APPRENTICE

UDACITY CAPSTONE PROJECT REPORT (MLEND)

Definition

A Teacher plays a very important role in the life of a student. It is the teacher who is responsible for how the student turns out, later in his/her life. However, a teacher spends a lot of time assessing and analyzing the answer sheets of the students from a test. Wouldn't it be better if a teacher is able to spend this time with the students instead? To solve, this very problem, I introduce Apprentice which is a service which automatically fact checks the student's answer against the teacher's given solution. It can return the number of facts that were correct in the student's answer out of the total facts in the solution given by the teacher.

This problem can be modelled as a semantic text similarity problem with binary labels (0 or 1). On a high-level, I approached this problem by using pretrained embeddings from the Universal Sentence Encoder (USE) and a two-input neural network with a sigmoid output to classify whether two sentences are semantically similar or not. I used the Quora Question Pairs dataset to train and evaluate my model. Based on the vocabulary of the dataset the model was trained on, apprentice can perform test assessment of subjects without exotic terms i.e. subjects except sciences, like English and Social Sciences.

I evaluated my model on four metrics- accuracy, precision, recall and F1.

Accuracy = no. of correct predictions / total predictions

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F1 = (2 * Precision * Recall) / (Precision + Recall)

[TP = True Positives, FP = False Positives, FN = False Negatives]

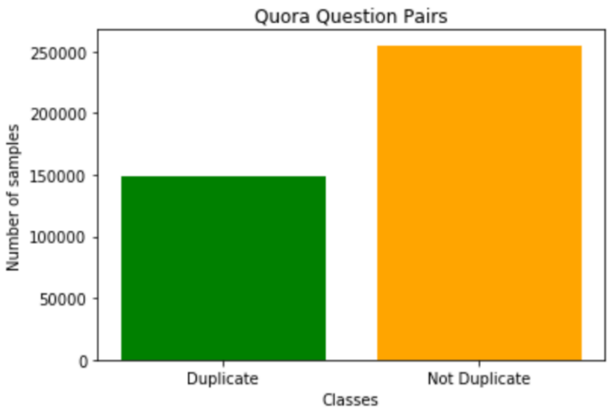
Intuitively, precision is the ability of the classifier not to label as positive a sample that is negative and recall is the ability of the classifier to find all the positive samples.

My model has an F1 score of 89% with a recall of 93%, thus reducing false negatives. Detailed analysis of the results of the evaluation can be found later in the report.

Analysis

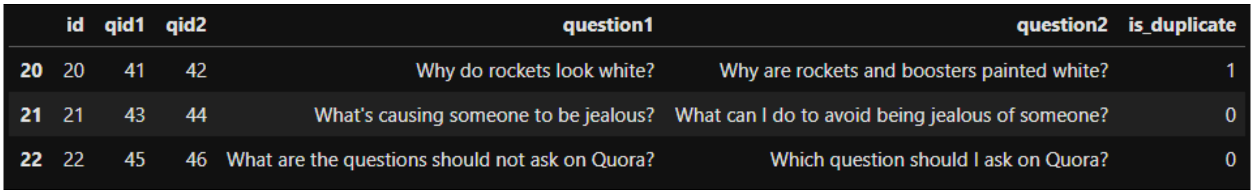
Data

I used the Quora Question Pairs dataset for training my sematic text similarity model. The dataset primarily contains a pair of text with a target binary value stating whether the text is duplicate or not (duplicate=1, not duplicate=0). The dataset contains more than 400k samples. Out of that 400k samples 1/3 of the samples are positive and the remaining are negative samples. A visualization can be seen beside.



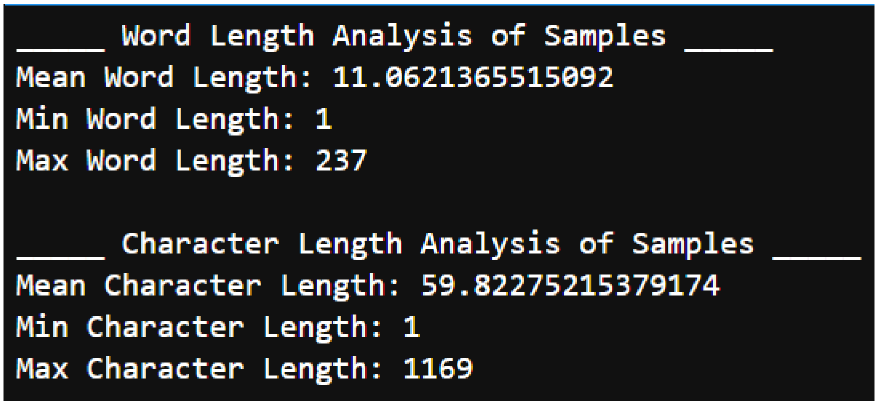
To balance the data, I randomly sampled half of the negative samples and concatenated it to the positive samples to create a balanced dataset. I split the entire dataset into training, validation and testing sets. The training set consists of approximately 200k samples, validation contains 90k and testing contains another 200k samples. There were a couple of nan samples so I dropped those rows.

A sample from the dataset can be seen below:



	id	qid1	qid2	question1	question2	is_duplicate
20	20	41	42	Why do rockets look white?	Why are rockets and boosters painted white?	1
21	21	43	44	What's causing someone to be jealous?	What can I do to avoid being jealous of someone?	0
22	22	45	46	What are the questions should not ask on Quora?	Which question should I ask on Quora?	0

The descriptive statistics on the word and character lengths of dataset are as follows:



```
_____ Word Length Analysis of Samples _____
Mean Word Length: 11.0621365515092
Min Word Length: 1
Max Word Length: 237

_____ Character Length Analysis of Samples _____
Mean Character Length: 59.82275215379174
Min Character Length: 1
Max Character Length: 1169
```

As you can see above, on an average the length of my samples at the word level is 11, meaning most of my sentences are quite short.

A small note here, in my initial capstone proposal, I proposed that I would be gathering data from real world answer sheets, however, I was unable to gather large enough data to be usable for my model. Also cleaning the data manually from the real world was not possible for me to do single-handedly. So, I instead used the QQP dataset.

Algorithm

I used Google's Universal Sentence Encoder to generate features/embeddings for my text. Universal Sentence Encoder encodes text into high dimensional vectors of size 512, that can be used for text classification, semantic similarity, clustering and other natural language tasks. USE comes in two variations of the encoder model, one is a Transformer Model and other is a Deep Averaging Network (DAN). The encoder model used in this project is the DAN. In the DAN based encoder, input embeddings for words and bi-grams are first averaged together and then passed through a feedforward deep neural network (DNN) to produce sentence embeddings.

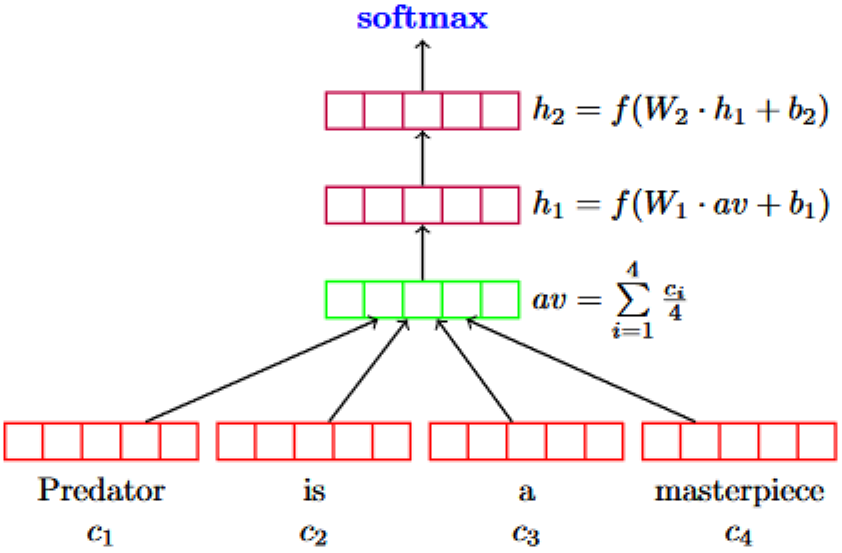
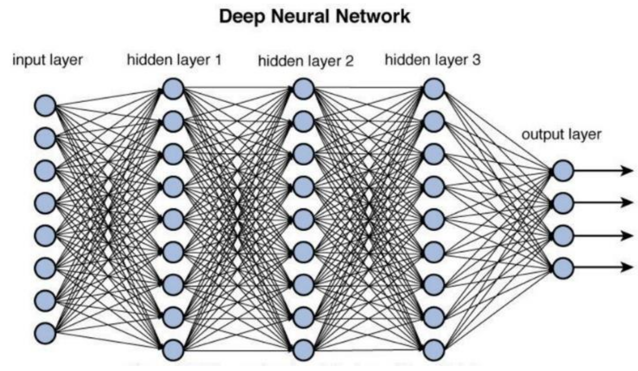

softmax
$h_2 = f(W_2 \cdot h_1 + b_2)$
$h_1 = f(W_1 \cdot av + b_1)$
$av = \sum_{i=1}^{4} \frac{c_i}{4}$
Predator
c_1
is
c_2
a
c_3
masterpiece
c_4

Fig. Deep Averaging Network (DAN)

A Deep Neural Network is basically a deep multi-layer perceptron model. A perceptron is a model inspired by the biological neuron which basically, performs a mathematical operation like Wx + b where x is the input, W is the weight, and b is t he bias.



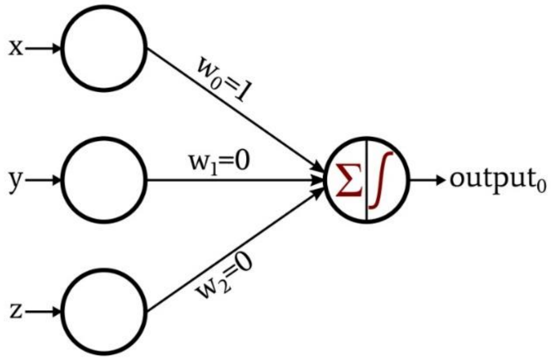


Fig. Deep Neural Network

Fig. Perceptron

After using USE to get the features, I built a two-input neural network with a single sigmoid output. The network primarily consists of 2 input layers, 5 hidden layers and 1 output layer. The embedding of each sentence in the pair is put through the two input layers, which passes through a dense layer to output a vector of size 256. The two 256 sized vectors from the two inputs are concatenated together and passed through a series of dense layers (with ReLU activation) to get a single value scaled between 0 and 1 because of the sigmoid activation at the last layer. The full network architecture can be seen in the image below.

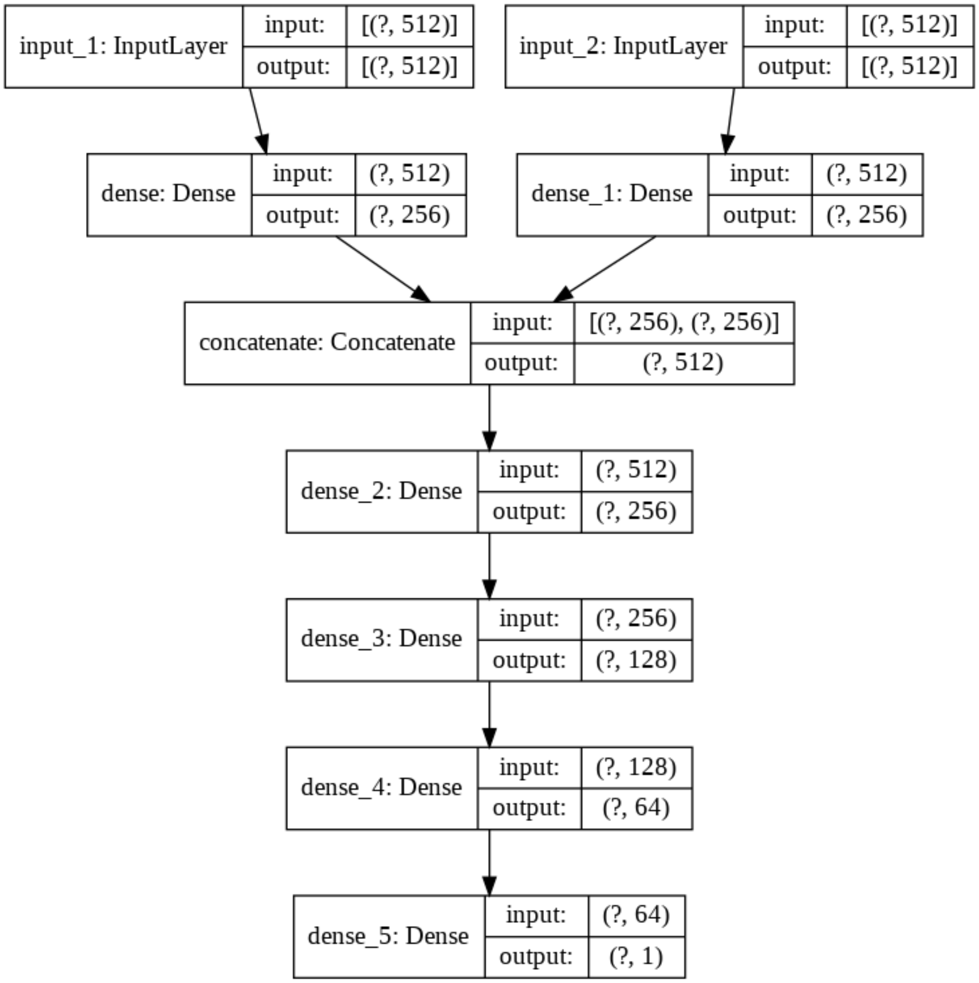


Fig. My DNN Architecture

Benchmark

To benchmark the results of my model, I compared my results to the three different kinds of models demonstrated in this blog post https://towardsdatascience.com/finding-similar-quora-questions-with-bow-tfidf-and-random-forest-c54ad88d1370 by Susan Li. She used three techniques to train her model on the Quora Question Pairs dataset and in her subsequent article she tested with word2vec too. The benchmark results on the validation set are as follows:

1. Bag of Words + XGBoost
    a. F1 Score = 75%
2. N-Gram level TF-IDF + XGBoost
    a. F1 Score = 67%
3. Character level TF-IDF + XGBoost
    a. F1 Score = 80%
4. Word2Vec + XGBoost
    a. F1 Score = 77%

Implementation

Data Preprocessing

As stated earlier, I used the Universal Sentence Encoder to generate the appropriate features/embeddings for my dataset (pairs of sentences). As USE is a pretrained encoder, I didn't need to perform any explicit preprocessing on the text.

Implementation

The results from my final solution are pretty reasonable with a F1 score of 89%. The steps I took to implement the algorithm and build the model are as follows:

1. After splitting the dataset into training, validation and testing sets, I encoded the X data i.e. the pairs of sentences using USE into fixed length vectors, each sentence of size 512. This process of getting the features/embeddings took around 1.5 hours on my Nvidia 1050Ti GPU. Then I saved my processed data which was in the format of NumPy arrays to .npy files so that I need not encode my data every time I start my notebook.
2. Then, I built my neural network model with the configuration as shown in the previous pages using Tensorflow 2.0's Keras API. I compiled the model with binary crossentropy loss and RMS Prop optimizer and accuracy as the metric for training. I also added some callbacks like tensorboard and early stopping.
3. After building the model, I trained my model for 20 epochs and reached a validation accuracy of 88% at the end of 20 epochs. And then I saved the weights of my model later.
4. I evaluated my model on the test set and the results of my metrics were:
    a. Accuracy: 88%
    b. Precision: 85%
    c. Recall: 93%
    d. F1 Score: 89%

    The detailed discussion of the evaluation can be found further along in the report

5. After the evaluation, I created a new notebook for inference which also acted as my backend server for the web app I built using Anvil. The communication between the notebook and the web app was made with anvil-uplink.
6. I created a predict function to make predictions given a pair of strings. It encoded the two strings using USE and used my trained model to give the result as a floating-point number in the range of 0 to 1 which can be later rounded off to absolute integer values.
7. To make the predict function usable in the real world, I created a check function which checks a given student's answer with the teacher's solution by matching every sentence in the answer to every other sentence in the solution using the predict function. This ensures that every point in the solution is covered by the student's answer. The function returns the number of facts/points correct out of the total number of points/facts in the teacher's solution.
8. The check function is connected to the web-app via anvil-uplink. The web-app calls the check function and passes the text entered by user as arguments and the response by the check function is displayed on the web-app to the user.

Refinement

The implementation described above was not my initial approach. I had to change my model architecture several times. I initially, modified the Short Answer Scoring (SAS) Dataset by transforming the dataset into a multi-label text similarity format by creating pairs of texts with one being the answer and the other being the solution and the target being an integer from 0 to 2. I used BERT for Sequence Classification in PyTorch using the Transformers Library by Hugging Face on that dataset. However, after trying different parameters in my Bert based model, there was something wrong as the loss stagnated at 0.6 even after several epochs and the accuracy was exactly 50%.

So, to find what went wrong, I modified the model architecture to a Siamese like model with pretrained BERT model giving the embeddings and then calculating the cosine similarity. After visually examining the distribution of cosine similarities of the SAS dataset, I found out that distribution is not separable even for a human. This made me realize the dataset needs a lot of cleaning which seemed very time consuming.

So, I changed my dataset from SAS to QQP which had a much better separable distribution. Also, I found out that Universal Sentence Encoder was better at generating embeddings for variable length sentences, So I switched to USE for feature extraction and ended up with the current model architecture which is also similar to a Siamese like network but without any intermediate distance calculations and instead directly feeding it through a neural network.

Results

I evaluated my model on four metrics i.e. Accuracy, Precision, Recall and F1.

Accuracy = no. of correct predictions / total predictions

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F1 = (2 * Precision * Recall) / (Precision + Recall)

Intuitively, precision is the ability of the classifier not to label as positive a sample that is negative and recall is the ability of the classifier to find all the positive samples.

Evaluating my model on the test set gave me the following results:

1. Accuracy = 88%
2. Precision = 85%
3. Recall = 93%
4. F1 Score = 89%

Comparing with the benchmark scores discussed in the Analysis section, it is clear that there is an improvement by a good margin. My model has a pretty good recall, so it is less likely that a student's answer is falsely marked negative. But the precision is a bit on the lower end, meaning it is slightly prone to falsely giving good marks. The overall F1 score is pretty balanced.

As the dataset isn't domain specific and doesn't contain a lot of exotic terminology as found in science-based subjects like biology, chemistry, physics etc. this model performs best at subjects which use much common words in its answers and solutions like English, Social Sciences etc. Based on real world testing, the current model is suitable for Grades 8 and below. With more domain specific data, I believe these results can be further improved and generalized to higher grades and wider range of subjects including science.

Conclusion

The current model is just the first step in fully solving the problem. Little human intervention is still required to keep a check on the models. Collecting domain specific data will surely improve the results and help generalize to a wider range of subjects. But my model works reasonably well with subjects like English.

Although there is still room for improvement in the application by adding support for OCR and creating a dashboard to create an answer wise performance report of the student and giving much detailed insights. This model is just the beginning in turning it into a full stack EdTech product for schools and educators.

Finally, I would like to conclude that my current solution gives adequate results to solve the problem for subjects other than science.