Solving Biology Analogies with Deep Learning

Justin Xu
Math and Computational Science
Stanford, CA
justinx@stanford.edu

Han Lin Aung
Computer Science
Stanford, CA
hanlaung@stanford.edu

Sajana Weerawardhena
Computer Science
Stanford, CA
sajana@stanford.edu

Abstract

For the purposes of automating the generation of textbook learning material, we formulate a prediction algorithm to automatically answer analogy tasks in the context of a biology textbook. Word embeddings have shown some success in solving analogical tasks by using cosine similarity metrics, but often times the analogies generated from word vectors, such as Glove vectors, face problems related to low dimensionality in the vector space or inadequate representation of the context the analogy was trying to show. In recent years, contextual word representations and deep learning architectures have shown greater success in caputring semantics of words given specific contexts. Hence, we present deep learning architectures including ones based on ELMO and BERT to generate analogy (e.g. animal:mitochondria :: plant:chloroplasts) in the context of the biology textbook. Since our analogies also have variable length components, utilizing deep seq2seq or seq2vec models are effective for predicting these analogies. Surprisingly ELMo models seemed to have outperformed BERT and even BioBERT, (a pretrained biology corpus BERT embedding) in this context, achieving 55% exact matches for unseen analogies.

1 Introduction

The task of analogy generation has been used in active research, in many different fields, such as psychology and linguistics. Analogy generation has not only been considered as an end-goal in itself but has also been used in evaluation tasks such as evaluating the performance of word vectors. Currently, this general task has had varying degrees of success. Some approaches that made use of word embeddings have faced problems related to representations being in too low dimensional space or are not able to capture the context the analogy is trying to represent.

Analogy generation in the natural sciences, especially biology, has not been as actively researched but is a beneficial space for analogy generation especially for educational purposes. In biology, there are many relationships between different concepts represented by scientific terminologies. Analogy generation can greatly assist the education of those who are starting to learn biology. Learning about relationships between different words from the analogies can enhance a student's understanding. Currently, there are models that serve to do so using hand-built glossaries such as from the LIFE biology textbook, but we are hoping to generate analogies in a more automated fashion.

Currently, most models use pure word embeddings or vanilla RNNs to generate these analogies. In this paper, we dive deeper into these word embedding models, including but not limited to GloVe and fastText. Furthermore, with the rise of transformers and contextual embeddings, we describe techniques that make use of various different deep learning architectures including ELMo and BERT to generate analogies. Finally, we also provide a dataset on the different analogies based on the LIFE biology textbook.

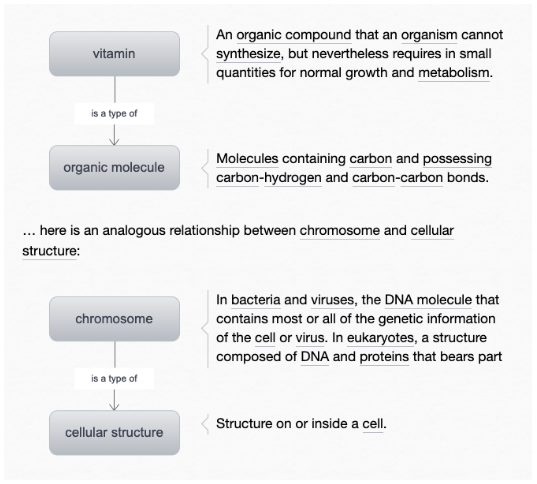

vitamin
An organic compound that an organism cannot synthesize, but nevertheless requires in small quantities for normal growth and metabolism.
is a type of
organic molecule
Molecules containing carbon and possessing carbon-hydrogen and carbon-carbon bonds.
... here is an analogous relationship between chromosome and cellular structure:
chromosome
In bacteria and viruses, the DNA molecule that contains most or all of the genetic information of the cell or virus. In eukaryotes, a structure composed of DNA and proteins that bears part
is a type of
cellular structure
Structure on or inside a cell.

Figure 1: Analogy example.

2 Related Work

Analogies have been used as evaluations for word vectors such as GloVe. Word embeddings such as GloVe embeddings have often shown that the representation: "King - Man + Woman" results in a vector very close to "Queen." [7] However, the training of these word embeddings, including word2vec and GloVe, are not based on learning these analogies. These analogies can be claimed to by-products. Character embedding techniques were also explored in FastText [1] to allow for out of vocabulary text to be generated and seems to be able to get relationships more closely related to word forms.

More recent analogy generation that have been used is through the use of deep learning architectures such as a vanilla RNN. One paper (Mikolov et.al, 2013) examines the vector-space word representations learned by the input-layer weights of an RNN [5]. In 2013, the model showed state-of-the-art results in capturing syntactic and semantic regularities in language through the analogies. After it generates the vectors for each word, it then captures analogies through the vector offset model.

In particular, recent models that have shown great successes in using pretrained modules such as ELMo [8] and BERT [2]. ELMo has shown great success in improving upon existing word embeddings by producing contextual representations of words. Many pretrained embeddings have been embedded in existing models and have shwon great success in many general NLP tasks. BERT is yet another of these models that have shown great successes. In fact, fine-tuned BERT representations have produced state-of-the-art results for a wide range of tasks, such as question-answering and language inference.

Additionally, given the challenges of biology vocabulary being very different from everyday language and with nuances that are hard to capture, Korean researchers developed BioBERT [4] to boost performance on biomedical text related tasks. Their paper showed performance improvements on various tasks like NER (named entity recognition), biomedical relation extraction, and biomedical question answering. While many of the tasks that these two pretrained modules have shown successes are not directly related to analogies, they show great promises in our specific task of generating analogies in the context of biology textbooks.

3 Approach

The task that we are looking to solve is, given the input (a,b,c) of the analogy a:b::c:d, where a, b,c and d maybe either words or phrases, predict the missing word/phrase d in the analogy. The model would either predict a sequence of words, which maybe in the vocabulary or UNK otherwise. We approached this task using a unigram dataset, analogies containing only single word inputs and also an n-gram dataset, analogies with multi-word phrases.

3.1 Unigram Dataset

Our approaches on the unigram dataset largely drew from the vector offset model [5].

1. First we find the corresponding embedding vectors for each of the words a, b, c: x_a, x_b, x_c which are all normalized to unit norm
2. We then compute $y = x_b - x_a + x_c$
3. Notice y is the continuous space representation of the word that is expected to be the best answer. Of course, since there is a possibility that there is no word at that exact position, the greatest cosine similarity is computed to find the closest word to y. We find the greatest cosine similarity as $w* = argmax_w \frac{x_w y}{\|x_w\| \|y\|}$

Based on the vector offset model, we built our word embedding models to generate analogies. Utilizing our corpus, we built multiple types of word vector representations to see which one showed the best initial results for our analogy detection algorithm.

GloVe

Our first choice of word embeddings was formed through running GloVe [7] directly on our biology corpus. These embeddings had a much smaller vocabulary size and as a result also had a lot more missing words when trying to generate the analogies. We trained this new word vector model from scratch using the GloVe script (https://github.com/stanfordnlp/GloVe). We also tried using GloVe vectors pretrained on the wikipedia corpus.

ELMo Pretrained

Afterwards, we experimented on a pretrained ELMO module [8]. We extracted the character-based word representations outputted from the module. The pretrained module is based on deep contextualized vectors paper which introduces a general approach for learning high-quality deep contextualized vectors [9].

FastText

Our final model based on unigram the dataset was experimenting on pretrained FastText as well as the biology corpus trained FastText vectors. We trained on the same biology corpus and evaluated in similar ways to the other embeddings.

3.2 N-gram dataset

3.2.1 Vanilla Seq2seq Model

We also implemented a simple sequence to sequence model that would learn the analogy d, directly. The vanilla seq2seq model is a simple Bidirectional LSTM Encoder, Unidirectional LSTM Decoder and multiplicative attention. Since we wanted to implement the simplest seq2seq model we thought it was best to concatenate A, B and C, as [A;B;C] with separation tokens '<SEP>' in between each of them. Then this single concatenated string was fed into the model as the input. For the output we expected the model to output a word or phrase that represented d.

The model was as follows:

1. The encoder was a bidirectional LSTM. While the decoder was a single LSTM cell.
2. In the decoding process we took a multiplicative attention with respect to each word in the concatenated ABC sequence.
3. We thought it was best to treat the problem as a multi-class classification problem- so we simply took the cross entropy loss between the predicted d and the ground truth d.

In our first experiment with this model we trained word embeddings from scratch. Then we also trained the model with a set of $GloVe$ vectors we trained from our biology text book data.

3.2.2 Seq2seq Model and Char-based CNN

In this model we attempted to strengthen our word embedding layer by concatenating the pre-trained word embedding with Char-based CNN word embedding. We also added a char-based greedy decoder to our decoder to handle 'UNK' tokens in output. This greedy decoder was based off the paper 'Character-Aware Neural Language Models'.[3]

We also trained a variant of this model where we didn't include the greedy decoder and we trained the model against ground truth converted into our pretrained word embeddings- instead of the char-based cnn word embeddings (like in our vanilla model).

3.2.3 Seq2seq Model, Char-based CNN, TriDaF

Our final modification on to our seq2seq model was to replace our basic multiplicative attention with a Tridirectional Attention Flow Layer (hence: TriDaF). This attention is a generalized version of the Bidirectional Attention Flow Layer described by Minjoon et Al. [10].

Recall that our input data to our model is three strings, A,B and C. In all our previous Seq2Seq models, we concatenated these three into a single string that we fed into our model. However in this model we feed each of A,B and C into three different LSTM Encoders. Then we use the hidden encodings we derive from each of them for our attention.

Our intention with the attention was to model $a2c, c2a, a2b, b2a, b2c$ and $c2b$, to produce *analogy aware* representations. Between two sequence inputs, like a and c, the attention was calculated as described by BiDaF. $a2c$ intends to model which words in c are important for a. $c2a$ models vice versa. We thought this was important because ultimately our analogies were meant to model the relationship $a : b :: c : d$. We hypothesized the attention weights would be suitable in helping model these relationships.

The specific implementation is as follows:
We modelled $a2c, c2a, a2b, b2a, b2c$ and $c2b$, just as BiDaF modeled $q2c$, $c2q$. Then we produced the following *analogy aware* encodings of A,B and C:

$$A = \{A; a2b; a2c; A \odot a2b, A \odot a2c\} \in R^{10h}$$
$$B = \{B; b2a; b2a; B \odot b2a, B \odot b2c\} \in R^{10h}$$
$$C = \{C; c2a; c2b; C \odot c2a, C \odot c2b\} \in R^{10h}$$

Analogy Aware encoding A, is the concatenation of of the original encoding of A with a2b, a2c and the element wise multiplication of A with a2b and a2c. Analogy Aware encodings of B and C follow this same pattern.

3.2.4 ELMo and BERT models

The ELMo model uses pretrained ELMo[8] embeddings and is embedded in both seq2seq and seq2vec models. For the seq2seq variant, the ELMo representations are embedded in a bi-LSTM encoder and a linear layer to produce the output analogy. For the Seq2Vec, the ELMo embeddings are embedded in a bi-LSTM encoder and a decoder with (dot-product) attention. This model was implemented directly using AllenAI's NLP [1] wrapper for Pytorch with the model architecture manually constructed using this framework.

The BERT model swapped out the embeddings with a pretrained BERT [2] embeddings from the pretrained "bert-large-uncased" model implemented through HuggingFace's pytorch-pretrained-BERT library. [2]

We also proceeded with similar experimentation with the pretrained BioBERT [4] embeddings.

[1] https://github.com/allenai/allennlp
[2]https://pypi.org/project/pytorch-pretrained-bert/

4 Experiments

4.1 Data

We were given raw text from a the Life Biology Textbook, which totaled around 30,000 lines, with the textbook being around 1500 pages long. The raw text file being of size around 3.5MB. To extend our corpus, we also incorporated a number of open source biology textbooks from OpenStax, scraped by the help of another collaborator. [11]

We also have labeled analogies that are generated by the concept graphs and glossaries of the Life Biology Textbook that are hand-built. Analogy of different relationships are captured. This can be formulated as word a having a certain relation with word b. The relationships we capture are: 1) taxonomic relationships, 2) possesses, 3) is-inside, 4) has-region, 5) has-part, 6)function-of, 7) encloses, 8) element-of. The total number of unigram analogy we can capture for all the relations amount to 2154 analogies. The total number of analogies for our n-gram dataset (including unigram) accumulates to 77,654 analogies.

4.2 Evaluation Method

4.2.1 Unigrams

For unigrams, we establish our own evaluation metrics, which parallel the metrics typically used for analogy tasks to evaluate vector embeddings.

Given analogy $(word1, word2) \rightarrow (word3, word4)$, we fed in $word1, word2, word3$ and found the top 10 words in our vocabulary that were outputted by our analogy detection algorithm. If $word4$ was included in that list of 10 words, then we counted the sample as a success. Otherwise, we counted it as a failure. From this information, our goal was to simply maximize the accuracy of our algorithm.

Another evaluation metric we are utilizing is the average cosine similarity between the vector representation of $b + c - a$ and d for our unigram analogies.

4.2.2 N-grams

For our N-gram models, we had more flexibility since we sometimes had logits and variable length sequences. As a result, we included **exact-match** to the top result and proportion of examples found in the **top-n results**.

Some of our examples included multiple (a,b,c) that would map to multiple d values. To try to fit this into our evaluation, we included top-n results, so that if those multiple d phrases were in the top-n results, than it would still be counted. This also brought up the **any match** accuracy metric, which given a certain tuple (a, b, c) would be a match if any of the resulting d values in our test set matched.

We also included corpus BLEU [6] score, a way of automatically calculating the similarity between two phrases using n-gram similarities. BLEU score is calculated by taking an array of reference sentences and a hypothesis. The reference sentences became any of the possible d values for a given tuple (a, b, c) and the hypothesis was our predicted phrase.

Call r_i the i^{th} reference sentence.

$$p_n = \frac{\sum_{ngram} min\Big(C_h(ngram), \max_{i=1,\ldots,k} C_{r_i}(ngram)\Big)}{\sum_{ngram} C_h(ngram)}$$

p_n denotes the score for respective ngrams of size n

$C_x(ngram)$ is the number of times the ngram occurs in sentence x.

$$\text{Brevity Penalty} = exp(min(0, \frac{n-L}{n}))$$

where n is the length of the hypothesis and L is the length of the phrase closest in length to the hypothesis

$$BLEU = \text{Brevity Penalty} * exp(\sum_{n=1}^{N} w_n \log p_n)$$

where w_n is the respective weight for the n-gram size. For our purposes, since most of our phrases were either 1 or 2 length, we decided to put weights $[.5, .5]$ as our n-grams, not utilizing anything larger than 2-grams in our BLEU score calculation.

Corpus BLEU is the micro-average of the BLEU scores for each example. This differs from the typical macro-average, in that it sums up the numerators and divides by the total denominators. Corpus BLEU is typically used for evaluating across a dataset so that easy to calculate smaller length phrases are weighted less in evaluation.

4.3 Results

Unigram dataset Our initial evaluations were on the set of all the unigram analogy examples, where each part of the analogy was only one word. This subset had a size of 2154 analogies.

	Correct / Total	**Top 10 Accuracy**	**Avg Cosine Sim**
ELMo pretrained	507/2154	**.235**	.54
Wikipedia GloVe	159/1510	.105	.42
Biology GLoVe	135/1203	.112	.424
Wikipedia FastText	330/1931	.171	.548
Biology FastText	181/1990	.0909	**.658**

Table 1: Unigram Vector Embedding scores

Because we had missing vocabulary for a number of these embeddings, the total number of evaluated examples varied, with ELMo able to include the entire vocabulary and the other corpus trained vectors with varying missing vocabulary errors.

N-gram dataset

	ELMo	**BERT**	**BioBERT**
Top 1 Acc	**0.507**	.277	.320
Top 2 Acc	**0.789**	.454	.516
Top 3 Acc	**0.930**	.572	.647
Top 4 Acc	**0.980**	.649	.722
Any Match Acc	**0.580**	.332	.381
Corpus BLEU	**56.35**	34.19	38.63

Table 2: AllenNLP ELMo and BERT embedded model performance for Seq2Vec models

ELMo trained models outperformed BERT models by a large margin. BioBERT having inferior performance, which was pretty surprising, given that BioBERT should've incorporated more domain specific knowledge in its embeddings. The BioBERT model also took a significant amount of time longer than it took to train ELMo, suggesting that the model was more complex to train. BioBERT did out perform BERT on all our evaluation metrics however, which was expected.

These results were also overall better than expected, especially as we look at top-n accuracy. It suggested that even if we could not get exact matches, the next top n-1 candidate words can be searched to find the true label, as ELMo achieved 98% accuracy for top 4 match.

The other Seq 2 Seq models performed as follows:

	Vanilla Seq2Seq	**Char Encoder/Decoder**	**Char Encoder Only**	**TriDaF**	**ELMo**
Top 1 Acc	**0.502**	0.277	0.439	0.477	**0.502**
Top 2 Acc	**0.788**	0.426	0.710	0.727	0.784
Top 3 Acc	**0.930**	0.515	0.848	0.852	0.926
Top 4 Acc	**0.980**	0.547	0.895	0.893	0.968
Any Match Acc	**0.580**	.332	.332	.381	0.579
Corpus BLEU	49.873	27.072	45.381	47.456	**54.08**

Table 3: Vanilla Seq2Seq, Char Encoder/Decoder, Char Encoder only, TriDaF model and ELMo Seq2Seq performance

Suprisingly, the best of these models was the vanilla Seq2Seq. The Char Encoder/Decoder had considerably less accuracy and BLEU scores. Even the TriDaF performed poorly with respect to the Vanilla Seq 2 Seq. The Vanilla Seq 2 Seq accuracy scores are surprisingly comparable to that of ELMO.

5 Analysis

We discovered that the current best-performing model based on our evaluation metric of simply accuracy(top 1) and BLEU score is the ELMo model embedded in the Seq2Vec model. In fact, this model takes about an hour to train to converge in about 20 epochs. This again supports the speed and performance of general pretrained embeddings for more specific tasks.

5.1 Analysis of vector offset models

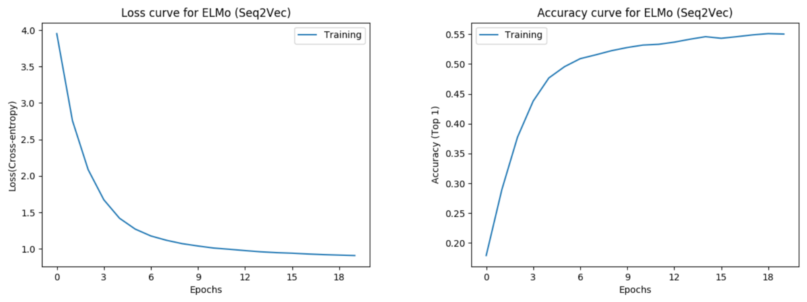
The vector offset model provides a reasonable framework and evaluation of our analogies. On unigram dataset, ELMo embeddings based on the character CNN performs best in accuracy, outperforming pure word embeddings, be it pretrained or trained on biology corpus. Furthermore, pretrained FastText character embeddings, also outperforms the word embeddings' counterparts on all metrics. Performance of solely training on the biology corpus is generally lower than the Wikipedia pretrained counterparts. This is probably caused by the relatively small data size from the biology corpus. However, the superior performance of ELMo and FastText shows promise in using more complex embeddings instead of pure word embeddings.

5.2 Analysis of Seq2Vec models

Seq2Vec models, which incorporate the pretrained embeddings, perform relatively well whether the inputs are ELMo, BERT or BioBERT. One surprising factor is that ELMo outperforms BERT by a huge margin, which may indicate that a reasonable convergence may not have been reached for BERT based models due to its complexity. Despite high performance of Seq2Vec, Seq2Vec models can suffer from OOV issues especially if the analogy that we are trying to predict is not in the corpus.

5.3 Analysis of Seq2Seq Models

Our class of Seq2Seq models proved to be quite a challenge to both design and evaluate. Our primary metric to evaluate them was the corpus BLEU score, where as above, we only weighted the one and



(a) Training loss curve (b) Accuracy curve

Figure 2: Loss and Accuracy of ELMo for Seq2Vec

two grams. We also report 'Exact Match' between the ground truth and the d outputs, with respect to the top 4 outputs of our model.

Our Vanilla Se2Seq model performed really well and outperformed our approach with the Char-CNN based Encoder and Decoder. The difference is in performance was in the order of thousands more examples correctly outputted. We think this is because our Char-CNN based approaches had to train the Char-CNN from scratch, during training. While the concatenation of our pre-trained vector and the char embedding did provide prior meanings about the input words, it was not adequate.

It was also surprising that our vanilla Seq2Seq also fared better than our TriDaF Model. We think this was because TriDaf had more weights to train and we did not have sufficient training data. The relationships between each of A,B and C which was modeled by the attention must have also either not helped or been weak relationships. Since the maximum length of each of A,B and C's components was around 5 words, a complicated attention did more harm than good. Empirically our simple multiplicative attention in our Vanilla mode fares far better. In fact, the vanilla Seq2Seq also outperforms the ELMo counterpart in a couple of our evaluation metrics.

The results of the Char Encoder Only model shows that training with pre-trained biology corpus vectors as opposed to a char CNN based word embedding performs better in this model.

6 Conclusion

There were some surprising results with less powerful deep learning models performing better than more complex architecture. However, overall, deep learning models, especially ones including ELMo embeddings, show powerful performance compared to regular word embeddings. To extend upon the analogy performance for improving the models themselves, one future potential direction is to research the interpretability of these models to provide an explanation to the output. Interpretability of neural networks has been a recent research interest in the NLP world, so it would be interesting to apply some of those methods to our context. Including the rationale for the output of the analogies will not only enhances the theoretical understanding of the underlying model but also increases students' understanding of how different concepts represented by these phrases are linked from the analogies generated.

Future work for this project would include incorporating more pipeline related functionality to our project. Given that our model can learn a good model to solve analogies given previous data, we'd like to extend this use case to be able to generalize to creating analogies from a textbook. Some kind of RNN related network could be utilized to generate analogies from the weights we have learned and the embeddings we used. To generalize to textbooks, it would also be interesting to allow for these weights to be learned through a corpus. This way, we can generalize to other subjects or possibly more specialized subjects.

Acknowledgments

Mentor: Vinay K. Chaudri
Scraping Help: Matthew Boggess

References

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[3] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. *CoRR*, abs/1508.06615, 2015.

[4] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *CoRR*, abs/1901.08746, 2019.

[5] Tomas Mikolov, Scott Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*. Association for Computational Linguistics, May 2013.

[6] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[7] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

[8] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[9] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[10] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[11] Rice University. OpenStax. `openstax.org`. Accessed: 2019-03-19.