Sentence Completion

Brian Xu
MIT
bwxu@mit.edu

Heeyoon Kim
MIT
heeyoon@mit.edu

Michelle Huang
MIT
mkyhuang@mit.edu

Xiaoxue Liu
MIT
sarahliu@mit.edu

Abstract

SAT vocabulary questions involve selecting the best word or words out of the choices given to fill in a blank for a block of text. This project will solve these vocabulary questions by applying the ideas of n-gram models, parsing, and recurrent neural networks in order to correctly rank and identify the best solutions for sentences with a single blank. In particular, we will score each option with our model and select the highest scoring answer as our solution. We compare the accuracy from our models with random guessing as our baseline of 20% accuracy. Currently, the state of the art model achieves approximately 50% accuracy. Using our models, we find that the highest accuracy comes from taking the semantic similarities of the sentence into account when choosing the answer.

1. Introduction

Sentence completion questions are a staple of standardized testing. These types of questions test not only sentence-level syntactic coherence, but also the semantic comprehension. The sentence completion problem introduces even greater complexity in that the answer choices are often all plausible, yet there is clearly a best answer, testing a high level of language processing ability.

Microsoft Research (MSR) hosts a sentence completion challenge to study this problem of algorithmic semantic coher-

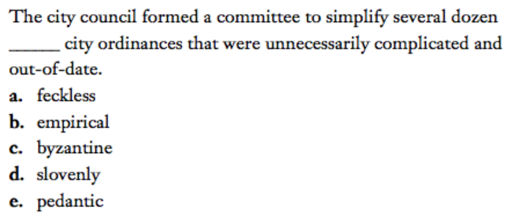
The city council formed a committee to simplify several dozen ______ city ordinances that were unnecessarily complicated and out-of-date.
a. feckless
b. empirical
c. byzantine
d. slovenly
e. pedantic

Figure 1: Example SAT sentence completion question [2].

ence. Using the data from Microsoft's Sentence Completion Challenge, we tried to see if we could generate better results.

In the later sections of the paper, we will first describe other studies tackling the sentence completion problem or related problems, and then describe the corpora we used for training, validation, and testing. We will then discuss the implementations of the bigram/unigram model (section 3.1), word embedding model (section 3.2), forward and backward LSTM models (sections 3.3, 3.4) and the bidirectional LSTM model (section 3.5). Finally, we provide a discussion of the results in section 4 along with challenges and future work in sections 5 and 6.

2. Related Work

Previous studies from the MSR Sentence Completion Challenge include using a fusion of local and global information to exceed 50% accuracy on this task (Zweig et al.) [5]. To put perspective to this benchmark, an average human scored 91% accuracy on a set of questions (Zweig and

Burges) [4]. Compared to the average human, the current state of the art models are still not even close to answering these questions with the same level of accuracy.

Apart from the MSR Challenge, there are several other tasks that are related to our work. In English Lexical Substitution (ELS) tasks, an algorithm is trained to find words that are able to substitute a target word in a given sentence [3]. This task has potential useful applications in summarization, paraphrasing, and question answering problems.

Other work in SAT include applying Latent Semantic Analysis (LSA), which was used in the above-mentioned MSR sentence completion study, to answer reading comprehension questions. The LSA model is based on capturing semantic relationships in high dimensional matrices and requires a very large data corpus [1].

3. Approach

For our technical approach, we implemented three models to solve the sentence completion problem. The first model is a basic n–gram model. The second model uses pre-trained word embeddings from the 6.864 problem set. The final model we consider is a Long Short Term Memory (LSTM) recurrent neural network.

3.1. Bigram/Unigram Model

The first model we implemented was a simple Bigram and Unigram model. During training, the model simply keeps track of the frequency of word occurrences as well as the frequency of transitions between pairs of words. We used the Holmes Training Data set provided by the Microsoft Sentence Sentence Completion Challenge. During testing, the model predicts the answer choice which has the greatest probability of transitioning from the previous word, which is computed using maximum likelihood on the transition frequencies as counts. If no transitions for any of the answer choices have been observed, then the model defaults to a unigram mode and predicts the most frequent word. If a word has not been seen before, it's assigned a frequency of 0.

3.2. Word Embedding Model

The next model we implemented used a set of pre-trained word embeddings for all of the words. This model conducts semantic analysis by using word movers distance to calculate the similarity of each of the answer options to the sentence the answers are trying to complete. Then, it chooses the most similar option as its prediction. The rationale behind this model is that the answer must be possible to derive from context clues in the sentence; so, the answer that is most similar is also most likely to be consistent with those context clues. In addition to word movers distance, we also tried using relaxed word movers distance and word centroid distance and achieved similar results.

3.3. Forward LSTM Model

Next, we used a forward feeding LSTM neural network. An LSTM is a RNN augmented with, among other things, remember and forget gates which allow it to remember relevant state for a longer number of cycles. Our LSTM takes in a one hot vector representing a word as an input and returns a softmax vector representing the probability of each word being the next word. To train the model, we used sentences from our WSJ corpus which contains on the order of a million words with a vocabulary size of around 50,000.

To get the results that we got, we trained the forward LSTM model using two hidden layers for one epoch, and then fed each of the 1040 partial question statements into the model. For each test sentence, the model predicts the answer which has the highest probability value in the last softmax output vector. If a word in the test sentence was not seen before, then we skip that word when entering the test sentence into the trained model. If an answer choice wasn't seen before in training, we set the probability of that answer to 0.

Because of the forward nature of the LSTM, this model will only account for words in the sentence before the blank we need to fill in. So, to take words from the other part of the sentence into account, we also created a backwards LSTM model.

3.4. Backwards LSTM Model

To implement a backwards LSTM model, we can simply change training and test inputs to the forward LSTM. For the training data, reverse the order of words and sentences in the entire corpus before feeding it into the training step. For the test data, we enter the words in the sentence backwards up to the blank that represents the answer we are to fill in. Other than these differences, the backwards LSTM is equivalent to the forwards LSTM.

While this model considers the words after the blank, it doesn't consider the words before. So, we then created a bidirectional LSTM which would consider both the words before and after the blank.

3.5. Bidirectional LSTM Model

For the bidirectional LSTM, the model first runs and saves the probability vectors from the forward and backwards LSTM. Then, for each test case, the model adds together the probability vectors from the forwards and backwards models and predicts the answer choice which has the highest probability in the new vector. Other than addition, we also attempted other methods for combining the probability vectors (e.g. multiplying probabilities). We discovered, though, that adding the vectors consistently provided the best results for both training and validation datasets.

4. Results

Before analyzing the results of our models, we calculate three baselines to evaluate our model (Table 1). The first baseline is random guessing, which has an expected 20% accuracy given that each question has 5 possible choices. The second baseline we use is based on the results of the Microsoft Research team. They achieved 44% accuracy with their n-gram language model and 42% accuracy with their RNN [4]. Finally, as the third baseline, human preformance achieves a baseline of over 90% [4].

Baseline	Accuracy
Random guessing	20%
MSR implementation	42–44%
Human performance	91%

Table 1: Baselines.

Model	Test	Validation
N-Gram	24.5%	27.9%
Word Embedding	29%	33%
RNN Forward	24.7%	21.6%
RNN Backwards	25.7%	22.1%
RNN Bidirectional	25.6%	22.5%

Table 2: Model Results.

As indicated by Table 2, each of the models that we implemented exceeded the random guessing baseline, though fell short of the state of the art models. The test set we used consists of the first 800 tests in the official MSR test set. The validation set we used consisted of the remaining 240 questions in the same MSR data. Our best model was the Word Embedding model which achieved an accuracy of 33% on the validation set. We believe that our results were negatively affected by a lack of data, particularly for the n-gram and RNN models. In particular, there are many entries in the RNN output data which are 0 indicating that the word was not seen in training, meaning that the words were not seen before and the model would not be able to make a good prediction. The code, as well as our output files, that was run to generate our results can be found at https://github.mit.edu/bwxu/6.864_project.

Now, we will do a more in depth analysis of each of our models.

4.1. Bigram/Unigram Results

Our bigram and unigram model scored 24.5% accuracy on our test set and 27.9% accuracy on our validation set. The test set is the first 800 sentences of the MSR data. The test set consists of the last 240 sentences from the MSR data. While this model was meaningfully above the random guessing baseline, it fell far short of the state of the art. As mentioned before, we believe this is due to a lack of training data. For many of the answer choices, the trained model either did not have any transition information on the word or simply did not see the word before at all. Because of how our n-gram model is implemented, it tends to disregard these answer choices, which severely affected the accuracy of the model.

4.2. Word Embedding Results

The word embeddings model scored 27% accuracy on our validation set and 30% accuracy on our test set. This is our best performing model. We found it strange that this model performed so much better than our RNN because in theory, a sufficiently trained RNN should learn the information provided by word embeddings. Again, the discrepancy we found is at least partly due to the insufficient amount of data we used for training. The model also outperformed the n-gram model, which is consistent with what we expected because the word embeddings model takes into account the semantics of the words as opposed to the n-gram model which only takes into account the ordering of the words.

4.3. Forward LSTM Results

The accuracy for forward LSTM was 24.7% on the test data and 21.6% on the validation data. This accuracy lower than we expected; it was outperformed by both the n-gram and word embeddings models. In addition to the lack of training data mentioned before, the subpar performance is probably also caused by the fact that we used an LSTM which was very simple, due to resource constraints. In particular, we only had 2 layers with 20 hidden units and only trained it for 1 epoch. If we increased the number of hidden units, layers, and trained it for epochs, the LSTM should perform significantly better. Unfortunately, we were limited in our computational resources and time so we could not test theory in practice. Also, the forwards LSTM only accounts for the words before the blank on the test questions. So, we implement a bidirectional LSTM, as discussed later, to account for this.

4.4. Backwards LSTM Results

The accuracy for backwards LSTM was 25.7% on the test data and 22.1% on the validation data. Similar to the forward LSTM, this model was outperformed by the n-gram and word embeddings models. We believe that the reasons for this is the same. This model only considers the words after the blank, and so we merged it with the forward LSTM to form a bidirectional model in hopes that considering the entire sentence would improve accuracy.

4.5. Bidirectional LSTM Results

The accuracy for the bidirectional LSTM on the test set is 25.6% and our accuracy for the validation set was 22.5%. We expected that this result would be better than the forwards LSTM and backwards LSTM, but the results it produces is about the same. When creating the bidirectional model, we tried different methods for combining the probability vectors form the forwards and backward directions trying methods such as addition, multiplication, etc. However, most of these operators seem to give us roughly the same results for the bidirectional LSTM, none of which was significantly better than a single direction LSTM.

5. Implementation Challenges

Throughout our experiments, there are a couple elements we could have changed to achieve better results.

1. **Lack of Training Data:**

   As mentioned earlier, one of our biggest challenges of solving the sentence completion problem is the lack of publicly available data to use for testing and training. The College Board specifically does not release any SAT problems for training or testing purposes so the only data we managed to find was from the MSR Sentence Completion Challenge data, which includes 1,040 sentences with a single word removed to act as a blank. This prevented us from using more specific learning related approaches for this task that would have required training data of this form.

   Our training data set for the LSTM was just the Wall Street Journal data set, which consists of approximately 50,000 sentences. If we had a larger set of training data, or supplemented the set with both the Brown corpus in the Penn Tree Bank-3 material and MSJ Challenge corpora, our LSTM would be more accurate, since more of the answer choices probably would have been seen during training. However, a larger set of training data would take much longer to run. If we had a larger set of training data, we would have more unique words in the English language to use in our prediction.

2. **Lack of Computational Resources:**

   Also, we were hindered by our lack of computational resources, especially when training the LSTM. The best option we found for training our LSTM on MIT Athena Cluster machines, since TensorFlow can only be used on Linux or Mac. These machines have a space quota which means we're limited in how much training data we can download. Furthermore, the communal nature of these machines, the model can only be trained while we are present and watching; so, this limits us to models which can train in the span of a couple of hours. With better computational resources, we would have been able to train larger LSTM models with more data for longer, which most likely would have resulted in better results.

6. Future Work

Of the models we made, we believe that the bidirectional LSTM has the greatest potential for improvement. For that model, we ultimately reached an accuracy of 25.6% using the bidirectional LSTM model and 29% using the word embedding model. Due to time constraints for training the model, we ran the model for only one epoch. Also, we used a network with only two hidden layers. We foresee that running the model for a greater number of epochs and a layers would give us a higher accuracy, since loss decays with a greater number of epochs. Similarly, tuning the learning rate would also increase accuracy.

We explored a few ways of combining the probability vectors for the bidirectional LSTM. However, there are some other ways we wish to have tried. For example, it's likely that the longer the partial sentence is, the more information it would contain to help complete the missing word. Hence, another way of combining the softmax layers in the forward and bidirectional models may be weighing each probability with the length of the part of the sentence it represents.

Also, we think finding and training on a larger dataset, such as the combined corpora of the Wall Street Journal, Sherlock Holmes, and the Brown corpus in Penn

Tree Bank-3, would have improved performance since the WSJ corpus only had a vocabulary of roughly 50,000 words, meaning that many of the 200,000 words in the english dictionary were not seen in training. Another possibility is editing the LSTM model so that it takes in word embeddings of words as input rather than one hot vectors. This way, the LSTM could better deal with words that it hasn't seen before.

7. Individual Contributions

7.1. Brian Xu

First, I wrote the code for the Bigram and Unigram model (NGram.py) and functions to read and parse the training and test data from the Microsoft Sentence Completion Challenge. Then, I wrote the code (word_embeddings.py) for the model which predicts answers based on word embeddings with the WMD, RWMD, and WCD metrics. Next, I changed the LSTM model in LSTM.py so that it would predict all of the test questions after training the model. Afterwards, I made a significant change to the LSTM to make it write output data into a file. Following that, I combined the data from the forwards and backwards direction by writing the script bidirectional_LSTM.py, which predicts the answer for a bidirectional model by combining the probabilities from the forwards and backwards directions. Finally, I helped write parts of the paper and cleaned up the git repo.

7.2. Heeyoon Kim

I helped to create a reader for the training data that would run through all the files in the Wall Street Journal corpus. I also fixed the reader for the test data to go with the LSTM model for our purposes. I then updated the structure of the model and reader, so that the LSTM model code is able to obtain the original sentences and word choices, rather than just the flattened list of word ids. I rewrote LSTM.py to include a proba variable, used it to predict the best answer choice, and made sure that this worked for an inputted test question. If the answer choice hadn't been seen during training, meaning that the choice didn't appear in the proba vector, I defaulted the probability to 0. Before we rewrote the code to handle multiple questions together, I ran multiple tasks to obtain initial accuracy results for the test data set. I wrote sections 3.3, 3.4, 4.3, 5 of the paper.

7.3. Michelle Huang

I created the initial reader.py and helped modify reader.py to take in additional files for our training data with Heeyoon and Sarah. I helped debug for the final LSTM in the forward direction after the base code was there to make sure we were getting the results in the format we wanted. I also created the slides used for the presentation and wrote the abstract, introduction, and results section of the writeup.

7.4. Xiaoxue Liu

Performed preliminary research about RNN and chose to use the LSTM model. Created the LSTM model using the TensorFlow tutorials and integrated it with our data reader. Implemented backward LSTM functionality. Modified the reader to use reversed sentences. Helped write reader functions for reading datasets. Ran experiments for forward LSTM. Contributed to general debugging and design decisions. Created figures for, contributed significant writing for introduction, related work, bidirectional LSTM sections of the writeup.

References

[1] C. Boonthum-Denecke, P. McCarthy, T. Lamkin, G. Jackson, J. Magliano, and D. McNamarab. Automatic natural language processing and the detection of reading skills and reading comprehension. *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference*, 2011.

[2] L. Express. 501 sentence completion questions. *LearningExpress, LLC*, 2004.

[3] D. McCarthy and R. Navigli. Semeval-2007 task 10: English lexical substitution task. *Proceedings of the 4th International Workshop on Semantic Evaluations (SemEval-2007)*, 2007.

[4] G. Zweig and C. Burges. The microsoft research sentence completion challenge. *Microsoft Research Technical Report*, 2011.

[5] G. Zweig, J. Platt, C. Burges, A. Yessenalina, and Q. Liu. Computational approaches to sentence completion. *ACL 2012*, 2012.