Finding Quora questions pairs with neural networks

Cody Ray Hoeft
School of EECS
Oregon State University
Corvallis, OR 97331
hoeftc@oregonstate.edu

Padraig Gillen
School of EECS
Oregon State University
Corvallis, OR 97331
gillenp@oregonstate.edu

Abstract

Anybody can ask and answer questions on Quora, the question answer site. With 100 million monthly users, questions with the same intent are frequently asked. Questions with the same intent can lead to more overhead for Quora, readers, and writers. Quora challenged the Kaggle Community to find a better way to identify question pairs. We started with a starter Kernel from the Kaggle Community and experimented with changing the word embedding and trying different neural networks.

1 Introduction

The goal of the Quora question pairs challenge was to determine if two questions were duplicate, that is they shared the same intent. Quora provided a training set of over $.4$ million questions, each labeled as a duplicate or not a duplicate. Quora also provided a testing set of over 2.3 million unlabeled questions of for judging the competition. The goodness of an algorithm was evaluated by log loss. Kaggle returns the log loss as a score after uploading the predicted labels for the test data.

We started with a Long Short-Term Memory (LSTM) neural network using word2vec embeddings based on Python code from Kaggle user lystdo ("LSTM with word2vec embeddings") but with some of the code for cleaning text removed. This achieved a score of 0.30907, but we wanted to know if we could do better with other word embeddings, a Gated Recurrent Unit (GRU) network, or Convolutional Neural Network (CNN).

We were unable to improve performance much beyond the initial code, but we had the opportunity to experiment with different approaches and found that given the same features most yielded comparable results. We found that Convolutional Neural Networks performed surprisingly well even when only using the letters of the question (without any pretrained model).

2 Approach

2.1 Preprocessing

Various methods of preprocessing the questions were investigated. To be used with word embeddings, words must be identified within the sentence. The simplest preprocessing is to split the sentence into words using white space. The weakness is that some punctuation can cause words to be different even though they are the same; for example, "love?" and "love" should be interpreted as the same word. Striping punctuation. Replacing punctuation with spaces (except for apostrophe) makes words touching punctuation the same. The capitalization of words can also cause words to be different, so words are made lowercase. Stop words are words so common that they carry no extra meaning and can be removed from the sentence. Additionally, some words have the same roots but are syntactically different, i.e., 'remove' and 'removing'. Stemming can be used to convert both words to the same

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

word. Additionally, many words in the training and testing set are misspelled. Automatic spell checking could correct the misspelled words.

2.2 Different Word Embeddings

Text data by itself doesn't contain much meaningful data. Word embeddings map words to vectors that have more meaning than the text itself. Building embeddings requires extremely large corpuses because they are finding a mapping from sparse data to dense data. Because the provided training and testing data are too small to train embeddings, we did not attempt to train one from scratch. Instead we tried using pretrained models including a model trained on a Google News corpus and Glove embeddings trained on a variety of sources.

Attempts were also made to improve the embeddings themselves. The embeddings contain both capitalized words and lowercase words. For example, "Trump" and "trump" both exist in the Google News embeddings, but 'Trump' will have occurred more in the corpus and therefore the vector should have more meaning. Since preprocessed sentences can never contain capital words, we can remap "Trump" to "trump" and discard the original "trump". Likewise, words that cannot possibly exist in the preprocessed sentences can be discarded to improve load time of the embeddings.

2.3 Different Neural Networks

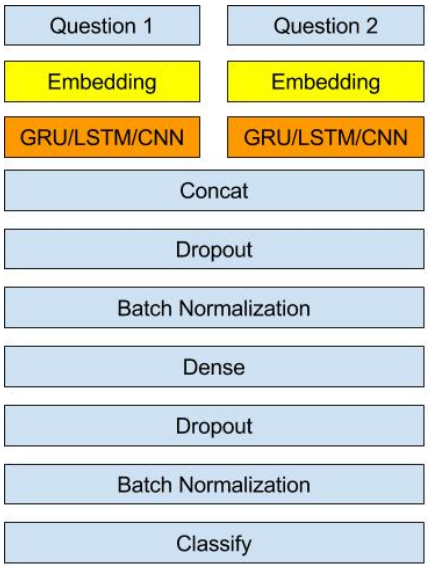

Figure 1: Outline of the structure used for GRU, LSTM, and CNN neural networks

The structure used in Figure 1 was used to support three distinct types of neural networks which replace the orange block in the neural network.

A Long Short-Term Memory (LSTM) network is a typical way to perform sentiment analysis on text. An LSTM is a reasonable choice for this problem because recognizing similar intent in a question is comparable to sentiment analysis.

A Gated Recurrent Unit (GRU) network is like an LSTM with similar performance but has fewer internal parameters to train.

A Convolutional Neural Network (CNN) is a network made of filters that is commonly used in image processing. A CNN is a reasonable choice for this problem because it has the potential to extract similarities in the sentences in various parts of the sentences.

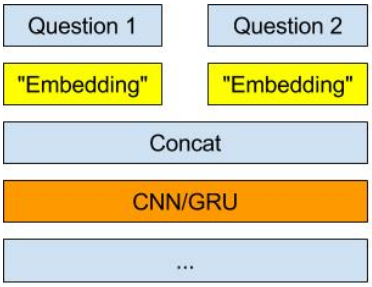

Figure 2: Outline of the structure used for letter-wise neural networks

In addition to the original structure, which used word embeddings. CNN and GRU neural nets were also attempted over letters instead of words. As shown in Figure 2, the location in the neural network where the two paths were concatenated was also moved before the CNN layer. To enable the encoding of the letters a 2 dimension word2vec instance was trained from scratch using each letter as a word. After the embedding each letter, the 2d vectors for each letter are joined into a 4d vector pairwise for the first 150 letters.

3 Results

3.1 Preprocessing

Stripping punctuation and making the sentence lowercase improved the validation loss. However, removing stop words and stemming were found to make the validation loss worse. Automatic spell check with a method explained on Kaggle, was attempted; but was found to be too resource intensive.

3.2 Word Embeddings

There was only a small difference between the validation loss of models trained on the Google News corpus and the Glove embeddings. In general, the Google News vectors performed better however the test loss difference was smaller than the test loss difference between the same model being run with different validation splits.

Efforts to improve the embeddings themselves yielded very small improvements in test loss. For Google News, the improvement was 0.00548 and for Glove the improvement was 0.00095. However, the size of the embeddings were reduced by 0.47 GB and 1.22 GB respectively. This helps them load faster.

3.3 Different Neural Networks

We started with an LSTM network which achieved a score of 0.30802 using Glove embeddings. The GRU network achieved a slightly better score of 0.30411. The CNN network was less successful, with a score of 0.33798.

The letter-wise GRU network failed to get less than a loss of 0.4 after several epochs of training. However, the letter-wise CNN worked surprisingly well and yielded a score of 0.36233.

3.4 Lessons Learned

As this was a project for an introductory class, most of our learning came in understanding the existing models and frameworks used in the field of machine learning. We learned a lot about the Python libraries available like `keras`, but found they obscured a lot of the details necessary to quickly see past the "magic".

We also learned that changing models often doesn't bring much in the way of accuracy gains, though it can speed up model loading and processing times. After reading the discussions by the winners of the Kaggle competitions, we noticed how important feature engineering is, and how finding the right combination will beat out naïve learning even with powerful tools.

For this to be most effective requires in-depth research into the datasets, trail and error with parameters, and a powerful computing platform to give rapid feedback. If we had a chance to work on this more, we would focus on longer training sessions for the letter-wise CNN, as well as looking for other patterns we didn't have time to explore, like TF-IDF.

4 Conclusion

In this project, we gained a practical exposure to LSTM, GRU, and CNN networks. We also got a chance to start testing our implementation quickly with embeddings, instead of making a best guess start like previous assignments. This allowed us to get a sense for training and testing accuracy within a few epochs, even on resource constrained systems.

This project was challenging to implement, and required a lot of time handling programming logic and server errors before we could dive into model analysis. However, it was rewarding to see how accessible the field is, and how we could tackle a hard problem with a comparatively small amount of code. We have enjoyed working with language processing as relative novices, and achieving fairly good accuracy in classification. Though we have more questions now than at the start, this project has provided a great foundation into practical machine learning, and should serve us well as we work on classification problems going forward.

A Work Done

A.1 Kaggle user "lystdo"

Without "lystdo" we wouldn't have been able to accomplish as much as we did, we learned a lot from his starter code.

A.2 Padraig Gillen

After a slow start to the project due to a high course load, Padraig focused on getting up to speed with the existing project code base, and started the documentation. He worked with IT staff on reproducing and solving the CPU deadlock issue we encountered, but these proved mostly unfruitful as a minimal working example was hard to produce, and the shared server could not be restarted often.

Later work involved edits to the presentation and report for grammar and clarity, and writing the final sections of the report. Total work done was around 35%.

A.3 Cody Ray Hoeft

Cody had a *very* light class load and an idle GPU. He was perhaps a little too eager to get started on the project and did some of the early experimentation with the LSTM after figuring out how to run keras on his Desktop computer. He also came up with the idea to clean the word2vec files to speed up their loading. Total work was around 65%