ONE-SHOT OF WINE: DEEP LEARNING ARCHITECTURES FOR DIFFERENT SETTINGS OF THE SENTENCE SIMILARITY PROBLEM

Category: Natural Language
Type: Application

Tom Knowles
Mathematical and Computational Science
Stanford University
tknowles@stanford.edu

Tatiana Wu
Mathematical and Computational Science
Stanford University
twu99@stanford.edu

ABSTRACT

This paper presents two main architectures for the general task of sentence similarity, which aim at tackling different settings of the following problem. Given two sentences, we wish to know whether they fall into the same labelled category—whether they're about the same thing, written by the same person, or have the same style or meaning. In *setting 1*, we say we can enumerate all the possible "categories" that a sentence may take on. This may be applicable in the context of intent recognition for a chat-bot, where only a limited number of options are available. In *setting 2*, we cannot enumerate all labels, and only know when two sentences have the same label; this is strictly harder than setting 1, and applicable to one-shot learning, where we may have only a few labels of one category. We present architectures for both of these settings, based off recent research in word embeddings, LSTMs, and Siamese models; it's useful to present these two settings together like this, since our model for setting 2 is simply a modification/generalisation of our model for setting 1. We also compare the results of these models, and present analysis as to why setting 2 is so much harder than the setting 1.

1 Introduction

Sentence similarity is an increasingly important task in the field of natural language processing. The problem of calculating the semantic similarity between sentences has broad applications, from text categorisation, to chat-bot implementations, to information retrieval. For this project, we chose to model sentence similarity using wine reviews, largely due to the accessibility of a large, high-quality dataset of textual wine reviews on Kaggle—though we focus more on the overall task of sentence similarity than the specific dataset used here. The task is as follows:

Given two textual sentences, which are categorically labelled, return a prediction of whether the two sentences fall into the same category.

For example, we may wish to tell whether two sentences have the same meaning, or are talking about the same object. We note that there are (at least) two settings of this problem: in *setting 1*, we can enumerate all possible labels that a sentences may take on (such as the user intentions in a chat-bot) and have training data associated with all these labels; in *setting 2*, we cannot enumerate all labels (such as general sentence meaning), and may have few or no sentences within some category. Setting 1 is closer to a traditional classification task, whereas the second falls under one-shot learning. We'll consider both settings of the task for this specific problem:

Given two wine reviews, we return a prediction of whether the wines being reviewed have the same "type" or "category." Examples of type in our dataset include "Pinot Noir," "Rose," and "Cabernet Sauvignon."

In this specific context, we see that calculating semantic similarity could play an important role in making recommendations for customers, based off their personalised preferences. The wine industry is a multi-billion-dollar industry, and every wine-store can provide recommendations, usually based off an in-person verbal or textual description. We might hope to automate and improve that personalised recommendation process —(and even provide it online)— by comparing a verbal description of what a customer wants, to real reviews and descriptions given by other customers who've tried the wine. In this way, we can find the

closest wine to what they want, and significantly improve the online shopping experience. This idea can be applied to any luxury online shopping business.

More generally, however, we hope that the architectures we use to solve this specific problem can be generalised to help solve the "sentence similarity" problem overall, and try to be as data-agnostic as possible.

2 Related Work

The general problem of text classification is increasingly being handled through a combination of word embeddings to process words; LSTMs to process sentences; and deep neural networks for classification, often with convolution and attention mechanisms. Word embeddings are essentially metric embeddings of words within a vector space, and the most popular embeddings are word2vec [1] and GloVE [2]. LSTMs are recurrent neural networks, that were introduced in 1997 by Hochreiter and Schmidhuber [3], and can handle memory over longer sequences (like sentences), effectively dealing with the "vanishing gradients" problem. Attention mechanisms have also been increasingly popular [4], with the intuition that they allow focus to be directed on certain hidden states of an LSTM, rather than just relying on the last.

An archetypal example of word2vec/LSTMs/CNNs is Zhou et al [5], which uses a combination of all three to classify textual data. We use all of these in our model for setting 1.

It has been unclear until recently whether these concepts can be applied successfully to setting 2 of the sentence similarity problem, however. Excellent progress towards this was made in one recent paper by Mueller [6] which we use extensively in our final implementation. It uses a Siamese LSTM—with two branches trained on the same model. We build this architecture, and a modification of it based largely on Chi et al [7].

This is not the only modern approach to the problem: Yang et al [8] present a largely unsupervised approach to the problem of identifying conversational input-response pairs, using deep averaging networks to encode sentences and responses into fixed-length vectors u and v, then take the dot-product to calculate a score.

3 Dataset and Features

We used the Wine Reviews dataset from Kaggle [9], which consists of over 130,000 reviews. We limited our project to the 50 most common wine varieties, which gives approximately 99,000 reviews. We also cleaned the data, removing duplicates, and those with invalid reviews.

We split off 20% of the data into a separate test set for both parts of the project. For the Siamese model, on both the training and test sets, we randomly sampled (with replacement) pairs of wine reviews—200,000 pairs for training and 100,000 pairs for testing.

After pre-processing, we converted every word into the dataset into a "word embeddings." We tested several different approaches to this, with notable differences:

1. 300-dimensional word2vec embeddings pre-trained on a large textual corpus, [1]
2. 300-dimensional GLoVe embeddings pre-trained on a large textual corpus, [2]
3. 100 to 300-dimensional word2vec embeddings trained on our data with a continuous bag of words model.

We present a brief description of how word2vec embeddings are produced: we train a shallow neural net with one layer, to predict a word from its surrounding context. For a vocabulary size V, we use V-dimensional one-hot embeddings to represent the words. From training the neural network shown in Figure 1—where we essentially get the intermediate vector by averaging the vectors in the context—we get a matrix $\mathbf{W} \in \mathbb{R}^{V \times N}$, such that the i^{th} row of $\mathbf{W}$ corresponds to a vector embedding representing the i^{th} word in the one-hot embedding. By nature of the neural network training, words with similar contexts have vectors closer to each other, making $\mathbf{W}$ a metric embedding based off similarity.

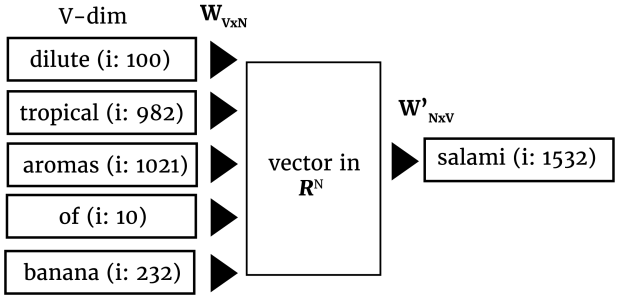

Figure 1: Training word2vec with a continuous bag of words model, and a context-size of 5

4 Methods and Experiments

4.1 Setting 1:

For setting 1, we can enumerate all categories, so we build a predictor over categories, and return that a pair is the same if the predicted category is the same. To do this, we train the neural network architecture in Figure 2. We now describe each layer of this.

4.1.1 Embedding Layer

This layer simply converts each word in a sentence into a vector, using precisely the matrix $\mathbf{W}$ described above.

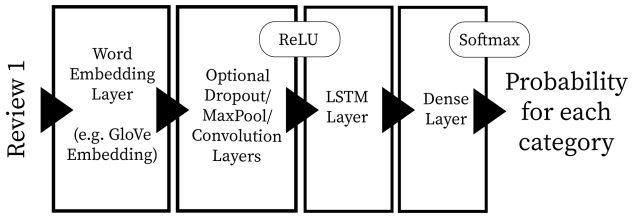

Figure 2: Recurrent network architecture for setting 1, to classify reviews by category

4.1.2 Dropout/Convolution/Pooling

These layers are intended to improve generalization, and adding them to the network did significantly decrease overfitting and improve the generalization.

Dropout: This involves randomly dropping out neurons during training, forcing other neurons to generalize without having that neuron present. It is mathematically fairly easy to show this is equivalent to regularization, with some assumptions. We used a drop–out factor of 0.2.

Convolution: A convolution on input vector v, with a kernel k of size m is given by

$$(v * k)(i) = \sum_{j=1}^{m} k_n \cdot v_{i-j+m/2}.$$

We may also apply an activation function (we used ReLU). This is equivalent to sliding a "summing window" over the input vector to produce a new output vector, to decrease the dimension and improve generalization. We used 1D convolution with $m = 5$ and output dimension of 128.

Pooling:

This is simply choosing the largest value from every consecutive "window" in the input vector of size k, so we ignore small values. We tested $k = 2$ to $k = 4$, with little difference.

4.1.3 LSTM

LSTM is a recurrent network with a memory cell, and a hidden state. It sequentially updates the hidden state, based on the memory cell and the value of the previous hidden state. Intuitively, this has the effect of "reading" a sentence one word at a time, also taking into account the results of previous readings and memory. The memory cell contains a memory state c, an output gate o, as well as an input gate i and a forget gate f. The updates are parameterised by eight weight matrices associated with these (four W, and four U) as well as biases. These are as follows:

1. $i_t = \text{sigmoid}\left(W_i x_t + U_i h_{t-1} + b_i\right)$
2. $f_t = \text{sigmoid}\left(W_f x_t + U_f h_{t-1} + b_f\right)$
3. $\tilde{c}_t = \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right)$
4. $c_t = i_t \odot \widetilde{c}_t + f_t \odot c_{t-1}$
5. $o_t = \text{sigmoid}\left(W_o x_t + U_o h_{t-1} + b_o\right)$
6. $h_t = o_t \odot \tanh\left(c_t\right)$

This gives a recurrent neural network with an excellent long-term memory, excellent for modelling sequential data like sentences. We varied the number of hidden units in our LSTM, but ended up using 132. We also add a dropout and recurrent dropout of 0.2 to improve generalization [10].

4.1.4 Dense

This is exactly the fully-connected neural networks described in lecture. We use softmax to output a probability for each category.

4.1.5 Prediction of pair similarity

The model in Figure 2 (trained with Keras/Tensorflow [11] [12]) is fairly successful at categorizing individual sentences into categories. We can either take the dot product $u^\top v$ of two output vectors u, v to get the probability of these vectors being the same category, or take the most likely category prediction, and return that pair is the similar if the most likely categories are the same. In practice, this had negligible effect on our accuracy, so we chose to use the simpler latter approach. (This has the benefit of making the multi-class classification accuracy of the model in Figure 2 very close to the F1 score over all possible pairs in the test data, so it makes intuitive comparison of the models easier.)

4.2 Alternate Models/Baselines

We train two simple baselines using the feature vectors, by simply averaging the vector embeddings for each word in a sentence to get a sentence vector (the "bag of words" model), and then training SVM/logistic regression models on these vectors. These were studied in the class, so we exclude the equations.

4.3 Setting 2:

Having built the neural network above, we hope to generalize it to the case of setting 2, where we cannot enumerate the categories and must simply pairs. We do this using a Siamese LSTM model best documented by Mueller [6]. We notice that the first three layers of Figure 2 provide a good framework for sentence understanding and meaning, even without the final categorisation. We therefore build two copies of these layers, one for each sentence; we will train each of these "branches" with the same weights, mak-

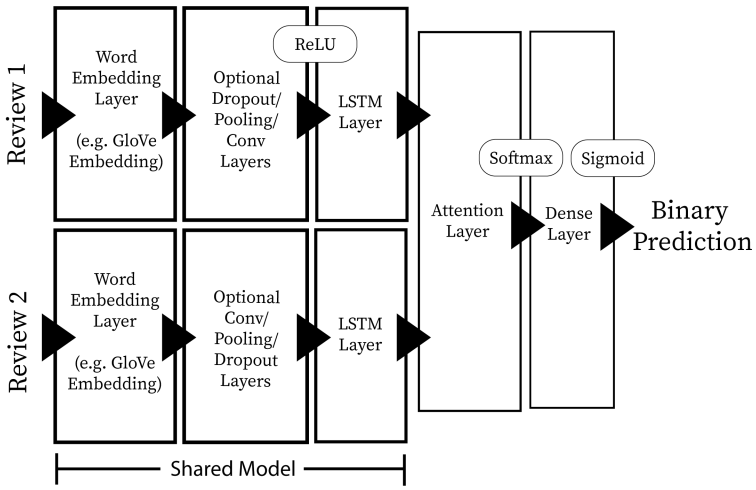

Figure 3: Recurrent network architecture for setting 1, to classify reviews by category

ing this a Siamese model. We then add an attention and dense layer for prediction, as shown in Figure 3.

4.3.1 Attention Layer

In updating the LSTM's hidden state h_i, we get representations of the hidden state at each time-step. In the previous model, we use only the last output h_T, but we can instead use all of them. Based off Yang et al.'s work [13], we introduce a "context vector" u_w, which is representative of what is an "informative word." The attention layer takes all of the hidden states associated with the LSTM layer, then scores it against u_w, normalizing with a softmax. It then sums over to get a sentence vector s:

1. $u_{it} = \tanh\left(W_w h_{it} + b_w\right)$
2. $\alpha_{it} = \frac{\exp\left(u_{it}^\top u_w\right)}{\sum_t \exp\left(u_{it}^\top u_w\right)}$
3. $s_i = \sum_t \alpha_{it} h_{it}$

Other than the attention layer, we chose similar hyper-parameters for the various layers to the previous model.

4.3.2 Alternate Manhattan Distance Model

We also build a more simple model using the Manhattan distance between the output of the LSTM layers to classify whether the pairs are the same. This is similar to the original model presented by Mueller, and identical to the model in 3, but with the last two layers changed to the Manhattan distance [6].

5 Results and Discussion

We trained the neural models for 20-50 epochs, stopping early when possible, and all other models until convergence. We then tested the resulting models on a separate test set. We summarize our results in two tables, both showing the F1 scores of the various models at classifying whether pairs are the same. (The accuracy is always above 95%, due to there being more pairs that are different than the same, so the F1 score is more useful.) The F1 score is the harmonic mean of the precision and recall:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

We note that the baseline F1 score is around 10% from predicting randomly, for our particular dataset.

Table 1 also contains the label ranking average precision score for the label classifier used in setting 1—(i.e. how accurately we classify elements before we even consider pairs)—since this metric is useful in understanding how the model works.

We note that the results for setting 1 are significantly better than those for setting 2. This is to be expected, since we claim setting 2 is much harder than setting 1, and we have strictly less information. There were a number of other more practical reasons for this, however. First and foremost was the very high training times for the model in Figure 3. This is partly because this model had many more parameters to train, but the loss also decreased at a much slower rate per epoch than it did for the model in 2, suggesting this problem is generally harder to fit. With our comparatively low compute power, and trying to standardise the epochs across the two settings, it proved fairly hard to train models for Setting 2 to a reasonable extent, given the time constraints. As Table 2 shows, they do not fully fit the training data—(by comparison, we were able to significantly over-fit the training data with neural networks in setting 1 with more training, with near 100% training accuracy in some cases.)

Moreover, there were significantly more hyper-parameters to choose from for the second model. Doing a grid search over these parameters would have taken an implausible amount of time, given the extreme training times. This was more plausible for setting 1, which we implemented

		Logistic Regression (Bag of Words)	SVM (Bag of Words)	LSTM Conv. NN
GLoVe (pre-trained)	*LRAP of single classification*	55.2%	49.8%	75.0%
	F1 Score of pair classification	39.0%	29.4%	66.4%
Word2Vec (trained on data)	*LRAP of single classification*	65.8%	64.0%	79.6%
	F1 Score of pair classification	50.0%	47.2%	70.0%

Table 1: F1 scores of models on the test data for setting 1. All classify sentences with a label, then compares labels for pairs

		Manhattan Distance	Neural Net Without Convolution	Neural Net With Convolution
GLoVe	*Train*	–	–	65.0%
	Test	–	–	26.6%
Word2Vec	*Train*	40.1 %	75.1%	68.8%
	Test	23.2%	26.7%	30.8%

Table 2: F1 scores of models for setting 2. These models all directly take in two pairs, and return a binary prediction

first. We are therefore unsure if our hyperparameters were chosen effectively.

There are more fundamental reasons for the differences in difficulty between these two tasks, however. First is the fact that pairwise training, as required if we only have pairwise comparison data, is far less efficient than training on individual data, even if we sample more pairs than we had single elements. Training on every possible pair would take a factor of $O(n)$ more time than it takes to train the network in Figure 2 on every single element of training data.

Secondly, the "discrete nature" of the sentence categories in models for Setting 1 seems to simplify learning good sentence representation. The sentence vectors produced by the LSTM layer of these models have higher dimension than the number of categories, so learning a good representation of the 50 categories is comparatively not as difficult. By comparison, the other model has to learn good sentence representation without any information about this "intrinsic lower dimensionality" [14] behind the data. We attempted to test this empirically by taking the best models for setting 1 and setting 2 (with some modification to make them more comparable), and measuring the L2 distances between a small sample of the sentence vectors produced by the LSTM layers. After normalizing these vectors to have the same average distance between vectors (of size 1) across the models, we found that the *average distance between vectors in the same category* for the "setting 1 model" was around 0.48x that distance for the "setting 2 model," while *the average distance between vectors in different categories* was slightly lower in the setting 2 model. This analysis (which we are aware is very unrigorous) likely indicates the LSTM layer for the setting 1 model is learning the underlying "discrete categorical" nature of the sentence space far better than the setting 2 model.

6 Future Work

There are a few obvious improvements to make: we'd try more hyperparameters for Setting 2, as well as a wider variety of different attention layer models. We'd also train these models for a lot longer, on GPUs, since we were unable to significantly overfit the training data.

More subtly, having noticed that the "discretization" of the training data is very useful to the Setting 1 model, we also believe that a model for Setting 2 that attempts to estimate to "discretize" the training data may have better results. First, we'd use unsupervised learning techniques to cluster the sentences, based off the pairwise comparisons we have in the data. We'd then use these clusters to assign discrete categories to each sentence in the training data. We could then train a Setting 1 model on the training data with these new labels. This would effectively convert the problem from Setting 2 to Setting 1, which with some iteration, may give reasonable results.

Lastly, we'd like to move away from wine reviews as a model of sentence similarity; we realize that reviews about the same wine are frequently not similar sentences. There

are many other available datasets that may be better for this task, including several released by the Stanford NLP group.

7 Code

The project code can be found at `https://github.com/tatianawu/cs229-project`.

8 Contributions

We collaborated on all parts of the project. Tom wrote most of the code for setting 1 and Tatiana wrote most of the code for setting 2.

References

[1] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

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

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[5] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.

[6] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[7] Ziming Chi and Bingyan Zhang. A sentence similarity estimation method based on improved siamese network. *Journal of Intelligent Learning Systems and Applications*, 10(04):121, 2018.

[8] Yinfei Yang, Steve Yuan, Daniel Cer, Sheng-yi Kong, Noah Constant, Petr Pilar, Heming Ge, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Learning semantic textual similarity from conversations. *arXiv preprint arXiv:1804.07754*, 2018.

[9] Kaggle wine reviews. `https://www.kaggle.com/zynicide/wine-reviews`. Accessed: April 2019.

[10] Alex Graves. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pages 5–13. Springer, 2012.

[11] François Chollet et al. Keras. `https://keras.io`, 2015.

[12] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[13] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[14] Elizaveta Levina and Peter J Bickel. Maximum likelihood estimation of intrinsic dimension. In *Advances in neural information processing systems*, pages 777–784, 2005.