A Low-Dimension Stacked Model for Paraphrase Recognition

Matthew Mauer
The University of Chicago / IL, USA
mmauer@uchicago.edu

Abstract

Paraphrase detection is a classical and ongoing challenge in Natural Language Inference. This paper combines several of the tried and tested methods for paraphrase detection into a few different lightweight models. Methods include Siamese LSTM, Naive Bayes, model stacking, TF-IDF, cosine similarity, character n-grams Jaccard, and other text mining techniques. The best accuracy on the easier test data was 0.899 and for the harder test data 0.615 — achieved with two different model architectures and training regimens.

1 Introduction

Here we lay out the underlying challenges of paraphrase detection that motivate the following modeling, feature engineering, and architecture design decisions.

First, it must be acknowledged that there isn't always complete consensus on whether two sentences are a paraphrase or not. Sense and semantic ambiguity can both lead to multiple interpretations of the same sentence. Furthermore, spelling and grammatical errors may cause a sentence pair to seem as though they are not paraphrases while the writer in fact intended them to be. There is then, necessarily, going to be noise in any model testing, and therefore this paper takes the view that the most generalizable model is the most desirable model.

Sentence similarity can be captured in several ways: The overlap in types that exist in either sentence, with varying importance depending on how rare the individual types are globally. The overall structure of the sentence as can be represented in the sequence or dependency of the tokens on one another. The interchangability of tokens or groups of tokens — eg synonyms. The logic or underlying meaning of the two sentences, which can be changed by a single negation. And the precision of certain nouns, especially people, places, and numbers — one change in a numerical digit can mean the difference between whether or not a pair are paraphrases.

We next describe the data provided for the challenge and go on to describe what tools are used to address the challenges and capture the types of similarity or discrepancy described above.

2 Data

Three general datasets were used for this task, two with dev and devtest partitions, and one larger file with only paraphrase pairs. For model training then, either the dev and devtest sets containing both positive and negative pairs were used for training, or the large set of paraphrases was used and negative samples had to be constructed. The exact methods used to address this are laid out in Training.

Here we discuss the difference between the two dev/devtest sets: One labeled the "easy" set and the other the "hard" set.

The easy set contained clear paraphrase pairs, clear negative samples, and several almost paraphrase pairs. In the easy set, the almost pairs each differed in one of several ways: The presence of additional negations. An exchange of one or two tokens that had special importance for the meaning of the sentence (subject nouns, verbs, descriptive numbers, etc).

The hard set contained pairs that are either paraphrases or just barely not paraphrases. Frequently, the negative pairs in the hard dataset differed by the swapping of nouns, so either the location or time of an event changed between sentences or the subject and object of a phrase changed. These structural differences are more difficult to capture.

Due to this structural semantic difference in negative pairs between the easy and hard datasets, different model architectures and features were optimal for the two. While more generalizable models trained and tested on the easy dataset were also

tested with the hard dataset, a few other models were constructed exclusively to address the challenges of the hard dataset.

3 Models and Neural Methods

A few non-neural models were trained and tested exclusively on manual features. These included logistic regression, Random Forests, and Naive Bayes with a Gaussian distribution. These three models were chosen for their differences to capture a variety of broader techniques. This tested whether linear, non-parametric, and probabilistic approaches to the same classifications problem with the same inputs.

The workhorse model of my approach was the Siamese Long Short-Term Memory (SiamLSTM) network — a common technique for the task.[1]. This involved each sentence being converted to a sequence of word embeddings that passed through the same LSTM. The final hidden vectors of each sentence were then joined in some way (several methods were tested and are explained in Architecture) and then passed onto a feedforward Multilayer Perceptron (MLP) which output two values. No gating was used in the SiamLSTM.

The initial hidden and memory cell vectors were initialized at $U(-1, 1)$ and included as parameters in the training of the model. The embeddings for this model came from Stanford NLP[2] pre-trained with GloVe on English Gigaword Fifth Edition and Wikipedia 2014 data. The pre-trained embeddings had 300 dimensions, 6 billion tokens, 400 thousand distinct embeddings in the case-insensitive vocabulary. These embeddings remained frozen and were not tuned in anyway during or before training.

4 Feature Engineering

Four manual features were selected for my models. These were by no means comprehensive but were intended to capture a wide variety of discrepancies in sentence meaning.

The first was TF-IDF cosine similarity. This feature combined a few techniques. A bag-of-words (BOW) TF-IDF vectorizer[3] was trained on the first 100 thousand samples of the raw, unlabeled training pairs, flattened to treat each sentence individually. The cosine similarity was then

$$similarity = \frac{TFIDF_1 \bullet TFIDF_2}{||TFIDF_1|| * ||TFIDF_2||}$$

used so that co-occurring, rare types gave higher scores. This way, sentences that contained the same tokens for proper nouns or domain-specific verbs and adjectives scored as more similar.

The second was the Jaccard score using 5-character-grams. This was used to supplement the TF-IDF similarity by providing information on token adjacency. Five was picked for the n-gram size as it was deemed small enough to not challenge computational resources while still capturing enough information on token adjacendy. The Jaccard score of the the pair was the the n-gram set overlap over the the union

$$Jaccard = \frac{A \cap B}{A \cup B}$$

Where A and B are the two sets of 5-character-grams.

The third feature was the absolute difference in the count of tokens that occurred in the two sentences that were part of a artificial predetermined set of negation types (e.g. "no", "not", "but", "'nt" etc) for more details on what was included in this set, see the documentation. The difference was scaled down a factor of 10 to preemptively regularize for models that could be skewed by inputs on various scales. An example: "I will not go home" and "I will not live in a home that isn't well lit" would have a score of $|2 - 1|/10 = 0.1$.

The fourth and final feature was the number of numerical discrepancies, or mathematically the length of the disjoint set of tokens containing valid numerical digits

$$\frac{|(A \cup B) - (A \cap B)|}{10}$$

also scaled down by 10.

5 Neural Architecture

We first delineate here the elements of the neural architecture that remained static between models and then go over those elements that were experimented on.

All SiamLSTMs were followed by a MLP of dimension $(D + M) \times 128$ where $D = J * 128$ was the dimension of the join of the SiamLSTM outputs, each 128 dimensions and $1 \leq J \leq 4$, and

[1] Bao et al 2018 and Mueller and Thyagarajan 2015

[2] (Pennington et al., 2014)

[3] scikit-learn

M was the number of manual features. The sigmoid activation function was applied on the output of this layer.

The next and final layer was then dimensions 128×2, outputting the two numbers for argmax classification or, during training, the Negative Log Likelihood Loss (NLLL, i.e. Cross Entropy Loss) .

The primary element of this neural architecture that was experimented on was the join operation of the two sides of the SiamLSTM (and therefore the first dimension of the first MLP). The joins operations attempted were: concatenation of the two vectors $J = 2$, the absolute difference of the $J = 1$, the element-wise product $J = 1$, and the concatenation of all three $J = 4$.

Also experimented on was whether manual features were included — always at the first MLP.

6 Stacking

One model involved stacking the neural network described above with another non-neural model. That is, a trained neural network matching the architecture described above (but always without manual features) passed the two outputs two another model that used these as features in concert with the four manual features previously described to then classify the sentence pair. The only output model used in this stacked approach was the Gaussian Naive Bayes, as it was deemed the best non-neural model in prior experiments.

In fact, the most generalizable and best performing model on the easy data used this stacked approach.

7 Training

Almost all training was done on the large set of raw paraphrase pairs, and therefore required the generation of negative samples. To achieve the most generalizable model, I took somewhat nuanced with a few steps of randomization re-sampling. During training, every other pairs was converted to a negative sample with the following approach.

With probability 0.5, the second of the sentences in the pair was replaced by another sentence from the select uniformly at random from the entire training set.

With a probability 0.5, tokens were swapped for other tokens that were themselves selected uniformly at random from the vocabulary of the word embedding data. The number of tokens to swapped was a number selected from $U(1, S)$ where $S = N * L$. N here is the number of the tokens in the sentence, and L, the "learning noise," was a tuneable parameter (either 0.05 or 0.1 in experimentation).

Stochastic Gradient Descent and a learning rate of 0.01 was use for all neural training. For any neural network that used manual features at the MLP level, a weight decay (or L2 regularization)[4] of $1e^{-4}$ was used to avoid the model overfitting to the manual features at the expense of learning the parameters of the SiamLSTM. All batches were single sample.

Early stopping was performed by checking accuracy on a DEV set every 50 thousand samples and halting training if the performance remained constant for three iterations or declined for two.

8 Experimentation and Results

The decision points that were tested most robustly were as follows:

The non-neural models were tested with the four manual features. Gaussian Naive Bayes outperformed all other options by about 0.04 in accuracy, achieving the highest accuracy on the dev data of 0.74.

The join operation was tested in the neural architecture after the SiamLSTM. In the two-vector concatenation and the complete four-vector concatenation, $concatenate(v1, v2, |v1-v2|, v1*v2)$, the model overfit the training data.

For the easy data, the absolute difference performed best: The baseline neural network with this join achieved an accuracy on the easy Kaggle test of 0.865. However it didn't perform as well on the hard data. For the hard data, the element-wise product performed best, but the difference between the latter two approaches for the hard set was marginal. All further experimentation for the easy set was performed with the absolute difference join and with the element-wise product join for the hard set.

The learning noise parameter was tuned. Performance appeared to improve marginally when it was changed from 0.1, the baseline, to 0.05. This was especially true for the hard dataset.

Manual features were then included — both for the absolute difference (AbsDiff) and the element-wise product (EProd) models. Interestingly, the inclusion of manual features reduced the performance of the AbsDiff model on the easy test by

[4] (Loshchilov and Hutter, 2019)

about 0.04 [5]. Alternatively, the manual features improved performance of the EProd model on the hard test.

Two final models were attempted, each designed and tuned for one of the two test datasets. They both were the highest performing model for their respective test set, and performed very low for the other set.

A new model EProd SiamLSTM without manual features was trained using a new method and training set. It was trained and cross-validated (for early stopping) on the hard DEV and DEVTEST sets. This training method no longer fabricated negative samples, as there already existed some in the set. While this new "training" set was far smaller (only 2,000), it far outperformed previous models for the hard data. It appears that the negative sample creation did not successfully replicate the kind of negative samples present in the hard data that consisted primarily of simple token swaps that changes the underlying meaning. The test accuracy on the Kaggle set was 0.615, but the accuracy on the easy DEV set was only 0.468.

The original AbsDiff SiamLSTM without manual features and with a learning noise of 0.05 was stacked beneath a Gaussian Naive Bayes the included the neural outputs and manual features. THis achieved a test accuracy on the easy Kaggle set of 0.899, but only poorly on the hard DEV data.

Results also displayed in Table 1.

9 Conclusions and Future Work

The two datasets represented very different challenges in paraphrase detection, and the Stacked model that performed best on the ease data, I consider to be the most useful and generalizable result. Solving the nuance of the semantic swapping present in the negative samples of the hard data would benefit from NER, dependency parsing, and possibly POS tagging prior to input in a Self-Attention LSTM. This more complex approach could potentially capture the change of meaning (or lack thereof) due to a swapping of tokens, but it requires more computational resources than are available to this researcher presently.

Some additional means of improvement to the current model include greater feature engineering, gated LSTM of GRU, mini-batch learning to reduce oscillation during learning, and a TF-IDF trained on a greater corpus.

[5] Rough estimate due to early stopping.

References

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

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

Model	Easy Test	Hard Test
AbsDiff SiamLSTM	0.865*	0.443*
EProd SiamLSTM	0.74	0.484
EProd SiamLSTM with Manual Features	0.69	0.43
Supervised Trained SiamLSTM	0.468	0.615*
Stacked Model	0.899*	—

Table 1: "—" indicates lack of test. Easy Test and Hard Test results are for the Kaggle score when possible (indicated with "*"), else the score on respective DEVTEST data.