Classify sentences that make sense: Applying MGNC-CNN to Textual Entailment

Xueguang Lu & his MacBook
Northeastern University
360 Huntington Ave
Boston, MA 02115
xueguang@ccs.neu.edu

Abstract

We document and report experiments training convolutional Neural networks (CNN) on the Stanford Natural Language Inference (SNLI) corpus. We show that by adapting Multi-Group Norm Constraint CNN (MGNC-CNN) model, hence simultaneously exploiting several pre-trained word embeddings, we achieve higher accuracy than using any of the word embeddings alone or combined using multi-channel model, which, on the contrary, typically yield better results in image processing. Although we did not achieve the accuracy showed in other papers of similar methods, we shall then discuss the logic behind this difference, and also describe a simple but highly accurate baseline model; this baseline model also exploits multiple word embeddings simultaneously. In this project report, we will start with discussing the literature that initiated and shaped my project followed by experiments we conducted along with results and implications.

1 Introduction

Textual entailment is a well-studied NLP task that has been applied many different models. It classifies sentence pairs that fall into three different kinds of relationships: entailment, contradiction, and neutral. For the corpus we have been studying, the SNLI corpus, the current state-of-the-art methods are carefully tuned LSTM-RNN models such as Syntactic-Tree LSTM (Chen et al. 2016) and Re-Read LSTM (Sha et al. 2016). These LSTM-RNN models pays special attention to distant words in the same sentence (or sentence pair), and it is proven effective in textual entailment to rely on certain keyword pairs that may appear more than dozen of words away from each other (Chen et al. 2016), making an n-gram model unsuitable in capturing this trait. The model we have been implementing and testing, MGNC-CNN, inherits the advantages on capturing global patterns from CNN family while utilizing several pre-trained word embeddings in a way that combines their advantages.

Word embeddings are vital in applying CNN model to textual entailment tasks because, in order to justify an entailment relationship, we would need to specify a logical deduction which always takes domain knowledge or background knowledge into account. Such knowledge, often not provided by the dataset itself, should be learned elsewhere and to be used as an input of our model; our experiment also shows that learning the word vectors on the fly creates massive over-fitting. Therefore, we utilize pre-trained vectors which have already converged on much larger corpora.

2 Preceding and Related Works

Our work is rooted from and influenced by Yoon Kim's CNN sentence classifier at New York University (Kim 2014). Kim compared different methods of using word embeddings:

1. Randomly initialize word vectors and modify them during training.
2. Use downloaded pre-trained word vectors (word2vec).
3. Combine two sets of word matrixes into a word tensor, and consider them ”channels”.

Kim's work shows that using pre-trained vectors (method 2) dramatically improve the accuracy of the model over training the words dynamically (method 1) on all the datasets. However, Kim also points out that multi-channel model (method 3) does not have a definite edge over one embedding (method 2); half of the time, in fact, using multi-channel model lowers the overall accuracy (see Kim 2014: Table 2).

Ye Zhang and Byron Wallace took Kim's idea further, they proposed a novel improvement over Kims model and named it Multi-Group Norm Constraint CNN (MGNC-CNN). The model filters, pools and penalizes norms word embeddings individually and then concatenate together to get a combined feature vector. Zhang and Wallace showed that by combining word embeddings this way, we are able to achieve higher accuracy than using individual embeddings, hence also superior to multi-channel model.

3 Experiment Setup

3.1 Dataset

The Stanford Natural Language Inference (SNLI) corpus contains 570,000 sentence pairs with labels entailment, contradiction and neutral. The corpus has an official train-test split, and the volume of each class is balanced.

3.2 Word Embeddings

I have tried the following pre-trained word embeddings.

- Google's original word2vec (W2V) by Mikolov at el.
- Global Vectors (GloVe) by Pennington at el.
- Dependency-Based word embedding (DBword) by Levy and Goldberg.

The above three word embeddings are widely accepted and used, yet they inherently differ in ways they were generated. While the first two embeddings context based (gram model and co-occurrences respectively), DBword is trained on dependency-parsed corpora using skip-gram model. Therefore, when combining these word embeddings, theoretically, we can not only retain the context information of the words, but the syntactic information as well.

4 Experiments

The entire implementation was written in Python. We wrote parsers for each of the word embeddings, as well as the corpus. The model was written using Googles TensorFlow library, which let us write the model in a declarative way and train the parameters efficiently. Because the SNLI is large in size, it takes more than a week to complete one training on a single machine; so, unfortunately, I only have the time to train it three times (twice on our own machine, one on school server).

4.1 Hyper-parameters

We used filter sizes of 3, 4 and 5 constantly throughout the experiment, and used 128 filters for each size, very similar to Zhang and Wallace's work. We originally planned to tuned the norm penalty threshold lambda when using multiple word embeddings over the range {1/3, 1, 9, 81} but because of time constraint (or lack of computational power), we did not start the tuning process and used (1,1,1) throughout. Finally, the drop out rate (probability of setting some feature to 0) is conventionally 1/2.

4.2 Baseline

Our baseline is by computing the sentence embedding using Arora et al.'s method and compute the euclidean distance (cosine) of the sentence pair(Arora et al. 2017). The sentence embedding model down weights frequent words in the vocab, hence highlight the infrequent word that often carries more semantic meaning in stead of syntactic.

4.3 Training

We First trained our own vectors by initializing a randomly distributed matrix and let the model modify the vectors on the fly, this approach turns out to be overfitting dramatically (see Table 1). We then used Googles word2vec and made it a constant of the model, at the end of the week, we saw a much better accuracy on the test dataset. At last, we added GloVe and DBword, we see a small decrease in train, and a small improvement on test.

Type of embeddings	Train	Test
Rand_init	**95.06**	72.95
word2vec	88.87	76.37
word2vec+GloVe+DBword	88.53	**77.92**
Baseline		71.19

Table 1: Training Performance

5 Problems Encountered

The first problem during implementation is how to deal with the sentence with difference lengths. At first it seems to be a problem without optimal solution, so we followed the conventional way:

padding the sentences to the length of longest sentence of the corpus and give the padding token a fixed vector.

We also find generating and indexing the matrixes/tensors quite overwhelming at first, we always forget which matrix is which (probably because of bad naming scheme); but it gets a lot easier when we found out that we should draw them out on paper and visualize them (which can be found in scratchpaper.pdf). It helps me (and maybe others) to pick up where we left of after a while.

During the training process, we found that due to high occurrences of out-of-vocab words, my accuracy was constantly below 60 even on the training set (out-of-vocab words are initialized to random vectors with same average variance as the downloaded vectors). After comparing our vectors with the downloaded matrix, we found that punctuation are not striped from the words, and our words are not converted into lower cases. Interestingly, I believe if we train the model just like that, the accuracy would still not be close to the cleaned result, so it is interesting to see that the model is quite robust probably because only a few words contributes to the entailment process.

Another problem with out-of-vocab words is that initially we found that we mapped the same word (at difference occurrences) to different random vectors when we try to generate a missing vector. And again, this flaw was barely noticed when executing the model at first, it was not until later when we try to add more embeddings that we found out this discrepancy.

6 Known Implementation Flaws and Future Works

As we mentioned before, the absolute next step is to tune over the lambda values for each word embedding. As mentioned by Zhang and Wallace, this process takes a embarrassingly long time to complete.

An unaddressed problem with the word embedding model is that I did not take phrases as a whole, instead, I treat each word as if they are independent of their immediate neighbors. For example, theme park was parsed as theme and park; whereas in some literature regarding word embeddings (such as word2vec), it is recommended to parse sentences into short phrases first (e.g. "theme-park"), and then apply the short phrase vectors. However, it create a potential difference of standards problem if we were to apply this technique to different implementations of word embeddings, since each implementation might segment phrases differently.

Another imminent improvement that can be done is to further breakdown the input matrix and separate the "ground truth" sentence from the "presumed entailment" sentence in the convolution layer and pooling layer(for which we do not have the time), instead of naively adding each pair together.

7 Conclusion

From experiment, we see that MGNC-CNN does have an small improvement over using only word2vec embedding. Although there are still untuned hyper-parameters, it is justifiable in some degree that this model out-performs or at lease same to using one word embedding.

References

Sha, Lei, et al. "Reading and Thinking: Re-read LSTM Unit for Textual Entailment Recognition."

Chen, Qian, et al. "Enhancing and combining sequential and tree lstm for natural language inference." *preprint* :1609.06038 (2016).

Frome, Andrea, et al. "Devise: A deep visual-semantic embedding model." *Advances in neural information processing systems*. 2013.

Levy, Omer, and Yoav Goldberg. "Dependency-Based Word Embeddings." *ACL (2)*. 2014.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." *EMNLP*. Vol. 14. 2014.

Kim, Yoon. "Convolutional neural networks for sentence classification." *preprint*:1408.5882 (2014).

Arora, Sanjeev, Yingyu Liang, and Tengyu Ma. "A simple but tough-to-beat baseline for sentence embeddings." *International Conference on Learning Representations*. To Appear. 2017.