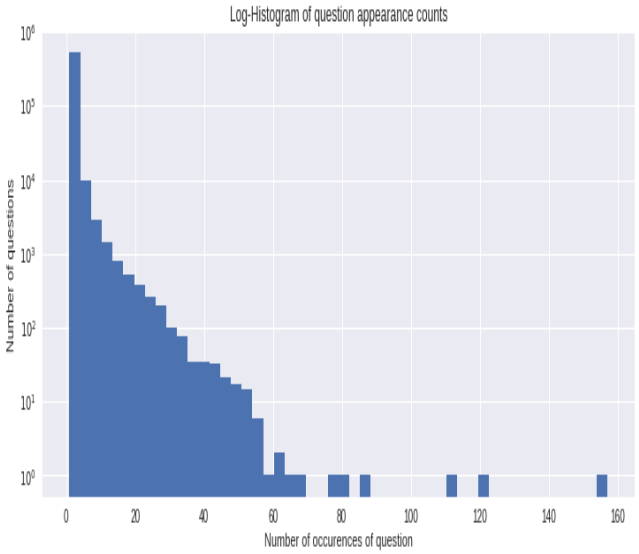
Deep Learning Approaches to detect Duplicate Question pairs

Rajat Mittal
Mathematics and Computing
Mathematical Sciences

Anubhav Anand
Mathematics and Computing
Mathematical Sciences

I. Introduction

The goal is to identify which questions asked on Quora or StackOverflow , a quasi-forum website with over 100 million visitors a month, are duplicates of questions that have already been asked. This could be useful, for example, to instantly provide answers to questions that have already been answered. We are tasked with predicting whether a pair of questions are duplicates or not.
Def. Two questions are semantically equivalent if they can be answered by the exact same answers.
For example, “How do I make $100k USD?” and “How do I earn a hundred grand?” could be duplicates!!



II. Dataset Insight:

Quora dataset provides a completely labeled dataset of pairs of questions.

We are given a minimal number of data fields here, consisting of:

id: Looks like a simple rowed

qid{1, 2}: The unique ID of each question in the pair

question{1, 2}: The actual textual contents of the questions.

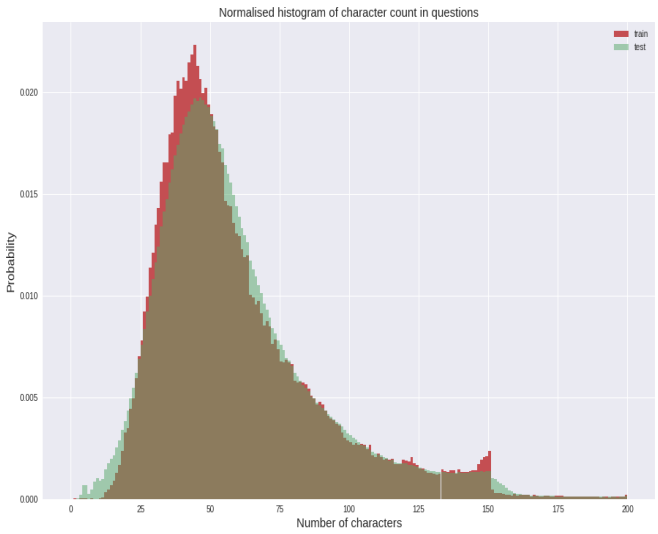
is_duplicate: The **label** that we are trying to predict - whether the two questions are duplicates of each other.

Total number of question pairs for training: 404290

Duplicate pairs: 36.92%

Total number of questions in the training data: 537933

Number of questions that appear multiple times: 111780



We can see that most questions have anywhere from 15 to 150 characters in them. It seems that the test distribution is a little different from the train one, but not too much so (I can't tell if it is just the larger data reducing noise, but it also seems like the distribution is a lot smoother in the test set).

III. Feauture Engineering:

- TF-IDF

We used TF-IDF (term-frequency-inverse-document-frequency). This means that we weigh the terms by how uncommon they are, meaning that we care more about rare words existing in both questions than common one. This makes sense, as for example we care more about whether the word "exercise" appears in both than the word "and" - as uncommon words will be more indicative of the content.

- Word Share

```
shared_weights  =
[weights.get(w, 0) for w in q1words.keys() if w in q2words] +
[weights.get(w, 0) for w in q2words.keys()  if w in q1words]
 total_weights = [weights.get(w, 0) for w in q1words] +
                 [weights.get(w, 0) for w in q2words]

 R = np.sum(shared_weights) / np.sum(total_weights)
 return R
```

- W-shingling

In natural language processing a **w-shingling** is a set of unique "shingles" (n-grams, contiguous subsequences of tokens in a document) that can be used to gauge the similarity of two documents. The w denotes the number of tokens in each shingle in the set.
For a given shingle size, the degree to which two documents *A* and *B* resemble each other can be expressed as the ratio of the magnitudes of their shinglings' intersection and union.

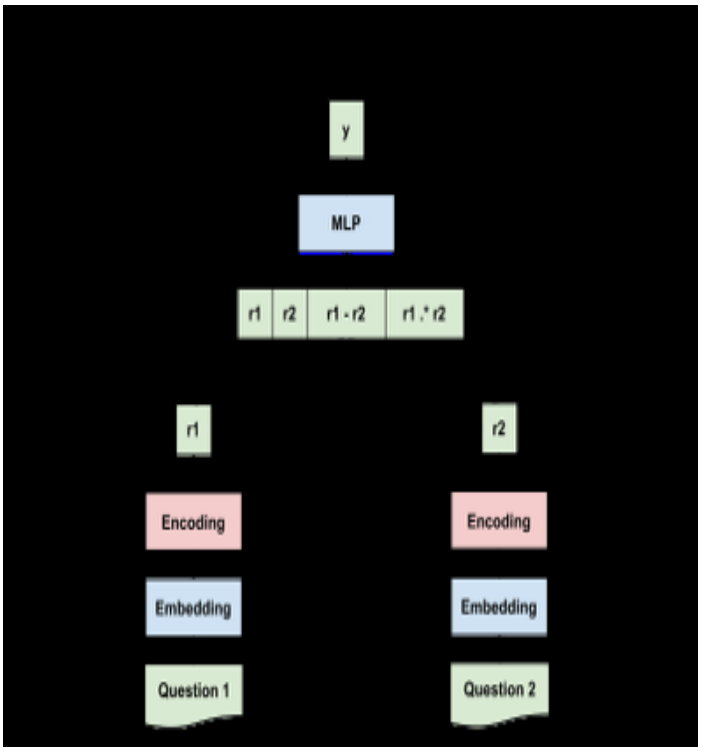
IV. Glove V/s Word2vec? For Word Embedding Matrix

Both models learn geometrical encodings (vectors) of words from their co-occurrence information (how frequently they appear together in large text corpora). They differ in that word2vec is a "predictive" model, whereas GloVe is a "count-based" model.
Predictive models learn their vectors in order to improve their predictive ability of Loss(target word | context words; Vectors), i.e. the loss of predicting the target words from the context words given the vector representations. In word2vec, this is cast as a feed-forward neural network and optimized as such using SGD, etc.

Count-based models learn their vectors by essentially doing dimensionality reduction on the co-occurrence counts matrix. They first construct a large matrix of (words x context) co-occurrence information, i.e. for each "word" (the rows), you count how frequently we see this word in some "context" (the columns) in a large corpus. The number of "contexts" is of course large, since it is essentially combinatorial in size. So then they factorize this matrix to yield a lower-dimensional (word x features) matrix, where each row now yields a vector representation for each word. In general, this is done by minimizing a "reconstruction loss" which tries to find the lower-dimensional representations which can explain most of the variance in the high-dimensional data. In the specific case of GloVe, the counts matrix is pre-processed by normalizing the counts and log-smoothing them. This turns out to be a Good Thing in terms of the quality of the learned representations.

V. Siamese neural network

Siamese neural network is a class of neural network architectures that contain two or more *identical* subnetworks. *identical* here means they have the same configuration with the same parameters and weights. Parameter updating is mirrored across both subnetworks. Siamese NNs are popular among tasks that involve **finding similarity or a relationship between two comparable things**. Some examples are paraphrase scoring, where the inputs are two sentences and the output is a score of how similar they are; or signature verification, where figure out whether two signatures are from the same person. Generally, in such tasks, two identical subnetworks are used to process the two inputs, and another module will take their outputs and produce the final output.



Siamese architectures are good in these tasks because

1. Sharing weights across subnetworks means **fewer parameters** to train for, which in turn means less data required and less tendency to overfit.
2. Each subnetwork essentially produces a representation of its input. ("Signature Feature Vector" in the picture.) If your inputs are of the same kind, like matching two sentences or matching two pictures, **it makes sense to use similar model to process similar inputs**. This way you have representation vectors with the same semantics, making them easier to compare.

VI. Three different encoding strategies used

Where r1 and r2 are the output feature vectors from the first question and the second
question, respectively.

1. The first encoding is a CNN .

We use three filters sizes of (3 x d), (4 x d), and (5 x d), where *d* is the dimensionality of the word embeddings. This corresponds roughly to 3-gram, 4-gram, and 5-gram features over our sentences. Each of these filter sizes is applied 128 times to the embedding matrix. We use a ReLU nonlinearity over the filtered output, and apply max-pooling to the result. During training, dropout with a 0.5 keep probability is applied to the pooled output. Finally, all the pooled outputs are concatenated together, resulting in a final feature vector .

2. In the second encoding, we apply a bidirectional LSTM

Each output state from the LSTM is encoded into a (1 x 5) vector. Since we perform both a forward pass and a backward pass over the word embedding matrix, this results in two (1 x 5) vectors per sentence position. Our sentences are constrained to be N words in length, which when concatenated together produces a (1 x 10N) feature vector, i.e., 5 element state * 2 directions * N words. We experimented with performing multiple layers over the LSTM, but found no improvement in performance, and so ultimately stuck with a single layer for the final results.

3. The third and final encoding strategy is a hybrid of the first two methods.LSTM + CNN

First, a bidirectional LSTM is applied to the word embedding matrix, with output states of size (1 xd) where *d* is again the dimensionality of the word embedding. The output state for each direction of the LSTM are then concatenated together back into a (N x d) matrix. The two matrixes for each direction are then stacked together to form a single (N x d x 2) matrix. Next, the CNN used in the first encoding strategy is applied to this matrix, with the same ReLU activation, max-pooling, and dropout policies. This again produces a single (1 x 328) feature vector. We explored using multiple convolutions over the outputs, but found no gain in performance, and so stuck with a single convolution.

Common to all three encoding methods is the final multilayer perceptron that combines the
siamese network feature vectors together. When introduced, provided a 2% performance gain over a simple concatenation of the two feature vectors. Specifically, we use as input:
(r1, r2, r1 - r2, r1 .* r2).

Where r1 and r2 are the output feature vectors from the first question and the second
question, respectively. The third concatenated vector is the difference between the first and
the second vectors (which will ultimately be symmetric given that each tower in the network
uses the same model weights). Finally, we use as a third concatenation the element-wise
product between the first and second vectors.

The single hidden layer of the MLP is constrained to have floor(sqrt(M)) nodes, where M is
the length of the concatenated input vector described above. ReLU activation is again used
for nonlinearity, and the final output layer consists of a single (1 x 2) vector, with the two
elements corresponding to each of the two classes: non-duplicate and duplicate. We take the
argmax over this vector to produce our final predication.

VII. Results

	CNN	LSTM + CNN	LSTM
Accuracy	0.8027	**0.8107**	0.8105
Precision	**0.7102**	0.6862	0.7004
Recall	0.7349	**0.8441**	0.7994
F1	0.7223	**0.7570**	0.7466

In running our experiments, we used 64 sample batch sizes and trained for a total of 20 epochs. We chose N=59 as our sequence length for all models. Adam optimization was used with a learning rate of 0.001. Softmax cross entropy with logits was chosen for the loss. All of the implementation was written in TensorFlow, with separate run scripts for training and evaluation.
Across all three of the models studied, we found a similar pattern of convergence. The model accuracy would sharply increase over the first training epoch, then quickly flatline after about 2 to 3 epochs. We observe a

similar pattern in the loss, where it decreases for the first few epochs, then begins to increase consistently as the magnitude of the weights increases. Overall, we found that the three deep learning models significantly outperformed the baselines across all metrics, but none of the three deep learning models stood out as being noticeably more expressive than the others. The LSTM was notable for having slightly better overall performance than the CNN, but being significantly faster to train and perform inference.

Most surprising was that the hybrid model did not provide any gain in performance over the LSTM whatsoever, and in some ways may have actually underperformed it. It's possible that the encoding of the LSTM captured some amount of information that was otherwise lost during CNN filtering process.

Model	Acc. (Val)	F1 (Val)
RNN, cosine distance	0.7737	0.7007
RNN, L_2 distance	0.7860	0.7077
RNN, weighted Manhattan distance	0.7991	0.7160
RNN, 1-layer similarity network	0.8061	0.7320
RNN, 2-layer similarity network	0.8094	0.7220
GRU, cosine distance	0.7708	0.7170
GRU, L_2 distance	0.8309	0.7735
GRU, weighted Manhattan distance	0.8422	0.7845
GRU, 1-layer similarity network	0.8633	0.8132
GRU, 2-layer similarity network	0.8631	0.8103
GRU, 1-layer similarity network, augmented data	0.8668	**0.8191**
GRU, 2-layer similarity network, augmented data	**0.8680**	0.8173

VIII. Data Preprocessing For LSTM and GRU

The input questions vary significantly in length, from empty (0-length) up to 237 words. In order to Batch our computations during training and evaluation using matrix operations, we needed the input questions to all have a fixed length. To do so, we padded the shorter sentences at the beginning with a designated zero-padding ID and truncated the longer sentences to the same standardized length, which is a hyperparameter to our model.Lastly, we split our data into training, development (validation), and test sets.

Sentence Encoding

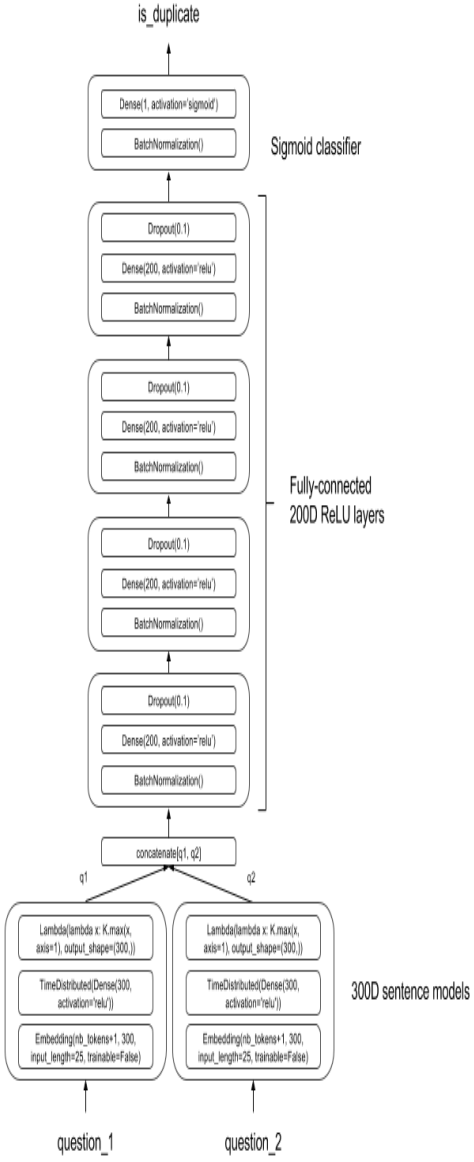
In this project, we explore two types of neural networks to encode each sentence: the recurrent neural network (RNN) and the gated recurrent unit (GRU). During training, we update the word embeddings as well as the weights and biases within the RNN/GRU cells. Each of these cells contains a single layer with the tanh activation function. The weights and the biases in the encoding layer are shared for both questions in the pair so that the network is smaller and easier to train. Each of these neural networks outputs a sentence vector of dimension H, the hidden state size in the RNN/GRU cells.

Distance Measure

Once the questions of each pair are individually encoded into sentence vectors, we need to combine
them in some way to predict whether or not the pair is a duplicate. The first method we use involves
calculating some distance between the sentence vectors and running logistic regression to make the
prediction,

$$\hat{y} = \sigma(a \cdot d(\boldsymbol{h}_1, \boldsymbol{h}_2) + b),$$

where _ is the logistic or sigmoid function, a; b 2 R are learned parameters, and $d : \mathbb{R}^{\mathbb{H}} \times \mathbb{R}^{H} \rightarrow \mathbb{R}$ is a distance function between the two sentence vectors.



IX. Conclusion

Our results build on previous research on using Siamese networks for identifying semantically equivalent questions. In particular, we provide three novel contributions.

First, passing the concatenated vector v of hidden state transformations through a neural network is much better at determining semantic equivalence than pure distance measures. Previous work by Bowman et al. only used cosine distance, which we show is the least promising distance measure.

Second, a GRU neural network is able to extract the majority of the semantic meaning from a sentence from the first 10 to 15 words. This result is somewhat surprising given Dey et al.'s claim that deep neural networks are unsuitable for learning semantic meaning on short-text content.

Lastly, we demonstrate that data augmentation can be more effective than L2 regularization for reducing over fitting, especially in asymmetric Siamese neural nets where there are natural data augmentation techniques. While we did not have enough time to perfect this technique, we believe it holds significant potential.

X. References

[1] Broder, A. (1997) On the resemblance and containment of documents. *Proceedings of the Compression and Complexity of Sequences 1997,* SEQUENCES'97, Washington, DC, USA. IEEE Computer Society.

[2] Yoon Kim. (2014) Convolution neural networks for sentence classification. *Proceedings of the 2015 Conference on Empirical Methods for Natural Language Processing*, pages 1746-1751. Doha, Qatar.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. (2013). Efficient estimation of word representations in vector space. *Proceedings of International Conference on Learning Representations,* ICLR 2013, Scottsdale, AZ, USA.

[4] Dagna Bogdanova, C´ıcero dos Santos, Luciano Barbosa, and Bianca Zadrozny. (2015). Detecting Semantically Equivalent Questions in Online User Forums. *Proceedings of the 19th Conference on Computational Language Learning*, pages 123–131, Beijing, China, July 30-31, 2015.

[5] Zhiguo Wang, Wael Hamza, Radu Florian. (2017). Bilateral Multi-Perspective Matching for Natural Language Sentences. https://arxiv.org/abs/1702.03814.

[6] Jeffrey Pennington, Richard Socher, Christopher D. Manning. (2014). GloVe: Global Vectors for Word Representation. *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532-1543. http://www.aclweb.org/anthology/D14-1162.

[7] Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, Christopher Potts. (2016). A Fast Unified Model for Parsing and Sentence Understanding. https://arxiv.org/pdf/1603.06021.pdf.

[8] Travis Addair, Duplicate Question Pair Detection with Deep Learning ，Department of Computer Science Stanford University

[9] Detecting Duplicate Questions with Deep Learning Yushi Homma, Stuart Sy, Christopher Yeh Stanford University

.