M. Eng. Project: Exploring Relation-Extraction with Word Embeddings

Kevin Huang
Cornell University

Abstract

Word embeddings are very popular tools for natural language processing, due to how well this method captures information about words in relation to each other. Given to the power of word embeddings, we explore various different methods to best extract relational information from a given sentence. The methods that we explore range from simple averaging techniques, to more complicated recurrent and convolutional models, each of which is piped into a final softmax layer for the final classification. Surprisingly, in general the more complex models do not tend to outperform the simpler models.

1 Introduction

Relationship extraction is an area of natural language processing that tries to determine if there exists some relationship between two objects in a sentence. This has a wide range of uses, such as in the medical field, where medicine-disease relationships can be automated. The current state-of-the-art models are recurrent neural networks, generally with long short-term memory modules. It is thought that these recurrent models are the most suited for relation extraction due to architecture being able to remember information from different parts of the sentence.

However, a paper recently released under review for ICLR 2018 [1] shows that for many natural language processing tasks, simple models such as averaging and max-pooling have the same, if not better results as the more complicated models. Some preliminary work done in this topic also show that simple dimensionality reduction techniques such as PCA, can also work well. These results are surprising as some of the simpler models used do not take into consideration the ordering of the words, and only looks at the information from each individually. This may be due to more information being contained in a word embedding than is previously believed. If simpler word models are indeed just as good as more complex methods at predicting, then

This work aims to explore the effectiveness of such models, and see if we can find or develop a model that is comparable to current state-of-the-art. We aim to test models on a synthetic dataset generated by snorkel's data programming scheme. This dataset contains around 22000 sentences that are labeled as to whether or not each sentence contains a relation. A relation in this instance is defined as some sort of relationship between two of the nouns within the sentence. The main metrics that we want to use to judge how well our models work is the test accuracy of the given model, as well as the F1 score. Both of these metrics combined gives us a better picture as to what kinds of predictions the model is making and whether or not it is reliable.

2 Methods

We aim to benchmark a series of methods that are trained on our dataset in order to learn more about how well they do on our datasets. Given a sentence on a each of the methods that we use to translate information about a sentence into a vector form is then fed into a logistic regression / softmax layer for a final classification. We started off trying simple word embedding models, such as average and max-pooling. After this, we move on to single layer recurrent and convolutional models. Lastly we try a model that uses a PCA layer to compress information from a sentence.

2.1 Averaging Word Embeddings

The first and most basic simple word embedding model is by simply averaging all of the word embeddings in a given sentence and then feeding the output of these word embeddings to a logistic regression classifier. This is a baseline approach to see how well our other methods compare. For a sentence W_i, and the word embeddings in the sentence $W_i = (v_1, v_2..., v3$, with m being the dimensions of the word embedding.

$$W_i = \frac{1}{n} \sum_{i=0}^{n} v_i \quad \forall v \in R^m$$

2.2 Max-Pooling

The next method we try is max-pooling. This was also suggested by a previous paper, and can be seen as taking the most important features of each word embedding. Since word embeddings are not necessarily all in the positive domain, (such as typical max-pooling for convolutional neural networks), we would take the maximum value in the negative direction as well.

$$W_i = maxpool(v_1, v_2..., v_n)$$

2.3 Recurrent Layers

The recurrent model uses a single LSTM layer to learn the dependencies of the words in relation to each other in a sentence. We feed the words into the LSTM in order, and we use the output from the last word to feed into a softmax layer at the end. This last output would theoretically contain information about all of the words previous to this in the sentence, and would be enough to classify well.

2.4 Convolutional Layers

We also try using a convolutional model in order to see if this would help as well. The idea behind this model is to learn filters that would be able to account for information being shared between a range of words. In this model, we only used a single convolutional layer to attempt to learn any dependencies between words. These filters are only between the word vectors, and can be see as a 1 dimensional convolution. For this method we used a filter size of 3. We also have to max-pool after these filters, and this is done in the same way as described before. Something that could be looked into with future work, is how deeper layers can affect the model.

2.5 Principal Component Analysis

The previous methods were all various sorts of benchmarks. A new method that we explore is using principal component analysis in order to compress information about a sentence. The intuition behind this is that because the PCA returns the "most important" dimensions for the data, the leading k dimensions would contain valuable information about how the words in a sentence relate to each other. Our model will take a matrix that contains the concatenated word vectors in a sentence and apply a PCA transformation on it. We would then take the top k dimensions from the PCA result, unroll it, and feed it to our softmax layer. In our experiments, we chose to use the top 3 dimensions for our PCA analysis.

3 Experiments

We aimed to test the effectiveness of each of our models - averaging, max-pooling, recurrent, convolutional and PCA layers. These models were all tested on our synthetic dataset with a training test split of around 20000/2000. All of the models read in a pre-trained word embedding dictionary for each sentence, with each embedding being a vector $v \in R^{300}$. In order to determine whether or not a relationship exists, a separate file containing the probabilities of such a relationship is given. The threshold is set at 0.5, such that anything above 0.5 is considered a relationship.

Each of these models were trained on up to 200 epochs, or if the difference of the loss from the previous iteration was ≤ 0.00001. In order to train, we used the Adam optimizer with a learning rate of 0.01 and weight decay of 0.

3.1 Results

After running these experiments, we see that the simple word embedding models do indeed perform better than the recurrent and convolutional models. This comes somewhat as a surprise, as both the recurrent and convolutional network take a lot longer to train, and take into consideration the ordering of the words in a sentence, compared to the simple models, where none of the ordering is preserved.

Model	Accuracy	F1-score
Average	62.5	54.6
Max-Pooling	62.4	55.63
Recurrent	67.35	52.24
Convolutional	57.92	35.24
PCA	**72.13**	**55.14**

Table 1: Model Comparison: Accuracy and F1

We see from the results that the recurrent and convolutional models do particularly bad in terms of the F1 score, even though it does about the same as the simpler methods in terms of actual classification accuracy. It is possible that the recurrent and convolutional models need more epochs to train in order to perform better than the simpler models, or perhaps deeper layers are needed.

For example, our convolutional network was only operating on 1 convolutional layer with a filter size of 3. This makes it very difficult for the network to extract important high level features out of the sentence. Not only is it hard to extract high level features, it is also hard to compare words that are not within the filter window, without additional convolutional layers.

Similarly for the recurrent networks, only 1 layer for the LSTMs were used. This does better than the convolutional model, most likely due to its ability to better capture long term dependencies, but still lags a bit behind the simple word embedding models.

Interestingly, having a PCA layer actually does quite well on its own in this task. It is comparable to max-pooling and averaging layers, while not as computationally intensive as the convolutional and recurrent methods. Therefore, we try to see if there are good extensions to this method.

4 Extensions to PCA

As shown from the results, we see that simpler models generally tend to do better, and that using PCA to compress information seems to do pretty well on its own. As we continue to explore these methods, we look to see if we can improve on just using PCA in order to help with our relation-extraction task.

4.1 Fine-tuning PCA

One of the ideas that we had was to introduce fine-tuning for our PCA model. This means that the input word embeddings would not be a constant anymore, and would be able to be changed through backpropagtion through the PCA layer. This means that we would have to reformat how we computed the PCA. Previously, we would use the SVD method of solving the PCA, but now we would have to formulate as an optimization problem. This is necessary in order to perform backpropagation through it, so that the word embeddings can be updated. This can be formulated as follows.

$$= min \sum |z_i z_i^T - w_i wi^T|^2$$

We can also see that this is differentiable, and that the derivative is

$$\nabla(min\sum(zz^T - ww^T)^2) = 4(zz^Tz - ww^Tz)$$

If we combine this with the loss function that we have from logisitic regression as well as allowing the word embeddings to vary, then the total loss function that is obtained is as follows.

$$min\sum_{i=1}^{N} l(< x_i, z_i >; y_i) + \frac{\mu}{2}|z_iz_i^T - w_iwi^T|^2 + \frac{\gamma}{2}||W - W_0||^2$$

where i is the sentence, and $l(< x_i, z_i >; y_i)$ is the logistic loss of the output of the PCA layer.

4.2 Max-Pooling PCA

Another idea that we tried out was to combine both max-pooling and PCA together as a layer to see its effects. Since PCA by itself does better than averaging, and max-pooling can be considered as pooling over first order moments, we would like to see how max-pooling PCA would do. We could consider the following problem as a max-pooling of the 2^{nd} order moments of the sentence matrix.

$$pca - maxpool(x_1, x_2...x_n) = argmin \quad tr(z)$$
$$s.t. \quad z - x_ix_i^T \geq 0$$

We use the result of this layer to feed into the softmax layer for classification.

4.3 Extension Results

Model	Accuracy	F1 score
PCA	65.65	55.42
Fine-Tuned PCA	65.20	54.92
Max-Pooled PCA	**67.20**	**57.81**

Table 2: PCA Extension Results

The experiment is set up exactly the same way as the previous experiments. However, fine-tuned PCA turns out to be computationally expensive, and was cut short on the number of epochs for practicality.

We see that by combining both max-pooling and PCA, we are able to obtain results slightly better than just doing either of the individual methods. Both the accuracy and the F1 score went up as a result, and the combination of these two methods are still not as computationally expensive as the recurrent, convolutional or fine-tuned models.

The fine-tuned PCA model takes a lot longer time to converge, and doesn't seem to improve the results. It is possible that we did not train for enough epochs, but due to time constraints, we were unable to continue training the model for an extended period of time.

5 Discussion

From our experiments, we were able to verify that in general simple word embedding models are able to perform surprisingly well at tasks that seem to be more suited for more complicated models. These simple models are much faster to train, and get better results. Our PCA models do quite well with respect to the simple models, and do not take much longer to train than these models as well.

The results lead us to believe that PCA has some properties when it comes to condensing information in a sentence that is very helpful for determining the existence of a relation within that sentence. It also seems that by incorporating max-pool and PCA, we can get a little bit of the best of the both worlds, and get some improvements in both accuracy and F1 score.

It seems that the more complex models cannot be just instantiated with a single layer to obtain state-of-the-art results. This is especially true for the convolutional models, where more convolutional layers would allow it to learn higher level details from the sentence. Fine-tuning also doesn't seem to improve the results, and takes quite a bit more time to train than the simple word embedding models.

Overall, we see that max-pooling with PCA does provide us with the best results over all of the methods we tried, with the simpler methods

6 Future Work

Some future directions that we would like to see with the work would be to fully understand why PCA works. We have a high level intuition of how it compresses down information, but more research in to the mechanisms would be interesting. With a better understanding of how PCA works, we would be able to improve on it further, such as the way we did with max-pooling PCA.

It would also be interesting to test fine-tuning on the other models to see if this has any effect on the results from these models. It might just be that fine-tuning acts like a regularization term and doesn't provide any new information about the sentence, but it would be interesting to flesh out the results and see what happens.

References

[1] Anonomous. On the use of word embeddings alone to represent natural language sequences. *ICLR*, 2018. URL `https://openreview.net/pdf?id=Sy5OAyZC-`.