Deep Learning for Semantic Similarity

Adrian Sanborn
Department of Computer Science
Stanford University
asanborn@stanford.edu

Jacek Skryzalin
Department of Mathematics
Stanford University
jskryzal@stanford.edu

Abstract

Evaluating the semantic similarity of two sentences is a task central to automated understanding of natural languages. We discuss the problem of semantic similarity and show that the use of recurrent and recursive neural networks can provide a 16% to 70% improvement over baseline models.

1 Introduction and Related Work

The fundamental challenge of natural language processing is to understand the meaning of a piece of text. However, judging whether a computer program "understands" a piece of text is an ambiguous task. We thus distill the abstract task of understanding text to the concrete problem of determining whether two pieces of text have very similar or distinct meanings. This is the challenge of "semantic similarity."

Although the problem of semantic similarity has a very simple statement, it has broad applications. This fact is a testament to both the importance and the difficulty of this problem.

Application: Automated Short-Answer Grading.

Grading short-answer questions on tests by hand is very time-consuming and expensive. Consequently, researchers have recently been exploring methods of automated essay grading (Dikli 2006). A functional semantic similarity evaluator could automatically grade short-answer questions by evaluating the similarity of a student answer with the corresponding correct answer.

Application: Machine Translation.

Although machine translation models have become very successful in recent years, it is not entirely clear how they should be evaluated. Possible evaluation metrics for machine translation include the Word Error Rate, the NIST score, the BLEU score, and the Translation Error Rate (Callison-Burch 2006, Vilar 2007). However, these schemes are ad hoc and provide only rough notions of what constitutes a "good translation." We propose that a semantic similarity system could evaluate a machine translator by measuring the similarity between the machine-produced translation and the gold standard.

Application: Image Captioning.

In the past few years, automated image captioning has become a much-studied area in computer vision and machine learning (Pan 2004). Many such systems are generative; the system takes as input an image and produces a number of possible outputs. A functional semantic similarity evaluator would be useful in the training of automated image captioning systems by providing a the system with a measure of goodness of the captions it produces.

Previous work.

The amount of research on semantic similarity has increased greatly in the past 5 years, partially driven by the annual SemEval competitions (Jurgens 2014). Results for the 2015 SemEval tasks were just published on June 5, 2015. Results and models of SemEval 2015 Task 1 for semantic similarity of twitter messages are described in (Xu et al. 2015). The highly successful ASOBEK system for semantic similarity (Eyecioglu and Keller, 2015) uses a SVM classifier with simple lexical word overlap and character n-grams features. The MITRE system (Zarrella et al., 2015) uses a recurrent neural network augmented with string matching features. Many other systems use a variety of supervised models using features such as n-gram overlap, word alignment, edit distance, cosine similarity of sentence embeddings. Additionally, the results of SemEval 2015 Task 2, which is the task our models are trained on, are described in (Agirre et al., 2015) with the best model achieving a mean Pearson correlation of 0.8015.

Our Approach.

In the last several years, training deep learning algorithms on large corpora of text has emerged as a general, powerful approach, performing as well as hand-designed algorithms based on many years of research and tuning. In this project, we use contemporary deep learning algorithms to determine the semantic similarity of two general pieces of text. Although we could build a better-performing system by training on a particular task (for example, image captioning), we instead seek to build a system which can evaluate the similarity of any two arbitrary sentences. The reason for this is two-fold. First, the lack of a very large publicly available dataset for any one specific task makes it difficult to use deep learning methods, which require very large datasets for training. More importantly, we feel that training for a particular task would not show the extent to which deep learning methods can truly evaluate semantic similarity of two sentences gathered from an arbitrary domain.

2 Technical Approach

The texts of the semantic similarity challenge are quite short, only consisting of individual sentences, and sometimes of very short sentences. Thus, any models trained only on the provided text are quite limited. To get around this limitation, we represent each word w by a vector $L[w]$. We construct these word vectors using GloVe, a weighted bilinear model which produces distributed word representations based on cooccurrence counts in a large corpus (Pennington et al., 2014). In particular we use 50- or 100- dimensional GloVe vectors pre-trained on Wikipedia and Gigaword 5 from `http://nlp.stanford.edu/projects/glove/`. We use pre-trained word vectors because the relatively small size of our dataset prevents us from learning word vectors directly.

We experiment with two deep learning models: recurrent neural networks (RNNs) (Mikolov 2011) and recursive neural networks (RNNs) (Socher 2011). Both neural networks take as input a structured set of words or tokens. Although each model treats its input differently, we train both models to report the probability that a given pair of sentences belongs to each similarity category.

Recurrent neural networks.

Recurrent neural networks are powerful deep learning models which take as input a sequence of tokens, each of which is used to update a hidden state. Concretely, for a sequence of works or tokens $(w_1, w_2, ..., w_n)$, we calculate

$$h_i = f(U \cdot h_{i-1} + W \cdot L[w_i] + b),$$

where U, W, and b are parameters leaned by the model. Here, f is a non-linear function, applied element-wise to the vector of its arguments. For each training example, we set h_0 to be the zero vector.

Given two sequences of words or tokens $(w_1, ..., w_n)$ and $(v_1, ..., v_m)$, we use the above equations to produce two hidden vectors $h_n^{(w)}$ and $h_m^{(v)}$. Finally, we produce a vector, the entries of which represent the probability that the sentence pair belongs in a corresponding similarity category:

$$p = \text{softmax}\left(W_s \cdot \left[\begin{array}{c} h_n^{(w)} \\ h_m^{(v)} \end{array} \right] + b_s \right),$$

where W_s and b_s are parameters learned by our model, and softmax is the function defined on k-dimensional vectors x by

$$[\text{softmax}(x)]_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}.$$

Finally, we train our model using the cross-entropy loss: given a vector of probabilities p for a training pair of sentences, if the correct similarity category corresponds to index i of p, our model will incur the loss $L = -\log p_i$.

We initialize the entries of U, W, and W_s using a random Gaussian in the interval $[-0.01, +0.01]$, and we initialize b and b_s to be the zero vector.

Recursive neural networks.

Recursive neural networks are deep learning models frequently used in applications of deep learning to natural language processing because, unlike recurrent neural networks, they can take advantage of known linguistic structure. Rather than working on a sequence of tokens, recursive neural networks take as input a (binary) tree. The leaves of this tree correspond to the words in a input sentence, and we infer the binary tree structure using a calculated parse of the sentence.

For each leaf with word w, we assign the GloVe vector $L[w]$. We then work up the tree, calculating for each node N with left child N_L and right child N_R a vector using the formula

$$h_N = f(W_L \cdot h_{N_L} + W_R \cdot h_{N_R} + b),$$

where W_L, W_R, and b are parameters leaned by the model. As with recurrent neural networks, f is a non-linear function, applied element-wise to the vector of its arguments.

Given two sentences S_1 and S_2, we use the above model to calculate two vectors h_{S_1} and h_{S_2}. The description of the rest of the model is entirely analogous to the recurrent neural network (with $h_n^{(w)}$ replaced by h_{S_1} and $h_m^{(v)}$ replaced by h_{S_2}).

3 Experiments

The Data.

To train and test our semantic similarity system, we will use data from the SemEval-2015 Task 2 (Agirre et al., 2015), which also includes all data from similar tasks in 2012, 2013, and 2014. This dataset provides pairs of sentences together with a semantic similarity score between 0 and 5. A score of 5 indicates that the two sentences mean exactly the same thing; a score of 0 indicates that the sentences are on completely different topics; and intermediate scores indicate varying amounts of relevance between the two sentences. Gold standard scores were computed as the average over 5 responses from Amazon Mechanical Turk per sentence pair.

The data from each year are split between several different groups, acquired from different sources. For example, the 2015 data fall into five groups:

1. Image captions from Flickr, as presented in (Rashtchian et al., 2010), (1499 sentence pairs).
2. News headlines from the RSS feed of the European Media Monitor, (1495 sentence pairs).
3. Student answers paired with correct reference answers from the BEETLE corpus (Dzikovska et al., 2010), (1491 sentence pairs).
4. Pairs of answers collected from Stack Exchange, (1942 sentence pairs).
5. Forum discussions about beliefs, from the DEFT Committed Belief Annotation dataset, (1904 sentence pairs).

Thus, the 2015 data comprises a total of 8331 sentence pairs. The 2014, 2013, and 2012 data comprise 3750, 1500, and 2234 sentence pairs respectively. For all tasks, we separate the data into train/dev/test sets comprising 70/15/15% of the data.

To generate the parse trees required for a recursive neural network, we train a CKY parser on the QuestionBank and Penn treebanks using pyStatParser. We then convert these parse trees to binary parse trees using simple heuristics (e.g., placement of punctuation).

Evaluation.

To score and evaluate our system, we first separate the similarity scores into 6 categories. We choose to evaluate our model using the macro F1 score. We choose this evaluation metric because it captures the desire for our model to classify each similarity category correctly. Models like logistic regression can achieve very high accuracies by learning to very accurately predict the two most common scores; however, a model that optimizes for a metric like accuracy will fail to capture the nuance captured by a range of similarity scores. Macro F1 score is calculated by averaging the F1 scores for each individual category:

$$F1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

where, if TP denotes the size of the true positive set, FP denotes the size of the false positive set, and FN denotes the size of the false negative set, precision and recall are defined by

$$\text{precision} = \frac{TP}{TP + FP} \qquad \text{and} \qquad \text{recall} = \frac{TP}{TP + FN}.$$

As the geometric mean of precision and recall, the F1 score penalizes heavily both false positive and false negatives.

Baseline models.

To establish a baseline performance, we train simple supervised models using three features: (1) the cosine similarity between the two sentences' bag-of-words vectors, (2) the cosine distance between the sentences' GloVe vectors (defined as the average of the word vectors for all words in the sentences), and (3) the Jaccard similarity between the sets of words in each sentence. We train a decision tree classifier, a logistic regression classifier, a nearest neighbors classifier (using the first nearest neighbors), a random forest classifier, and a support vector machine (using the one-vs-the-rest approach).

Neural network training.

Neural networks were trained using stochastic gradient descent with a batch size of 30. Gradient step sizes were governed by a learning rate hyperparameter and adapted according to training history using AdaGrad (Duchi et al., 2011). In this adaptive scheme, the learning rate of each parameter is reduced proportional to the distance the parameter has moved over the course of training.

We found that our neural network models heavily overfitted the training data. To combat this tendency, we added an L2 regularization term with regularization strength ρ chosen as a hyperparameter. Additionally, we tested applying dropout to the hidden layers. Dropout probabilities tested ranged from 5% to 40%.

The word vectors are a crucial input to the network. To explore the effect of word vector choice on performance, we trained networks based on pre-trained 50-dimensional and 100-dimensional word vectors. Furthermore, we tested back-propagating gradients into the word vectors, allowing task-specific information to affect individual word representations to better optimize performance. Network hidden layers were chosen to be the same size as the input word vectors.

Finally, the imbalanced distribution of training similarity categories was another factor that could affect training results. Indeed, the predictions of some baseline models were predominantly skewed towards the two most common categories (0 and 4). To account for this, we tested training networks from equalized data, whereby an equal number of training examples from each similarity category were randomly chosen to form each batch.

Neural networks were implemented in Python using Numpy and trained on a CPU cluster. Networks were trained for 300 epochs (around 3 hrs) to explore parameters and 1000 epochs (around 10 hrs) for final production runs.

We explored the performance of each model across a variety of learning rate and regularization values and report the highest-performing networks here.

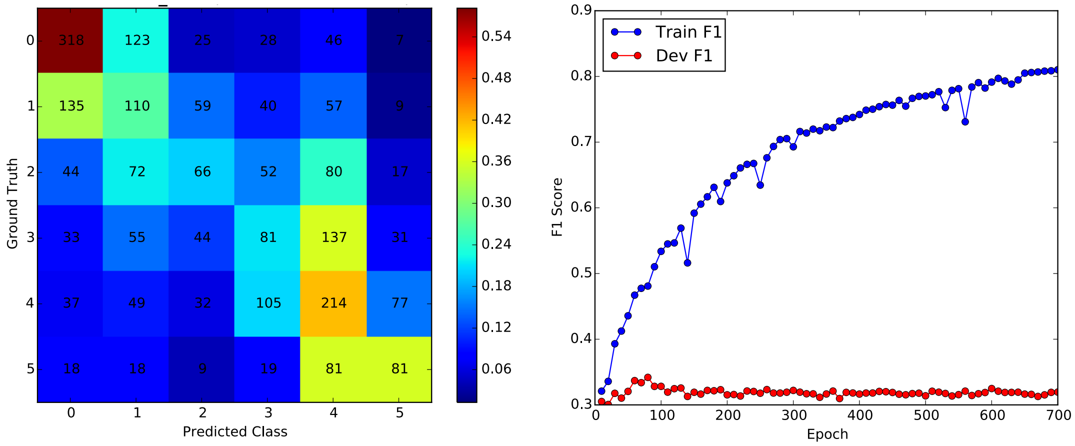
4 Results.

Results of our recursive and recurrent neural networks are listed in Table 1. Unless otherwise stated, each network was trained using 50-dimensional word vectors for 1000 epochs. Networks using 100-dimensional word vectors were trained for 700 epochs.

Model	Train F1	Test F1
Nearest neighbors	0.500	0.287
Logistic regression	0.192	0.199
Decision tree	0.997	0.261
Random forest	0.962	0.292
SVM	0.235	0.242
Recurrent NN	0.492	0.303
Recurrent NN, equalized	0.494	0.316
Recurrent NN, dropout 5	0.471	0.315
Recurrent NN, 100D, dropout 20	**0.812**	**0.338**
Recursive NN	0.520	0.305
Recursive NN, equalized	0.527	0.298
Recursive NN, dropout 5	0.509	0.302
Recursive NN, 100D, dropout 30	0.849	0.313

Table 1: Neural network performance.

The best performing model was a recurrent neural network with 100-dimensional word vectors and 100-dimensional hidden layers, trained with 20% dropout, learning rate of 0.01 and regularization constant of 0.01. This model achieved train/dev/test F1 scores of 0.812/0.342/0.338 respectively.



	0	1	2	3	4	5
0	318	123	25	28	46	7
1	135	110	59	40	57	9
2	44	72	66	52	80	17
3	33	55	44	81	137	31
4	37	49	32	105	214	77
5	18	18	9	19	81	81

Figure 1: Confusion matrix and training performance of the best model.

Surprisingly, providing training data in equalized schedules did not improve network performance. Increasing the regularization constant did decrease the amount of overfitting in the networks. For 50-dimensional networks, this also caused the network performance to drop overall. However, 100-dimensional networks (which have a higher capacity) trained with dropout tended to perform slightly better.

Model	Vector dim	dropout %	Train F1	Test F1
Recurrent	50	0	0.492	0.303
Recurrent	50	5	0.471	0.315
Recurrent	50	10	0.478	0.304
Recurrent	50	20	0.464	0.314
Recurrent	50	30	0.439	0.304
Recurrent	100	5	0.831	0.306
Recurrent	100	10	0.796	0.310
Recurrent	100	20	0.812	0.338
Recurrent	100	30	0.548	0.317
Recursive	50	0	0.520	0.305
Recursive	50	5	0.509	0.302
Recursive	50	10	0.529	0.295
Recursive	50	20	0.486	0.278
Recursive	50	30	0.481	0.295
Recursive	100	5	0.897	0.310
Recursive	100	10	0.898	0.302
Recursive	100	20	0.851	0.307
Recursive	100	30	0.849	0.313

Table 2: Effect of dropout on network performance.

Finally, back-propagating gradients into word vectors also significantly increased overfitting, shifting the train/dev F1 score after 300 epochs from 0.408/0.324 to 0.937/0.291.

5 Conclusion

Evaluating the semantic similarity of two sentences is a task central to automated understanding of natural languages. Here, we have explored the effectiveness of recurrent and recursive neural networks, combined with word vector representations, to predict the semantic similarity of sentence and phrase pairs. Our best model, a recurrent network with 100-dimensional word vectors and 20% dropout, outperforms all baselines by a substantial margin.

Results of the 2015 SemEval challenge were recently released, though the models were not described in great detail (Agirre et al., 2015). Entries to the SemEval challenge are evaluated by Pearson correlation between the predicted and the gold-standard semantic ratings. Our best model achieves a Pearson score of 0.560, while the best model of the SemEval challenge achieved an average Pearson score of 0.802 across all the tasks. However, there are two crucial differences between our approach and the SemEval challenge that make our task more challenging. First, we converted the task into one of classification, grouping all similarity scores into six categories. Second, models in the SemEval challenge were trained and tested on the different subtasks individually, whereas our models had the challenge of learning all tasks simultaneously. Therefore, direct comparison of the two scores may not be informative. Nonetheless, we still conclude that neural networks are an effective method for advancing computational understanding of semantic similarity.

References

Agirre, Eneko, et al. ”SemEval-2015 task 2: Semantic textual similarity, English, Spanish and pilot on interpretability.” Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), June. 2015. `http://alt.qcri.org/semeval2015/task2/`.

Callison-Burch, Chris, Miles Osborne, and Philipp Koehn. ”Re-evaluation the Role of Bleu in Machine Translation Research.” EACL. Vol. 6. 2006.

Dikli, Semire. ”An overview of automated scoring of essays.” The Journal of Technology, Learning and Assessment 5.1 (2006).

Duchi, John, Elad Hazan, and Yoram Singer. ”Adaptive subgradient methods for online learning and stochastic optimization.” Journal of Machine Learning Research, (2011).

Dzikovska, Myroslava O., et al. ”Intelligent tutoring with natural language support in the Beetle II system.” Sustaining TEL: From Innovation to Learning and Practice. Springer Berlin Heidelberg, 2010. 620-625.

Eyecioglu, A. and Keller, B. (2015). ”ASOBEK: Twitter paraphrase identification with simple overlap features and SVMs.” In Proceedings of SemEval.

Jurgens, David, Mohammad Taher Pilehvar, and Roberto Navigli. ”SemEval-2014 Task 3: Cross-level semantic similarity.” SemEval 2014 (2014): 17.

Mikolov, Tomas, et al. ”Extensions of recurrent neural network language model.” Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2011.

Pan, Jia-Yu, et al. ”Automatic image captioning.” Multimedia and Expo, 2004. ICME’04. 2004 IEEE International Conference on. Vol. 3. IEEE, 2004.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. ”Glove: Global vectors for word representation.” Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014) 12 (2014).

Rashtchian, Cyrus, et al. ”Collecting image annotations using Amazon’s Mechanical Turk.” Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon’s Mechanical Turk. Association for Computational Linguistics, 2010.

Socher, Richard, et al. ”Parsing natural scenes and natural language with recursive neural networks.” Proceedings of the 28th international conference on machine learning (ICML-11). 2011.

Vilar, David, et al. ”Human evaluation of machine translation through binary system comparisons.” Proceedings of the Second Workshop on Statistical Machine Translation. Association for Computational Linguistics, 2007.

Xu, Wei, Chris Callison-Burch, and William B. Dolan. ”SemEval-2015 Task 1: Paraphrase and semantic similarity in Twitter (PIT).” Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval). 2015.

Zarrella, G., Henderson, J., Merkhofer, E. M., and Strickhart, L. (2015). ”MITRE: Seven systems for semantic similarity in tweets.” In Proceedings of SemEval.