Document Embeddings via Recurrent Language Models

Andrew Giel
BS Computer Science
agiel@cs.stanford.edu

Ryan Diaz
BS Computer Science
ryandiaz@cs.stanford.edu

Abstract

Document embeddings serve to supply richer semantic content for downstream tasks which require fixed length inputs. We propose a novel unsupervised framework by which to train document vectors by using a modified Recurrent Neural Network Language Model, which we call DRNNLM, incorporating a document vector into the calculation of the hidden state and prediction at each time step. Our goal is to show that this framework can effectively train document vectors to encapsulate semantic content and be used for downstream document classification tasks.

1 Introduction

Word embeddings, also known as word vectors, have proven to be one of the most powerful building blocks for advancements in natural language processing in recent years. These distributed representations have shown the capacity to encapsulate the semantic and syntactic structure of language while remaining a fixed-length real-valued vector, allowing for easy usage and extension into a variety of tasks. Despite these advances for single words and short phrases, it is still unclear how to represent collections of many words such as sentences, paragraphs, and documents in as powerful and accessible a format. Traditional techniques for vector representations of documents such as bag-of-words models have obvious weaknesses such as the lack of word order preservation and the lack of semantic encapsulation. This paper builds upon the work of others in creating semantically and syntactically expressive general purpose fixed-length document representations. These document embeddings are trained using a modified Recurrent Language Model known as DRNNLM.

2 Related Work

Our work is inspired by prior research into document embeddings and Recurrent Neural Network Language Models (RNNLMs).

Quoc Le and Tomas Mikolov presented a paper titled *Distributed Representations of Sentences and Documents.* [3] which is the primary motivation and inspiration for our research. In this paper the authors present an unsupervised model as an extension of the Word2Vec model [5], capable of training what the authors refer to as Paragraph Vectors. These Paragraph Vectors are trained in two manners, very similar to the way that word vectors are trained. One method, known as Paragraph-Vector Distributed-Memory or PV-DM, averages or concatenates the Paragraph Vector into the context window for all windows of the document. PV-DM incorporates word order while training. The other method, which ignores word order, is very similar to the Continuous Bag-of-Words method for training word vectors. Known as Distributed Bag of Words version of Paragraph Vector, or PV-DBOW, this technique creates Paragraph Vectors by training to predict words within a window of the paragraph. The resulting PVs created by these methods proved to be very effective in downstream tasks, with results outperforming many (supervised) state-of-the-art methods on the

Stanford Sentiment Treebank and IMDB data sets. Even more remarkable is that both of these techniques are completely unsupervised (or create supervised tasks from unstructured data), meaning the amount of data that accessible to the model is nearly infinite. This paper is our primary inspiration for researching document embeddings, as it showed that the task was both possible but also extremely useful.

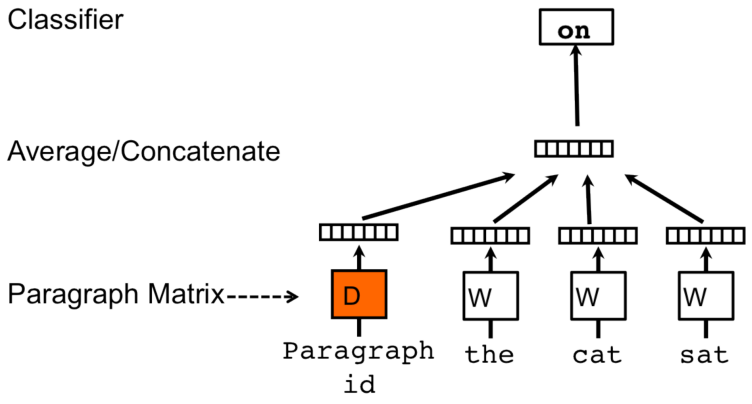

Figure 1: The PV-DM model

While Le and Mikolov's paper served as inspiration for this paper, Mikolov and Zweig's paper *Context Dependent Recurrent Neural Network Language Model*. [4] served as the guiding force for our model formulation. In this paper the authors demonstrated state-of-the-art results for the task of creating Recurrent Neural Network Language Models (RNNLMs) by supplementing the standard RNNLM architecture with a 'feature layer', which served as a context vector, incorporating this vector into both the hidden layer and output layer calculation. The authors found that a vector created by running Latent Dirichlet Allocation on the previous 50 words served as the best context vector, giving them state-of-the-art results in terms of perplexity over multiple datasets. This model was known as RNNLM-LDA. The RNNLM-LDA model architecture was highly influential when designing our own network, as will be elaborated upon later.

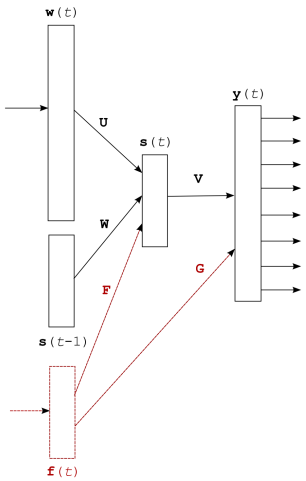

Figure 2: The RNNLM-LDA model

3 Model

We propose a novel model used to train document vectors int the form of a modified Recurrent Language Model, which we call DRNNLM, that incorporates the document vector in both the hidden layer and prediction at each time step.

3.1 DRNNLM

In this section we give a formal mathematical description of our model, which we call DRNNLM.

We begin with a word to vector mapping, where each word has a unique column within our matrix L. L can be randomly instantiated or with a set of pre-trained vectors such as GloVe or Word2Vec. Additionally, we have with a matrix D, representing the document matrix, with each document mapped to a unique column within D. This matrix will be randomly instantiated.

Just as for traditional n-gram language model, our DRNNLM is given a series of words $x_{m-n}...x_m$ with the goal of predicting x_{m+1}. Additionally, our DRNNLM takes a document d_i where the series $x_{m-n}...x_m \in d_i$. The values of our hidden layers h_t and output y_t are defined as follows

$$h_t = \sigma(Wx_t + Hh_{t-1} + d_i + b_h)$$

$$y_t = g(Uh_t + Gd_i + b)$$

where $\sigma(z)$ is the sigmoid function and $g(p)$ is the softmax function. This incorporation of the document vector into the hidden layer and prediction is very similar to the respective functions used by Mikolov for context dependent RNNLMs [4].

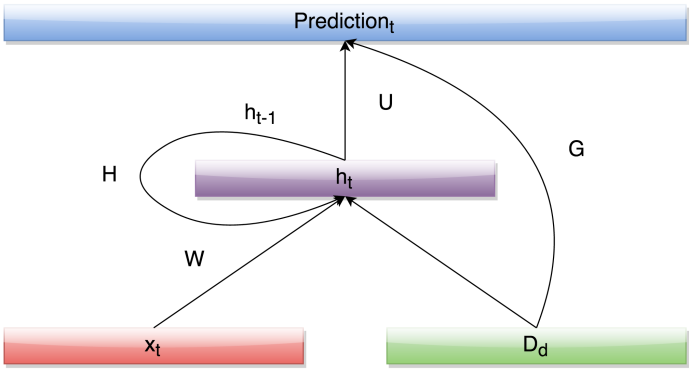

Figure 3: The DRNNLM model

3.2 Training

Training is accomplished via backpropagation of errors. In particular, we minimize the cross-entropy loss for our language model. More formally, our loss function $J(x_i, y_i, d_i)$ is defined as follows

$$J(x_i, y_i, d_i) = CE(x_i, y_i, d_i) = -\sum_{t \in x_i} y \log(\hat{y}_t)$$

Since the focus of this paper is to train our document vectors, D, we explicitly derive the gradient for D_{d_i}, the update for a single document.

$$\frac{\partial J}{\partial D_{d_i}} = \sum_t G^T(\hat{y}_t - y_t) + \sum_{i=0}^{C}(\sigma'(z_{t-i}) \cdot U^T(\hat{y}_{t-i} - y_{t-i}))$$

where $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ and C is the number of timesteps backwards we propagate through time.

It is possible within this framework to train only document vectors once the other parameters of the network have been trained. In this scenario, 'training' takes place in two stages. In the first, we minimize the cross-entropy loss for our language model over our corpus by backpropagating the errors into all of our parameters: L, D, W, H, U, G, b_h, b. Secondly, given a new corpus of documents, we expand D and for each new document train the corresponding column vector, once again by minimizing cross-entropy loss but only propagating errors to D. This second stage allows us to create vectors for arbitrary numbers of new documents quickly that can be used in downstream tasks.

3.3 Intuition

The intuitions for this network formulation and its capacity to the create of rich, meaningful document vectors are clear. Firstly, by training the vectors with a language model, we believe word order will be encapsulated. Secondly, by incorporating the document vector at each time step of the language model we believe the resulting document vector will be amply contextual and representative. In particular, our formulation mirrors that of Mikolov for context dependent RNNLMs [4]. Whereas Mikolov found improvements over standard RNNLMs by incorporating a context vector created via Latent Dirichlet Allocation, we use our document vectors as the context vector and backpropagate into them. The intuition here is that we can learn the context vector and use that context vector as the document vector.

4 Experiments

4.1 Data

For our experimentation we used the 20 Newsgroups dataset [2]. This dataset consists of 20000 short documents grouped into 20 distinct categories. We compared the effectiveness of several document representations by looking at the F1 score of each representation given the task of document classification.

We instantiated our L with GloVe [1] vectors, using a vocabulary size of 100,000. We found 100,000 to be large enough to contain almost all of the words found in the dataset while not too large. Any words found in the dataset that were not within the vocabulary were replaced with an 'UUUNKKK' token whose vector was randomly instantiated.

4.2 Implementation

Our model was written in the Python library Theano. Theano allows for optimized matrix operations, automatic differentiation, and the usage of Graphics Processor Units (GPUs). This allowed our model to be clearly defined and very fast. We ran all experiments on an Amazon AWS EC2 GPU instance (g2.2xlarge) in order to take full advantage of Theano's capabilities.

4.3 Challenges

One of the biggest challenges we faced was training time. Even using Theano and GPU instances, we found that our training time was far too large for the scope of this project. To make a full epoch through the training set was going to take approximately 24 hours, making training very expensive. This is most likely due to two challenges: training RNNs is very computationally expensive since the recurrence cannot be vectorized but must occur sequentially, and two of our network's dense matrices U, G are very large ($U, G \in R^{|V| \times h_{dim}}$. These two challenges forced us to choose our experiments wisely and limited our ability to tune hyperparameters.

4.4 Baselines

Our first attempt at creating a document representation was to average the word vectors of every word in the document. Using 300-dimensional GloVe vectors trained on the Wikipedia 2014 and Gigaword 5 datasets we averaged each word embedding to leave us with a 300-dimensional document vector.

These document vectors were used as input to a linear SVM (squared_hinge loss, C = 1.0, tolerance = 1e-4) to classify each document into one of twenty classes. The results of using the average-word document vector with a linear SVM were compared with the use of simple bag-of-words and tfidf document representations. The results show that the average-word document representation produces similar results to the bag-of-words model, both of which underperform the tfidf model.

4.5 DRNNLM

We trained the DRNNLM on our dataset for 25 epochs (limited by computation time). We ran the model over sentences from each document training each document vector by backpropogating the error of each incorrect word predicition into the corresponding row of D. D, the matrix containing the document representations of each document was used to run experiments measuring the representational capacity of our model using implicit and explicit evaluation metrics.

4.6 Results

4.6.1 Implicit

As shown in the figure below the representation of each document in 300 dimensional space the documents show little separation based on document class. K-nearest neighbor classification provides essentially random classification results. The document vectors produced from running our model do not show clear separability based on the class of the document.

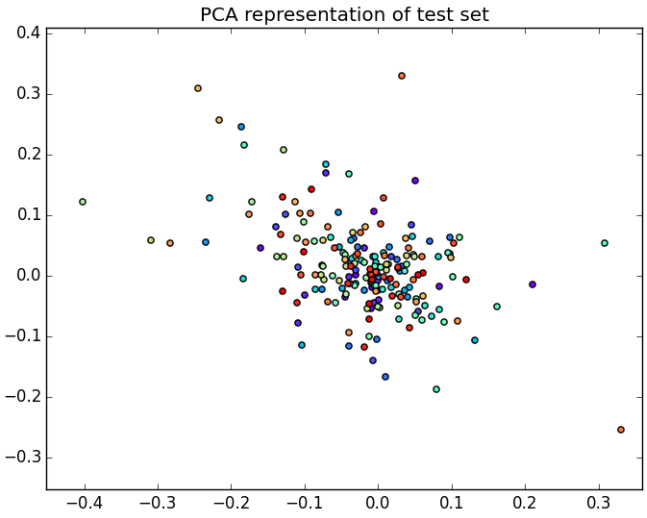

Figure 4: Sampling of 200 document vectors plotted via PCA

4.6.2 Explicit

Averaging the word vector representations of each word in each document for a document representation provides comparable results to other document representation techniques that use the word counts in a document to represent it in a fixed length vector. Our language model construction provides random classification and thus is not listed.

Table 1: Classification using a Linear SVM

Model	Precision	Recall	F1-Score
Average GloVe	0.69	0.70	0.69
Bag-Of-Words	0.69	0.69	0.69
tf-idf	0.81	0.80	0.80

5 Future Work

Future work on this model would include better training methods. Making this model more efficient to train and training for more epochs over that datset would make the model more representative of the content of each document. Starting training with a pre-trained language model on the dataset could help in training. The complexity of the model aims to create a deep representation of each document, but overall the entire model likely needs to be redesigned to find a better way to extract semantic meaning. Core aspects to the task of document embedding creation need to be improved.

6 Conclusions

The final results of this model were not impressive in terms of both implicit and explicit evaluation metrics. The complexity of the model versus the Mikolov document vector model led to the model's lack of practicality. Training both a language model and document vector simultaneously took a long time for a reasonably sized data set. Training the model for more epochs was not practical given our computational resources, but given much longer training time it is possible this model could show improved results. The complexity of our model and training through backpropagation meant that the document vector was only adjusted slightly for each training example, most of the classification error was used to train the language model. In the end traditional document representation techniques vastly out performed our model, but finding new ways to optimize and adjust our document representation system could potentially prove useful in future research projects.

References

[1] C. D. M. Jeffrey Pennington, Richard Socher. Glove: Global vectors for word representation. 2014.

[2] K. Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339, 1995.

[3] T. M. Quoc Le. Distributed representations of sentences and documents. 2014.

[4] G. Z. Tomas Mikolov. Context dependent recurrent neural network language model. 2012.

[5] K. C. G. C. J. D. Tomas Mikolov, Ilya Sutskever. Distributed representations of words and phrases and their compositionality. 2013.