need for someone to pre-train and maintain high quality vectors in many different languages, with common words combined into phrases. Along with this there is a need for a rigorous test to define the quality of vectors.

Another direction is to investigate transfer learning between tasks. Word vectors pre-trained on larger than a task-specific dataset can be viewed as useful universal low level features across multiple tasks.

We also think that understanding the nature of the hidden states in Neural Network Language Models could reveal some new interesting properties of pre-trained word vector spaces and help us to understand a bit more about the fundamental building blocks of natural language.

7 Conclusion

Through a series of experiments encompassing hundreds of different networks, we have shown that using pre-trained word vectors in RNNs text classifiers has many advantages over the more traditional approach of training embeddings as part of an RNN.

The main advantages are a significant improvement to test accuracy, and a decrease in training time and memory usage.

Pre-trained word vectors also help to generalise and prevent overfitting, allow for the usage of very large vocabularies.

References

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

Mikolov Tomas: Statistical Language Models based on Neural Networks. PhD thesis, Brno University of Technology, 2012.

Andrej Karpathy: Connecting Images and Natural Language, PhD Thesis, 2016.

A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of Tricks for Efficient Text Classification, 2016.

Y. Kim, Convolutional Neural Networks for Sentence Classification, in EMNLP, 2014

G.E. Hinton. Learning distributed representations of concepts. In Proceedings of the Eighth Annual Conference of the Cognitive Science Society, pages 1–12, Amherst 1986, 1986. Lawrence Erlbaum, Hillsdale.

Yoshua Bengio, Reejean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. The Journal of Machine Learning Research, 3:1137–1155, 2003.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, pages 160–167. ACM, 2008.

Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S.; Dean, Jeff . Distributed representations of words and phrases and their compositionality. Advances in Neural Information Processing Systems. 2013.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

Omer Levy and Yoav Goldberg. Linguistic regularities in sparse and explicit word representations. In Proceedings of the Eighteenth Conference on Computational Natural Language Learning,pages 171–180, Baltimore, Maryland. 2014.

J. Elman. Finding Structure in Time. Cognitive Science, 14, 179-211, 1990.

S. Hochreiter, J. Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8):1735-1780, 1997.

K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259, 2014.

Peng, Hao and Mou, Lili and Li, Ge and Chen, Yunchuan and Lu, Yangyang and Jin, Zhi. A Comparative Study on Regularization Strategies for Embedding-based Neural Networks. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. 2015.

Yarin Gal, Zoubin Ghahramani: A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. NIPS 2016: 1019-1027. 2015.

Dumitru Erhan and Yoshua Bengio and Aaron Courville and Pierre-Antoine Manzagol and Pascal Vincent and Samy Bengio. Why does Unsupervised Pre-training Help Deep Learning? Journal of Machine Learning Research: 625-660. 2010.

Waltz, D.L. & Pollack, J.B. Massively parallel parsing. Cognitive Science. 9, 51-74. 1985.

Y. Bengio. Deep Learning of Representations for Unsupervised and Transfer Learning. JMLR W&CP 27:17–36, 2012.

Alexis Conneau, Holger Schwenk, Loic Barrault, and Yann Lecun. Very deep convolutional networks for natural language processing. arXiv preprint arXiv:1606.01781. 2016.

Xiang Zhang and Yann LeCun. Text understanding from scratch. arXiv preprint arXiv:1502.01710. 2015.