

Felipe Pazos (fap26), Michael D'Alessandro (msd248), Oscar So (ons4)
Group Name: Duck Duck Goose
Project 4
12/04/2019

Story Cloze Test

Part A: Applying NLP Knowledge

I. Description: Our system consists of the feed-forward neural network we had worked on in Project 3. However, Instead of the bag of words vector representation that was given to us in *ffnn1fix.py*, we decided to represent each x_i using word embeddings from spaCy's "en_core_web_lg" dataset. This means that each word will be a 1x300 dimension. To represent a given entry, we averaged the words for each sentence to create a 1x300 vector for each sentence. We then concatenated the five resulting vectors to attain a 1x1500 vector to represent the document. This will be done through calling `load_data()` from the main function, where it then passes the spacy word embeddings to `encode()` and `avg_vec()`, which averages the vectors of the sentence. We then train the FFNN on the training set until it reaches the max accuracy, set at 75%, for the validation set. From then on, we will use the trained model to predict the test results of the testing data. By comparing the results, if `ffnn(example[1]) > ffnn(example[2])`, then it would predict as 1, ie. the first random sentence is correct, meaning that the first sentence was the correct ending and vice versa for example 2 as the ffnn results compare the probability of whether the sentence is labeled as 0 or 1 for our feature set.

II. Feature Set: Our feature set consists of a binary feature that records whether a certain sentence was correct or wrong. Initially, we appended two training examples for each document - one with the correct ending, and one incorrect ending. The correct endings were labeled with a 1 and the incorrect ending labeled with a 0. However, we realized that the vectors would've been too similar if we had two data points that only have 1 out of 5 sentences that were different. Thus, we decided that we would only use the first column out of the two random sentence endings. For this, we took all documents ending 1 as the "correct" examples and documents with ending 2 as the "incorrect" example.

The code snippet below gives an in depth view of both our thought processes.

```
with open("train.csv", newline='') as f:
    reader = csv.reader(f, dialect='excel')
    for i, row in enumerate(tqdm(reader)):
        e = row
        if e[7] is "1":
            x = encode(e[1],e[2],e[3],e[4],e[5], nlp)
            train_data.append((x,1))

            #x = encode(e[1],e[2],e[3],e[4],e[6], nlp)
            #train_data.append((x,0))

        else:
            #x = encode(e[1],e[2],e[3],e[4],e[6], nlp)
            #train_data.append((x,1))

            x = encode(e[1],e[2],e[3],e[4],e[5], nlp)
            train_data.append((x,0))
```

Furthermore, we also think this will be effective because by looking at the average word embedding vectors, it provides a strict and insightful context of the paragraph preceding the 2 random sentences that we have to predict to follow after. We postulate that the correct sentence should be in some ways more closely related to the input 4 sentences than otherwise. Therefore, since the output sentence is part of the vector, the neural network can learn what weights to apply to the part of the vector to correctly predict whether it is part of the previous 4 sentences or not.

III. Extracted Features: Before we put our train.csv in the model, we did some feature extractions and preprocessing in order for the data to be more easily understood by our FFNN model. To start off, we used NLTK's stopwords set and removed all stopwords from our dataset. This is to remove any unnecessary training given that those words rarely have significant meaning when determining the closure of the paragraph. Furthermore, we also used NLTK's "PorterStemmer", which translates all words from its form into its stem form. For example, "programs", "programmer", and "programming" will all have the stem word "program". Since these words all are similar in vector form, we thought it would be useful to use the stems for consistency in values when we actually translate them into word embedding vectors. With this, we output the result from the jupyter notebook into the new "train.csv" and then do the last step inside the python file. Under the functions "encode" and "load_data", we include the method that takes the average vectors of the particular sentence, given the paragraph has 4 sentences and 2 probable sentences to follow it, and for the two probable paragraph combinations, we label them 0 and 1, legend is talked about above. Furthermore, after averaging the word embedding vectors in the sentence, we concatenate the vectors into a giant vector of length 1500.

Code snippet of preprocessing:

```
def final_preprocessing(df_test):
    stop_words = set(stopwords.words('english'))
    ps = PorterStemmer()
    id_list = []
    results_list = []

    for i in tqdm(range(df_test.shape[0])):
        sentence1 = word_tokenize(df_test.at[i, 'InputSentence1'])
        sentence2 = word_tokenize(df_test.at[i, 'InputSentence2'])
        sentence3 = word_tokenize(df_test.at[i, 'InputSentence3'])
        sentence4 = word_tokenize(df_test.at[i, 'InputSentence4'])
        random1 = word_tokenize(df_test.at[i, 'RandomFifthSentenceQuiz1'])
        random2 = word_tokenize(df_test.at[i, 'RandomFifthSentenceQuiz2'])

        filtered_sentence1 = [ps.stem(w) for w in sentence1 if not w in stop_words]
        filtered_sentence2 = [ps.stem(w) for w in sentence2 if not w in stop_words]
        filtered_sentence3 = [ps.stem(w) for w in sentence3 if not w in stop_words]
        filtered_sentence4 = [ps.stem(w) for w in sentence4 if not w in stop_words]
        filtered_random1 = [ps.stem(w) for w in random1 if not w in stop_words]
        filtered_random2 = [ps.stem(w) for w in random2 if not w in stop_words]

        df_test.at[i, 'InputSentence1'] = ' '.join(filtered_sentence1)
        df_test.at[i, 'InputSentence2'] = ' '.join(filtered_sentence2)
        df_test.at[i, 'InputSentence3'] = ' '.join(filtered_sentence3)
        df_test.at[i, 'InputSentence4'] = ' '.join(filtered_sentence4)
        df_test.at[i, 'RandomFifthSentenceQuiz1'] = ' '.join(filtered_random1)
        df_test.at[i, 'RandomFifthSentenceQuiz2'] = ' '.join(filtered_random2)
```

Before:

Out[103]:

	InputStoryId	InputSentence1	InputSentence2	InputSentence3	InputSentence4	RandomFifthSentenceQuiz1	RandomFifthSentenceQuiz2	AnswerF
0	138d5bfb-05cc-41e3-bf2c-fa85ebad14e2	Rick grew up in a troubled household.	He never found good support in family, and tur...	It wasn't long before Rick got shot in a robbery.	The incident caused him to turn a new leaf.	He is happy now.	He joined a gang.	
1	bff9f820-9605-4875-b9af-fe6f14d04256	Laverne needs to prepare something for her fri...	She decides to bake a batch of brownies.	She chooses a recipe and follows it closely.	Laverne tests one of the brownies to make sure...	The brownies are so delicious Laverne eats two...	Laverne doesn't go to her friend's party.	

After:

Out[105]:








	InputStoryId	InputSentence1	InputSentence2	InputSentence3	InputSentence4	RandomFifthSentenceQuiz1	RandomFifthSentenceQuiz2	An
0	138d5bfb-05cc-41e3-bf2c-fa85ebad14e2	rick grew troubl household .	He never found good support famili , turn gang .	It n't long rick got shot robberi .	the incid caus turn new leaf .	He happi .	He join gang .	
1	bff9f820-9605-4875-b9af-fe6f14d04256	lavern need prepar someth friend 's parti .	she decid bake batch browni .	she choos recip follow close .	lavern test one browni make sure delici .	the browni delici lavern eat two .	lavern n't go friend 's parti .	

Final Preprocessed Training Example - Word Embedding and Feature Set Labelling:

(tensor([0.0808, 0.3055, -0.0204, ..., 0.0946, -0.0471, 0.0919]), 1)


IV. Analysis: We believe that our system works pretty well given the fact that our cut-off for training was 75% and the neural network reached that level in around 100 epochs. We have also tried, instead of breaking out of the program, to continue training, and the results on the validation set have been to as high as 85%. However, we decided to stick with our 75% benchmark as overfitting and more training might not conclude the best results for the testing dataset. We think that removing stop words definitely helped the speed of our computation and prevented any unnecessary from being included in our vectors. Furthermore, we think that stemming was really helpful as we would get more accurate vectors out of words that had the same meaning with different endings. Due to the way our model is set up, where it stops learning after the validation accuracy has reached above 0.75, our final validation accuracy score was 0.750425. One minor problem we found with vectorization of our sentences is that when the model predicted a particular sentence for the development set: “Clare wanted to go for a walk. So she went outside and walked around her block. But the sun began to rise. And Clare began feeling hot.” The choices were: “She took her jacket off to cool off.” versus “Clare wished she brought a warm jacket.” The correct response is the first, but the model gave us the later as the correct ending. This could be because the word “hot” in the last input sentence is completely opposite from the word “cool” in the second random sentence. Thus, the algorithm picks the first random sentence since it also has a word associated with a hot temperature, “warm”. This could’ve been one reason why the Part 1 model incorrectly picked the second random sentence. In this case, we think that BERT would do better because it is bidirectional and it relates the subjects together with the words in the sentence. In other words, it doesn’t simply look at the word embeddings itself and tries to apply deeper contextual understanding with the prior paragraph. Given we reached >71% on the test set, we think that our idea was sufficient for this project and doesn’t need more work.

V. Kaggle Submission:

#	Team Name	Notebook	Team Members	Score ?	Entries	Last
📍	human performance			1.00000		
1	sea97ddr42		 	0.73490	6	12d
2	Mahayana Gang		 	0.72207	13	10h
3	Wes Tan			0.72100	8	9h
4	Duck Duck Goose		 	0.71726	5	20h

Your Best Entry ↑

Your submission scored 0.71726, which is an improvement of your previous score of 0.54890. Great job!

 Tweet this!

Part B: Fine-tuning Pre-trained Models

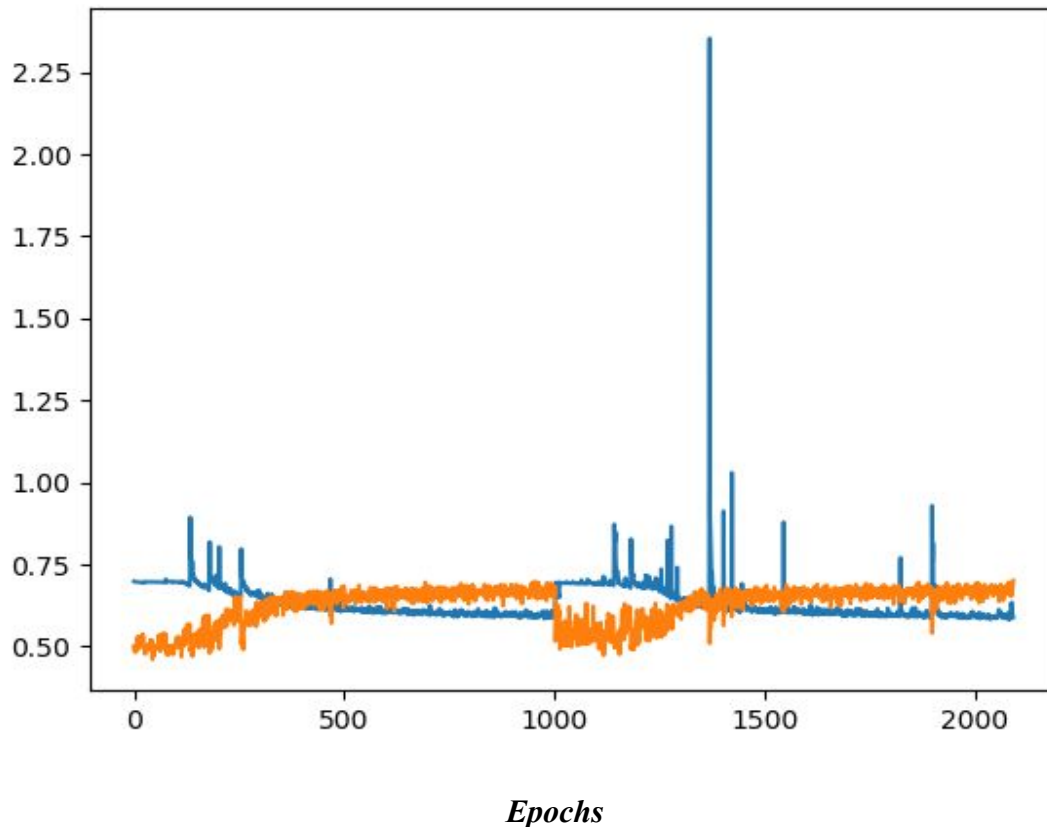
I. Description: For Part B, we utilized the BERT transformers from HuggingFace's transformer python library as well as a feed forward neural network to classify the training documents. To create the training and development set, each document would first be encoded using a BertTokenizer object. The first four sentences would be passed in together as the first BERT sentence, and the final sentence would be passed in as the second sentence. Once an example is tokenized, it is passed into a BERT model to encode. At this stage, we take the BERT model's pooler output and use this as the features to train on. Unlike for Part A, the training and development set have two examples for each document - both the correct and incorrect completion to the sentence. The correct example is encoded and labeled 1, while the incorrect example is encoded and labeled 0.

Then, the training and development set are used to train a FFNN with input dimension of 768, hidden dimension 32, output dimension 2, learning rate of 0.1, and a LeakyRELU activation function. The model is set to train until the development set accuracy reaches 0.75. At this point, the model stops training and evaluates on the test set.

To evaluate on the test set, each document is first encoded with both possible endings. The resulting pooler output for each possible ending is then passed into the FFNN. However, we aren't able to argmax the outputs to check whether the ending is correct or not like we can during training. To predict which of the two inputs is more correct, we have to look at the second element of the FFNN output vector. This corresponds with how confident the FFNN is that the prediction is correct. Whichever output vector has a higher value in this field is the one the FFNN is more confident is correct. This is the output we choose.

II. Learning Curve Plot:

Loss is shown in blue, accuracy in orange.



III. Analysis: The system we implemented in part B works well, having reached a validation accuracy of over 70% in 1390 epochs. However, while part B took more epochs to reach a similar validation accuracy as part A, it is important to note that the epochs were trained and tested relatively quickly, and this did not take much longer in terms of computing time to train.

```
Validation completed for epoch 1390
Validation accuracy for epoch 1390: 0.7024456521739131
Validation time for this epoch: 0.19308209419250488
```











Screenshot of Validation Accuracy for Part B

We decided to stop at 70% because we did not see any significant increase in validation accuracy at higher epochs, and the validation accuracy actually began to decrease after a certain extent. Similar to part A, we took a deep dive into the types of sentences that our model in part B was getting routinely incorrect on the dev set in order to see ways it could be improved. It appears that our model in part B did a good job at getting contexts in a sequence bidirectionally, which was accomplished with BERT, but got more complex sentences wrong, especially when they

involved negation. For example, take the following sentences: “May invited her friends for a picnic in the park. After she arrived, she realized she had forgotten the ice! Now none of the drinks would be cold. Luckily, her friends had a sense of humor.” with the two responses being “They filled their cups with ice.” and “They had a wonderful picnic, despite the warm drinks.” The correct answer is clearly the second response, but our model chooses the first response. We believe this has to do with “ice” in the first response being similar to the given sentences. Additionally, we think the model had trouble understanding the negation that the people at the picnic did not have ice, and picked the answer even though it involved ice. Despite this, we believe this example was advanced and would be difficult for most language models.


IV. Comparison with Part A: Our system implementation for Part 1 is definitely way quicker and the encoding is more efficient. However, the representation for Part 1 is less efficient for space because the vectors are larger. One problem we found in Part 1’s system is that when the model predicted a particular sentence for the development set: “Betsy comes home late one night and wants to eat something simple. She notices that she has spaghetti and sauce in the house. She quickly cooks and eats a pot of spaghetti. After eating, she falls asleep on the couch.” The choices were: “She was full and happy.” versus “Besty likes cheeseburgers.” The correct response is the first, but the model gave us the later as the correct ending. This could be because given that the words in the previous sentences mainly consisted of food words like spaghetti, sauce, and cook, the vectors tend to lean more towards the <food> category. In this new fine-tuned BERT model with FFNN, it tended to work better on sentences like these and the one written above in Part 1’s response because the bidirectional BERT model looks at the entire paragraph and links all possible words with each other. By applying this deeper context of the paragraph in a 768 x 1 vector for this Part 2 model, the model definitely learned the correct results quicker and more space allocatively efficient even if it took longer to reach the validation accuracy rate we set at 75% in the long run.

V. Kaggle Submission:

16	Courtney McBeth		0.75521	4	2d
17	snoww	  	0.75360	3	2d
18	MusicNerds	  	0.74986	5	10d
19	Duck Duck Goose	  	0.74665	4	2d

Your Best Entry ↑

Your submission scored 0.74665, which is an improvement of your previous score of 0.52699. Great job!

 [Tweet this!](#)

Part C: Team Contributions

Every team member worked on a significant part of this project as we all tried coding and implementing different aspects of the assignment. We met up multiple times and called each other to work on the code and report over Thanksgiving break. We also made sure that we understood each other's code before the submission. Each task was evenly shared across the team. For example, we had one person try working on preprocessing, another work on Part A, and another person working on Part B. We all met up to work on the report and submitted it together. Altogether, we were very efficient and good partners. We all shared in the writing of the report.

Part D: Feedback

Overall, we found that this project was of perfect difficulty. It was a great relief that this project didn't require as much time, effort, and stress as project 3 had given us. We thoroughly enjoyed the project and actually learned more about word embeddings, preprocessing, feature extraction, and neural networks. We spent no more than 25 hours cumulative across all members on this project for coding and no more than 3 hours writing the report.