Detecting Duplicate Questions

John Herd
Department of Mathematics
University of California San Diego
jherd@ucsd.edu

Abstract

In this paper we will be looking into a solution to the problem of identifying semantically equivalent questions. This study is on a semi-recent dataset with included binary classification labels using low complexity models in an attempt to discover what parameters of the model will affect the precision and recall the most.

1 Introduction

Detecting semantic similarity or equivalence in sentences has many applications from sequence to sequence translation to Q&A forums online. In this project we focus on the Q&A application of detecting duplicate questions. Having the ability to detect such questions helps to improve the efficiency of online forums and in turn bring more traffic to a service.

In this project, the focus is on a dataset published by Quora.com which contains roughly 430,000 labeled question pairs[1]. These labels include identification numbers, but most importantly a binary classification of semantic equivalence. Given the labels, size, and quality, this dataset presents a great opportunity to implement a model to detect semantic equivalence. With semantic equivalence being defined by Quora[2] as "alternate phrasings of the same question".

To minimize the pretraining needed to implement the model and optimize its performance, the word embeddings are taken from GloVe's pretrained 300 dimension embeddings. The added benefit of using GloVe embeddings is that they already cover a large percentage of the dataset's vocabulary, so only a few tokens will be unrecognized by our model.

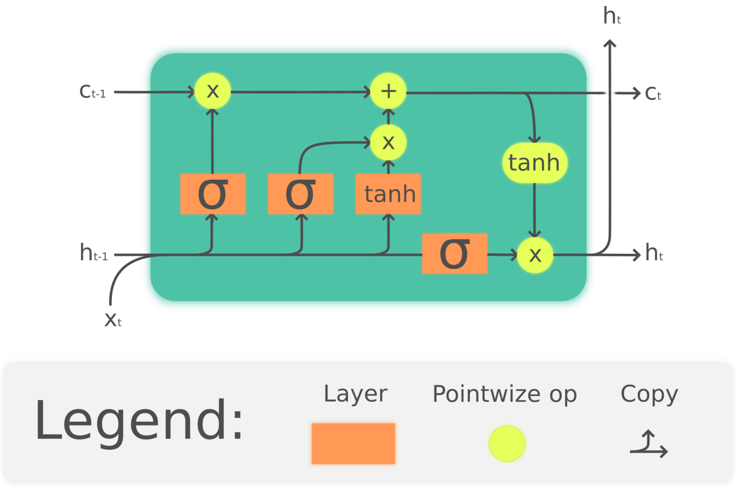
[1] https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs
[2] https://www.quora.com/Whats-Quoras-policy-on-merging-questions

2 Model

The general model architecture is a Siamese LSTM. This means that for each of two distinct inputs sentences a single model will generate an embedding individually. These output embeddings are then put into a classifier to determine a binary classification. Another interpretation is to have two models which share weights encode sentences separately, but this lends to the same results, only more parallelizable.

For each input, there is also an accompanying hidden and cell states. The hidden state is the same as any regular RNN architecture, while the cell state is the "memory" of the model. This state is responsible for storing important information for longer timesteps and is generally not used outside of the model itself.

To start, here's what the internals of a single LSTM look like.



[3]

The accompanying mathematical expressions:

$$
\begin{aligned}
i_t &= \text{sigmoid}(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \text{sigmoid}(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
o_t &= \text{sigmoid}(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
$$

("*" is element-wise multiplication)

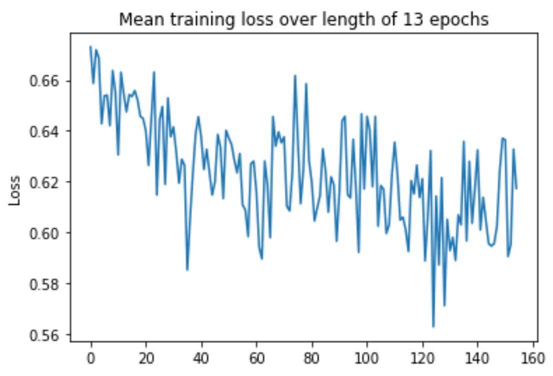
[3] https://en.wikipedia.org/wiki/File:The_LSTM_cell.png

2a Model Characteristics

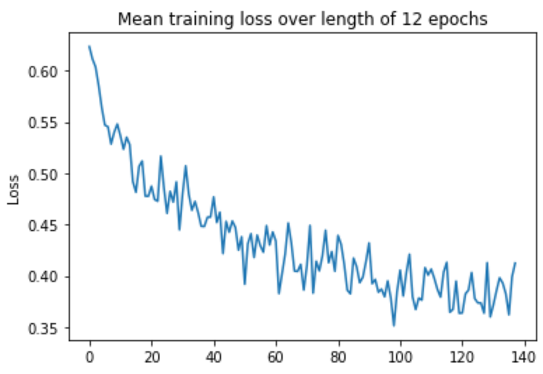
<u>All Models:</u>
All models[4] are using GloVe embeddings, specifically the 300-dimensional pretrained on Wikipedia 2014 and Gigaword 5[5]. These embeddings cover on average +80% of the vocabulary in our training set, with all unknown words in the training set receiving a randomly generated embedding vector.

<u>Baseline Model:</u>
Two baseline models were implemented and trained for 25 epochs on a training/validation set of size 60,000. At the epoch where the validation loss was lowest, the training loss values were recorded and graphed, giving the graphs below. The first model was using the Adam optimizer, and the second using SGD. Each point on the graph is the mean of 1000 loss values, one for each input during the training process.

Adam Optimizer



SGD Optimizer

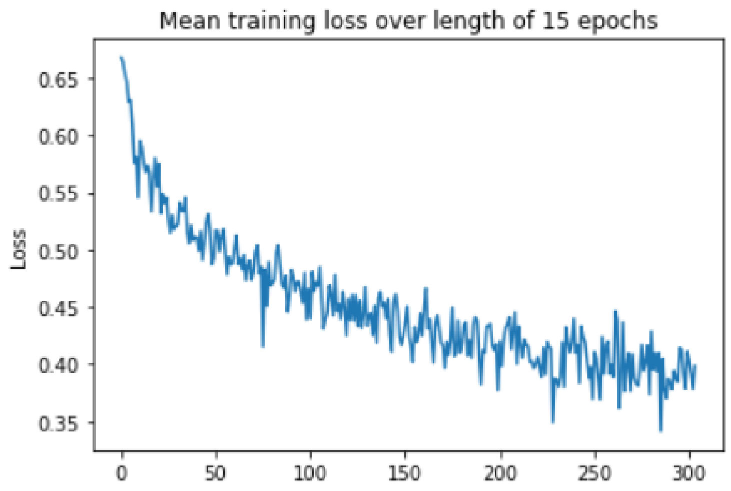


Based on these preliminary results, the final model was trained using the SGD optimizer, as the loss trended downwards more quickly and more consistently with SGD.

<u>Final Model:</u>
The final model introduces two more ideas, layering and dropout. This model is using a 2-layered lstm for its internals, in which inputs are sequentially run through multiple distinct lstm layers producing a more enriched hidden and cell states. Dropout is a regularization method where input and recurrent connections to lstm units are probabilistically excluded

[4] For training purposes, the built in pytorch LSTM is being used, as it was faster than my custom written LSTM. The custom LSTM can be seen in the modelV1.py file and is based on the math shown above.
[5] https://nlp.stanford.edu/projects/glove/

from use and weight updates while training a network. The end result gave an improved F1-Score overall and the loss trends over the first few epochs can be seen below.



2b Approach

For each entry in our training set, we are presented with two sentences and a binary classification. In order to assess if the two sentences are equivalent, we must encode each sentence. After we encode them we get outputs of v1 and v2. We then take a concatenation of a few operations of the output encodings and pass the concatenated vector into an MLP. This vector is the concatenation of each of the vectors, the pointwise difference, the pointwise multiplication, and the pointwise average. The vector is as shown below:

(v1, torch.abs(v1 - v2), v2, v1*v2, (v1+v2)/2)

The MLP in this model is a sequential double linear classifier which takes in the concatenated vector and outputs a length 2 tensor. This tensor is then put through a softmax function which outputs a continuous value between zero and one.

2c Testing

In order to get predictions from continuous values in testing, values below 0.5 are mapped to 0 and all other values are mapped to 1. While this would cause issues in our training and skew the loss in our validation, this is necessary to evaluate metrics like precision, recall, and accuracy.

3 Dataset

3a Data Structure

As mentioned in the introduction, the dataset we are working with here is the question pairs dataset, released by Quora.com. Each entry in the dataset contains 6 labels:

1. Id -- the id for the question pair
2. Qid1 -- the id for the question 1 in the pair
3. Qid2 -- the id for the question 2 in the pair
4. Question1 -- the text for question1
5. Question2 -- the text for question2
6. Is_duplicate -- binary classification of semantic equivalence (1=yes, 0=no)

For the training of the model, half of this information is not required, so the training and validation data is modified to only include the values [Question1, Question2, Is_duplicate]. And the testing data is modified to only include the sentences.

3b Preprocessing

The next step is to do some light preprocessing of this data. This includes reading the file into a dataframe, dropping the unnecessary columns, cleaning the text, and finally splitting the data into training and testing sets. The cleaning process here includes tokenizing the sentences into its individual words and removing punctuation. Since all text in this dataset is in the form of a question, the punctuation serves no purpose other than to add computations, hence its removal. Also of note, given the model structure used here, there was no need to pad the sentences.

4 Results

The two baseline models from above, saw their training loss and validation loss start to separate after reaching roughly 20 epochs, but this was to be expected as the testing data size was only 60,000. The final model saw its loss quickly converge within reason, though as the model continued to train, there was only a small change to the F1-Score. This can be seen in the table below.

	Training Size	Precision Final[6]	Recall Final	F1-Score Final	Accuracy Final	F1-Score Best[7]
Baseline SGD	60,000	0.6484593 83753501	0.4057843 99649430	0.4991913 74663072	0.6903333 33333333	0.5023303 98757120
Baseline Adam	60,000	0.5290787 44209984	0.4504820 33304119	0.4866272 18934911	0.6385	0.5162272 18519112
Two layer w Dropout = 0.5	100,000	0.7584	0.66905	0.71093	.7805	.7219

5 Conclusion

Our results give us a few notable talking points on how various parameters affect the ability for our model to accurately predict semantic equivalence.

Firstly, when choosing the best optimizer for this model, SGD gave better results than Adam. While both losses trended downward, SGD trended down faster and with less variation.

Second, when implementing baseline models, it was observed that a single layered lstm tended to overfit within roughly 20-25 epochs when using the built in pytorch models. With a custom LSTM, the speed to which this occurred was roughly the same at about the 20 epoch timer.

Lastly, when using more layers, the model was observed to achieve better results when predicting, but this ability seemed to plateau after the 30 epoch mark. We believe with more time to train a model, adding a few more layers and tuning the dropout ratio would lend better results.

In future work, we would like to test these changes and try using a CNN to replace the sequential classifier to see if this would lead to better results. Along with this, having the computational time to train on a larger portion of the dataset would prevent any amount of overfitting that might occur when training.

[6] "Final" being the final epoch which is epoch 15 on baseline and **25** with final
[7] "Best" being where the validation loss was the lowest recorded value