Charlie Furrer - cfurrer@stanford.edu
Priya Chatwani - priyac5@stanford.edu
Charles Vidrine - cvidrine@stanford.edu

Paragraph-Level Feature Engineering

Overview

Motivation: Our project focuses on the SQuAD 2.0 dataset to perform question answering given a context paragraph and various questions. Drawing on Chen et al.'s work on the Dr.QA model, we sought to produce better word embeddings via feature engineering.

What We Built: We added character embeddings, vectorized TF-IDF scores, and exact matching indicator variables to the vanilla word embedding vectors fed into the encoding LSTM.

Results: Our model performed best with character-level embeddings and TF-IDF scoring. We were unable to successfully implement exact match.

F1: 61.925 EM: 58.225

Data

Data: We used the SQuAD 1.0 reading comprehension dataset. The SQuAD dataset, presented in 2016 by Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang, revolutionized the domain of question-answering by providing 100,000+ questions posed on Wikipedia articles, in addition to the text segments containing the corresponding answers. We used the SQuAD 1.0 dataset to train and evaluate our model.

Retrieval: Our data retrieval methods did not differ from that of the baseline model given to us.

Baseline: For our baseline model, we used the Bidirectional Attention Flow (BiDAF) model given to us by the default project without a character embedding layer.

Approach & Experiments

Character Embedding: The character embeddings are created using a Linear Embedding matrix. Since the resultant embeddings have an extra dimension, we flattened the vector accordingly.

This model took longer to train due to the increased number of parameters and size of the embeddings. In return, we saw significant improvement in our accuracy.

TF-IDF Scores: Term Frequency - Inverse Data Frequency (TF-IDF) reveals how important a word is to a document within a collection of documents. TF-IDF score is usually used within the context of page ranking for search results. We had the original idea to use a slightly modified version of it as a feature for our embeddings.

TF-IDF is mathematically defined as TF * IDF where:

$TF = tf_{d,i} * 1 / \sum_d tf_{d,i}$
$IDF = \log(N / df_i)$

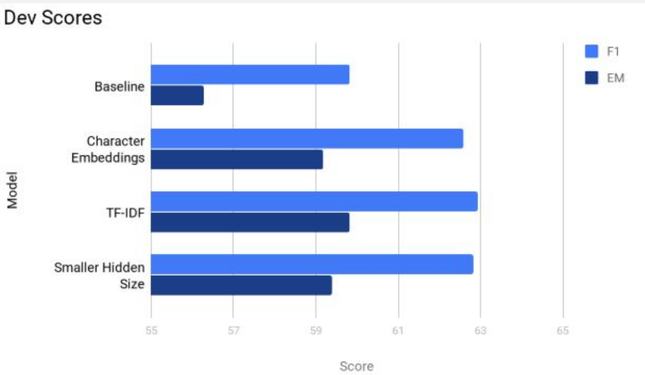
Where $tf_{d,i}$ is the term frequency of term i in document d, N is the total documents in the corpus, and df_i is the number of documents that word i appears in across the entire corpus.

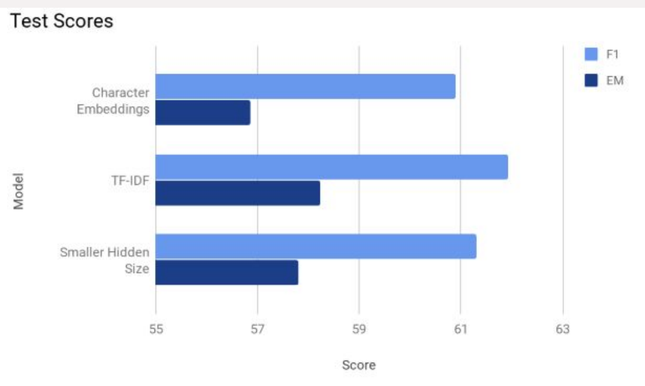
To produce the TF-IDF scores, we stored the total number of documents and document occurrences in a dictionary during preprocessing. Then, we used a vector the length of our vocabulary to map word indices to relevant TF-IDF information. We used Pytorch's batch indexing capabilities to retrieve the information at training time, allowing for much faster training than a typical iterative approach.

This model required more epochs to converge, leading to a much longer training time.

Exact Match: We attempted to implement exact matching, which tagged words in the context that also appeared in the query. However, the exact match feature actually made performance significantly worse compared to the baseline.

Results





Baseline Results:

F1: 59.82 EM: 56.28

Character Embedding Test Results:

F1: 60.9 EM: 56.87

TF-IDF Results:

F1: 61.925 EM: 58.225

Smaller Hidden Size: After implementing our desired features, we proceeded to tune the hidden size of the model to see if we could make improvements. By using a hidden size of 75 instead of 100, we were able to achieve the following scores on the test set:

F1: 61.3 EM: 57.8

Analysis: As you can see, our best performing model was the model that included the TF-IDF feature in the embeddings. However, we found that the TF-IDF model with a hidden size of 75 instead of 100 still outperformed the character embedding model and was only slightly worse than the TF-IDF model. This model was able to train much faster, taking 20 hours to reach convergence as opposed to the 40+ hours that we saw for the other models. Thus, we showed that even in reducing the complexity of our model, we were still able to train a model that learned from the character-level and TF-IDF features we added. Even though it performed slightly worse, this trade-off made it the best model in our opinion.

Discussion & Future Work

Main Findings: Although the implementation of TF-IDF was non-trivial in terms of time, space, and complicated vectorized indexing, the outcome was satisfactory. Thus, our results show that while TF-IDF scoring is traditionally used in document retrieval, it is also helpful in context-specific query retrieval.

Feature Engineering: Due to our focus on feature engineering, we did not get a chance to change the model or tune as many hyper parameters as we would have liked. In order to accurately measure the effects of each feature, we had to preserve the model across training and testing. Given more time, we would have complemented our feature engineering with the tuning of various hyper parameters so that we could better optimize our F1 and EM scores.

Additional Features: In a future iteration of this project, we would like to adjust our exact match implementation and incorporate more context-specific features, like in Dr.QA. For example, we could have trained on noun and pronoun tagging, named entity recognition, lemma matching, etc.

We also would add an interaction term to our model, such as the TF-IDF score times the Exact Match score. This may have helped the model learn that rare words found in both the document and query are more relevant than common articles of speech found in both.

References

[1] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. *Reading Wikipedia to Answer Open-Domain Questions*. CoRR abs/1704.00051, 2017.

[2] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. *Squad: 100, 000+ questions for machine comprehension of text*. EMNLP, 2016.

[3] Luong, M., and Manning, C. D. *Achieving open vocabulary neural machine translation with hybrid word-character models*. CoRR abs/1604.00788, 2016.

[4] Ramos, J. *Using TF-IDF to Determine Word Relevance in Document Queries*. Rutgers University, 2003.