Sequential Convolutional Architectures for Multi-Sentence Text Classification *CS224N - Final Project Report*

Alfredo Láinez Rodrigo
alainez@stanford.edu

Luke de Oliveira
lukedeo@stanford.edu

Abstract

We describe a natural extension of deep learning to the multi-sentence text analysis domain – in particular, we construct a novel recurrent convolutional architecture by drawing inspiration from video analysis to exploit the hierarchical and sequential nature of text. We provide not only a high-performing and general model, but also a more linguistically plausible text classification scheme than bag-of-n-grams or averaged word-vector based approaches.

1 Introduction

Text classification has always been an important topic for both the natural language processing and machine learning communities. Recently, with the advent of social networks and social media, vast amounts text are available for analysis. This has led to an increase in interest and demand for text classification, in particular sentiment analysis, which can be used to easily gauge the public opinion and is invaluable to marketers.

Classically, the problem of text classification has been solved by methods that find words and combinations of words that are representative of a certain category. For instance, in a text sentiment task, it is relatively easy for a simple classifier to track words that convey positive or negative meaning and use those to predict whether the text is positive or negative. While this approach works surprisingly well, it is not hard to find examples that will be misinterpreted and wrongly classified (such as varying scopes of negation irony, and hyperbole).

To build linguistically richer models, researchers in NLP are turning to deep learning so as to exploit lower level, structured information. Deep learning has been successfully applied to many tasks in natural language processing ranging from named entity recognition to neural machine translation. However, we note that in terms of text classification, most work in the NLP community has been focused on word (Pennington et al., 2014) (Mikolov et al., 2013) and sentence level models (Socher et al., 2013) and representations.

While the clear linguistic shortcomings of classic methods for classification are less problematic when applied to long texts (which have more information), interest in applying new deep architectures to general, multi-sentence text has very recently arisen, and is still seen as an area relatively untouched by the huge boosts in performance usually likened with deep learning.

In this paper, we review some basic deep learning models and architectures than can be adapted to perform text classification. In particular, we consider mean of word vectors, recurrent neural networks (RNNs) and convolutional neural networks (CNNs). Then, we present a novel deep architecture based on recurrent convolutions of sentences that takes advantage of structure in text in order to perform classification of long documents. Finally, we test our model in two classifications tasks: sentiment analysis in IMDB reviews and humor detection in Yelp reviews.

2 Review of deep learning models for text classification

2.1 Average of word vectors

Word embeddings, such as *Word2Vec* (Mikolov et al., 2013) and *GloVe* (Pennington et al., 2014), are one of the most important recent advances in natural language processing. The representation of words in a semantic vector space offers many advantages to using the previous atomic representations. Given a text, one option to perform classification is to take all the word vectors and average them into a text vector that can be be used as input for a general classifier. As experienced by the authors in (de Oliveira and Lainez, 2015), this approach performs surprisingly well when combined with deep neural network architectures, but not so well when used with other classifiers such as SVM or Random Forests.

2.2 Recurrent neural networks

The *average of word vectors* approach heavily resembles the classic bag of words models in that word order is totally disregarded. One option to take this order into account is to use recurrent neural networks. Recurrent neural networks (RNNs) operate over a sequence of inputs and produce an output for each time step, with the goal of enabling the propagation of context information through the sequence. The sequential structure of text makes RNNs a strong candidate method to model language. Furthermore, more advanced architectures have been developed to better capture long-term dependencies, such as LSTMs (*Long Short-Term Memory*) and GRUs (*Gated recurrent units*).

Recurrent neural networks can be adapted for text classification. Feeding the sequence of tokens that constitute a text, we can train a network by taking the last output of the sequence as the classification result. This approach looks appropriate as a way to model text. However, the difficulty of the network to keep long-term relationships and the existence of problems like vanishing gradients make it difficult for the architecture to correctly capture the important characteristics of long texts. As we saw in (de Oliveira and Lainez, 2015), for the particular problem of humor detection, RNNs performed similarly to the linguistically-unmotivated model of taking the average of word vectors.

2.3 Convolutional neural networks

Finally, we examine convolutional neural networks (CNNs) à la (Kim, 2014). These networks can bridge many levels of abstraction, and so they may be able to capture higher order structures constituting classification labels.

In order to apply CNNs to text, we let $\bigoplus$ be the horizontal-wise concatenation operator. The model takes as input a word vector matrix over a vocabulary of size V and provides word vectors of size w. We let this matrix be defined as $U \in \mathbb{R}^{V \times w}$, where $U[i]$ is the word vector (a row vector) for word i. Note that we set $U[0]$ to be the zero vector for the padding. So, for a given sentence $s = (t_1, t_2, \ldots, t_L)$, we define the *sentence image* as

$$X_s = \bigoplus_{i=1}^{L} U[t_i]^T$$

Where now the ith column of X_s is the word vector associated with the ith word (t_i) in s. For a given n-gram size n, we can construct convolution filters of shape $(w \times n)$ followed by pooling of shape $(L - n + 1 \times 1)$.

A way of using this architecture, as described in the paper, is using n-grams of different sizes, where each filter has many features. We then apply dropout, and have a fully connected layer that feeds into a final classification layer, which depends on the classification task in hand.

Although originally intended for sentence classification, the CNN architecture described here can be equally applied to full texts. However, apart from being computationally expensive when applied to long texts, it makes less sense due to the fact that a lot of information might be lost when max pooling *across sentences*. Similarly to what happens with recurrent neural networks, we hypothesize that at some point the higher order structures and contextual information captured by the network are lost when classifying a big document.

3 Recurrent Convolutional Architecture

As seen in (de Oliveira and Lainez, 2015), the three approaches yielded good results and showed improvements over classical classifiers, at least in the task of humor detection. However, since these architectures were not created for long sequences of text, they have some common limitations apart from the ones already mentioned for each one in particular. First of all, and due to practical considerations for GPU batching, one must select a predefined text size and cut down or zero-pad the sequence to get the desired length. In domains with high variance of text size, a trade-off between training time and information loss must be decided.

Furthermore, and more importantly, these models are not based on the language structure of long text. While RNNs and CNNs take into consideration word order and context, they do not account for natural language organization. When using the previously explained models, we hope that the text will be short and clear so that the model will capture the characteristics of the classes that we are interested in classifying. However, written language is transmitted in the form of compositional elements that convey different levels of information. Paragraphs have a concrete theme, which is expressed by means of several and interrelated sentences. Sentences, in turn, aim to transmit a simpler concept by means of a sequence of words. Complex transmission of ideas such as contrasts between sentences or dependencies between not consecutive phrases will be irremediably lost when using models based on word sequences and on a vague context of the sequence.

To solve this problem, we expand on the ideas of (Kim, 2014) and draw inspiration from video analysis (Srivastava et al., 2015). In particular, we recast sentences as *frames* in a video, and use recurrent neural networks to model time evolution. Formally, as in 2.3, suppose our vocabulary is of size V and provides word vectors of size w. We further define a matrix U as $U \in \mathbb{R}^{V \times w}$, where $U[i]$ is the word vector (a row vector) for word i. Note that we set $U[0]$ to be the zero vector for the padding. We define a *text* $\mathcal{T}$ to be some sequence of sentences

$$\mathcal{T} = (s_1, s_2, \ldots, s_T)$$

where each sentence s_i is a sequence of tokens

$$s_i = (t_1, t_2, \ldots, t_{L_i})$$

For a given sentence s_i, we define the *sentence image*, once again, as

$$X_{s_i} = \bigoplus_{j=1}^{L_i} U[t_j]^T \in \mathbb{R}^{w \times L_i}$$

Note that the choice of U is arbitrary. We can use multiple sentence images per sentence, leading us to the notion of multiple word vector channels.

Zooming out, we see that for a text $\mathcal{T}$, we have a sequential, convolved representation

$$(X_{s_1}, X_{s_2}, \ldots, X_{s_T})$$

for each channel of word vectors considered. For each sentence image, we apply many n-gram convolutions, For a given n-gram of size n, we can construct convolution filters of shape $(w \times n)$ followed by pooling of shape $(L - n + 1 \times 1)$. If we do this across channels, we obtain a fixed vector representation for each timestep, or in this case sentence. We note that the application of a max-pooling operator is less problematic here than in the case of a CNN directly applied to a longer piece of text without sentence separation. This is due to the fact that the pooling is applied across one sentence rather than many, meaning that less information is lost due to the down-sampling that it produces.

At a higher level, after convolutions and max-pooling at the sentence level with various n grams sizes, we apply dropout, flatten, and concatenate to obtain a sequence of fixed length inputs, which we then feed into a bi-directional GRU. We then concatenate the two directions, and attach a fully connected, maxout unit in top. Finally, we attach a logistic regression unit on top of the outputs of the maxout. The visualization of our global architecture is in Figure 1.

To fully parametrize our model, we have the following list of architectural decisions / constraints.

- GloVe size, w
- Number and source of *word vector channels*, k, and whether or not they are *trainable*
- Set of convolved n-gram sizes, $\mathcal{N}$
- Number of feature maps per convolution filter size, m
- Max-pool size, though this is fully determined by sentence length
- Post-convolution + maxpool dropout rate
- Gated Recurrent Unit representation size + dropout rate upon bi-directional concatenation
- Number of maxout units and associated choice of the number of piecewise components + dropout rate

4 Implementation

For any given text, the first step prior to applying our deep architecture is perform sentence recognition and word tokenization. At first glance, finding sentences might seem to be a trivial task, but it is actually a difficult problem due to the fact that punctuation marks are often ambiguous (a period might denote an abbreviation, a decimal point, an email address, etc.). To solve both sentence boundary detection and word tokenization, we use

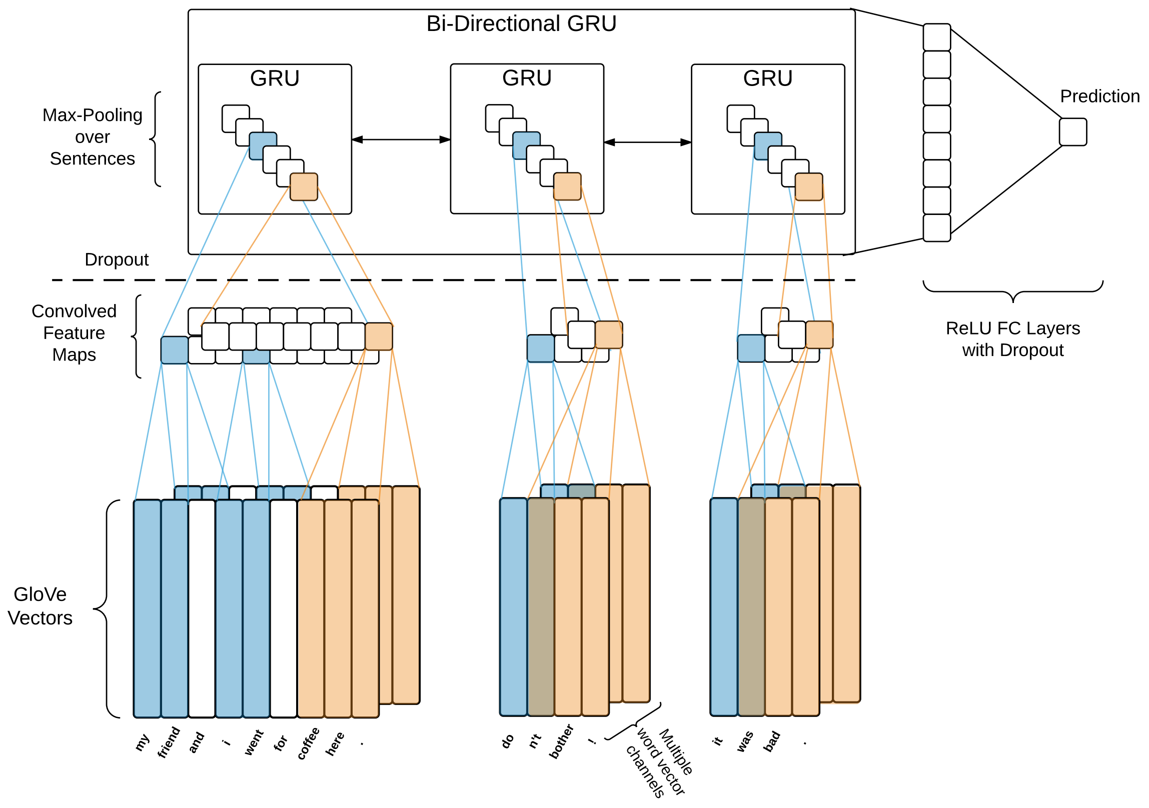

Figure 1: Our Recurrent Convolutional Architecture

spaCy (Honnibal, 2014), a lightweight cythonic library for natural language processing. Our preprocessing, which is easily parallelizable, takes a given text and converts it into a list of lists, organized in such a way that each outer list is a sentence in the text, and each sublist contains the words associated with that sentence. Due to practical considerations for GPU efficiency, both number of sentences and number of words per sentence must be predefined for a specific text problem, leading to the need of zero-padded or cut-down texts. As mentioned before, a trade-off between information loss and training time must be decided, although here we assume that we are dealing with multi-sentence texts. For our specific datasets, we computed histograms of sentence and paragraph sizes to find a point that allows us to lose less than 1% of trailing words across sentences.

Once we have a list of sentences consisting of word tokens, we transform these tokens to word vectors. For this project we used *GloVe* (Pennington et al., 2014). In particular, we considered the general 100 and 300-dimensional word vectors, trained on Common Crawl. After initial experiments, we settled on using 300 dimensional vectors, and we leave a thorough examination of the effect of word vector dimension to later studies. Using the original implementation from (Pennington et al., 2014), we also train dataset-specific glove vectors for our datasets, allowing us the opportunity to exploit multi-channel models. In all cases, we allow for a trainable `<unk>` token.

To implement our architecture, we utilize the Keras (Chollet, 2014) library, a Theano-based (Bastien et al., 2012) neural network library. In addition, we make heavy use of the CuDNN (Chetlur et al., 2014) library, extensively using heuristics to choose both forward and backward convolutional implementations suited to our uniquely shaped convolutional filters. Through our studies, we found that for our architecture, usage of CuDNN led to a factor of 3 speedup for training time per epoch.

We now provide a walk-through of data flow through our framework.

1. Data is initially ingested as a 2D `uint8` tensor, flattened across the paragraph dimension. *Note* that a separate stream is used for ingestion across each word vector channel to allow for differences in vocabulary coverage. At this stage, tensor is of shape `(n_samples, n_words)`

2. Data is passed through a lookup [1] table. As a result, we have `k` 3D `float32` tensors (one for each channel) of shape `(n_samples, n_words, w)`, where `w` is the word vector dimension.

3. The 3D tensors from each channel are joined together to make a 4D tensor of shape `(n_samples, n_words, k, w)`

4. We reshape the resultant 4D tensor into a 5D tensor of shape `(n_samples, n_sent, n_words_sent, k, w)`. Finally, we roll across the 3rd and 4th axes, and we have our final tensor shape for ingestion: `(n_samples, n_sent, k, n_words_sent, w)`

After data manipulation, we use our customized, CuDNN-aware temporally distributed convolutional layer to act on the final three axes of our 5D tensor.

5 Experiments

To test our architecture, we utilize two datasets – the humor detection task in the Yelp Challenge Dataset, first outlined in (de Oliveira and Lainez, 2015), and the IMDB Large Movie Review dataset (Maas et al., 2011). These datasets represent two tasks that rely on the ability to deal with long term relationships in text that span across multiple sentences and even across paragraphs.

All experiments were run on NVIDIA GRID K520 and Titan X graphics cards, with CUDA 7.5, CuDNNv3, and bleeding-edge installations of Keras and Theano. As reference, we list the following timings for one epoch on the IMDB dataset (20000 samples with a batch size of 32).

- On a 2.6 GHz Intel Core i7 CPU, with a custom-compiled OpenBLAS installation, one epoch of IMDB training takes $\sim$ 3.25 days

- On a GRID K520 *with* CuDNN, one epoch of IMDB training takes $\sim$ 30 minutes

- On a GRID K520 *without* CuDNN, one epoch of IMDB training takes $\sim$ 1 hour

[1] An `Embedding` layer in Keras

- On a GTX Titan X *with* CuDNN, one epoch of IMDB training takes ~ 11 minutes

5.1 IMDB Dataset

5.1.1 Data description

The IMDB dataset is a repository of movie reviews together with a binary sentiment polarity label. Negative reviews are associated with scores of less than 4 out of 10, while positive reviews have scores from 7 to 10. There is a total of 50,000 labeled reviews, divided into two balanced training and test sets, each with 25,000 reviews. The dataset includes an additional 50,000 unlabeled documents for unsupervised learning purposes.

5.1.2 Setup

For word vectors, we use two channels – one from the pre-trained 300D common crawl vectors (Pennington et al., 2014), and the other custom trained on the IMDB unlabelled partition using the C-based GloVe package released in (Pennington et al., 2014). For practical purposes, we only consider the first 120,000 most frequent words obtained by Common Crawl.

We consider convolved n grams $\mathcal{N} = (2, 3, 4, 5, 7)$, and for each convolutional unit, we allow for 96 feature maps.

For our input streams, we zero-pad (or truncate, as necessary) to ensure a sentence length of 50 tokens (including punctuation). We also blank-pad sentence images to ensure that there are 50 sentences per review.

We allow the word vectors to be *trainable*, but we train them with respect to the constraint that each word vector must have unit ℓ_2 norm. For each convolutional unit, we apply an ℓ_2 norm penalty of 10^{-5} to each kernel[2]. We apply the ReLU activation function, with a dropout of 0.10 applied before a temporally distributed flattening operation.

We then take the flattened representations from each convolved n-gram unit and concatenate them, feeding into both forward and backward pass GRU layers (Chung et al., 2015), each with 64 units. We apply a dropout fraction of 0.7 to each direction after concatenation along the last axis.

With a fixed representation in hand, we then use a fully connected maxout (Goodfellow et al., 2013) unit with 64 output units and 16 degrees of freedom. Finally, we connect this to a simple logistic regression output, and train against negative log likelihood (cross entropy). To optimize the parameters of our architecture, we use the RMSProp algorithm (Tieleman and Hinton, 2012) with early stopping and 30 periods of patience. We split the training set into 20000 training and 5000 dev points, and we retain the full 25000 testing points as per the original partition.

[2]Tuned and included to combat observed large filter weights.

5.1.3 Results

We compare our model with the state of the art – in particular, against Paragraph Vectors (Le and Mikolov, 2014) and their enclosed results from (Wang and Manning, 2012). We are comparing against simpler models that either use an order-independent representation of words or a learned vector representation of a paragraph that is not representable as a function. In Table 1, we see that our Sequential Convolutional Network achieves nearly state-of-the-art results, and lags behind only Paragraph Vectors. We note that a key advantage of our model is that a Sequential CNN is a *function*, which allows application to unseen paragraphs – something which a Paragraph Vector based model cannot do trivially.

Model	Error rate
BoW (bnc) ()	12.20 %
BoW (bΔt'c) (Maas et al., 2011)	11.77%
LDA (Maas et al., 2011)	32.58%
Full+BoW (Maas et al., 2011)	11.67%
Full+Unlabeled+BoW (Maas et al., 2011)	11.11%
WRRBM (Dahl et al., 2012)	12.58%
WRRBM + BoW (bnc) (Dahl et al., 2012)	10.77%
MNB-uni (Wang and Manning, 2012)	16.45%
MNB-bi (Wang and Manning, 2012)	13.41%
SVM-uni (Wang and Manning, 2012)	13.05%
SVM-bi (Wang and Manning, 2012)	10.84%
NBSVM-uni (Wang and Manning, 2012)	11.71%
NBSVM-bi (Wang and Manning, 2012)	8.78%
Paragraph Vector (Le and Mikolov, 2014)	7.42%
Sequential CNN	**8.43**%

Table 1: Performance of Sequential Convolutional Networks compared to other approaches on the IMDB dataset. The error rates of other methods are reported in (Wang and Manning, 2012) and (Le and Mikolov, 2014).

5.2 Yelp Dataset

5.2.1 Data description

The data set consists of 147,444 English reviews of businesses extracted from the *Yelp Dataset Challenge* data with two labels: funny or not funny. Funny reviews are selected as those that have three or more "funny" votes, while not funny reviews do not have any vote. Both classes are balanced and divided into a train, dev and test sets.

5.2.2 Setup

We use a very similar setup to the one described for the IMDB dataset. The main difference is that we use only one channel for training due to poor results when adding more than one. Similarly to the previous case, we compute specific word vectors for Yelp reviews and allow them to change during training. We present results for a convolution over n grams $\mathcal{N} = (1, 2, 3, 4, 5, 6)$ and 21 feature maps, with a GRU of size 20 and an input of 30 words per sentence and 30 sentences per review.

Model	Error rate
Logistic Classifier (BoW)	27.71%
SVM (BoW)	26.76%
Random Forest (BoW)	24.58%
Average of word vectors	21.79%
RNN (GRU)	21.80%
CNN	21.72%
Sequential CNN	**21.63%**

Table 2: Performance of Sequential Convolutional Networks in the task of predicting humor, compared with previous results as seen in (de Oliveira and Lainez, 2015)

5.2.3 Results

As we can see in Table 2, our Sequential CNN obtains improved results over classic classifiers, while it is in line with the other deep learning classification methods commented on in this project, showing a slight improvement over them when using a larger testing sample than in our original work. However, we note that the model does not present significantly better results than other deep learning approaches, or even a simple average of word vectors. As we hypothesized in our previous work, sequential structures that constitute humor – if any – may be too complex for any general model to leverage. It is encouraging, however, that our Sequential CNN architecture obtains very good performance with minimal parameter tuning, indicating that we have a model that can be generally applied to multi-sentence text samples.

6 Conclusion and Outlook

We have presented a general framework and novel convolutional architecture for multi-sentence text classification, building upon natural text structure and the latest developments in deep learning for natural language processing. Despite little tuning of hyperparameters, we obtain very competitive results in a well-studied problem in sentiment analysis, as well as in a complex and open problem such as humor detection.

For the future, we plan on testing the model in other common classification tasks and work on architectural changes and extensions. In particular, we would like to examine extensions involving an external memory unit, which we hypothesize will help with longer term text relationships. We would also like to test the model in the information retrieval domain. Finally, we would like to experiment with the inclusion of additional layers over the outputs of all the GRU sequences, in essence stacking the bidirectional recurrent layers.

References

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. 2012. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759.

François Chollet. 2014. Keras. `https://github.com/fchollet/keras`.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2015. Gated feedback recurrent neural networks. 02.

George E. Dahl, Ryan Prescott Adams, and Hugo Larochelle. 2012. Training restricted boltzmann machines on word observations. *CoRR*, abs/1202.5695.

Luke de Oliveira and Alfredo Lainez. 2015. Detecting humor in yelp reviews using deep learning.

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Maxout networks. 02.

Matthew Honnibal. 2014. spacy. `https://spacy.io/`.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. 08.

Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

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

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. 2015. Unsupervised learning of video representations using lstms. *CoRR*, abs/1502.04681.

T. Tieleman and G. Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Sida Wang and Chris D. Manning. 2012. Baselines and bigrams: Simple, good sentiment and text classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, 2012*.