Natural Language Understanding within the context of Question Matching

by
Michael Li

A Thesis submitted in partial fulfillment of the requirements for Honors
in the Department of Applied Mathematics and Computer Science at Brown University

Providence, Rhode Island
April 2019

This thesis by Michael Li is accepted in its present form by
the Department of Applied Mathematics and Computer Science as satisfying the research requirement
for the awardment of Honors.

Date _____ _____

Eugene Charniak, Advisor

Date _____ _____

Matthew T. Harrison, Reader

Abstract

In this thesis we examine the AI-hard problem of natural language understanding. We use the Quora Questions dataset as motivation - strong performance on this dataset requires a classifier that can perform reading comprehension and compare meanings accurately. We explore improvements upon existing approaches to this dataset, such as the SiameseLSTM model. This is primarily accomplished by modifying the LSTM unit to its Bidirectional GRU variant, allowing the model to make better use of question context, as well as utilizing new optimization techniques to train the model. The GRU is often easier and faster to train than the LSTM and gives better results on less training data. We use the Quasi-Hyperbolic Adam optimizer from Facebook AI Research to train our model, which is a variant of the classical Adam algorithm by Kingma & Ba (2014). Our result of 88.4% testing accuracy over arbitrary train/validation/test splits is competitive with existing results and shows the importance of considering context when determining question similarity as well as the choice of optimization technique.

Acknowledgements

I'd like to thank Professor Eugene Charniak for his invaluable guidance with this thesis as well as being the person who introduced me to neural networks by way of his Deep Learning course. I'd also like to thank Professor Matt Harrison for being my official reader, and for all that I have learned from him in applied mathematics.

I am also grateful to Orion Reblitz-Richardson, Soumith Chintala and the rest of the team at Facebook AI for allowing me to spend Summer 2018 with them as an engineering intern working on new features for the PyTorch deep learning framework. PyTorch is the framework of choice in this thesis.

I'd also like to thank Aaron Gokaslan and Andrew Hou for their suggestions and help with Brown's GridEngine system, as well as numerous friends for moral support.

Lastly I'd like to thank my parents for providing me with never-ending encouragement and motivation.

Contents

Chapter 1

Introduction

Formally, semantics is the branch of linguistics and logic that deals with meaning. In this thesis we look at Natural Language Understanding (NLU), the sub-field of Natural Language Processing (NLP) that deals with a machine's ability to understand semantics. It is an AI-hard (AI-complete) problem in general, which means that it is one of the most difficult problems in artificial intelligence. If it were to be solved, then it would be an example of giving computers true human understanding and intelligence (i.e. Strong AI). The potential applications of such an achievement are unbounded.

The process of building machines with the ability to seemingly process and extract meaning from input information has been widely explored. The first NLP methods were quite naive and involved applying a large, complex set of rules to input in order to produce reasonable output. Due to the hardcoded nature of the methods, their success was limited. Such methods include ELIZA (Weizenbaum, 1965), which simply parsed input and substituted words into premade rules in order to generate output. In the 1980s, machine learning techniques and increased computational power allowed for further advancement. Examples of these systems include IBM Watson. Later on deep learning techniques were applied with successful results, such as Collobert et. al (2011), which proposed a general neural network architecture that worked well for a variety of natural language tasks. Most recently, OpenAI announced their GPT-2 model, which achieves exceptional results on tasks such as reading comprehension and coherent fake text generation.

The ultimate goal is Artificial General Intelligence (AGI), in which a computer would be able to perform any test of intelligence at a human level.

The question then is: how well can we teach a computer program to demonstrate the ability to understand meaning?

We examine this overarching question within the context of the Quora Questions dataset. The dataset consists of over 400,000 pairs of questions and corresponding labels indicating whether the two questions in a pair have the same intent. For example, a sample pair of questions from the

dataset is the following:

$$\begin{cases} \text{Are exocytosis and endocytosis examples of active or passive transport?} \\ \text{What are examples of passive transport in cells?} \end{cases}$$

These questions contain several similar words, but ultimately are not asking for the same information (differing intent). Thus, they have a corresponding label of 0.

A pair of questions from the dataset with the same intent (true label of 1) is also provided below for reference:

$$\begin{cases} \text{What are some special cares for someone with a nose that gets stuffy during the night?} \\ \text{How can I keep my nose from getting stuffy at night?} \end{cases}$$

This dataset was prepared from questions asked organically on Quora, although the dataset is not representative of the entire population of Quora questions due to the selection process . The dataset is available at `https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs`. The goal is to be able to learn a classifier that can accurately determine whether two questions have the same meaning. As it is possible to permute, extend or otherwise modify a question in many ways, it is not feasible to apply more naive methods such as a rule-based system. In general this task is quite difficult, and we require deep learning techniques to obtain a high-performance classifier. Due to the nature of organic content, there are numerous instances of grammatical and spelling issues which additionally leaves room for clever pre-processing techniques to improve performance.

Chapter 2

Background Information

Here we define and elaborate on some of the key terms and concepts used in this thesis.

2.1 Recurrent Neural Networks

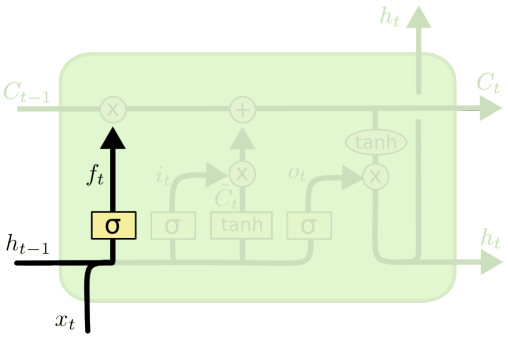
An **RNN (Recurrent Neural Network)** is a network containing a cycle, allowing the output from the network to be re-used as input to the network. This is helpful because it simulates intelligence more accurately. For example, when humans are processing new information, they draw upon old information that they have previously processed and use that to make new decisions. This is especially valuable in Natural Language Processing, because the network can draw upon previous information from sentences or questions it has seen and use that information in making future inference decisions.

The basic version of an RNN is limited by the fact that it operates on rather short term memory. If previously processed information is not too far off (in a dependency graph) from the next site at which it can be used, then the RNN typically does quite well. The issue arises when the RNN *could have* benefited from information further down the dependency graph. With a vanilla RNN, it cannot because the information will have mostly been lost due to the nature of data processing by the time it reaches the site where it needs to be used.

This is where the LSTM was introduced. An **LSTM (Long Short Term Memory)** network is a special version of an RNN, first introduced by Hochreiter Schmidhuber (1997). The main advantage of the LSTM is that it is able to remember information for long periods of time as opposed to the purely short-term memory of a vanilla RNN.

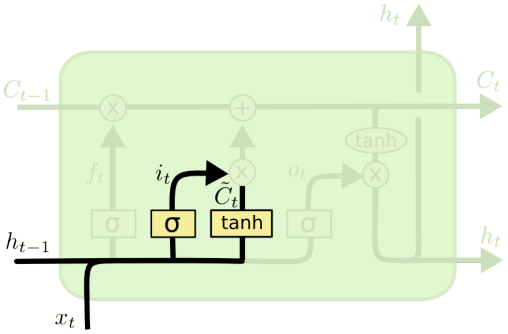
There are two main paths in an LSTM: there is (1) the **cell state** and the (2) **hidden state**. The cell state records what we have selectively remembered over time, and the hidden state is more of a collection of all the information that we have seen so far.

f_t in this diagram (Figure 2.1) is known as the "forget gate". Here the LSTM uses the new input x_t at time step t and combines it with the previous hidden state h_{t-1}, then applies a linear $W \bullet x + b$ operation and squashes the results to be between 0 and 1 using the sigmoid σ activation



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.1: LSTM Forget Gate Detail (Olah, 2015)



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.2: LSTM Update Candidate Detail (Olah, 2015)

function. The result of this operation consists of one real-valued number in [0, 1] that represents the corresponding percentages of information that we want to "forget" from the cell state C_{t-1}.

In Figure 2.2, the next two gates decide how much new information to add to the cell state and create a new candidate vector $\tilde{C}_t$ respectively. The $tanh()$ operation is applied to squash the values of $\tilde{C}_t$ to be between -1 and 1 (we essentially either make a subtractive or additive update).

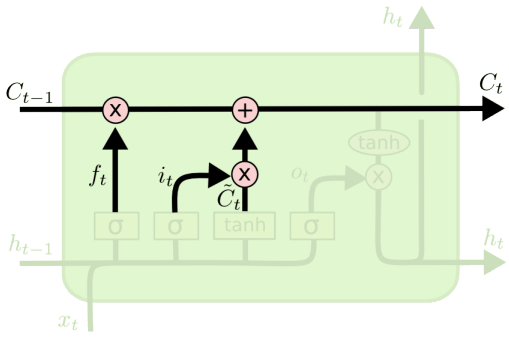
In Figure 2.3, the "forget" operation is applied on the previous cell state C_{t-1} and the new scaled candidate vector $\tilde{C}_t$ is added to the next cell state C_t.

Lastly, in Figure 2.4, the output gate o_t controls the output of our LSTM: we decide how much we want to scale the output by and again apply a $tanh()$ operation This output becomes the next hidden state.

There are also variants of the LSTM, including the GRU (Cho et. al, 2014), which can be thought of as a simplified version of the LSTM. Instead of a separate forget and input gate, it has a combined gate which controls all additive and subtractive operations on the cell's memory. There is also no notion of a separate cell state and hidden state, among other changes. A diagram is given below:

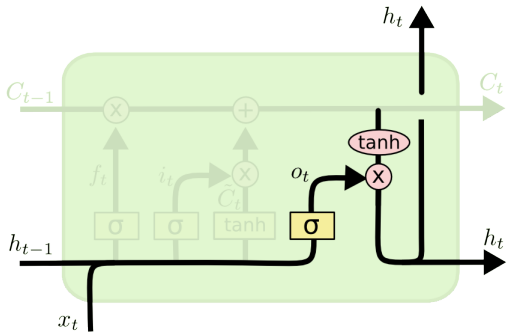
In Figure 2.5, $\tilde{h_t}$ is the new candidate vector for the hidden state and z_t controls the percentage of the new candidate vector we want, while retaining $(1 - z_t)$ of the previous hidden state.

Chung et. al (2014) give some analysis of the performance of GRUs versus LSTMs and show that on some datasets the GRU is able to outperform the LSTM in terms of time to converge as



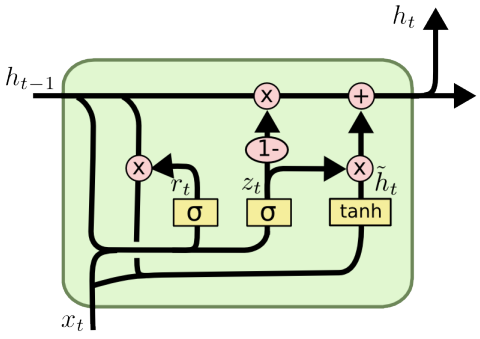
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.3: LSTM Cell State update operation (Olah, 2015)



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.4: LSTM Output (Hidden State) detail (Olah, 2015)



$$z_t = \sigma\left(W_z \cdot \left[h_{t-1}, x_t\right]\right)$$
$$r_t = \sigma\left(W_r \cdot \left[h_{t-1}, x_t\right]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot \left[r_t * h_{t-1}, x_t\right]\right)$$
$$h_t = \left(1 - z_t\right) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 2.5: GRU Unit Detail (Olah, 2015)

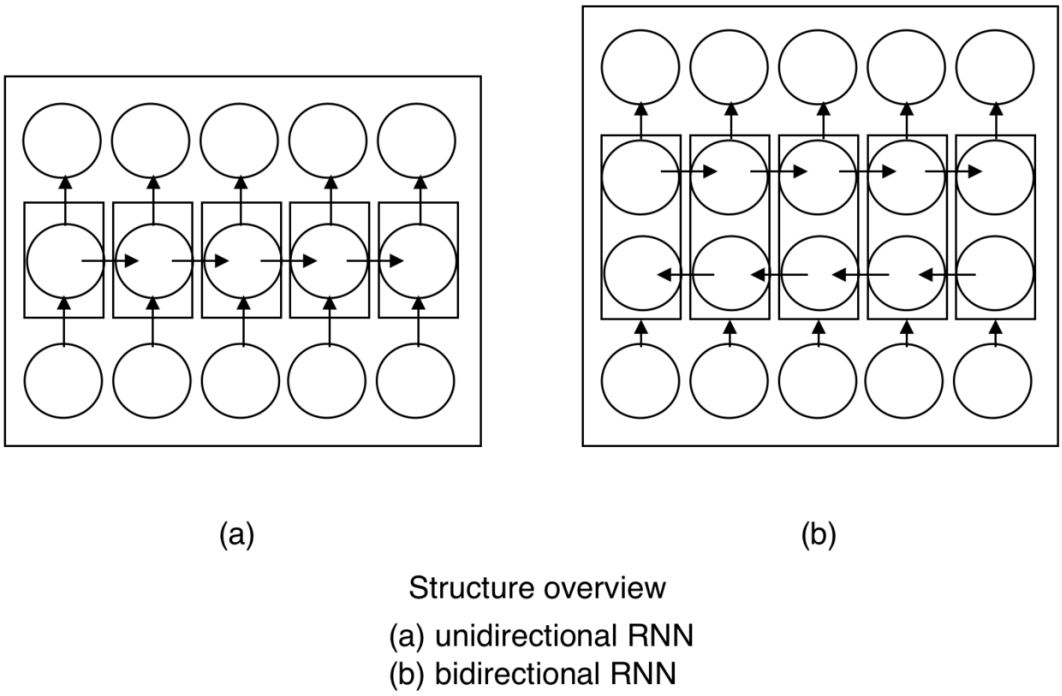


Figure 2.6: Unidirectional vs Bidirectional RNN structure (By Incfk8 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45392097)

well as parameter updates and generalization beyond the training dataset. Intuitively, the GRU is less computationally intensive to train because it has less overall complexity than the LSTM. It is also more likely to perform better than the LSTM when less training data is available for this reason. The LSTM may be more optimal for longer input sequences as it can better model long-term dependencies in its cell state. In our use case, the questions are not quite long enough on average (roughly 10-15 tokens) and we do see that the GRU outperforms the LSTM.

Additionally, RNNs can be made *bidirectional*, an advancement by Schuster & Paliwal (1997). The bidirectional RNN has two hidden state paths which are oriented in opposite directions (one forward, one backward) and connect to the output layer. This allows information from both the past and the present to be reached from the current state. Bidirectional RNNs experience some of the greatest performance gains when there is a need for context. In our use case here, we benefit from a Bidirectional GRU because when looking at words in questions it helps significantly to know what occurs before and after each word in the question. Figure 2.6 provides a diagram that illustrates how a bidirectional RNN differs from a standard unidirectional RNN.

An (advanced) application of the LSTM model is the SiameseLSTM model, which is leveraged in this work and explained in further detail in Section 3 (Architecture). In this work we modify the SiameseLSTM model to become a SiameseGRU model.

2.2 Word Embeddings

An **embedding** is a representation of a word as a vector in an n-dimensional space. The most basic form of an embedding will assign all distinct words in the corpus (the "vocabulary" or "dictionary") distinct indices in sequential fashion. However, for most neural networks this is far too basic. We usually use anywhere from 50-dimensional to 300-dimensional embedding vectors. Embeddings which have dimensionality lower than 50 typically are not rich enough in terms of model complexity, and embeddings with dimensionality greater than 500 to 1,000 typically see very little gain in exchange for a massive performance compromise during the training and testing processes. Word embeddings are learned given a corpus of training data. Sample training data for embeddings, for example, includes text from Wikipedia articles and Twitter.

The use of word embeddings is highly popular in NLP and there are multiple packages available for both research and production use. One of the original packages is word2vec by Mikolov et. al (2013) at Google Research, Other packages include fasttext, which is developed by Facebook and focuses on using low-dimensional word embeddings to improve performance, and GloVe, which was developed at Stanford by Pennington et. al (2014) and uses an unsupervised learning algorithm for obtaining vector representations for words. Ideally, a good embedding will represent similar words with similar vectors, i.e. have the vector for "automobile" be very similar to that of "vehicle". An embedding can also model other relationships, i.e. it could have the distance between the "Paris" vector and the "France" vector be roughly the same as the distance between the "Beijing" and "China" vectors.

In our code, we will use the embedding for a word whenever such an embedding exists, and if no embedding exists for the word, we will generate a random vector of the same dimensionality as all of the existing embeddings and assign that as the word's embedding vector.

2.3 Loss Function

A **loss function** is a measure of how well an algorithm/architecture models a given problem. Often it can be interpreted in terms of a "distance" between the true distribution and the distribution that your model learns. For this task I use cross entropy loss (`nn.CrossEntropyLoss` in PyTorch). Cross entropy loss is appropriate when the output of the model is a probability value between 0 and 1. Here the model is outputting a probability of the two questions having the same semantics. For example, in the case that two questions do have the same semantics (ground truth label of 1), cross entropy loss would penalize the model for outputting a low probability (i.e. if the model thinks there is a 1.5% chance that the questions have the same semantics, it would receive a $-\log(0.015) \approx 1.824$ penalty, which is quite large! The ideal model would have a loss value of 0, meaning that it is perfectly able to capture the complexity of the problem at hand. Unfortunately this never happens in applications beyond toy examples.

The formula for cross entropy in the case of binary classification (0-1) is given below:

$$(y \log(p) + (1 - y) \log(1 - p))$$

where y is the true label (either 0 or 1) for the example, and p is the predicted probability. For generalized n-class classification, the cross entropy formula is given by:

$$\sum_{c=1}^{n} y_{e,c} \log(p_{e,c})$$

where

$$y_{e,c} = \begin{cases} 0 & \text{example } e \text{ does not have true class label } c \\ 1 & \text{otherwise} \end{cases}$$

and $p_{e,c}$ is the predicted probability that example e is of class c.

2.4 Data Augmentation

Data augmentation is the process of permuting the existing training dataset to create new training examples. It is commonly used in computer vision applications where the input training images may have transforms applied to them, such as slight rotations or translations to generate new training images. This is especially effective when the original set of training data is limited, and it can help the model learn invariance to rotation or translation, for example, in future test input.

The training data set (400k training examples) is large enough such that data augmentation is not a completely necessary technique, but there is an easy data augmentation here that we can perform given the nature of the problem. Given that the questions come in pairs, if one question has the same semantics as the other, then it should not matter what order of the two questions we provide the model with. Thus, as an augmentation, for each pair of questions (x, y), we add the corresponding pair (y, x) to the training data.

This helps the model learn invariance as to which question appears first in the pair, which is invaluable for this task. It also immediately doubles the size of the training dataset to over 800,000 examples.

Chapter 3

Architecture and Methods

3.1 Pre-processing methods

We use a `"STOP_PAD"` word with index 0 to represent padding for questions, and we also use a `"UNK"` word with index 1 to represent any words in our test set that we do not have in our corpus. We then assign new indices to words as we see them, and keep track of a global dictionary mapping words to their indices.

We first take the two input questions, and cap them at a maximum question length - question pairs containing one or more questions with a length exceeding this limit are discarded. In my model the limit is generously set at 35 words total. The two questions are padded with however many STOP words are necessary to meet the limit.

We also perform text cleaning on the individual words in each question. This helps with standardization and avoids us adding extraneous words to our global dictionary. We create vectors representing each question, with the vectors consisting of word indices in the global dictionary.

3.2 Generating embeddings

Then, we load the GloVe embeddings and create an embeddding matrix - if we find an embedding matching the words in the input question vectors, then we place the corresponding embedding vector in the embedding matrix at the $(i + 2)$th index (assuming that the word corresponds to index i in our dictionary). We shift by 2 because we have reserved index 0 and 1 for the STOP and UNK words respectively.

With the embedding matrix created, we load this into our PyTorch model and set the `requires_grad` option to `False`, because we are using pretrained embeddings and do not need to train our own.

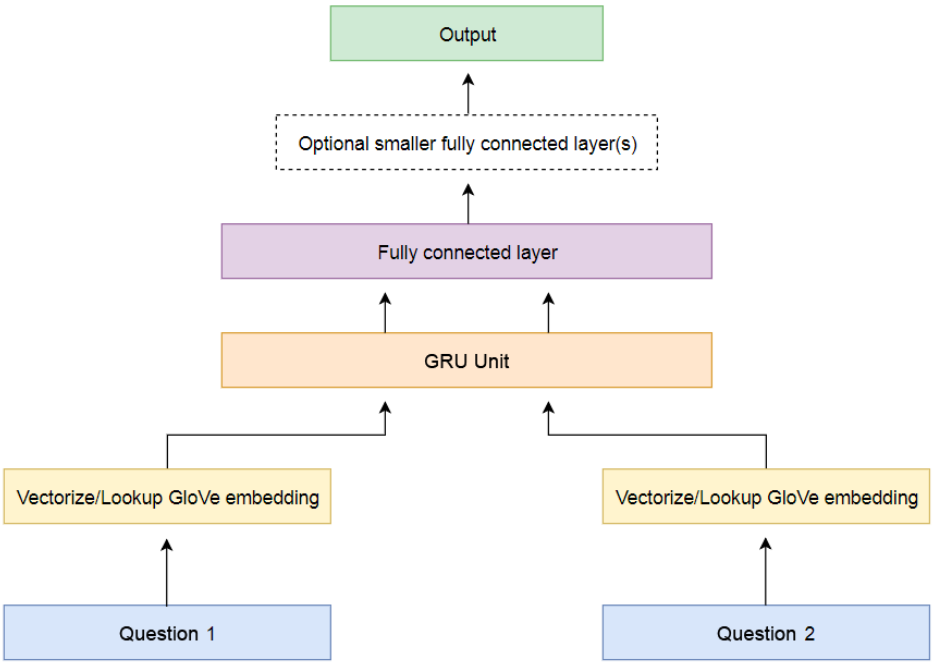


Figure 3.1: SiameseGRU detail

3.3 Model

I used the PyTorch deep learning framework developed by Paszke et. al (2017) at Facebook AI Research to build the model.

The backbone of our model is inspired by the SiameseLSTM model. We use the GRU instead of an LSTM to create a SiameseGRU model. A diagram of the architecture can be seen in Figure 3.1.

For each training example, we will have a vector of word indices in our global dictionary. For a pair of questions (x, y), each of length n, we will have

$$x = [x_1, x_2, ..., x_n]$$

$$y = [y_1, y_2, ..., y_n]$$

We look up the embedding vector for each x_i and y_i. After we lookup the embedding vectors corresponding to each word in the two question vectors, we will obtain two matrices E_x, E_y filled

with the embedding vectors sourced using the indices in x and y.

$$E_x = \begin{bmatrix} [x_{1,1}, x_{1,2}, ..., x_{1,m}] \\ [x_{2,1}, x_{2,2}, ..., x_{2,m}] \\ \vdots \\ [x_{n,1}, x_{n,2}, ..., x_{n,m}] \end{bmatrix}$$

$$E_y = \begin{bmatrix} [y_{1,1}, y_{1,2}, ..., y_{1,m}] \\ [y_{2,1}, y_{2,2}, ..., y_{2,m}] \\ \vdots \\ [y_{n,1}, y_{n,2}, ..., y_{n,m}] \end{bmatrix}$$

The GRU is used for both questions. We initialize two hidden states, one for each questions. The questions in embedding form are then passed through the GRU, one at a time, and their outputs and resulting hidden states are collected.

We then use a fully connected layer that consists of three linear $W \cdot X + b$ layers with a ReLU layer, a Dropout layer and a softmax layer. The detailed view is shown in Figure 3.2.

The ReLU (Rectified Linear Unit) is an activation function in deep learning. It applies the function

$$f(x) = x^+ = \max(x, 0)$$

to its input. ReLU was first proposed by Hahnloser et. al (2000) and was motivated both biologically and mathematically; it is currently one of the most popular activation functions and has been shown to improve the training process for deep networks.

Dropout is a regularization technique. With some probability p, the "keep probability", it will randomly zero out elements in its input with probability $(1 - p)$. Usually a "keep probability" of around 0.7 - 0.8 works well. If the "keep probability" is set lower, there is a risk of losing too much information in the dropout layer.

Lastly, we have the softmax layer, which takes in an input vector $\mathbf{z}$ and applies the following function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ on it, where σ is defined as

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, ..., K \text{ and } \mathbf{z} = (z_1, ..., z_K) \in \mathbb{R}^K$$

The function of the softmax layer is to normalize the final logits produced for each input example into a valid probability vector. In our case, we output a 1×2 vector for each input example, with the first element representing the probability that the class is 0 (questions do not have the same meaning) and the second element representing the probability that the class is 1.

Thus, ideally when the true class is 1, the model will assign a probability of 1 to class 1. Using our cross-entropy loss function, we penalize our model: the further the probability of the correct class deviates from 1, the more the model is penalized.

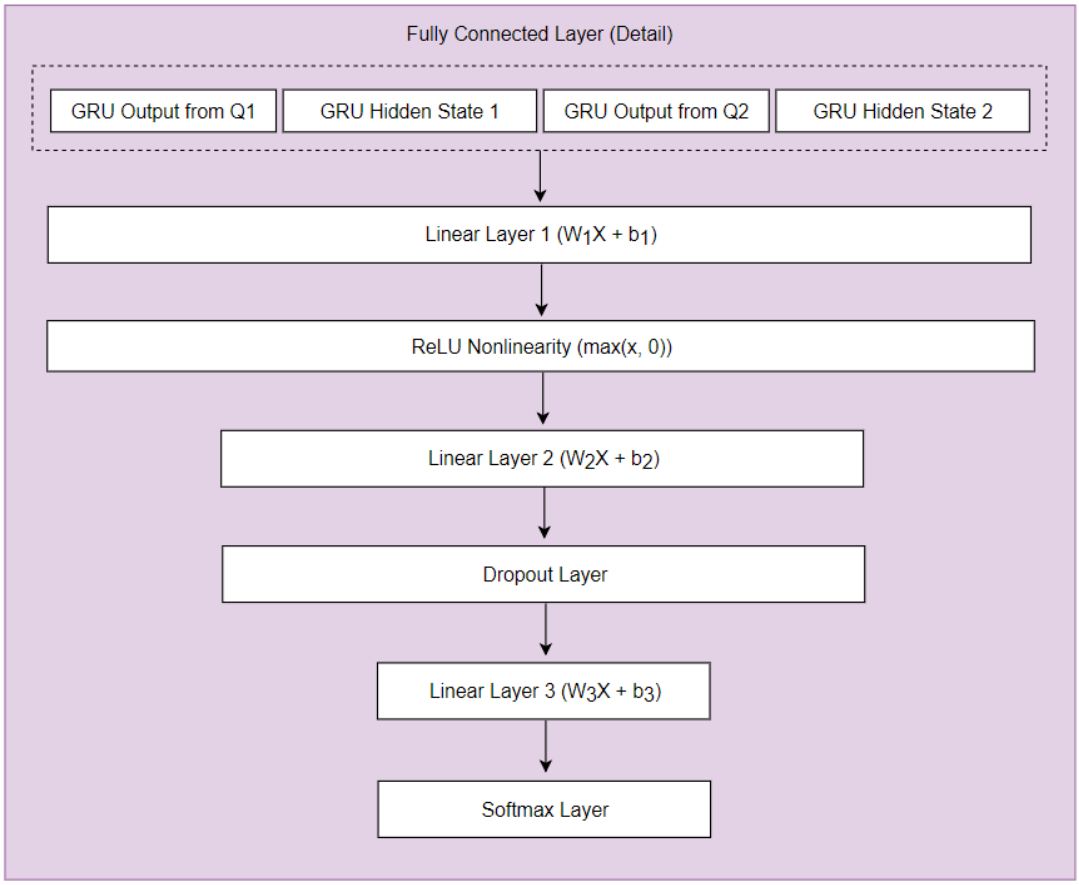


Figure 3.2: Fully connected layer detail

3.4 Training process

To train our model, we use the CUDA functionality of PyTorch where the tensors (data) are placed on the GPU during computation for highly increased speed. I used both an Nvidia Tesla P100 (from Google Cloud Platform) as well as a Nvidia GeForce GTX 1080 Ti (from Brown's GridEngine system) to train the models. Evaluation of 75 epochs (loss and training accuracy) and testing on the test dataset takes under an hour in total.

Of the initial 404k examples provided as part of the training set, we augment so that we have 808k total examples, and then partition into a training set with 486,000 examples and a dev and test set of roughly 160,000 examples each. The splits are different every time in order to more accurately assess the performance of our model on unseen data. As the Quora Questions Dataset is imbalanced (roughly 63% of the labels are of one class and the remaining 37% of the other) this pre-existing bias is removed before performing splitting.

Chapter 4

Results

As the content is organic, there are instances of spelling errors in the questions. I tried some cleaning techniques with the Python `autocorrect` and `editdistance` packages, replacing misspellings if the edit distance of the suggested correction is less than a small percentage of the original word length. This preprocessing technique actually slightly hurt the performance of the classifier (by around -0.1% or so) and also was very computationally expensive, so I discarded it in the final model.

I found that my best results were from using GloVe 300-dimensional embedding vectors, a hidden dimension (for the GRU) of 100, a minibatch size of 256, 75 epochs, and a learning rate of 0.0015 with a staircase decay factor of 0.9 every 2 epochs. I used dropout with a "keep probability" of 0.6.

I experimented with several optimizers including Stochastic Gradient Descent (SGD) with Nesterov accelerated gradient (NAG), a technique developed by Russian mathematician Yurii Nesterov. I also explored the usage of AdaGrad/AdaMax. AdaMax is a variant based on infinity norm of the popular Adam optimization algorithm by Kingma & Ba (2014).

My final choice of optimizer is the Quasi-Hyperbolic Adam (QHAdam) optimizer by Ma & Yarats (2019), developed at Facebook AI Research. The code is available at `https://github.com/facebookresearch/qhoptim`. The advantage of this optimizer is that it provides two additional hyperparameters over the existing Adam optimizer for controlling the rate at which parameters are updated. This allows for finer control in optimization.

My best test accuracy is 88.4% over arbitrary train/validation/test splits, showing that the model performs well and has sufficient complexity for the problem at hand. The Quora engineering team's own best result is 87% test accuracy, however, this was evaluated over their own private test set and thus the two results are not directly comparable.

For reference, human-level performance on this task is 92.7% accuracy (taken from the GLUE benchmark by Wang et. al, 2018). While this might seem low, it is important to recall that there exists some degree of noise in the test labels. Additionally, certain questions may have multiple interpretations.

A plot of the average minibatch loss and training accuracy with respect to the number of epochs is included in Figure 4.1.

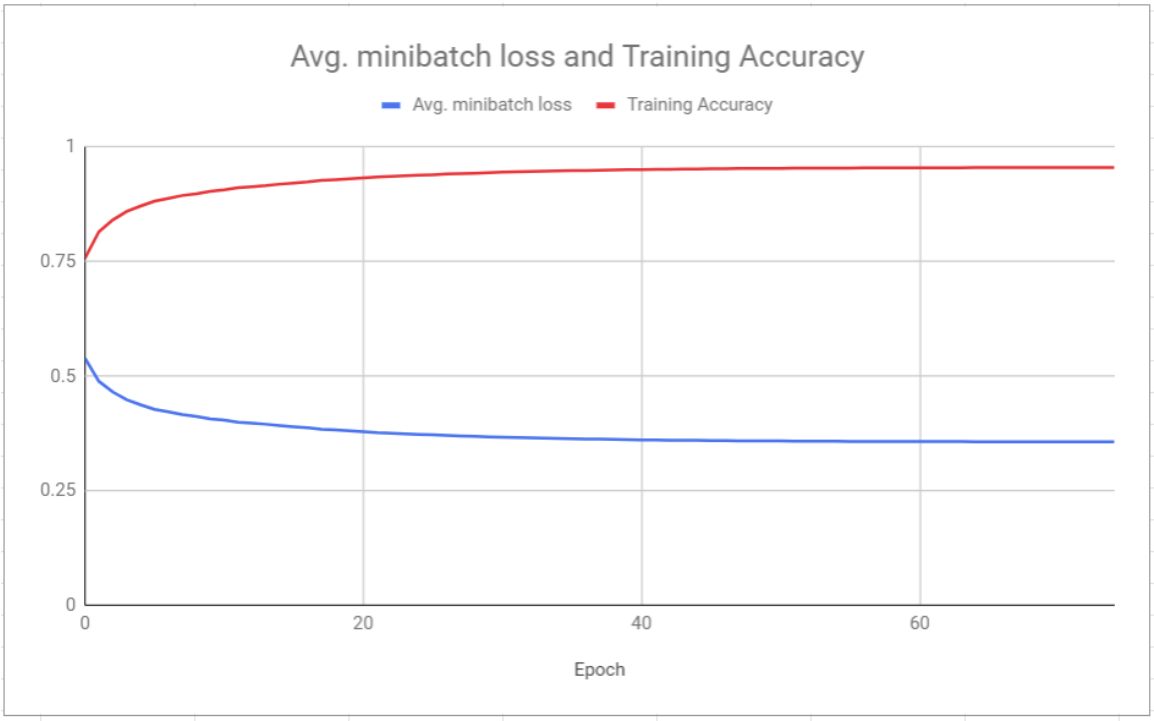


Figure 4.1: Average minibatch loss and training accuracy per epoch

The code is immediately available at `https://github.com/ml7/nlpresearch`.

Chapter 5

Related Work

5.1 Kaggle Competition

The Quora Questions data set was the subject of a Kaggle competition in which teams of people compete to produce the best classifier. The winner of the competition was a team of French data scientists and PhD students. They used the following three approaches:

1. **Deep Learning**: Their first approach is most similar to the approach followed in this thesis. Using some popular word embedding packages (word2vec, doc2vec, sent2vec), they convert questions to sets of vectors in 300-dimensional space and pass them through a Siamese Neural Network with a fully connected layer and softmax activations.

2. **Natural Language Processing**: Their second approach involves engineering many different types of features and using them in a classifier. They consider the following:

   - Similarity measures from Latent Dirichlet Allocation (LDA) and Latent Semantic Indexing (LSI).
   - Similarity measures on bags of character n-grams where $n \in [1, 8], n \in \mathbb{Z}$. Optionally reweighted using `tf-idf` (term frequency-inverse document frequency), which reflects how important a word is relative to the document or corpus that contains it.
   - Edit distance between character strings, including Levenshtein distance, where for two strings a and b, with lengths $|a|$ and $|b|$ respectively:

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{\{a_i \neq b_i\}} \end{cases} & \text{otherwise} \end{cases}$$

     as well as Bray-Curtis distance and Jaro-Winkler distance.

- The percentage of words shared by the two questions, where the word length is limited between 1 and 6 characters. This feature is used when the questions start and/or end with the same word.
- The length of the two questions individually and relative to each other. The number of capital letters, question marks, etc. are also considered.
- Indicator variables for the question word that a question starts with ("Can", "How", "Do", "What", etc).

3. **Graphical Models**: Their third approach involves graphical models. Using the test and train datasets of question pairs as a whole, they build a graph where individual questions are nodes and edges represent question pairs. Due to the fact that many questions are asked repeatedly on Quora, they found it appropriate to exploit this pre-existing relationship in the hope that the frequently asked questions can provide similarity signals for other questions.

   For each pair of questions q_1, q_2 (nodes connected by an edge), they compute the following statistics:

   - Minimum number of, maximum number of, and size of intersection of neighbors of both questions. This is extended to second and third degree neighbors as well which aren't neighbors of any lower degree set of neighbors.
   - Shortest path from q_1, q_2 when $E_{q_1 \longleftrightarrow q_2}$ (the edge connecting the two questions) is cut.

   For each **connected component** in the graph (a connected component is any subset of nodes in a graph such that any two nodes in the set are reachable by a path, and no node in the set is connected to a node outside of the set) they compute the size of the connected component, the percentage of the nodes in the connected component that are part of the training set, and the percentage of duplicated pairs in the connected component.

With these three techniques, they apply **model stacking**, which is the process of using models sequentially. The output of one model becomes the input to another. This is a powerful technique because different stages of the learning process are better suited by different models. However, this technique is exceptionally prone to overfitting. They have four main phases which stack in excess of 450 smaller models. This is perhaps one of the most extreme approaches to this dataset, and required 1 week to run all of the models on 10 GPU machines with 32GB RAM working in conjunction with 80 CPU machines with 120GB RAM. This related work is interesting because it pushes the boundaries on this dataset; however it is neither feasible to use in production models nor does it represent a particularly intuitive advancement in approaching the problem.

5.2 Quora's Approaches

Quora (Dandekar et. al, 2017) currently uses a Random Forest model in production to detect duplicate questions. They utilize a number of handcrafted features on the order of 10^4, including

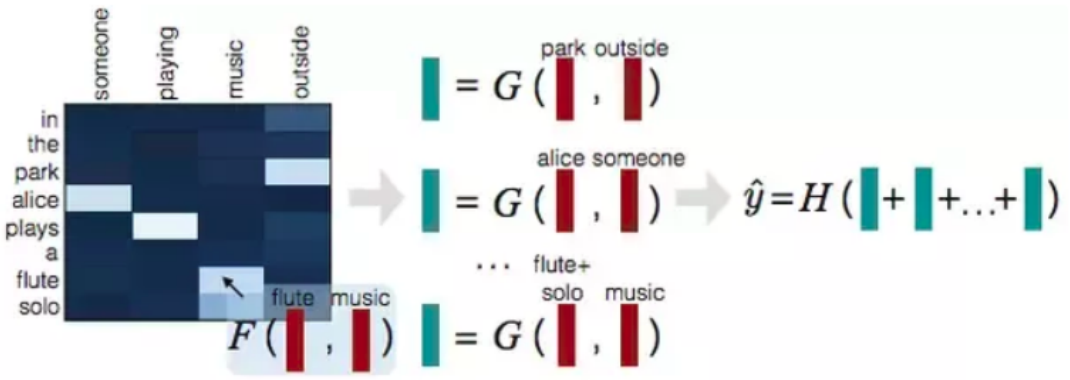


Figure 5.1: Decomposable Attention Model Detail (Quora, 2017)

- Cosine similarity of the average of the `word2vec` embeddings of tokens. For a pair of questions (u, v), the cosine similarity of the average would be calculated as follows.

  Let E : String $\rightarrow \mathbb{R}^n$ be a function that maps an input string (token) to an n-dimensional embedding vector, and let $(u_1, u_2, ..., u_m)$ and $(v_1, v_2, ..., v_m)$ be the set of tokens making up u and v respectively. Assume that the vectors are capped to a fixed length and padded with `STOP` words as necessary. Then the cosine similarity of the average of the embedding vectors is defined as:

$$\cos(\theta) = \frac{\frac{1}{m}\sum_{i=1}^{m} E(u_i) \bullet \frac{1}{m}\sum_{i=1}^{m} E(v_i)}{||\frac{1}{m}\sum_{i=1}^{m} E(u_i)||_2 * ||\frac{1}{m}\sum_{i=1}^{m} E(v_i)||_2}$$

- Number of common words shared by a question pair
- Number of common topics labeled on the questions
- Part-of-speech tags of the words.

They also tried LSTM-based deep learning approaches similar to those described in this thesis combined with hand-crafted features such as distance and angle between embedding vectors.

Their last approach involved "decomposable" attention which was inspired by work from Parikh et. al (2016) at Google Research (Figure 5.1). This approach involved decomposing the input questions into pairs of words from both questions, comparing phrases, and using comparison results in their final model for classification. Decomposable attention scored 1% higher in testing accuracy compared to their LSTM-based approaches.

5.3 Improvements on LSTM Networks

Tai et. al (2015) proposed the TreeLSTM model as an enhancement to the standard LSTM described in Section 2 (Background Information). A standard LSTM features a linear structure. The motivation for the TreeLSTM lies in the fact that humans often process words in a piece of text as

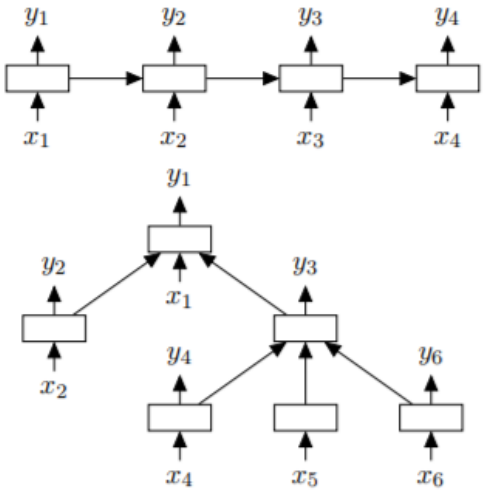


Figure 5.2: TreeLSTM vs Standard LSTM structure (taken from Tai et. al, 2015)

groups rather than individually one-by-one. The tree-like structure helps introduce this dependency to better represent the human cognitive process.

The TreeLSTM model does exceptionally well in some tasks such as the Sentences Involving Compositional Knowledge (SICK) dataset (Marelli et al., 2014), in which there are sentence pairs and a corresponding score from 1 to 5 representing how related the two sentences are. It outperforms existing baselines from standard LSTMs, both single and multi-layer.

5.4 GLUE: General Language Understanding Evaluation

The Quora Questions Pairs dataset is also featured in the GLUE benchmarking system by Wang et. al (2018). Directly from the paper, GLUE is an "online tool for evaluating the performance of a single NLU model across multiple tasks, including question answering, sentiment analysis, and textual entailment, built largely on established existing datasets".

Not surprisingly, most of the top performers on this benchmark use what is known as multi-task learning (MTL) (Caruana, 1997), where a classifier is built to tackle several tasks at once by taking advantage of commonalities between the tasks. MTL can be a powerful technique because it prevents the model from overfitting too much to any one particular task and it is often observed that MTL models have strong generalization results to unseen data. Examples of leaders on the GLUE results board include Microsoft AI's MT-DNN (Multi-Task Deep Neural Networks) and Stanford Hazy Research Group's Snorkel MeTaL, with 89.6% and 90.1% accuracy on the test set respectively.

There are also numerous approaches involving BERT (Bidirectional Encoder Representations from Transformers) by Devlin et. al (2018) from Google Research, which does not use a fixed word to embedding mapping such as that provided by GloVe. Instead, the embedding of a word or a

token can vary depending on the context in which it is used. The result is a far more powerful and descriptive embedding, and BERT has been used to achieve state-of-the-art results on many existing NLP tasks.

MTL and BERT-based approaches score anywhere from 87.3% to 90.3% test accuracy on the GLUE benchmarking system for Quora Question Pairs.

Chapter 6

Conclusions

The problem of determining whether two questions are the same is a very hard problem due to the large number of ways in which one question can be permuted or modified while retaining its original semantics.

It is clear that rote systems such as rule-based approaches would experience poor performance on this as they are not able to fully model the complexity of this problem.

Deep Learning approaches represent a significant advancement as we can now learn question structures far more accurately through deep networks, especially those that humans are not be able to intuitively pick up on.

The key enhancement to existing Deep Learning approaches is the introduction of context. There are several recent techniques that help a deep language model become context aware including bidirectional RNNs (which is a superset of LSTMs and GRUs), as well as BERT, which provides models for context-aware embeddings for words (tokens). BERT represents a significant upgrade over previous techniques such as GloVe, in which a word receives the same embedding regardless of the context it is used in. It is not surprising that both bidirectional RNNs and BERT have been applied in various tasks to achieve state-of-the-art performance. The use of context when determining if two questions are the same is critical because it both simplifies the problem and increases accuracy. We are able to focus on just two things: (1) the words in the question and their meaning with regards to the overall context and (2) how their meanings relate and compare to those from the other question.

Chapter 7

Future Work

7.1 Feature Engineering

Some of the low-hanging fruit that could be explored includes feature engineering, which is the process of designing additional features based on the data and using them as input to the network to improve performance. Possibilities in this case include Euclidean distance between word embeddings and maximum number of common words shared between questions. As seen in Section 5 (Related Work), feature engineering is heavily leveraged in numerous other approaches to this dataset. The main drawback is that it takes significant time to design and compute these features. There is also risk of poor generalization if the training-testing split is not done properly.

7.2 Alternative Approaches and Enhancements

Kim (2014) has published promising work on using CNNs (Convolutional Neural Networks) in NLP, by first converting text to a set of vectors using pre-trained word embeddings and then joining together the vectors to make an image. Work has been done with CNNs in NLP with good results. Yin et. al (2017) provide some analysis on the relative performance of CNNs vs RNNs for language tasks. The intuition is that CNNs would pick up better on tasks such as sentiment analysis as they can learn to act on the image representation of key phrases, and RNNs would pick up better on tasks such as language modeling where context is more important. Although there is no decisive agreement on which one is better, Yin et. al find that CNNs and RNNs could provide complementary information for text classification tasks. They also find that GRUs tend to do better in specific cases, i.e. when sentiment of the entire sentence is required, as well as when the sentence or sequence is longer in length (above 10 tokens). Combining the two in a hybrid approach could achieve better performance.

Performance could also be improved through the use of TreeLSTMs mentioned in Section 5 (Related Work). Due to the computational cost to train and evaluate them, they were not explored

in this thesis, however, they have offered 1% accuracy gain in tasks such as the SICK dataset over standard LSTMs.

Lastly, BERT (Bidirectional Encoder Representations from Transformers) and multi-task learning could be applied to this task to potentially improve performance. The BERT pre-trained embedding vectors have higher dimensionality (768 for the base version, and 1000+ for the large version) which allows for a richer feature space. More importantly, an embedding vector for a word is able to capture context of how the word is being used in text. The reasons BERT was not attempted in this thesis are as follows:

- The Quora dataset is composed of a total vocabulary of almost 90,000 words, which is significantly larger than the vocabulary that was available with BERT's pre-trained model of 30,000 words. It would be difficult to match embedding vectors and we would end up having to initialize random embedding vectors for tens of thousands of words in our vocabulary.
- The BERT (`bert-base-uncased`) vocabulary includes non-ASCII characters and words which simply do not appear in the Quora dataset.
- Due to the first two reasons, we would have had to fine-tune the BERT model for our own use case, which is computationally expensive and best done on a TPU (Google's specialized hardware for training).

7.3 Moonshots

I am also interested in exploring adversarial approaches to the Quora Questions dataset. For example, if we could build a "generator" to generate fake question pairs that could fool our "discriminator" (which is the SiameseGRU model here), that would be incredibly interesting. The difficulty (and the difficulty in applying adversarial methods to natural language tasks in general) lies in generating fake question pairs that still have valid semantics. A basic approach could involve (1) nearest-neighbor lookup in the embedding space for either the embedding vector of a word or its opposite (negation), which is essentially finding synonyms or antonyms and using those as replacements or (2) converting questions to images and then training a Generative Adversarial Network (GAN). A more advanced approach could take advantange of OpenAI's recent GPT-2 model, which is able to generate coherent paragraphs of fake information.

Bibliography

[1] R. Caruana. Multitask learning. Machine learning, 28(1), 1997

[2] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, Learning phrase representations using rnn encoder-decoder for statistical machine translation, arXiv preprint arXiv:1406.1078, 2014.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555

[4] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, Natural language processing (almost) from scratch, Journal of Machine Learning Research, vol. 12, no. Aug, pp. 24932537, 2011.

[5] Sebastien Conort, Lam Dang, Guillaume Huard, Paul Todorov, and Maximilien Baudry. Kaggle - Winning Submission on Quora Question Pairs Dataset, 2017.

[6] Kornél Csernai, Shankar Iyer, and Nikhil Dandekar. First Quora Dataset Release: Question Pairs.
`https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs`, 2017.

[7] Nikhil Dandekar, Hilfi Alkaff, Shuo Chang, Kornél Csernai and Lili Jiang. Semantic Question Matching with Deep Learning.
`https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning`, 2017.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[9] Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. The Journal of Machine Learning Research, 12:21212159, 2011.

[10] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature 405, 6789 (2000), 947.

[11] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural computation, vol. 9, no. 8, pp. 17351780, 1997.

[12] Y. Kim. Convolutional neural networks for sentence classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 17461751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. CoRR, abs/1412.6980, 2014.

[14] Ma, Jerry and Yarats, Denis. Quasi-hyperbolic momentum and Adam for deep learning. In Proceedings of the International Conference on Learning Representations, 2019.

[15] Marelli, Marco, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. 2014. SemEval-2014 Task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In SemEval 2014.

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 31113119.

[17] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. In Soviet Mathematics Doklady, volume 27, pages 372376, 1983.

[18] Olah, Christopher. Understanding LSTM Networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015.

[19] Ankur P Parikh, Oscar Tackstrm, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2016.

[20] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In NIPS 2017 Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques, Long Beach, CA, US, December 9, 2017, 2017.

[21] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2014.

[22] Radford, Alec and Wu, Jeff and Child, Rewon and Luan, David and Amodei, Dario and Sutskever, Ilya (2019). Language Models are Unsupervised Multitask Learners.

[23] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. Signal Processing, IEEE Transactions on, 45(11), 26732681.

[24] K. S. Tai, R. Socher, and C. D. Manning, Improved semantic representations from tree-structured long short-term memory networks, arXiv preprint arXiv:1503.00075, 2015

[25] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461, 2018

[26] Weizenbaum, J. (1965) Eliza - a computer program for the study of natural language communication between man and machine. Communication of the Association for Computing Machinery 9:36 45. [JRS]

[27] W. Yin, K. Kann, M. Yu, and H. Schtze, Comparative study of CNN and RNN for natural language processing, CoRR, vol. abs/1702.01923, 2017. [Online]. Available: http://arxiv.org/abs/1702.01923