Scholastic Aptitude of Neural Networks:
Improving the performance of word embeddings on analogy questions

Felipe Campos **Robert Foster** **Zachary Ingbretsen**

Abstract

Understanding the relationships between abstract concepts is a valuable skill for both humans and computers. A somewhat surprising property of word embeddings trained by CBOW or skip-gram neural network models is that linear relationships form between words. We are putting word embeddings trained on different corpora to the test. First, we show how the embeddings perform on a standardized set of analogy questions from Turney, et. al. In the future, we will explore methods of improving the performance of a subset of these word embeddings.

1 Introduction

This paper studies different approaches to computationally process analogies. A good understanding of analogies is crucial for better understanding the meaning and usage of words in human-written texts and may be generalizable to other areas of AI development. First, we use pre-trained word embeddings (Word2vec and GloVe) to establish a baseline of how word embeddings perform at this problem. We establish performance differences based on the training corpus, number of dimensions, and distance metric used. Given that understanding, we attempt to improve on the performance of the embeddings by analyzing their shortcomings and augmenting them. Specifically, we use ELMo representations of our words to improve on the GloVe embeddings, and we train a deep neural net on analogy questions.

2 Background

The analogy task from Turney, et. al. has been subjected to various approaches since the early 2000's. There are many approaches that exceed the baseline performance of random guessing, however no automated approaches excel at this task. Somewhat consolingly, humans also do not excel at the task; The average US college applicant only scores a 57% on this particular test (Turney and Littman, 2005). The current state of the art on this task is a hybrid ConceptNet implementation which scores a 56.1%, approaching the performance of the average college applicant(Speer et. al., 2017).

2.1 Challenges

Understanding analogies requires identifying the correct meaning of individual words and the relationships between pairs of words. Correctly answering a seemingly simple analogy could involve first disambiguating homonyms, and then identifying complex/non-linear relationships between words, among other challenges. Our work tries to address the following problems.

2.2 One-to-Many Relationships

A relationship such as capital-to-country has a straightforward interpretation. Italy has only one capital, and that capital is Rome. Such a one-to-one relationship can be applied to all countries in order to find the capital. However, there is no such relationship between any arbitrary city and its country. Naples is a city in Italy, but the city's relationship in high-dimensional space does not occupy a privileged position, as does the capital city. For this reason, a linear transformation may not be effective at correctly answering this type of analogy. One solution idea to this problem was to use a web search to find the words (Bollegala et. al., 2008). Overall this was only moderately effective

2.3 Polysemy

Having just one embedding per word can be problematic if a word has more than one meaning. Over the course of training the embedding, all meanings of the word will contribute to the final

embedding. This will reduce the usefulness of the embedding for all senses of the word.

2.4 Out of Vocabulary Words

Word2vec and GloVe are both trained on a large corpus, and filter out words that do not pass a certain count threshold. If a word in the test set was not found in the original corpus or did not surpass the count threshold, it will be impossible to predict an embedding vector for that word.

3 Methods

Given the challenges inherent in this task, we first benchmarked the performance of pre-trained Word2vec and GloVe vectors. In order to get a direct comparison of Word2vec and GloVe on this task, we then trained our own embeddings on a full dump of English Wikipedia. We used these benchmarks to identify strong and weak aspects of the different embeddings.

Using this knowledge, we set out to improve our performance by modifying the embeddings via ELMo, as well as trying to train a neural network to use the word embeddings to capture nonlinear relationships in the analogies. By modifying the word embedding vectors to introduce different representations for individual words via a process such as generating CoVe embeddings or ELMo representations, we believe that we can rival the performance of a human agent (ELMo CITE #TODO). (#TODO MOVE UP McCann, et. al., 2017)

3.1 Embedding Evaluation

As noted by Mikolov et. al., word embeddings can encode relational similarities in their vector representations and, when well-trained, analogies in the form "*king* is to *man* as *queen* is to ..." can be answered with very simple arithmetics. Therefore, in a word analogy of form a:b::c:d, where (a,b,c,d) are words, it is expected that the vector resulting from the operation d - c would share some similarity with the vector resulting from b - a (Li and Summers-Stay, 2017) . Therefore, we could answer an analogy question such as a:b::c:? by calculating:

for every

Where *sim* can be any similarity function such as cosine, when other works usually refer to the equation by the name *3CosAdd* (Levy and Goldberg, 2014) or Euclidean. In this paper, we use both similarity functions for experimentation purposes.

We also evaluate the performance of word embeddings when used in a neural network trained and focused on solving analogy questions, based on a deep neural network model used for visual analogy-making in a previous work (Reed et. al., 2015).

3.2 Datasets

An evaluation dataset of 373 SAT questions was obtained from Peter Turney. Each analogy question has four or five choices as well as part of speech tagging. We start with benchmarks for the dataset as a whole, but looked at accuracies for part of speech and question source as well, using the algorithms below.

Common pre-trained word embeddings (GloVe 6B, 42B, and 840B, and Google News word2vec) were downloaded from different sources, and we converted them into the standard Word2vec format to standardize the model scoring interface.

For training the analogy-focused neural network models, we have used the Google Analogy Dataset as a smaller training sample (with approximately 20,000 analogies) and the Bigger Analogy Test Set as a more complete training sample (with more than 2,000,000 analogy questions). The trained neural network was then benchmarked against the SAT dataset.

3.3 Code

The code is written in Bash and Python scripts and is available at:

Instructions on how to run the setup scripts to use the virtual environment are contained in the Readme.md file in the github repository. Please email your GitHub username for access to the repository if you do not have access.

4 Results

4.1 Baseline Algorithms

We used pre-trained word embeddings (word2vec and GloVe trained on different sources) as a baseline using both Euclidean and cosine distance measurements. The summary of the baseline models of each are in Table 1 below:

From this baseline, we decided to do a direct comparison between corpora, embeddings and distance metrics. Using the GloVe 42B Twitter Vectors, the Cosine Distance outperformed the

Euclidean Distance with a 26% vs 22% accuracy. Also GloVe outperformed word2vec by a small margin - 40.5% to 37% accuracy when they were both trained on the Wikipedia 4.2B word dataset.

We have so far found the best performance from embeddings with higher dimensionality, trained on more data, and using cosine distance as the distance metric. However, there may be cases when the Euclidean distance is more appropriate to use (e.g., when the analogy has more to do with the magnitude of the difference between words than direction).

We intend to experiment with different ways of modifying the embeddings to better encode relationships between words as well as try to train our own analogy-focused word embeddings to see how they compare against the baseline.

4.2 Error Discovery

Next we examined errors from the best performing embeddings, the GloVe840b, 300 dimension embedding. The model appeared to have more difficulty with a couple of relationships more often than others. For example, below is a sample question that all the embeddings got wrong (all models had child,parent as the answer that scored the 'best'):

Question 370: ['sandal', 'footwear']

Sorted distances:

Words: ['volume' 'bookcase'], score: -0.13121460378170013

Words: ['watch' 'timepiece'], score: -0.053831081837415695

Words: ['wax' 'candle'], score: -0.03754980117082596

Words: ['monarch' 'castle'], score: 0.010474562644958496

Words: ['child' 'parent'], score: 0.03693845123052597

Best answer found: child,parent

Correct answer: ['watch', 'timepiece']

A linear relationship does not capture this relationship well. A sandal is a type of footwear. However, there are many types of footwear, and all may have different types of linear relationships. The same is true for the correct answer: a watch is a type of timepiece. We will need a different approach to capture relationships like this.

We also investigated errors in the difference between a variety of relationships using the Google analogies dataset published by Mikolov et al. The results are shown below in Table 3. As noted in the table, most Semantic Questions are answered correctly fairly accurately (with the exception of currency) while most Syntactic Questions are answered less accurately.

For these two reasons, we thought that a purely linear approach to solving the problem was limiting the types of questions the model could get correct. We decided to try a couple of approaches to see if we could overcome this hurdle.

4.3 Improving on Baseline

To solve the analogy problem, we tried a few different solutions. The first was to train new embeddings that has more context included for the words using the ELMo embedding algorithm. The second was to train a deep convolutional neural network to understand the analogies and make better guesses at the answers.

4.4 ELMo Encoding Algorithm

The ELMo algorithm is deep contextualized word representation that models both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (Peters et. al. 2018). The ELMo algorithm can be trained on words or sentences to produce word embeddings that include more context around each word. ELMo word representations are functions of the entire input sentence and are computed on top of two-layer bi-directional LSTMs with character convolutions as a linear function of the internal network states. (Peters et. al. 2018). Each layer of the ELMo encodings can be used as the embedding for our problem set which allows multiple end 'outputs' to the algorithm.

For our ELMo testing, we broke down this phase into two separate training tests: on each analogy word and on each pair of words. The baseline vectors in ELMo are the GloVe6B 100 dimension set with a baseline accuracy of 37.27%. Table 4 contains the accuracy from different stages.

From Table 4, the single word embeddings taken from the first layer provide the biggest increase in accuracy with a total of 8.3% from the baseline accuracy.

4.5 Neural Network

We also wanted to test the behavior of embeddings when used in a Neural Network focused

on analogy-making, under the rationale that it could perhaps find patterns between the encoding of words that is not obvious with simple vector operations. The neural network architecture was inspired by the paper "Deep Visual Analogy-Making", by Scott Reed et al, that used a neural network in the context of visual analogies (e.g. where images were rotated, changed color and/or changed size). This was done by encoding pictures using a function f and decoding using a function g. Using the a:b :: c:d framework for analogies, the pictures were inferred by the relationship between a and b by subtracting the encoded values and then used a transformation query T on c to then decode the result and get d. In the paper, they used three different types of transformation query for T: *3CosAdd* (as we have done previously), *3CosMul* (similar to *3CosAdd*, but instead of summing the difference of a and b to c a multiplication is used) and a deep neural network. We used the latter as inspiration for our neural network model.

We tried to reproduce a similar approach for words. For this, we changed the encoding function f to a word embedding lookup, based on the different embeddings we used above. We used the similar approach of inputting a concatenation of the embedding of the word (c) and the difference between the embeddings for two given words (a: b) into a fully-connected neural network, and then summed the results with the word (c) to achieve the final embedding. This embedding was then decoded back using decoder function g to a word (d) by picking the closest word in the embedding matrix through either cosine similarity or euclidean distance. The architecture diagram for the neural network is shown in figure XX.

The neural network was trained using the following objective function, similar to the one used in the original paper, which aims to reduce the sum of the square of norms for the vector resulting from the subtraction of the encoding of target word d and the decoded result as shown below:

As the original paper did not mention the parameters used for the fully-connected layers, we have tried with several different network sizes. Initially, we tried with a single 600-dimensional layer, and then tried with a bigger model using 4096-2048-1024 dimensional layers. Training was performed for 4500 epochs with mini-batches of 50 using Adam Optimizer with gradient clippings with maximum gradient norm of 1.0, learning rate of 0.00001, and with dropout rate of 0.5 using the Google Analogy dataset. The word embeddings were fixed and made not trainable in the model, as the interest of our study was to use the neural network to find word features encoded in the embeddings and not refine them with analogy data.

After the set amount of epochs, the training still showed room for loss reduction in case more epochs were executed. The resulting model, however, failed to produce useful results. It is worth mentioning, however, that the model did produce word embeddings in the vicinity of the correct answer in vector space when decoded using Euclidean distance, which indicates that perhaps with some more training epochs to reduce loss the model could provide better results.

We also developed two other models, which haven't been as thoroughly tested. In our second model attempt, we tried a modified version by removing the subtraction between the embeddings of the two given words (a:b) and inputting a concatenation of all three known words (a:b :: c:) into the neural network to check whether it would behave better or worse than the previous model. The rationale was that maybe performing the subtraction before could be restricting the model to transformations that are visible in a linear space. The diagram for this architecture is depicted in figure YY.

Our third model added a secondary, trainable embedding that is concatenated with the original, primary embedding through the secondary encoding function h. This aims to augment the original embeddings with data from training on analogies and perhaps make more complex analogies clearer on the embedding. The diagram for this architecture is depicted in figure ZZ. Unfortunately, both models also failed to produce a noteworthy result that was better than random chance.

5 Conclusion

There are a few lessons to be learned. Unsurprisingly, the quality and site of the training corpus is important for the performance of word embeddings on the analogy task, and using the cosine distance as the distance metric was universally more accurate. Given the results of our direct comparison of Word2vec and GloVe, we would recommend using GloVe as the base for future experimentation over Word2vec for this particular application. However, further improvements can

be made on top of the embeddings alone.

Using ELMo to improve the GloVe embeddings proved to be particularly promising. The key insight was to use the middle layer of ELMo's deep net instead of the top layer. Since the lower layers encode more of the syntactic information, using the middle layer improved performance in nearly all syntactic questions, and in one semantic type of question. The ELMo representations did seem to perform poorly on questions relating to places (i.e., countries or states), but otherwise improved significantly on the performance of the GloVe 100 dimensional vectors.

The ELMo model that was trained on the Glove 6B 100D emeddings outperformed even the Glove 42B 300D embeddings. Given that particularly encouraging result, training an ELMo model on the Glove 840B 300D vectors would be a promising next step. Given the raw embeddings' performance on the task, the addition of the ELMo representation could offer performance that is competitive with the state of the art.

Finally, although the neural network approach ended up not providing satisfying results, we did find that it does return word embeddings close to the expected words. Given that there was still margin to reduce the loss with more training epochs, it is possible that with a larger analogy dataset for training, more training epochs and further tuning of the model parameters the neural network model could prove to be a useful method for solving analogy problems and thus worth the effort of more study and experimentation.

Embeddings	Euclidean	**Cosine**
word2vec 300d	0.3190	0.4236
GloVe.6b 50d	0.3190	0.3566
GloVe.6b 100d	0.3432	0.3727
GloVe.6b 200d	0.3405	0.4075
GloVe.6b 300d	0.3405	0.4155
GloVe.42B 300d	0.3780	0.4450
GloVe.840B 300d	0.3619	0.4906
Twitter.42B 200d	0.2172	0.2601

Table 1: Baseline accuracy of embeddings.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.