Semantic Analysis from Verbs

Raju Khanal
Computer Science Department
Stony Brook University
rkhanal@cs.stonybrook.edu

Akshay Verma
Computer Science Department
Stony Brook University
akverma@cs.stonybrook.edu

Abstract— **The general aim of this framework is to provide a scalable method for learning relations and/or verbs as operators (represented as tensors). These operators should take in one or more words, and map them to a new embeddings space. The goal of this new embedding space is to capture the semantics of the phrase (rather than the semantics of a single word).**

I. Introduction

In order to provide a scalable method for representing verbs as operators, it is desirable to have the ability to compute these tensor representations directly from the word embedding representation for the verb. The idea is that the word embedding for the verb should already give us plenty of information about the verb, so it is not unreasonable to believe that the tensor representation for the verb can be derived from it. This drastically reduces the storage and parameter requirements, from a tensor for every verb, to a single tensor used for computing verb tensors from verb embeddings.

II. Related work

Previous work has looked at learning tensors for individual verbs, though this proves to be unusable in practice due to the large number of parameters needed and unreasonable storage requirements (in addition to storing word embeddings, you must also store verb tensors of order 3). Because of this, previous methods could only learn operators for a small subset of verbs (< 400). One example for instance is the paper on "Low-Rank Tensors for Verbs in Compositional Distributional Semantics" in which they attempt to do similar work to find low-rank tensors for verbs. In the paper by Stanford University[1] , entities are represented as an average of their constituting word vectors. We adopt a similar technique to compute the embeddings for the entities.

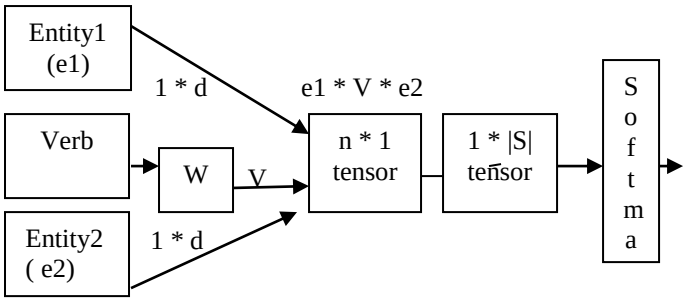
III. Dataset

We have used New York times article dataset. The entire dataset was of 32GB which spanned across years 2001 to 2007. As the dataset was too huge, we used the dataset of 2007 to extract the tuples <e1 , r1 , e2>. The total number of sentences were 850000. We used the StandfordCoreNLP to extract the tuple in reverb format[3]. Also we have used the Reverb format for extraction from University of Washington's site[4]. In total the reverb format contained 18 tab separated values. The order of the data extracted from the tool was as shown below:

i. The filename (or stdin if the source is standard input)
ii. The sentence number this extraction came from.
iii. Argument1 words, space separated
iv. Relation phrase words, space separated
v. Argument2 words, space separated
vi. The start index of argument1 in the sentence. For example, if the value is i, then the first word of argument1 is the i-1th word in the sentence.
vii. The end index of argument1 in the sentence. For example, if the value is j, then the last word of argument1 is the jth word in the sentence.
viii. The start index of relation phrase.
ix. The end index of relation phrase.
x. The start index of argument2.
xi. The end index of argument2.
xii. The confidence that this extraction is correct. The higher the number, the more trustworthy this extraction is.
xiii. The words of the sentence this extraction came from, space-separated.
xiv. The part-of-speech tags for the sentence words, space-separated.
xv. The chunk tags for the sentence words, space separated. These represent a shallow parse of the sentence.
xvi. A normalized version of arg1. See the BinaryExtractionNormalizer javadoc for details about how the normalization is done.
xvii. A normalized version of rel.
xviii. A normalized version of arg2.

We used the POS tag on the relation to extract the verb. For the entities, we took the average of the embeddings of the words as suggested by Socher in "Reasoning with Neural Tensor Networks for Knowledge Base Completion" [1]. This is one of the most practical methods currently being used in the NLP space.

IV. PROPOSED APPROACH



Let's assume that word embeddings describe vectors in some semantic space S. We can think of our embeddings as being described in terms of some basis E = {e1...en}. We can think of these basis as describing some basic semantic concept of which all words are composed of. For example, one dimension of a word embedding may describe the animateness of the word, while another 2 may describe how feminine the word is. Of course no dimension has an explicit meaning, they are learned automatically in order to perform well on some task. However, the fact that the dimensions of word embeddings have some type of meaning is apparent when looking at the results of vector arithmetic done with these embeddings (such as the famous king - man + woman example). We would like to find tensors for verbs in order to produce embeddings for phrases. These embeddings we would like to also be mapped into some semantic space U. However it is unlikely that the basis used for U would be the same as the basis used for E. This is because phrases are different semantic entities from single words. Some of the concepts needed to capture the semantics of words may be unnecessary, or we may need to describe phrases using some other basic concept not needed for words (for example, how much something relates to movement might be important when describing phrases, but not in describing words). Thus we would like our verb tensors to map into a new subspace U that is spanned by some basis B = {b1...bn}, with this basis describing the semantic concepts needed to concisely describe the meaning of phrases.

For the remainder of the document, we will refer to this function that produces verb tensors from embeddings as T. One possible formulation for T is as follows: Let W be a matrix of weights that parameterize the function T (with w_i being the i^{th} column of W). Given a word embedding for a verb v, we can compute i^{th} column of its matrix V_i as:

$\mathbf{V_i = w_i w_i^T v}$. We can rewrite this as: $\mathbf{w_i w_i^T v = <w_i, v_i w_i>}$. From this it can be seen that the final matrix will be a mapping to a subspace that is generated by the column vectors of W, with the i^{th} column giving the projection of v onto w_i. We can thus interpret W as being a sort of basis (though they may not all be linearly independent, so they may not be a true basis) for the semantic space U (since when we use this operator on a word, it will map the word to some linear combination of the columns of W). Note that in this case, all that is required is a matrix to parameterize T. If we want to represent our verbs as tensors, then T would be parameterized by a single tensor.

The Algorithm

Let n denote the number of instances and d the embedding size. Then:

1. We first initialize a tensor W of size <d , d> randomly.
2. Form a tensor for all the <e1,V,e2>. Each tensor's dimension is < n , d > where n is the number of tuples in our training set.
3. Compute a tensor V of shape <nd * d * 1> by doing a batch matrix multiplication of <nd * d * 1> (which repeats the embeddings from 0…d n times from weights tensor i.e repeats the row embeddings) and <nd * 1 * d)> (which repeats the embeddings of 0^{th} column d times , 1^{st} d times and so on till n from weights tensor i.e. repeats the column embeddings).
4. Reshape e1 tensor to <n * 1 * d> and multiply with V above to get <n * 1 * d>. Reshape e2 to <n * d * 1>. Multiply these two to get a tensor of shape, <n * 1 * 1>. Reshape it to n * 1.
5. Compute the actual one hot vectors called w2 which is of size <1 * |S|> where |S| = length of the unique words in the corpus for each training instance.
6. Multiply the result from (4) with (5) to get a tensor of shape <n * |S|>.
7. Run cross entropy to find the probabilities of words closest in the vocab to the particular tuple.

   In our case, W and w2 are the two variables we are trying to learn. We store the result of these two tensors and use them later on to compute the accuracy.

V. EVALUATION

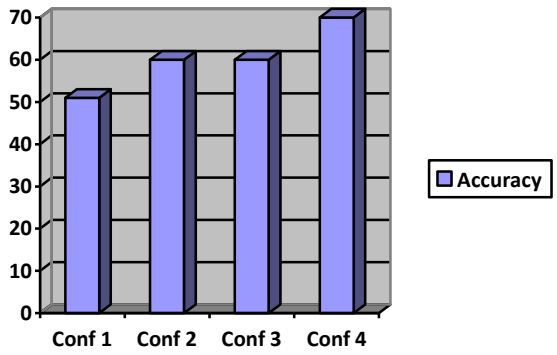
For the sake of evaluation, we have extracted 1000 instances from the test set where each instance contains two tuples t1 and t2. Both the tuples are from the same sentence. Our goal is to compute the cosine similarity of these two tuples by following the same algorithm as described above, but instead using the learnt W and w2 tensors from the training set. If the cosine similarity is above the given threshold, we conclude that the verb was able to capture the semantic notion of the sentence properly. In our case we are using 0.5 as the threshold value.

VII .RESULTS

To start off, we checked the result with Weights initialized randomly and found that only 8 instances had a cosine similarity of more than 0.1. In our results, we will use a threshold value of 0.5.

Configuration	Embedding_size	Training Sample size	Accuracy
1	50	5000	50.99%
2	50	10000	
3	128	5000	
4	128	10000	

Below is the graphical representation for the above table:



VI. CONCLUSION

We find that low-rank tensors for verbs achieve comparable or better performance than full-rank tensors on sentence similarity tasks, while reducing the number of parameters that must be learned and stored for each verb by at least two orders of magnitude, and cutting training time in half. Thus memory and space and time are all optimized as per our liking.

This is important for extending the model to many more grammatical types (including those with corresponding tensors of higher order than investigated here) to build a wide-coverage tensor-based semantic system using.

.

VII. FUTURE SCOPE

We need to investigate whether the orthogonality or independence of the columns of W is important (this determines the size of the subspace for phrases). Should any constraints be placed on the columns of W to ensure the subspace for phrases is rich enough? Or should we go with what it output from training?

Negative sampling might be used as well to make sure that nonsense phrases (such as multiplying an adjective with the verb tensor) map somewhere near the zero vector. In linear algebra terms it would be desirable for the kernel of the verb tensor to be the set of words that are not grammatically compatible with that verb (this may be too much to ask for, but we might be able to approximate it by using negative sampling).

VIII. REFERENCES

[1] Socher, Chen, Manning and Ng in “Reasoning With Neural Tensor Networks for Knowledge Base Completion”

[2] Daniel Fried, Tamara Polajnar, and Stephen Clark in “ “Low-Rank Tensors for Verbs in Compositional Distributional Semantics”

[3] https://stanfordnlp.github.io/CoreNLP/

[4] http://reverb.cs.washington.edu