Exploring Different Matching/Attention Mechanism for Machine Comprehension Task on SQuAD dataset

Nattapoom Asavareongchai
Department of Electrical Engineering
Stanford University
nasavare@stanford.edu

George Pakapol Supaniratisai
Department of Computer Science
Stanford University
pakapol@stanford.edu

Abstract

For our default final project, we implemented a model for machine reading comprehension through the use of Deep learning. We trained our model using the SQuAD dataset. The aim of our model is to create a high-performance question answering tool through the incorporation of different techniques proposed by recent literatures, two in particular, and our own modifications of each layer in the pipeline.

1 Introduction

One prevalent NLP problem that is a big part of research today in machine learning and artificial intelligence is the problem of machine comprehension (MC). This problem aims to create a machine, or an AI model, that is able to reliably read a passage with a certain context, extract information and knowledge from this context, and answer sets of question asked about the passage correctly.

In order to accomplish this task, we aim to build a neural network model that uses the Stanford Question Answering Dataset or SQuAD [1], to train. The SQuAD dataset consists of pairs of context paragraphs and corresponding questions about the passage within the context paragraphs, as well as answers to the question. The dataset can then be processed and consumed by the MC model in the following way. The input of the model are the pair of paragraphs and questions and the output are the answers to the questions.

This problem is a popular problem with very recent ongoing research with literatures that have proposed different techniques to have the machine incorporate contextual information from the context passages and the questions. With the many existing models in literature, our model will try to replicate (with some simplifications) and modify plus build on techniques used by Wang Zhiguo, et al. called "Multi-Perspective Context Matching" [2]. We will also then combine Minjoon Seo, et al.'s "Bidirectional Attention Flow" [3] technique to create an ensemble and improve our model.

2 Related Works

Both works by Wang Zhiguo, et al. on "Multi-Perspective Context Matching" [2] and Minjoon Seo, et al. on "Bidirectional Attention Flow" [3] use similar pipeline structure for the MC model. The model contains roughly 5 stages. The first stage consists of preprocessing. Both papers utilizes both word embeddings and character-composed embedding to represent each word in the passages and questions. The next 4 stages in the model includes encoding the word representations, matching passage and question representations, decoding the matched outputs, and predicting the answers (more specifically, predicting the start word and end word of the answer sentence within the passage). However, the details of the 4 stages differ for both paper's approach.

2.1 Multi-Perspective Context Matching

Wang Zhiguo, et al's work on multi-perspective context matching encodes the word representation of the passages and questions by filtering using a cosine similarity matrix and applying bi-directional LSTM [7]. They then fed the output of the encoder to a *matcher* layer. This layer matches the states from the passage and questions through three different techniques, including 'Full Matching', 'Maxpooling-Matching', and 'Meanpooling-Matching' [2]. These matching technique use l different perspectives to capture different dimensions of their relationships. They then decoded the matched outputs using another bi-directional LSTM and predicted using softmax prediction.
Their experiments gave out good results and thus we will use this as our baseline model. Details on the literature's techniques will be described later in our baseline model description.

2.2 Bi-directional Attention Flow

As opposed to filtering using a cosine similarity matrix in the encoding step, Minjoon Seo, et al's model uses a Highway Network [4] to first process the passage and question word and character embeddings and then feed them into the bi-directional LSTM layer for encoding. The output of the encoder layer is then passed into the *attention flow* layer. This layer calculates a context-to-query attention and a query-to-context attention using a similarity matrix calculated from a learned linear function [3]. The outputs of this layer are combined and fed into a two layer bi-directional LSTM for decoding (different from the one layer used by the multi-perspective matching). Prediction of start word is then done through softmax prediction with a further one layer bi-directional LSTM and softmax for the end word.
The results obtained by this model is also comparable to multi-perspective matching model and thus we will also build a second baseline off of this model. Details on the replication of the paper with simplification will be described in the model & approach section.

3 Model & Approach

3.1 Baseline model 1

Our first baseline model initially tries to replicate the "Multi-Perspective Context Matching" pipeline and techniques with some simplifications and some of our additions.

► ***Preprocessing:***
For preprocessing, we loaded in the pre-trained GloVE word embeddings [5] with a dimension of 300. To simplify our preprocessing model, we did not include a character-composed embedding representation that was included in the paper. Each words in the passage and questions are then mapped to the list of GloVE word embeddings through integer IDs and thus the passages and questions are represented by a list of integer IDs. We then set each passage to have a fixed length of 768 words and set each question to have a fix length of 32 words. We enforce this by concatenating with zeros if the sentences are too short or cut the sentences short if they are too long.

► ***Encoder:***
The first step of our encoder is to filter the word embeddings of the passage/context word using the relevance matrix R. Each element $r_{i,j}$ in the relevance matrix R is defined as:

$$r_{i,j} = \frac{q_i^T p_j}{\|q_i\| \cdot \|p_j\|}$$

Where q_i is the i-th word of the question for $i \in (1, ..., 32)$ and p_j is the j-th word of the passage/context for $j \in (1, ..., 768)$. This relevance matrix is the same defined by Wang Zhiguo, et al.'s paper [2]. Filtering is done by creating $p_j^{'} = r_j \cdot p_j$ where $r_j = \max_{i \in (1,...,32)} r_{i,j}$

We then applied a one layer bi-directional GRU [6] (different from the LSTM used in the paper) on the list of passage words and another one layer bi-directional GRU on the list of question words. We applied dropout [8] to both the inputs and outputs of each bi-directional GRU at the rate of

0.15 (0.2 used by Wang Zhiguo, et al [2]). In addition to this, we passed the outputs of the passage bi-directional GRU as input of the bi-directional GRU as well by setting it as an initial state of the bi-directional GRU. We implemented this so that the question GRU can take into account the contextual representation of the passages as well (to learn their relationship early on).

► ***Matcher:***
The output of the encoders are then fed into our matcher where we utilized the 'Full-matching' technique used by Wang Zhiguo et al. We did not use Maxpooling nor Meanpooling for simplification of the baseline and for efficiency, since these two techniques requires a lot of memory. As defined in the paper [2], mutli-perspective matching first uses a function f_m to create a l-dimensional vector:

$$\mathbf{m} = f_m(v_1, v_2; \mathbf{W})$$

$$\mathbf{m} = [m_1, m_2, ..., m_l]$$

$$m_k = cosine(W_k \circ v_1, W_k \circ v_2) \text{ for } k\text{-th perspective}$$

where $\mathbf{W} \in \mathbb{R}^{\mathbf{l} \times \mathbf{d}}$ is a parameter to be trained and d is the dimension of v_1 and v_2. Then for full-matching, for both the forward and backward pass of the encoder output we use:

$$\overrightarrow{\mathbf{m}}_j^{full} = f_m(\overrightarrow{\mathbf{h}}_j^p, \overrightarrow{\mathbf{h}}_{32}^q; \mathbf{W^1})$$

$$\overleftarrow{\mathbf{m}}_j^{full} = f_m(\overleftarrow{\mathbf{h}}_j^p, \overleftarrow{\mathbf{h}}_1^q; \mathbf{W^2})$$

Note that only the first and last output of the question encoder is used for matching, but all outputs from the passage encoders are used. (Note: Equations are cited from Wang Zhiguo et al.'s paper [2])

In addition to the paper's full matching technique using v_1 and v_2, we also added an additional vector v_3 to the full-matching technique.

$$v_3 = v_1\mathbf{W}_{v_1} + v_2\mathbf{W}_{v_2} + \mathbf{b}_{v_3}$$

where W_{v_1}, W_{v_2}, and b_{v_3} are variables to be learned. v_3 is then used instead of v_1 to calculate $\mathbf{m}^*$.

$$\mathbf{m}^* = f_m(v_3, v_2; \mathbf{W})$$

$$\overrightarrow{\mathbf{m}^*}_j^{full} = f_m((\overrightarrow{\mathbf{h}}_j^p\mathbf{W}_{v_1} + \overrightarrow{\mathbf{h}}_{32}^q\mathbf{W}_{v_2} + \mathbf{b}_{v_3}), \overrightarrow{\mathbf{h}}_{32}^q; \mathbf{W^3})$$

$$\overleftarrow{\mathbf{m}^*}_j^{full} = f_m((\overrightarrow{\mathbf{h}}_j^p\mathbf{W}_{v_1} + \overrightarrow{\mathbf{h}}_1^q\mathbf{W}_{v_2} + \mathbf{b}_{v_3}), \overleftarrow{\mathbf{h}}_1^q; \mathbf{W^4})$$

The hyper-parameters we used are l = 48 perspectives and the hidden state size of the bi-directional GRU is 96 (as opposed to 100 used by the paper).

► ***Decoder:***
The output of the matcher are then concatenated into a vector $[\overrightarrow{\mathbf{m}}_j^{full}; \overleftarrow{\mathbf{m}}_j^{full}; \overrightarrow{\mathbf{m}^*}_j^{full}; \overleftarrow{\mathbf{m}^*}_j^{full}]$ and bi-directional GRUs are used with application of dropout [8] in both the input and output with dropout rate of 0.15. This is to aggregate the matching outputs and learn the contextual relationships.

► ***Prediction:***
For our prediction stage we apply softmax on the output of the decoder to classify each word in the passage as either a starting word or end word of the answer. Cross-entropy loss of our prediction is used as an objective function. We did not enforce the constraint that the start indices should come before the end indices, for simplicity.
The model is trained to minimize cross-entropy loss using the AdamOptimizer [9] with a learning rate of 0.0001.

3.2 Baseline model 2

To create an ensemble of techniques in our model, we decided to build from scratch a second baseline model that tries to replicate Minjoon Seo, et al.'s "Bidirectional Attention Flow" model, with some simplification. This model is independent to our first baseline. We will then try to combine these two models later.

► ***Preprocessing:***
Similar to our first baseline model we used pre-trained GloVE word embeddings [5] of size 300 to represent the words in the passages/contexts and questions/queries. We also represented each word through an integer ID as explained earlier. However, for this model we decided to cut the length of the passage to a fix length of 256 words instead of 768 as before. We fixed the length of the question at 32 words.

► ***Encoder:***
Before encoding the passages and questions, we first pass the word embeddings of these two structures through a Highway Network [4], where the parameters will be trained by the model. We used a tanh activation function for the network. The output from the highway network is then passed into a one layer bi-directional GRU [6] (different from the LSTM used by the paper [3]). We set the state size of the GRUs to 96 and applied dropout [8] at a rate of 0.25 (an increase because our first baseline model over-fits, as shown in the results section).

► ***Attention Flow:***
The first step of the attention flow layer is to calculate a similarity matrix **S**. This similarity matrix is different from R specified by the first baseline. Lets denote **H** to be the matrix of the outputs of the passage encoder (dimension $\mathbb{R}^{state\_size \times time\_step}$). Let **U** be the matrix of outputs of the question encoder (dimension is the same as **H**). According to the paper [4]:

$$\mathbf{S}_{i,j} = \mathbf{W}_{(\mathbf{S})}^{T}[\mathbf{H}_{:,i}; \mathbf{U}_{:,j}; \mathbf{H}_{:,i} \circ \mathbf{U}_{:,j}]$$

where $\circ$ means element-wise multiplication, **W** is trainable weight and i is the i-th column of **H** and j is the j-th column of **U**.
S is then used to calculate a context-to-query attention matrix and a query-to-context attention matrix.
The *context-to-query attention matrix* $\tilde{\mathbf{U}}$ is defined as follows:

$$\tilde{\mathbf{U}}_{:,i} = \sum_{j} \mathbf{a}_{i,j} \mathbf{U}_{:,j}$$

$$\mathbf{a}_i = \mathrm{softmax}(\mathbf{S}_{i,:})$$

The *query-to-context attention matrix* $\tilde{\mathbf{H}}$ is defined as follows:

$$\tilde{\mathbf{h}} = \sum_{i} \mathbf{b}_i \mathbf{H}_{:,i}$$

$$\mathbf{b} = softmax(\mathrm{max}_{col}\mathbf{S})$$

The resulting $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{H}}$ are then combined to form a matrix **G** where **G** = [**H**; $\tilde{\mathbf{U}}$; $\tilde{\mathbf{U}} \circ$ **H**; $\tilde{\mathbf{H}} \circ$ **H**]. (NOTE: All these equations are cited from Minjoon Seo, et al's model paper [3])

► ***Decoder:***
The output of the attentions flow **G** is then decoded using a two layer bi-directional GRU [6] with dropout [8] rate of 0.25. The output of the decoder is denoted **M**.

► ***Prediction:***
To predict the starting and end word of the answer within the passage, we used all the same techniques given by the literature's model [3]. We then enforced the constraint that the ending word must come after the start word. We did this by maximizing the sum of the softmax scores of the

starting word and end work through dynamic programming, such that $k < j$, where k = index of start word and j = index of end word.
The start word index is predicted using softmax prediction on the linear transform of matrix $[\mathbf{G}; \mathbf{M}]$.
The end word index is predicted using softmax prediction on the linear transform of matrix $[\mathbf{G}; \mathbf{M}^2]$ where $\mathbf{M}^2$ is the output of a one layer bi-directional GRU with $\mathbf{M}$ as inputs.
We then trained the model to minimize the cross-entropy loss using the AdamOptimizer [9] (Minjoon Seo, et al used AdaDelta optimizer [3]) with a learning rate of 0.01.

3.3 Final Model

For our third and final model, we decided to build on and improve our second baseline model since the model is easier to debug and understand. We figured out that having the cosine similarity function in our first baseline caused numerical stability problems and so we decided not to use it further in our improvements. Results from the two models (as shown in Table 1 and Table 2 below in *Results & Analysis* section) are very similar and so building on either ones are fine.

► ***Modifications:***
The modifications we made for model 3 that's different from our baseline 2 are:

- We decayed the learning rate from 0.1 down by 0.1% (or rate of 0.999%) every iteration. This is because our baseline model takes many epoch to train and seems to improve slowly. With a decaying learning rate, the learning speed sped up quite significantly. We initially had to run nearly 100 epochs for our baselines for learning to plateau and start to overfit, however, with the decaying rate, around 10 epochs were sufficient.
- We decided to use LSTM [7] instead of GRU for our bi-directional RNNs. This is because LSTM cells in Tensorflow allowed us to set the forget gate bias to 1 allowing more learning to happen. We figured that it was harder to do this with GRU cells.
- We concatenated $\mathbf{H}$ and $\mathbf{U}$ from the encoder layer with word embeddings from the preprocessing layer to allow the vector/matrix to store previous information.
- We changed the initialization of $\mathbf{W_{S)}}$ and variables used to calculate softmax in our prediction layer. We initialized the parts that will multiply with $\mathbf{h}$ and $\mathbf{u}$ to all 0's instead of randomization and the parts that will mutliply with $\mathbf{h} \circ \mathbf{u}$ to very small positive values, less than 1. This is to default the similarity to 1 and because there are no symmetry there is no need for randomization.
- In our baseline models, since we padded and truncated the passage and questions, we needed to mask the output of the RNN. However, we initially though the *actual_length* function in the GRU/LSTM cell did this for us. However, we noticed that this is not the case and so we had to mask the output ourselves in this model.

4 Results & Analysis

We evaluated all our models using F1 and EM evaluations. We both looked at the results with our training set as well as the development set to test our generalization of our model. The tables below show our results for all our models.

Train Eval (%) / Model	F1	EM
Baseline 1	52.4	36.6
Baseline 2	57.0	40.7
Final model	65.5	51.1

Table 1: Evaluation on our training set

Dev Eval (%) / Model	F1	EM
Baseline 1	42.1	28.0
Baseline 2	42.2	26.8
Final model	54.5	39.7

Table 2: Evaluation on our Development set

Our baseline models (both 1 and 2) seems to produce not so good results with F1 of around 40 and EM around 28 for the development set. However, with some tuning and small modifications, we were able to bring F1 up to 54.5 and EM to 39.7, which were relatively good.

We then plotted the changes in loss, F1 and EM while training using our final model. This is done to visualize how the model learns through time.

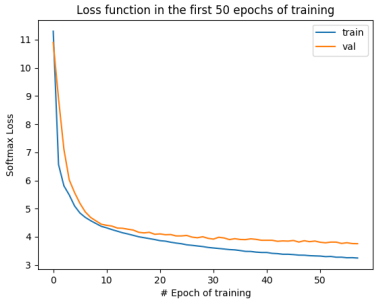

Figure 1: final model train and validation loss as function of epoch

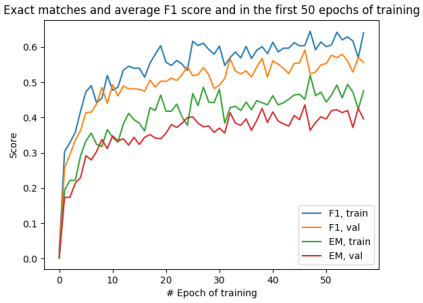

Figure 2: final model train and validation F1 & EM as function of epoch

To further try to investigate the patterns of the answers our model predicts, we tried to record the frequency of the length of each answers in the development set and how well each length categories' F1 and EM scores are. We separate the lengths into answers of length 1 word to 7 words and a final 8 or more words (only mentioned as 8 in the graph/chart). We will then compare the frequencies with the ground truth frequencies for comparisons. Since the ground truth contains more than 1 answer, we will average and round the length of answers for each question. Figures below shows the outcome:

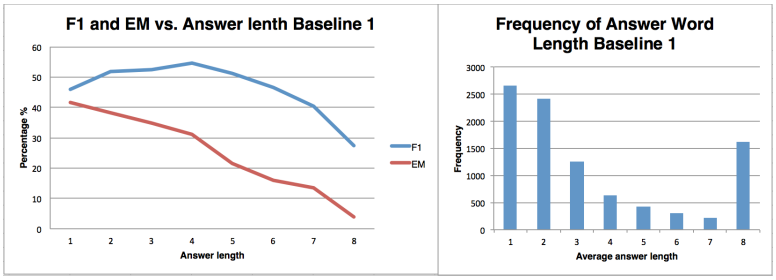

Figure 3: Baseline 1 Word length Frequencies analysis of F1 and EM (NOTE: answer length of 8 means 8 or more words in the answer)

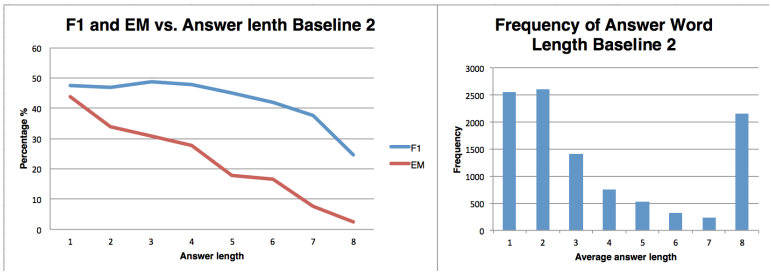

Figure 4: Baseline 2 Word length Frequencies analysis of F1 and EM (NOTE: answer length of 8 means 8 or more words in the answer)

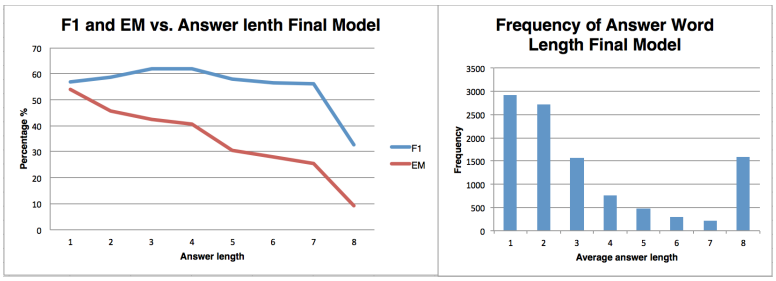

Figure 5: Final Model Word length Frequencies analysis of F1 and EM (NOTE: answer length of 8 means 8 or more words in the answer)

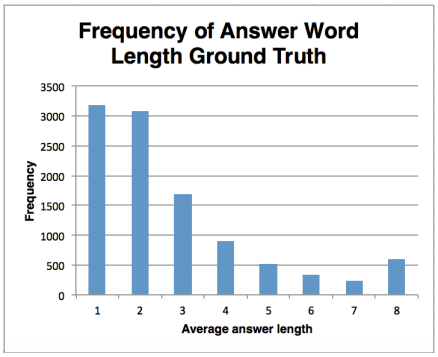

Figure 6: Ground truth Word length Frequencies analysis (NOTE: answer length of 8 means 8 or more words in the answer)

From our baseline models, it seems like the high F1's were answers between 1 and 6. Whereas for our final mode, high F1's were consistent from word length of 1 through 7. All F1 scores were decreased dramatically with word lengths 8 or more. As for EM scores, all our models show steady decrease as word length increases.

However, we also noted the frequency of the number of words in each answer. As we see from the ground truth, most answers contain between 1,2, or 3 words and longer length answers are minimal. This can also be seen in our models except for the fact that for all our models we created a lot more answers with 8 or more words. The models high prediction frequency of 8 or more word answers shows a big con/mistake our model makes and a point for improvement.

For visualization of the similarity matrix produced by our final model during the attention flow layer, we plotted the figure below:

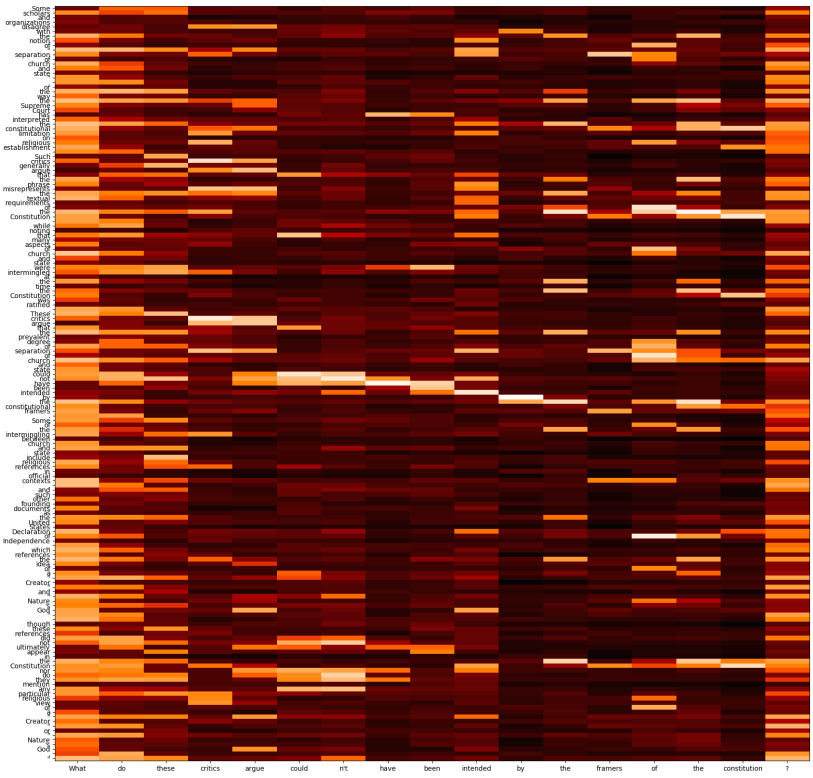

Figure 7: Visualization of the similarity matrix. Y-axis words are words in a sample passage and the x-axis words are words in the corresponding question. The brighter the color in the matrix representation, the more similar the words are in the context.

5 Conclusion & Future Works

Our model are far from perfect and more time is needed to tweak hyper-parameters and fix numerical issues to bring F1 and EM accuracy up to par with recent literatures. We concluded that both bi-directional attention flow model and multi-perspective matching model produce similar results but because the matching model uses more memory and has more numerical issues, the attention flow model can be run more smoothly.

With more time in the future, we could improve our model and try out new ideas. These could include:

- Using reinforcement approach to solve the problem - this is by having the model read the question first and from the question information read the passage with a set of action policies.
- Incorporate sentence chunking - we noticed that many answers were contained in only one sentence. Our current model does not separate sentence and combined them into one whole passage. We could do an initial step of separating out sentences and choosing one that is most likely to contain the answer, then run the current model on only that sentence.
- Visual Question Answering - we could extend the task of machine comprehension to visual comprehension or by answering questions asked about images instead of text passages.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. CoRR, abs/1606.05250, 2016.

[2] Wang, Zhiguo, et al. "Multi-Perspective Context Matching for Machine Comprehension." arXiv preprint arXiv:1612.04211 (2016).

[3] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603, 2016.

[4] Srivastava, Rupesh Kumar, Klaus Greff, and Jrgen Schmidhuber. "Highway networks." arXiv preprint arXiv:1505.00387 (2015).

[5] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." EMNLP. Vol. 14. 2014.

[6] Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555 (2014).

[7] Hochreiter, Sepp, and Jrgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[8] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of Machine Learning Research 15.1 (2014): 1929-1958.

[9] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).