9-2018

Recursive Neural Networks for Semantic Sentence Representation

Liam S. Geron
The Graduate Center, City University of New York

RECURSIVE NEURAL NETWORKS FOR SEMANTIC SENTENCE REPRESENTATION

by

LIAM SIMON GERON

A master's thesis submitted to the Graduate Faculty in Linguistics in partial fulfillment of the requirements for the degree of Master of Arts, The City University of New York

2018

Recursive Neural Networks for Semantic Sentence Representation
by

Liam Simon Geron

This manuscript has been read and accepted for the Graduate Faculty in Linguistics in satisfaction of the thesis requirement for the degree of Master of Arts.

Date	William Sakas Thesis Advisor
Date	Gita Martohardjono Executive Officer

THE CITY UNIVERSITY OF NEW YORK

ABSTRACT

Recursive Neural Networks for Semantic Sentence Representation

by

Liam Simon Geron

Advisor: William Sakas

Semantic representation has a rich history rife with both complex linguistic theory and computational models. Though this history stretches back almost 50 years (Salton, 1971), recently the field has undergone an unexpected shift in paradigm thanks to the work of Mikolov et al., 2013(a & b) which has proven that vector-space semantic models can capture large amounts of semantic information. As of yet, these semantic representations are computed at the word level, and finding a semantic representation of a phrase is a much more difficult challenge. Mikolov et al., 2013(a&b) proved that their word vectors can be composed arithmetically to achieve reasonable representations of phrases, but this ignores syntactic information due to the commutativity of the arithmetic composition functions (addition, multiplication, etc.), causing the representation for the phrase *"man bites dog"* and *"dog bites man"* to be identical. This work hopes to introduce a way of computing word level semantic representations alongside a parse tree based approach to composing those word vectors to achieve a joint word-phrase semantic vector space. All associated code for this thesis was written in Python and can be found at https://github.com/liamge/Pytorch_ReNN.

Acknowledgements

Thanks to my parents, who, despite me being in university for an unreasonably long amount of time, never complained once (to me) about it.

Contents

List of Tables

List of Figures

1 Introduction

Good numeric semantic representation is an extremely important task in the field of Natural Language Processing. Semantic representation can offer us direct routes towards modeling high-level ideas, and can provide essential features for some popular tasks such as sentiment analysis, information retrieval, sarcasm detection, word sense disambiguation, etc. It would be safe to say that any task that has an inherent semantic component could benefit from good semantic representation. Currently, the state of the art for sentiment representation is very good; a popular sequence of papers from Mikolov et al., 2013a & 2013b introduced a new method of computing dense vectors that capture good semantic information very quickly. These vectors have been extremely useful tools in many different tasks, but they are not a panacea to the semantic representation challenge.

One desirable feature of semantic representation is to have a semantic space, or more formally euclidian space in which word vectors are points. This semantic space can have very interesting properties, as Mikolov et al., 2013a & 2013b show, such as analogous linear transformations for analogous word pairs like: *"paris":"france"* and *"berlin":"germany"*. This space isn't limited to simply words, Mikolov et al. demonstrated that it extends to phrases as well by simple arithmetic functions. While certainly impressive, these phrase-level representations have a fatal flaw in that they are essentially a bag-of-words approach to phrase representation.

In this thesis we propose a model capable of modeling a joint semantic space for words and phrases directly, explicitly taking syntax into account. This model can take any variable sequence of words and embed it in the same space as it's word vectors, allowing for a shared semantic space between words

2 Background

2.1 Theoretical Background

In 2013, Thomas Mikolov et al. created a now-ubiquitous algorithm for computing word level semantic representations called Word2Vec. This algorithm relied on the seminal idea popularized by Firth that "a word is characterized by the company it keeps" (Firth, 1956). This distributional approach had been used in the past to much success, traditionally relying on term frequency matrix factorization techniques like Latent Semantic Analysis (Landauer and Dumais, 1997). One problem that models like LSA encounter is that they scale inefficiently with a time complexity of $O(min\{mn^2, m^2n\})$ for an $m \times n$ matrix (Holmes et al., 2007), and as such become unsuitable for the increasingly large corpora of modern datasets. Mikolov et al. mitigate this problem by not utilizing traditional matrix factorization methods, opting for methodology based on the proven effectiveness of utilizing neural networks to latently model these dense representations (Bengio et al., 2003).

Mikolov et al. base their model on the approach designated by Bengio et al., 2003, in which a Language Model is trained using a Feed Forward Neural Network (FFNN) by directly predicting the surrounding words for a given word. Formally, a word embedding matrix is randomly initialized as a $|V| \times d$ matrix, where V is the set of the vocabulary and d is the dimensionality of the word vectors. Each word of the vocabulary is then assigned a unique index, i, or a row of the embedding matrix. The task is then defined as: for each word w_t in the training corpus, where t is the order in which it appears in the corpus, the network attempts to predict w_{t+1}, or the next word in the sequence. The way this is achieved is by projecting the word vector that represents w_t, the ith row of the embedding matrix, using a hidden layer and a non-linear

transformation into an H dimensional vector. The network then directly predicts the next word by projecting that into a $|V|$ dimensional vector, where each index of that vector corresponds with the unique indices assigned to each word in V. This final projection uses the softmax function, which "squashes" the values into a range of (0, 1), and guarantees they sum to 1, thereby turning the values into probabilities. The ith place of the final $|V|$ dimensional vector then is the probability the network assigned to ith word being the next word in the sequence. The loss is measured using the cross-entropy function and back propagated to maximize the log likelihood of the $t+1$th word. These underlying principles are what Mikolov 2013a uses to build the Word2Vec algorithm, though they change the model slightly to either predict several words around the target word, or use the surrounding words to predict the target word.

Bengio et al., 2003 proved that semantic models based on Neural Networks are capable of latently modeling many desirable semantic properties inside of the word vectors, clustering semantically similar words together. One glaring problem with this model however, is that a $|V|$ dimensional softmax is hugely expensive computationally, and scales poorly to large datasets much like matrix factorization. Mikolov 2013b presents several ways of resolving this, ultimately settling on a sampling based method of approximating a softmax called Noise Contrastive Estimation (NCE) introduced in Gutmann et al., 2010.

NCE works by drawing k vectors picked from a unigram distribution in addition to the true target words, and running a sequence of binary classifications on those . The goal is then to distinguish the noise vectors from the true positive. This drastically reduces the number of parameters needed, and approximates the task done by a softmax over the vocabulary (Gutmann

et al., 2010). Both the usage of NCE and the overall model task presented in this thesis (i.e. predicting context words from a given word) are borrowed from Mikolov et al., 2013a & 2013b.

One shortcoming of the model presented in this thesis however, is that while the vectors learned by it capture much semantic information, using them to represent longer phrases is non-trivial. In Mikolov et al., 2013b, they show that the vectors can be combined arithmetically (summations, products, or averages of vectors) to have sensible representations of phrases, however due to the commutativity of these operations the phrase representations essentially become bag-of-word models where order (and syntax) doesn't matter. Formal semantics on the other hand relies heavily on syntactic representations of phrases in order to compose a semantic representation, and many semanticists believe the two to be deeply intertwined. It is reasonable, then, to assume that taking syntactic structure into account would improve these representations.

Socher et al., 2013 took this approach for the task of Sentiment Analysis. Sentiment Analysis is generally a task in which good semantic representation is key, particularly in the case of negational constructions such as “this movie was ***not*** good”. Being able to properly model a negation is essential to predict the overall sentiment of a phrase. In order to do this, Socher et al., 2013 employ the usage of what are called Recursive Neural Networks (ReNNs), which iterate recursively over a binary tree structure. This network uses a form of syntactic representation of a phrase called a binary parse tree, and combines daughters using a composition function *f* until it reaches the root of the tree, thereby representing the tree as compositions of its constituents. For example, the parse tree for “The cat is black” is found in Figure 1. Note that the order of compositions start from the leftmost leaves, where vectors *a* and *b* representing “the” and “cat” are composed into P_1 using the composition function *f*. The only requirement for *f* is that the

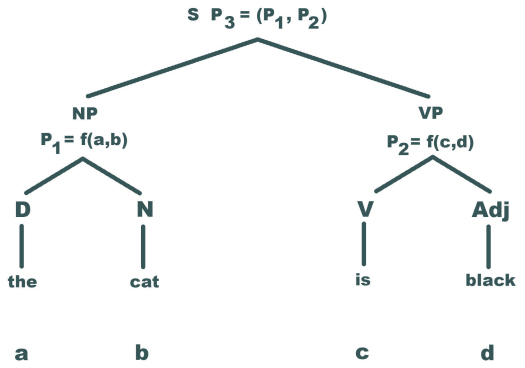

Figure 1. Example of the order of composition for a Recursive Neural Network over a binary parse tree

dimensionality of the output is identical to the dimensionality of the word-vectors, *v*. P_2 is then computed in the same way. Finally, P_3 is computed by recursively applying f to P_1 and P_2. The loss function (which is task dependent) is then used compute the loss at each node in the binary tree with an additional prediction. In the case of Socher et al., 2013, each node then predicts the sentiment of that constituent with an additional projection. Formally, in their simple Sentiment Analysis task there are three possible sentiments: positive, negative, and neutral. Each constituent vector is then projected using a *d x 3* matrix, *M*, into a 3 dimensional vector and evaluated using the standard cross entropy measured against a labeled dataset.

This model is capable of jointly training word-vectors and a composition function, thereby having a joint word-vector/phrase semantic space. Additionally, it takes syntax explicitly into consideration when computing the phrase vectors, allowing the model to capture complex semantic structures like negation (Socher et al., 2013). One problem that arises from this particular task however, is in the gathering of a sufficiently large dataset that has every constituent labeled for sentiment. Socher et al., 2013 relied entirely on human effort for this, which is not scalable to datasets similar to those of Mikolov et al., 2013a & 2013b, which contain billions of word tokens. This thesis presents a semi-supervised adaptation of Socher et al.'s model that directly trains for good generalized semantic representation, circumventing the reliance on human labeling.

2.2 Problem Description

The work of Mikolov et al., 2013a & 2013b are capable of modeling complex semantic constructions, however a shortcoming is in how their word vectors are used to represent multi-word phrases. While arithmetic composition of the Word2Vec word-vectors models semantic composition relatively faithfully (Mikolov et al., 2013b), it fails to capture word order information, making modeling negational constructions difficult to approximate due to how heavily those constructions rely on syntax.

Le et al., 2014 attempt to directly model longer phrases using a similar model as in Mikolov et al., 2013a. In this work however, they assign phrases a unique vector and use that vector to predict both the context words and the words contained within a phrase. They found that the vectors learned by this process were extremely capable for most semantics-based text classification tasks, oftentimes beating the previous state of the art (Le et al., 2014). While this model is capable of representing phrases semantically well, it still lacks the ability to take advantage of an explicit representation of syntax.

Socher et al., 2013, on the other hand, use explicit syntactic representation in order to capture a reasonable semantic representation of specifically negational constructions. The task the model directly trains for however, is Sentiment Analysis and does not directly train for good semantic representation like the model in Mikolov et al.'s.

This work attempts to remedy these shortcomings by combining the two influences into a Recursive Neural Network that directly trains for good semantic representation using the influence of distributional semantics. We posit that this new model is capable of representing any

phrase that can be represented syntactically, and that the addition of explicit syntactic representation improves the overall semantic representation of longer constructions.

3 Data

The data used for this experiment is the dataset both gathered and used in Socher et al., 2013, the Stanford Sentiment Treebank[1], hereby referred to as the SSTB. The SSTB is a collection of 11,855 sentences extracted from movie reviews parsed in binary tree format by the Stanford Parser (Klein and Manning, 2003). While the domain is relatively narrow and the dataset is small for the task of distributional semantics, we hope that it is enough to evaluate how the new model leverages parse trees to learn semantic representations.

4 Methodology

4.1 Model

This work hopes to expand upon Mikolov et al., 2013a, 2013b, and 2014 by introducing an adaptation of the Recursive Neural Network for the computation of reasonable generalized semantic representation. We structure our problem similarly to that of Mikolov et al., 2013a, though with the additional constraint of using a binary parse tree to assign a label. Formally, for a given constituent vector c, we attempt to predict a word w_i that is randomly assigned from a neighboring constituent of the tree. For example, in Figure 1 we predict from P_1 a word randomly assigned from all the leaves of P_2, namely “is” or “black”. This is the same principle of Mikolov et al., 2013a & 2013b where during the task of predicting context words the model

[1] Found here: https://nlp.stanford.edu/sentiment/code.html

latently computes a reasonable semantic representation of words and a reasonable composition function during the process of learning this task.

Formally, the parameters for this task are $E \epsilon |V| \; x \; d$, the embedding matrix, H, the composition matrix (dimensions rely on choice of composition function), and P, the projection matrix. For the most basic task, for each tree t, we first find the leftmost node in which all of its children are leaves, n, and compose the embeddings of it's left and right children, n_l and n_r respectively. For this work, the choice of composition function is either one layer of a Feed Forward Neural Network, or a layer of a Recurrent Neural Network.

In the instance of the FFNN composition function, $M \epsilon \; 2d \; x \; d$, where d is the dimensionality of the word vectors. In this case, we first concatenate the embeddings and then multiply that resulting $2d$ vector with H to get a d dimensional resultant vector. This vector then has a non-linear function applied, in our case the Rectified Linear Unit (CITATION) in order to capture some potential for non-linearities in the composition function. Figure 2 demonstrates this structure of a composition function where V represents the concatenation of the two children vectors, and H is the constituent representation such that $H = ReLu(MV + b)$, where b are typical bias terms.

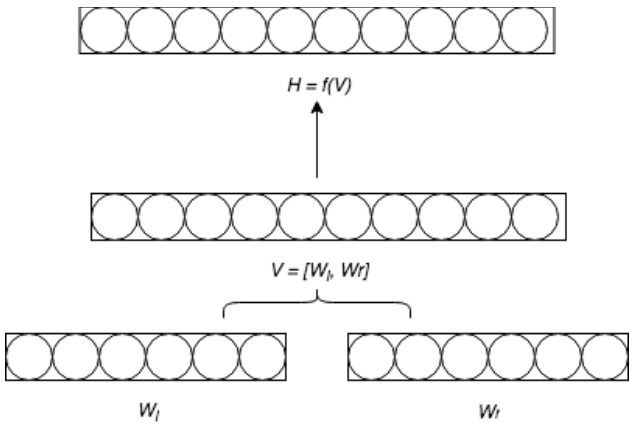

Figure 2. Example composition function using a Feed Forward Neural Network

In the instance of the recurrent composition function, *H* is actually many different parameters corresponding with a Long Short Term Memory (Hochreiter, 1997), or LSTM, layer. Recurrent neural networks are a good fit for Natural Language Processing problems due to their ability to model long term dependencies, an integral requirement for much of NLP. LSTM networks particularly are good at modeling long term dependencies (Hochreiter, 1997), an ideal quality for this task. In an LSTM layer, there are what are referred to as "forget gates" which choose what information to keep and what information to throw away at each step of computation. Essentially, this allows the layer to keep a working memory that it can refer back to and write to. This working memory acts as a ledger that allows for the network to learn to refer to in order to capture longer term dependencies. Figure 3 demonstrates this style of composition function, where *H* is the hidden state, or the working memory. Note that *H* is updated at both time steps. *H* is also the final constituent representation.

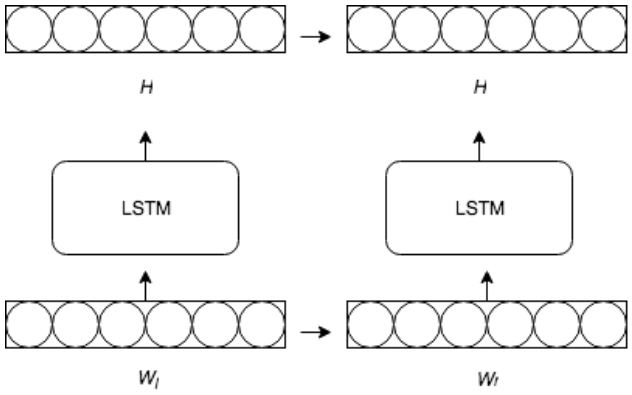

Figure 3. Example composition function using an LSTM Recurrent Neural Network

The loss for this model is calculated via Noise Contrastive Estimation (NCE) for each composition. The final constituent vector is used in the NCE task to distinguish between the true context word and noise vectors. Formally, each node is computed recursively through some

composition function f (either FFNN or RNN) into a node representation H. H is then used to compute the NCE loss for that given node. This loss is then summed with all the other node losses to get the full tree loss. The full tree loss is then back propagated using the standard Back Propagation algorithm, and the weights for the composition function and the word-vectors themselves are updated accordingly.

4.2 Experiments

One problem with evaluating tasks like semantic representation is that they are inherently non-quantitative; we cannot simply look at the loss and see how well the vectors are representing the semantics of a given phrase because there is no numerical way to say how a vector represents semantics. Because of this, other forms of experimentation are needed to examine how the algorithm at large is working.

The first experiment is a simple one: tuning proper hyper parameters and seeing how a FFNN and a RNN composition function compare to each other. The way to do this for something that is inherently non-quantitative, however, is to observe the results in the second and third experiments respectively. There are many possible hyper parameters to tune, and we do so using the standard cross validation technique. Some of the hyper parameters that require proper tuning are gradient clipping to prevent vanishing/exploding gradients, dimensionality of the word/ constituent vectors, number of negative samples for NCE, whether to initialize the word vectors from scratch or use retrained Word2Vec vectors initially, and finally whether to use the FFNN or the RNN composition function.

The second experiment is that of examining how word/constituent vectors cluster when projected down to 2 dimensional space. Mikolov et al., 2013a uses this method to evaluate how their vectors represent semantics in terms of vector space. To project their vectors, they use the standard Principal Components Analysis to factorize their embedding matrix into two dimensions. Those two dimensional vectors are then plotted and their relationship to one another is observed. For our experiment, we both project word vectors and constituents down into 2 dimensions and observe how they cluster respective to one another. We try to observe how negation is captured given that Socher et al., 2013 proved that ReNNs were very capable at modeling negational constructions. We also try to observe how adjectival/adverbial modifiers change the position of nouns/adjectives/verbs respectively.

The third experiment is seeing how well the word/constituent vectors perform on downstream tasks that require good semantic representation such as sentiment analysis. By observing how the word/constituent vectors perform on tasks such as this, we can observe how much semantic information is contained within the vectors, and specifically how much of it can be leveraged by a classifier. For our purposes, we use the already labeled Stanford Sentiment Treebank to use as a sentiment analysis task against Mikolov et al., 2013a, 2013b, & 2014, or Word2Vec and Doc2Vec respectively.

5 Evaluation

5.1 Hyper Parameter Tuning

The difficulty of direct hyper parameter tuning is that it doesn't necessarily reflect how well the model represents phrases semantically. Maximizing our loss function (NCE) does not have a one to one correlation with good semantic representation. Overfitting a dataset of our size (11,855 sentences) is a distinct possibility considering the powerful nature of our model, which would make the overall loss low but the semantic representations bad. The reason behind performing this sort of minimization in general is simply because we posit that there is *some* correlation between a low NCE loss and a good semantic representation, and by minimizing the NCE loss while being careful of overfitting will get us better semantic representation in the long run. In this way we can say that if one composition function results in a lower loss than another, then that composition function captures more semantic information.

The results of this experiment can be seen in Table 1 below. Predictably, utilizing pre-trained Word2Vec vectors to initialize the ReNN word vectors reduces overall loss across the board. The Recurrent composition function also is the only composition function that works with this architecture. This is due to the exploding gradient problem, in which a deep neural network's gradients are unstable, exploding out towards extremely large values and essentially snowball, causing unstable learning and massive weight updates during training. Recurrent Neural Networks were invented in part to mitigate this problem, and we see that their ability to prevent exploding/vanishing gradients is particularly useful for the ReNN's architecture, in which arbitrarily deep networks are a possibility. Specifically, however deep a parse tree is, that is how many "layers" of computation the network has, and that makes unstable gradients more likely for more layers.

Beacause of the exploding gradient problem, the only results we have are for a Recurrent composition function, and we had to clip the gradients at 0.1 for all of our models trained. This prevents the gradient from exploding, though it also prevents the network from learning as efficiently as possible. Another potential problem in our architecture is in the amount of time and memory this network takes to train. One full epoch over the 11,855 training examples takes (on average) roughly 8 hours to complete, making proper hyperparameter tuning exceedingly difficult. In addition, it indicates that this algorithm scales very poorly, and the advantages gained by using the ReNN may be offset by this.

Model	Average Loss
ReNN (+ pre-trained, + recurrent, 25 neg samples)	**0.299**
ReNN (- pre-trained, + recurrent, 25 neg samples)	5.95
ReNN (+ pre-trained, + recurrent, 5 neg samples)	2.84

Table 1. The results of hyperparameter tuning ReNNs

Interestingly, the number of negative samples has a highly significant effect on the overall loss of the network. In identical networks (i.e. both with recurrent composition functions and with pre-trained vector initialization), when we decrease the number of negative samples from 25 down to 5, the error increases by almost 850%. Similarly, by simply not initializing with pre-trained vectors, the error rate increases by about 1890%. All of this makes the ReNN with pre-trained vector initialization, a recurrent composition function, and 25 negative samples the clear best performing model.

5.2 Vector Clustering

Vector clustering, while inherently qualitative, is a straightforward method of evaluating semantic word vectors that relies on the assumption that good semantic representation will cluster vectors that represent words that are similar in meaning (e.g. “good” and “not bad”) closer than words that are different in meaning (e.g. “good” and “bad”). This is one of the main reasons to use this style of representation in the first place, considering that the traditional one-hot encoding of words does not capture this quality.

One-hot encoding is a method of representation that relies on word counts. For the *i*th word in the vocabulary, w_i, the vector representing that word will be a sparse vector of zeros where the *i*th index in that vector is a 1. For example, in a corpus of data that consists of the sentences: “the cat is black” and “the dog bites”, it will have a vocabulary $V = \{$*“the”, “cat”, “is”, “black”, “dog”, “bites”*$\}$. If we assign each word an index such that *“the”* is the 1st index, *“cat”* is the second, etc., the vector $v_{the} = [1, 0, 0, 0, 0, 0]$, the vector $v_{cat} = [0, 1, 0, 0, 0, 0]$, and so on. A problem with this style of representation is that the vector representing *“cat”* and the vector representing *“dog”* are just as far away as the vector representing *“cat”* and the vector

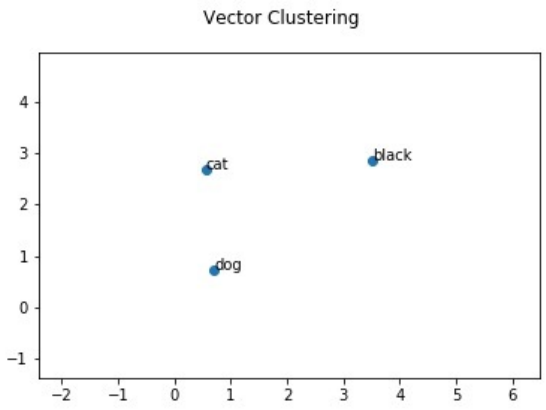

Figure 4. ReNN with a Recurrent composition function vectors projected into 2D space using Principal Components Analysis (PCA)

Figure 5. ReNN with Recurrent composition function vectors projected into 2D space using PCA

representing *"black",* despite being more semantically similar.

In this way, when similar words cluster we can reasonably assume the model has learned that they exist in similar distributional contexts, and are therefore semantically similar. In Figure 4 we can see that a Recursive Neural Net with a Recurrent composition function is able to cluster *"cat"* and *"dog"* closer together than *"black"*. Additionally, in Figure 5 we can see that this same ReNN with a Recurrent composition function can capture some negational elements, due to *"not bad"* and *"good"* being closer together than *"good"* and *"bad"*. The ability model negation in semantic representation is essential, as negation is notoriously difficult to properly represent. For example, in Figure 6 we see that Word2Vec trained on the same raw data as the ReNN (i.e. the same sentences only without the parse trees) fails to capture negation properly.

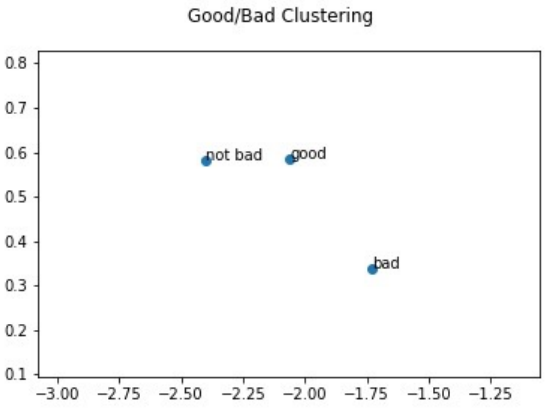

Figure 6. Word2Vec vectors projected into 2D space using PCA

In order to properly compare our algorithm with the current state of the art, we also project both FastText (Bojanowski et al., 2016) and Doc2Vec (Le et al., 2014) vectors, where both of these models are capable of modeling longer phrases. All alternative algorithms and their vectors were calculated using the open source toolkit Gensim[2]. As can be seen in Figures 7-8, neither model is as capable of learning negational elements from such a comparatively small

[2] https://radimrehurek.com/gensim/

dataset to those traditionally used for distributed representation training (Word2Vec used corpora with more than 1 billion tokens[3]).

As we can see, the ReNN is capable of modeling negation and basic semantic representation with comparatively very little data. This is a huge advantage given the ReNN's shortcomings; namely, that it needs binary parse trees in order to represent a phrase and that in

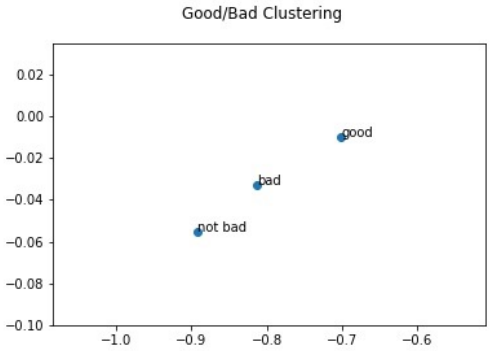

Fig 7. Doc2Vec vectors

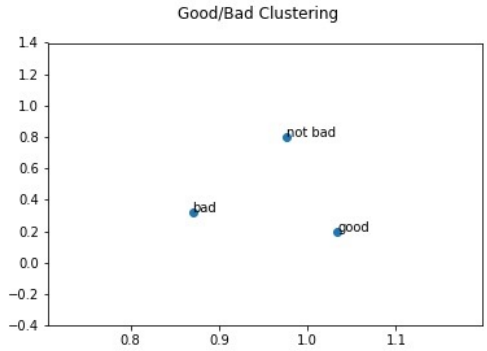

Fig 8. FastText vectors

comparison to the alternative models, the ReNN takes significantly more time to train. Despite these shortcomings, we can see that the results are encouraging.

5.3 Downstream Task

The final form of evaluation that we can perform for semantic vectors is evaluating their performance on some downstream task that relies on good semantic representation. For this purpose, we chose sentiment analysis using the Stanford Sentiment Treebank[4] (SSTB). The SSTB task breaks down into both a more fine-grained and a more simplistic task, we chose to do the fine-grained task. For the fine-grained task, each sentence is manually tagged as one of five possible sentiments: very negative, negative, neutral, positive, and very positive. In the simplistic

[3] https://code.google.com/archive/p/word2vec/

[4] https://nlp.stanford.edu/sentiment/treebank.html

task, there are only 3 possible sentiments: negative, neutral, and positive, where negative/positive is every phrase that is labeled either negative/positive or very negative/positive in the fine-grained task.

In order to do a proper comparison, we calculate all algorithms using only the SSTB as a corpus, and keep similar hyper parameters identical (i.e. number of negative samples, vector dimension, etc.). We compare several of our own models with Word2Vec (both mean and sum composition), Doc2Vec, and FastText, and the results are found in Table 2. Note that for the ReNN, the pre-trained parameter refers to initializing the word vectors with pre-trained Word2Vec vectors that are then fine-tuned in the process of training. Additionally, the number of negative samples that are used for NCE is reported, where 25 is the the number used for all alternative models. We only use the corresponding word/sentence vectors as features that are then fed into a Logistic Regression.

Model Type	Accuracy	F1
ReNN (+ pre-trained, + recurrent, 25 neg samples)	0.31	0.18
ReNN (- pre-trained, + recurrent, 25 neg samples)	0.27	0.22
ReNN (+ pre-trained, + recurrent, 5 neg samples)	0.26	0.12
Word2Vec (mean)	0.25	0.08
Word2Vec (sum)	0.25	0.08
Doc2Vec	0.3	0.23
FastText	**0.32**	**0.26**

Table 2. Results of sentiment analysis task on the SSTB

Notably, we can see that the results clash slightly with those from the hyperparameter tuning stage in that the ReNN with pre-trained vector initialization did not have the highest F1 score out of all the ReNNs. One explanation for this is that because the ReNN naturally models features extremely important for sentiment analysis (e.g. negation), starting from vectors not specifically predisposed to modeling those features can bring the overall representation further from the actual goal.

Another notable result is that all the ReNNs outperform all the Word2Vecs consistently in both accuracy and F1 score. This shows that the ReNN is capable of capturing more information per example than the more simplistic Word2Vec model. The best ReNN even comes within 0.01 F1 to Doc2Vec, which was the previous state of the art in sentiment analysis. The clear dominant model is FastText, though again not by a huge margin (0.04 F1). These results are encouraging, though many improvements still are necessary in order to get the ReNN to be a competitive model. Most notable is the time difference in training; every other model took at most about one minute to train completely on the dataset, whereas the ReNN took about 8 hours per epoch. The time advantage and performance advantage makes the choice of semantic representation model clear, though it is still too early to declare the ReNN unfeasible.

6 Conclusion & Future Work

In this work we presented a possible alternative model for word, phrase, and sentence level representation using the Recursive Neural Network. While there are many distinct advantages of using such a model such as taking syntax explicitly into account, modeling negational constructions with ease, and requiring comparatively little data for good results, the advantages

are offset by the difficulty and time intensiveness of the training process. While the results are encouraging, and there is plenty of room for additional research, the usage of ReNNs for semantic representation cannot be justified when alternative options that give better performing semantic representations in a fraction of the time can be accessed for free.

There is much room for improvement in the ReNN, such as finding a proper solution for out of vocabulary tokens. In its current state, the model replaces any out of vocabulary item as a vector of zeros. This solution works, however more clever and linguistically motivated solutions can be used. Bojanowski et al., 2016, for example, use sub-word features to allow for some morphological information to be captured. This allows the FastText model to infer an out of vocabulary item based on it's characters. This style of solution can be adapted to a ReNN in a similar fashion.

Another area of improvement is in speeding up the processing. Currently, the network only processes one training example at a time. However, if a method for batch processing can be applied then it could speed up the training process dramatically by parallelizing the computation required. The difficulty of this, of course, is that each training example is of variable size, and so parallelization is difficult.

If the inherent problems can be mitigated, then the ReNN can become a competitive and linguistically motivated alternative to the more popular distributed semantic representation models. Increasing the presence of formal Linguistics within Computational Linguistics is an important goal to strive for, as the communication between these two fields is essential for understanding natural language.

References

Salton, G. (1971). *The smart retrieval system: Experiments in automatic document processing.* Englewood Cliffs, NJ: Prentice-Hall.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137– 1155.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword infor- mation. *arXiv preprint arXiv:1607.04606*.

Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*.

Salton, G. (1971). The smart retrieval system experiments in automatic document processing. *Englewood Cliffs*.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 confer*ence on empirical methods in natural language processing, pages 1631–1642.

Gutmann, M. and Hyva¨rinen, A. (2010). contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artifi- cial Intelligence and Statistics*, pages 297–304.

Hochreiter, S. and Schmidhuber, J. (1997). Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473– 479.

Holmes, M., Gray, A., and Isbell, C. (2007). Fast svd for large-scale matrices. In *Workshop on Efficient Machine Learning at NIPS*, volume 58, pages 249– 252.

Klein, D. and Manning, C. D. (2003). Accurate un- lexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics.

Landauer, T. K. and Dumais, S. T. (1997). A solution to plato's problem: The latent semantic analysis the- ory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211.

Le, Q. and Mikolov, T. (2014). Distributed repre- sentations of sentences and documents. In *Inter- national Conference on Machine Learning*, pages 1188–1196.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.