Exploring Deep Learning Solutions for Question-Answering and Reading Comprehension Tasks

Kimberly Wijaya
Department of Computer Science
Stanford University
Stanford, CA 94305
kcwijaya@stanford.edu

Rodrigo Grabowsky
Department of Computer Science
Stanford University
Stanford, CA 94305
rmgrab@stanford.edu

Abstract

Machine Comprehension has long been a desired goal for the machine learning community. In an effort to build such a Question Answering system, this paper will describe a model that incorporates several approaches in deep learning, inspired by leading QA systems, to achieve fairly competitive F1 and Exact Match (EM) scores when evaluated on the Stanford Question Answering Dataset (SQuAD). In particular, this model employs Global Vectors or Word Representation (GloVE) pre-trained word embeddings augmented with a character-level convolutional neural network (CNN) embedding layer; a bidirectional recurrent neural network (RNN) composed of Gated Reccurent Units (GRUs) for question and passage encoding; a bidirectional attention flow layer; a 2-layer bidirectional RNN composed of Long-Short Term Memory Cells (LSTMs) for modeling; and an output layer that uses the joint probability distribution of answer start and end positions to find the optimal answer span.

1 Introduction

In his comprehensive 2013 paper, Christopher J.C. Burges, at the time a member of Microsoft Research (MSR), states that "a machine comprehends a passage of text if, for any question regarding that text that can be answered correctly by a majority of native speakers, that machine can provide a string which those speakers would agree both answers that question, and does not contain information irrelevant to that question." [1] This challenge of utilizing a question-answering paradigm to train and validate a machine's reading comprehension abilities is at the center of this project.

To meet this challenge, we sought to design and construct a model that, when given a paragraph and a corresponding question, is able to answer the question correctly. The model's success in this endeavor would thus parallel its ability to "understand" text that it is given. In particular, this model is centered around the Stanford Question Answering Dataset (SQuAD). This dataset is composed of paragraphs from Wikipedia, with its approximately one hundred thousand questions and answers crowdsourced using Amazon Mechanical Turk, which is the field's standard.

To help mitigate the cumbersome task of enumerating such possible dependencies and features, recurrent neural networks (RNN) have become a popular solution to such tasks due to their usefulness in processing sequential data, as is the case with natural language. Our model uses many layers of different types of neural networks and other techniques such as attention to identify answer candidates within the passages for a given question. The words themselves are represented as word and character embeddings. The start and end indices for the ultimate answer span are identified by building a joint probability distribution of the answer start and end positions and finding the span

that maximizes that probability. The specifics of the constructed model will be further explained in later sections of this paper.

Our model improves upon a baseline model give to us by experimenting with ways to make each of its layers more sophisticated. While not quite reaching the levels of performance of some other existing SQuAD systems, the model still performs fairly well.

2 Related Work

2.1 SQuAD

SQuAD, briefly mentioned in the introduction of this paper, is an interesting dataset in that it was not designed to specifically test out reading comprehension, as it is defined by most other fields. Rather, as noted by Yoav Goldberg, a lecturer at Bar Ilan University, SQuAD "was designed as a benchmark for machine learning methods, and the human evaluation was performed to assess the quality of the dataset, not the humans' abilities." [2] As such, it serves as a particularly suitable dataset for the task of exploring this frontier of machine learning. Additionally, its unique characteristic of having answers span across several words, not necessarily of a named-entity category, sets it apart form other close-style queries. This places SQuAD firmly in open-domain question answering, with special attention made to the diversity of answer types, the difficulty of questions, and the divergence in syntax between questions and their corresponding answers.

2.2 Traditional Models

Traditionally, there are two major modern paradigms for question answering: IR-based question answering and knowledge-based question answering. The former tends to focus on passage retrieval, as is the case with SMART [3] and the works of Soubbotin et al [4], and Ravichandran et al [5]. The knowledge-based paradigm is grounded in natural language processing, and is often centered around converting text input into formal, mathematical representations [6]. In the creation of the aforementioned SQuAD, Rajpurkar et al implemented a logistic regression model with a wide range of features. [7]. A number of issues presented themselves in this endeavor: the model's success was inversely proportional to the number of answer types and the divergence between the syntax of the question and the sentence. This logistic regression model, as with many other traditional models, is composed of approximately 180 million features. Despite this, its performance is significantly worse than human performance, with an F1 score of 51.0% as opposed to the 86.8% achieved by humans. This leaves ample room for improvement.

2.3 Deep Learning Models

To address this performance gap, as well as the complexity and inefficiency of these traditional models, researchers have begun exploring the use of neural nets for question-answering tasks.

Yu et al [8] proposed the dynamic chunk reader (DCR), described as an "end to end neural reading comprehension model that is able to extract and rank a set of answer candidates from a given document to answer questions." This model uses deep networks in order to learn better representations for candidate answer spans. This addresses the problem of having features in the order of hundreds of millions, as in the traditional model baseline. Additionally, the answers candidates are represented as titular chunks, rather than word-representations, in order to further differentiate among candidates. DCR works in four steps: an encoder layer composed of a bidirectional RNN; an attention layer to rank word relevance; a chunk representation layer to extract chunks from passages and encode contextual information, and, finally, a ranker layer that scores chunk relevances with a softmax layer. This model achieves a 71.0% F1 and 62.5% Exact Match score for the test set, significantly higher than the traditional logistic regression model.

Wang et al [9] utilized a model whose architecture is based on match-LSTM and Pointer Net (Ptr-Net) in order to complete this task. The match-LSTM system was originally developed for textual entailment, but was adapted for SQuAD after the observation that many questions paraphrase sentences from the passages. The Ptr-Net allows the model to generate answers composed of several

tokens from the passage itself. This end-to-end neural network achieves an F1 test score of 77.0% and EM 67.9%.

The Bi-Directional Attention Flow (BiDAF) network was introduced by Seo et al [10] as a hierarchical multi-stage model for modeling context paragraph representations at different levels of granularity. It uses a variety of different embeddings and attention flows in order to ensure its representation of contexts are query-aware. There is particular emphasis on its attention layer, which allows information to flow through the modeling layer, reducing the information loss. By simplifying the attention layer to be memory-less, it also distributes the work that needs to be done between both the modeling *and* the attention layer. BiDAF outperforms all other SQuAD systems by achieving a test F1 score of 81.1% and Exact Match score of 73.3%.

3 Approach

Our approach to the SQuAD channel incorporates a combination of the improvements from the models presented in the previous section into the provided baseline model.

3.1 Provided Baseline

The provided baseline model is composed of three main layers. The *RNN Encoder Layer* encodes the context passages and questions into hidden states by feeding pre-trained GloVe word embeddings into a 1-layer shared bidirectional GRU RNN. The context hidden states attend to the question hidden states in the *attention layer* by applying basic dot-product attention. The *output layer* feeds the blended representations, composed of context hidden states concatenated with attention outputs, into a fully connected layer and ReLU non-linearity. A score is then assigned to each context location through downprojection, after which individual softmax layers are applied to compute probability distributions for the start and end positions of the answer span. The loss function is the sum of the cross-entropy loss for the start and end points. To predict answers, the model simply takes the argmax over these start and end positions separately to obtain a span (start, end) given a passage context and corresponding question. More detailed equations can be found in the default project handout. [10]

3.2 Our Model's Architecture

Our model augments the baseline by adding and editing both the layers and their corresponding inputs. Figure 1 is a visual representation of this model. We also regularize our non-bias parameters using L2 loss to avoid over-fitting.

3.3 Bidirectional attention flow

This layer substitutes the basic dot product attention included in the baseline. It allows attention to flow in both directions: first, from context to query (C2Q), then from query to context (Q2C). Let the context and question hidden states be $\boldsymbol{c}_1, ..., \boldsymbol{c}_N \in \mathbb{R}^{2h}$ and $\boldsymbol{q}_1, ..., \boldsymbol{q}_N \in \mathbb{R}^{2h}$, respectively. At the center of this of bidirectional attention is a similarity matrix, defined as $S_{ij} = f(\boldsymbol{c}_i, \boldsymbol{q}_j)$, where f is a trainable scalar function of context and query representations representing a similarity score between them. We used the same function as the original paper that introduce this method [11]:

$$S_{ij} = f(\boldsymbol{c}_i, \boldsymbol{q}_j) = \boldsymbol{w}_{sim}^T [\boldsymbol{c}_i; \boldsymbol{q}_j; \boldsymbol{c}_i \circ \boldsymbol{q}_j]$$

Then, C2Q attention outputs $\boldsymbol{a}_i$ are calculated using the row-wise softmax of S:

$$\boldsymbol{\alpha}^i = \text{softmax}(S_{i,:}) \in \mathbb{R}^M \quad \forall i \in \{1, ..., N\}$$

$$\boldsymbol{a}_i = \sum_{j=1}^{M} \boldsymbol{\alpha}_j^i \boldsymbol{q}_j \in \mathbb{R}^{2h} \quad \forall i \in \{1, ..., N\}$$

Q2C attention is similar except we first reduce the similarity into a vector taking its row-wise max:

$$\boldsymbol{m}_i = \max_j S_{ij} \in \mathbb{R} \quad \forall i \in \{1, ..., N\}$$

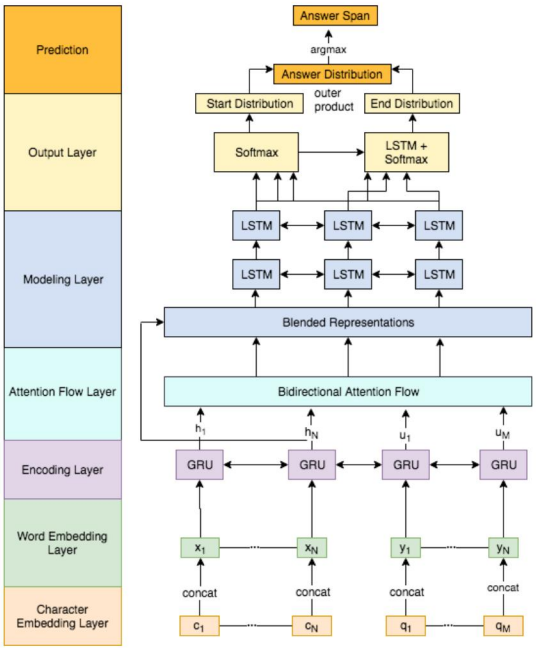

Figure 1: Our model is composed of six main layers: a character embedding layer, a word embedding layer, an encoding layer, an attention flow layer, a modeling layer, and an output layer.

$$\boldsymbol{\beta} = \text{softmax}(\boldsymbol{m}) \in \mathbb{R}^N$$

$$\boldsymbol{c}' = \sum_{j=1}^{M} \boldsymbol{\beta}_i \boldsymbol{c}_i \in \mathbb{R}^{2h}$$

The two attention outputs are combined and then concated with the context hiddens to form a blended representations $\boldsymbol{b}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{c}'] \in \mathbb{R}^{8h}, \forall i \in \{1, ..., N\}$.

3.4 Modeling Layer

We then feed these blended representations into a two-layer bidirectional LSTM. This layer is similar to the GRU enconding used on the context embeddings in that it captures the interactions between context words. However, it is different in that it uses a different kind of RNN unit and that the inputs at this point in the system are query-aware representations of the context word. The modeling output matrix $M \in \mathbb{R}^{2h \times N}$ (the LSTM outputs are of size h and there is one for each direction) has column vectors that encode context words aware of the entire context paragraph and the query.

3.5 Output Layer

The input to the output layer is the blended representations concatenated with the modeling output. Imagine each $\boldsymbol{b}_i$ is a column vector in the matrix $B \in \mathbb{R}^{8h \times N}$. Then, the probabilities of the start and end positions are calculated similarly to the baseline method except with different inputs.

$$\boldsymbol{p}^{\text{start}} = \text{softmax}(\boldsymbol{w}_{\text{start}}^T [B; M]) \quad \text{and} \quad \boldsymbol{p}^{\text{end}} = \text{softmax}(\boldsymbol{w}_{\text{end}}^T [B; M^2])$$

The weights $\boldsymbol{w}_{\text{start}}^T, \boldsymbol{w}_{\text{end}}^T \in \mathbb{R}^{10h}$ are trainable vectors and the softmax function here corresponds to the Simple Softmax Layer provided in the baseline which consists of a fully connected (linear transformation) layer and a masked softmax. We create M^2 by feeding M into yet another layer of bidirectional LSTMs. During training, we're optimizing the cross-entropy loss for the start and end locations. At test time our system outputs an answer span by finding the indices (i, j) that maximizes the product of $\boldsymbol{p}_i^{\text{start}}$ and $\boldsymbol{p}_j^{\text{end}}$.

3.6 Character-level CNN

Our model augments this baseline embedding layer by incorporating a character-level convolutional neural network (CNN). Thus, the model is better able to condition on morphology as well as handle out-of-vocabulary words. In our CNN, filters slide over full rows of our word embedding matrix, allowing us to learn the compositions of words.

Our character vocabulary is composed of the ASCII vocabulary, along with a PAD_ID and an UNK_ID to handle padding and unknown characters. Unicode characters are converted into ASCII characters. The context character embeddings and question character embeddings ($\mathbf{e}_1, ..., \mathbf{e}_L \in \mathbb{R}^{d_c}$) are fed into a three-layer CNN with a ReLU activation function and default-initialized bias vector in order to compute a sequence of hidden representations. Each of these hidden representations is computed based on a window of characters centered at some position i. Our resulting final character embedding for both question and contexts come from applying elementwise max-pooling to these hidden variables, such that $\text{emb}_{\text{char}}(w) = \max_i h_i \in \mathbb{R}^f$. We concatenate these embeddings to our word embeddings to serve as hybrid representations for each word. This improvement was inspired by the BiDAF paper [11].

4 Experiments

4.1 Evaluation Metrics

SQuAD provides three crowdsourced answers for each question. Due to the nature of this dataset, we evaluate the performance of our model based on two metrics:

- **Exact Match** (EM): binary measure that represents whether the answer given by the system matches the ground truth exactly.
- **F1**: harmonic mean of precision and recall.

We take the maximum of these two metrics between the three provided ground truth answers for a given question. The final F1 and EM scores are the average of all the EM and F1 scores for questions across the dataset.

4.2 Hyperparameters

4.2.1 Preventing Memory Exhaustion

The addition of our improvements to the baseline meant that there would be a significant increase in the number of parameters. Thus, we had to decrease the values of a number of our hyperparameters to ensure that our machines did not run out of memory during training.

We chose to reduce `context_len` from 600 to 300 after plotting a histogram of the lengths of context passages in the dev set. This is due to the fact that only 1.69% of the context passages exceed 300. In fact, 0.0058% of the context passages are of length or exceed 600 words. Thus, 300 is a much more reasonable maximum length to handle (Figure 3).

The `batch_size` was decreased through experimentation. We ran experiments with our model starting with the default batch size of 100. If the model ran out of memory during training, we decreased the batch size by 10. Often, models would run out of memory early in the training run. Through this experimentation, we found that a batch size of 60 was the highest that our machines could handle for training.

4.2.2 Preventing Overfitting

We did a few experiments to tune our regularization and dropout rates. Given more time, we would have liked to tune these more. Our experiments resulted in us using a learning rate of 0.001, regularization factor of 0.00001, and dropout rate of 0.2.

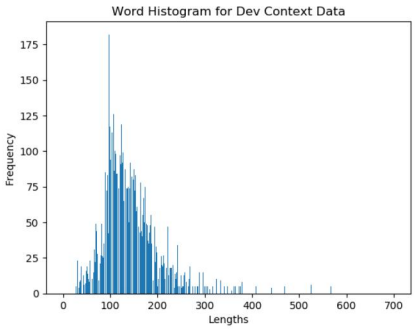

Figure 2: 98.31% of all context passages within the dev set are of length 300 or less.

4.3 Overview of Experiment Results

The following figure is a brief summary of the performance we achieved throughout the experiments we detail below. The table also provides some scores from the models from which we took inspiration (BiDAF, DRQA, etc), as well as baseline measurements. These are the F1 and EM scores for the dev set. Note that the BiDAF improvements we made include not only the attention layer but the modeling and output layers. The bolded model is our final model. We include iteration numbers when available.

Model	F1	EM	iters
Human	91.2	82.3	N/A
Regression	51.0	40.0	
BiDAF	81.1	73.3	
DRQA	79.0	70.0	
R-NET	83.7	76.7	
Our Baseline	39.7	28.8	17k
Baseline + BiDAF	73.7	62.7	15.5k
Baseline + BiDAF + Self-Attention	66.9	51.0	14.01k
Baseline + Features	54.7	40.3	9.42k
Baseline + BiDAF + CNN	**74.0**	**63.1**	38.5k
Baseline + BiDAF + Features	67.2	53.4	23.25k

We have charted the F1, EM, and Loss functions of our final model alongside those of the baseline in the following figure.

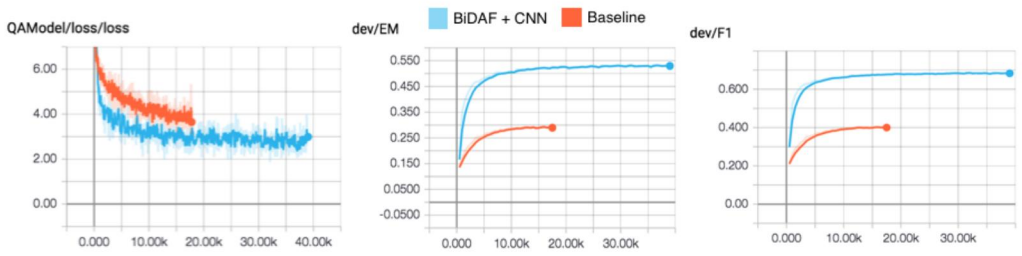

Figure 3: There is a clear improvement between our final model and the baseline, as indicated by the steeper decrease and minimum in the loss function and the steeper increase and higher plateau for the evaluation metrics.

4.4 Bidirectional Attention Flow

Substituting the Basic Attention provided in the baseline model with the BiDAF attention layer gave a modest increase in the development metrics. The training evaluation metrics increased more

significantly and that led us to conclude that the model was overfitting and include L2 loss of non-bias parameters. That improved the development numbers slightly. The modeling layer significantly increased the development dataset performance (most of the improvement) shown in Fig. 3.

4.5 Self-Attention

We were able to implement the Self-Attention layer and plug it in between the bidirectional attention flow and modeling layers, similarly to what Microsoft researchers did for their R-Net model [12]. We also made the encoding layer deeper by using three layers of GRUs instead of one. We introduced this changed after implementing the BiDAF attention, modeling and output layers. It decreased our performance. We did not have time to test this change independently and to tune its hyperparameters. We suspect the performance decrease is actually because the model change and the number of parameters increased which means that our use of the same hyperparameters became very suboptimal in the new context.

4.6 Character-Level Convolutional Neural Network

4.6.1 Baseline

We conducted a number of experiments with our character embedding layer. The first two experiments we conducted were built atop only the baseline model in order to see if a character-level CNN, alone, would make a sizable difference. We initialized our character embeddings in two ways:

- Trainable, pre-trained vectors supplied by Max Woolf [13].
- Trainable, randomly initialized vectors

We used the default settings of a learning rate of 0.001. Interestingly,both of these options performed approximately the same:

Experiment	Iterations	F1 Dev	EM Dev
Pre-trained	12k	37.6	26.4
Randomly Initialized	7k	36.9	26.1

Additionally, they showed little to no improvements to the original baseline. Due to the similarity between these models, we have decided to present the associated graphs of these experiments in our supplementary material.

4.6.2 BiDAF

Interestingly, it does appear that after implementing the bidirectional attention layer and the modeling layer, adding a character embedding actually does help. The increase in performance is only a few percentage points, but such an increase did not seem to occur without these attention and modeling changes. This is because this model learns out-of-vocabulary words better and the attention layer is able to find slightly more meaningful patterns in the relationship between the questions and contexts.

After looking at some example predictions, we made some interesting observations:

- Our model tended to predict answers that contained the correct answer but either added extraneous words (”catholics and prohibited emigration” rather than ”prohibited emigration”) or omitted words (”catholic” rather than ”mostly catholic”). This could be due to the fact that the addition of character embeddings causes the model to learn and depend more on morphology, and thus answer forms become more ambiguous: the answer is technically correct, but not specific or general enough.
- It seemed to excel when questions were quantitative in nature (”How many major ice age have occurred?” - six; ”How many examination boards exist in India?” - 30). This could be because of the brevity of the answers or the bidirectional attention flow layer has associated meaningful relationships between ”how many” questions and words composed of numbers at the character level.

- Hardly any words were identified as unknown tokens. This is likely due to the robustness of the GloVE vectors and the addition of character embeddings

4.7 Additional Input Features

For a context word t_i we augmented our word embeddings with:

- **Exact Match Per-Token Binary Features**
  - **Exact Match**: Whether t_i is found exactly in the question. Since we are guaranteed that the answer for the question is found in someway within the passage, this helps with pattern matching.
  - **Lowercase Exact Match**: Whether lowercase t_i is found in the question. Syntactic divergence between the question and answer means that we need to account for when a word appears, but not in the exact same letter cases.
  - **Lemma Exact Match**: Whether t_i's lemma form is found in the question. This also accounts for syntactic divergence.
- **Part of Speech Tagging**: Its part of speech tag. This can help the model as different question types tend to correspond to answers composed of tokens of a particular part of speech.
- **Normalized Term Frequency**: Its normalized term frequency. This is calculated by counting the number of times t_i appears in the passage and normalizing it by the maximum term frequency for any term in the passage. This helps the model learn the interaction between "rarer" and often "more significant" words, questions, and stop words.

4.7.1 Baseline

We experienced a fairly significant increase in the performance of our model. The increase in F1 and EM scores, 54.7% and 40.3%, compared to the baseline of 39.7% and 28.8%, suggests that feature engineering can make a significant impact on performance.

We believe that this makes sense due to the nature of the dataset. Since SQuAD was designed such that the answer to a question is always contained within the accompanying passage, this task of QA task is essentially a pattern matching problem. By singling out features that aid in this task, we are helping the model less"understand" the semantic meaning of the text, but instead find how syntax and patterns between the question and context could point to an answer. It is akin to memorizing how correct answers should look like versus understanding why they are correct.

4.7.2 BiDAF

Adding these features to the BiDAF model significantly increases the amount of time it takes for the model to train and converge. Additionally, performance actually decreased. It may be that the bidirectional attention flow does not work well in conjunction with these more robust representations of words. Perhaps the associations that are being made change to be unhelpful. Or, it could also be an issue with hyperparameters like regularization and learning rate. With additional time, we would have liked to explore this area further.

5 Conclusion

We were surprised with the huge positive the effect that a more sophisticated bidirectional attention layer coupled with re-encoding of the blended representations had on our model's performance. The fact that the character embeddings helped little and that self-attention and additional features hurt our performance at first puzzled us. But upon further inspection, that makes sense given the itemized analyses in the experiments section above. Given more time to improve this model, we would probably systematize our hyperparameter search to tune the model better and try to integrate the self-attention layer and some additional features (especially ones that are more lightweight) by trying out different overall model sizes and a different balance of sizes between the different layers.

6 References

[1] Christopher J.C. Burges. Towards the Machine Comprehension of Text: An Essay. Microsoft Corporation, 2013, Towards the Machine Comprehension of Text: An Essay.

[2] James Vincent. No, Machines Can't Read Better than Humans. The Verge, 17 Jan. 2018, www.theverge.com/2018/1/17/16900292/ai-reading-comprehension-machines-humans.

[3] Yi Chang, et al. A Re-Examination of IR Techniques in QA System. A Re-Examination of IR Techniques in QA System.

[4] Martin M Soubbotin, and Sergei M Soubbotin. Use of Patterns for Detection of Answer Strings: A Systemic Approach. InsightSoft-M, Use of Patterns for Detection of Answer Strings: A Systemic Approach.

[5] Deepak Ravichandran, and Eduard Hovy. Learning Surface Text Patterns for a Question Answering System. ACL '02 Proceedings of the 40th Annual Meeting on Association for Computational Linguistics.

[6] Andrea Andrenucci, and Eriks Sneiders. Automated Question Answering: Review of the Main Approaches. Information Technology and Applications, 4 July 2005.

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

[8] Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end answer chunk extraction and ranking for reading comprehension. arXiv preprint arXiv:1610.09996, 2016.

[9] Shuohang, Wang and Jing, Jiang. Machine comprehension using match-lstm and answer pointer. arXiv preprint arXiv:1608.07905, 2016.

[10] Abi See. CS 224N Default Final Project: Question Answering. http://web.stanford.edu/class/cs224n/default_project/default_project_v2.pdf

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

[12] Natural Language Computing Group. R-Net: Machine Reading Comprehension with Self-Matching Networks. Microsoft Research Asia. 2017. https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf

[13] Max Woolf. char-embeddings. MIT. https://github.com/minimaxir/char-embeddings