Final Project Report: SQuAD

Michael Chung
Department of Computer Science
Stanford University
mchung96@stanford.edu

Winston Wang
Department of Computer Science
Stanford University
wwang13@stanford.edu

Abstract

For the final project for CS 224N: Natural Language Processing with Deep Learning, we implemented a machine comprehension model that utilizes a character-level CNN, Bi-directional Attention Flow (BiDAF), and a modeling layer. We achieved a maximum F1 score of 0.6744 and an EM score of 0.5232 on the dev set.

1 Introduction

For humans, reading and understanding text, images, audio, and other inputs seems like second nature. For computers, however, this remains an open problem. Done correctly, machine comprehension has the potential to streamline a myriad of real-world problems, from answering Jeopardy questions to searching through medical articles and records. Rather than having to manually search through relevant articles to find answers to questions, an operational machine comprehension system would allow automatic detection of key pieces of text within the actual articles, saving many hours of time that could otherwise be spent elsewhere. Therefore, we attempt to improve on current NLP and deep learning models for this question and answering problem.

Our approach was primarily based off of the previous work of Seo et al., as described in the paper *Bi-Directional Attention Flow for Machine Comprehension* [2]. Beginning with the provided baseline model for the assignment, we implemented each of the layers of the "BiDirectional Attention Flow Model," and tuned some extra parameters along the way. This approach involved implementing a character-level CNN, bidirectional attention, and a modeling layer, all of which are discussed further in the Approach section.

2 Background

2.1 Problem Statement

Given a context $\boldsymbol{c}$ of length L and a query $\boldsymbol{q}$ of length J, we want to learn some function that takes $\boldsymbol{c}$ and $\boldsymbol{q}$ and outputs two predictions: a start index s and an end index e such that the answer to the query is $\{c_s, c_{s+1}, \ldots, c_{e-1}, c_e\}$. Note that in the SQuAD dataset there are actually a maximum of three possible answers for each question, since there is some variability in the way even humans would answer such questions, which means that there could be multiple pairs of s and e that are correct.

2.2 Relevant Work

Although the SQuAD dataset has only been around for a few years, the problem of question answering has been a prominent area of research for decades. Recently, with the spread of deep neural techniques, performance on question answering has skyrocketed, especially for more general datasets (contains a wide range of source material), like SQuAD.

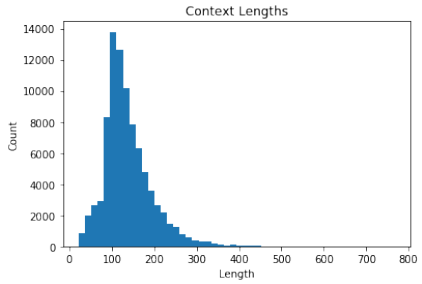

Figure 1: Histogram of context lengths in the training set. Notice that the vast majority (99.789%) fall below 400 words

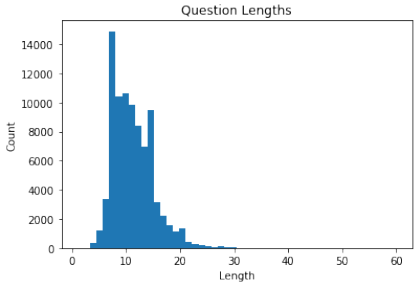

Figure 2: Histogram of question lengths in the training set. Notice that the vast majority (99.933%) fall below 30 words

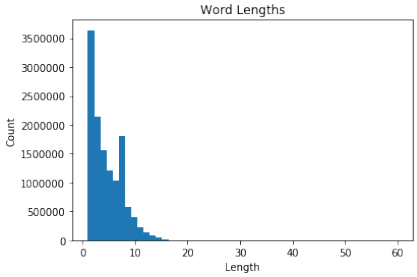

Figure 3: Histogram of word lengths in the contexts in the training set. Notice the vast majority (99.996%) fall below 25 words

Performance on SQuAD can be easily tracked through the public leaderboard, which shows the top models' performances on the secret test data set. Some of the top performing models, from research centers like Google Brain and Microsoft, have been able to achieve F1 and EM scores similar to human performance through methods such as ensembling and complex attention.

3 Approach

3.1 Analysis of Training Data

For this project, we utilize the Stanford Question Answering Dataset (SQuAD) [3], which contains over a hundred thousand question-answer pairs. There tend to be multiple questions per article, and each question has a maximum of three possible exact match answers. The distributions of context lengths (Figure 1), question lengths (Figure 2), and context word lengths (Figure 3) can be seen above.

3.2 Architecture

Our model consists of the following layers (each discussed in greater detail below):

1. **Character Embedding Layer:** Uses a character-level CNN to convert words to vectors.
2. **Word Embedding Layer:** Uses pre-trained word embeddings to convert words to vectors.
3. **Contextual Embedding Layer:** Uses a bi-directional LSTM to capture context between words.
4. **Bi-directional Attention Flow Layer:** Produces query-aware features for context words.
5. **Modeling Layer:** Uses a pair of bi-directional GRUs to capture interactions between context and query.
6. **Output Layer:** Produces the answer to the query.

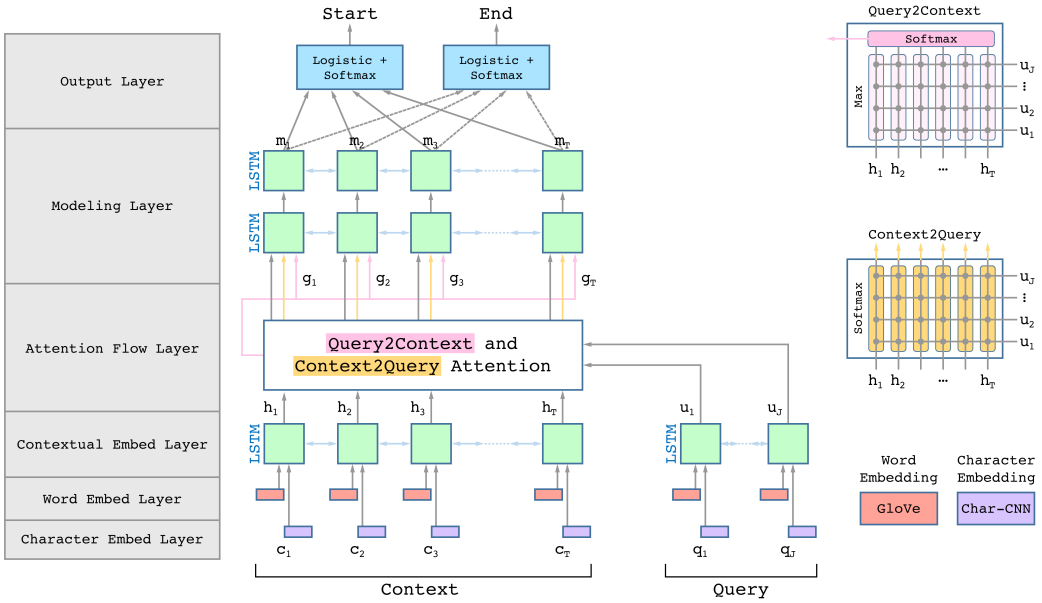

Figure 4: Architecture of the Model

3.2.1 Character Embedding Layer

The purpose of the character-level CNN is to find a meaningful way to represent words not found in the vocabulary. Rather than representing the word as <UNK>, the character-level CNN is able represent words based on their morphology, meaning out of vocabulary words can be represented much more accurately. The character embeddings are trained specifically for this task.
We use Xavier initialization for the initial character embeddings. The character embedding size $d_c = 20$. Each word w is converted into a list of character embeddings $\boldsymbol{e}_0, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_L \in \mathbb{R}^{d_c}$, which were then convolved with a kernel size $k = 5$ in order to obtain a list of hidden representations $\boldsymbol{h}_0, \boldsymbol{h}_1, \ldots, \boldsymbol{h}_L \in \mathbb{R}^f$, where $f = 100$. These hidden representations are then max pooled to obtain a single vector $\text{emb}_{\text{char}}(w)$.

3.2.2 Word Embedding Layer

In this layer, we utilize pretrained GloVe word vectors [5] to represent each word w. Without any modifications, or further training, we convert the word into its word embedding $\text{emb}_{\text{word}}(w)$. This word embedding is then concatenated with the character based word embedding to get the final word embedding, $\text{emb}(w)$

3.2.3 Contextual Embedding Layer

This layer captures the interactions between context words via a bi-directional LSTM, which encodes the interactions into vector representations. These representations model interactions such as grammar, which are entirely independent of the query.

3.2.4 Bi-directional Attention Flow Layer

The purpose of the attention flow layer is to draw connections between the context and query words, linking them together into a combined representation where each word is "aware" of the others.

The inputs to the attention flow layer are $\boldsymbol{C}$ and $\boldsymbol{Q}$, the context and query vector representations from the previous layers. The outputs of the layer are the embeddings from the previous layer, and the query-aware context embeddings.

To calculate the context-to-query and query-to-context attentions, we first compute the similarity matrix $\boldsymbol{S} \in \mathbb{R}^{T \times J}$, where $\boldsymbol{S}_{tj}$ is the similarity between the tth context word and the jth query word. This matrix is given by

$$\boldsymbol{S}_{tj} = \boldsymbol{w}_{sim}^T [\boldsymbol{c}_i; \boldsymbol{q}_j; \boldsymbol{c}_i \circ \boldsymbol{q}_j] \in \mathbb{R},$$

where $\boldsymbol{c}_i \circ \boldsymbol{q}_j$ is the elementwise product, and $\boldsymbol{w}_{sim}$ is a trainable weight vector.

Context-to-query attention is the measure of how relevant each query word is to a given context word. We first calculate the attention distributions α^i by taking the softmax of $\boldsymbol{S}$ (row-wise). Then, we use the attention distributions to find the weighted sums of the query hidden states $\boldsymbol{q}_j$, which we will call $\boldsymbol{a}_i$. In other words:

$$\alpha^i = \text{softmax}(\boldsymbol{S}_i) \in \mathbb{R}^J, 1 \leq i \leq T$$

$$\boldsymbol{a}_i = \sum_{j=1}^{M} \alpha_j^i \boldsymbol{q}_j \in \mathbb{R}^{2h}, 1 \leq i \leq T.$$

Query-to-context attention is the measure of which context words are most similar to a given query word, and thus should be given higher weight for answering a question. We take the max of each row of the similarity matrix and call it $\boldsymbol{m} \in \mathbb{R}^T$. Then, take the softmax of $\boldsymbol{m}$ and call it β. Finally, use β to find the weighted sum of the context hidden states $\boldsymbol{C}$, and call it $\boldsymbol{c}'$. In other words:

$$\boldsymbol{m}_i = \max_j \boldsymbol{S}_{ij} \in \mathbb{R}, 1 \leq i \leq T$$

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

$$\boldsymbol{c}' = \sum_{i=1}^{N} \beta_i \boldsymbol{c}_i \in \mathbb{R}^{2h}.$$

Finally, we combine together $\boldsymbol{a}$ and $\boldsymbol{c}'$ with $\boldsymbol{c}$ to form $\boldsymbol{G}$, as follows:

$$\boldsymbol{g}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{c}'].$$

3.2.5 Modeling Layer

The modeling layer takes in $\boldsymbol{G}$, the query-aware vector representation of the context words, and outputs a representation that encodes higher-level interactions between the context and query words. The key difference between this layer and the contextual embedding layer is that this one captures representations of how context and query words interact with each other, instead of merely how context words interact independent of a query.

For this layer, we simply pass $\boldsymbol{G}$ through back-to-back bi-directional GRUs.

3.2.6 Output Layer

This layer remains largely the same in our model and the baseline. The blended representations are fed through a fully connected layer with a ReLU nonlinearity. A simple matrix multiplication with an added bias gives the logits vector, which is passed through softmax to create a probability distribution for the start position. This same process is repeated, with different weights and biases, in order to get the probability distribution for the end position. When the end position is being conditioned on the start position, before the blended representation is fed to the end position output layer, the blended representation is concatenated with the probability distribution for the start position.

4 Experiments

We ran a total of 18 experiments, to varying degrees of success. Some were cut short after a few thousand iterations due to obvious flaws, while others were allowed to play out until they either flatlined or began to overfit. The following subsections will discuss the results of these experiments.

Note that with the terms "successful" and "failed," we refer not to the inherent value of the test itself, but rather to indicate whether or not the features improved upon the previous model. Every single test, no matter how poor the results, provided valuable insights that we used to continue improving our model.

4.1 Successful Experiments

Of the 18 total experiments, 8 of them (including the baseline) were deemed "successful," meaning they surpassed the scores of the previous best model. Table 1 summarizes these successful experiments, reporting their EM and F1 scores on the development data set.

Table 1: Successful Experiments

NAME	Dev EM / F1	NOTES
baseline	0.2934 / 0.4050	default
lstm	0.3077 / 0.4182	LSTM in contextual embed layer
bidaf	0.3412 / 0.4598	BiDAF
modelinglayer	0.5025 / 0.6509	two LSTMs for modeling layer
l2_endonstart	0.5074 / 0.6580	l2=0.0001, condition end pred on start pred
cnn_grumodeling	**0.5232 / 0.6744**	char-level CNN, GRUs for modeling layer

Note: there were 8 successful experiments, but two of them are not worth including in this list. One was a precursor to the "bidaf" experiment with only slightly lower scores. The other was the first run of the character-level CNN, which did not change the modeling layer to use GRUs; we omitted it for the closeness of its scores to those of the "cnn_grumodeling" experiment.

We will now discuss in detail each of these successful trials. These improvements are cumulative, so any given trial kept the features from the previous, and added its own. Note that this section is primarily for the purpose of assessing performance gains; see the Architecture section for technical implementation details.

4.1.1 LSTM

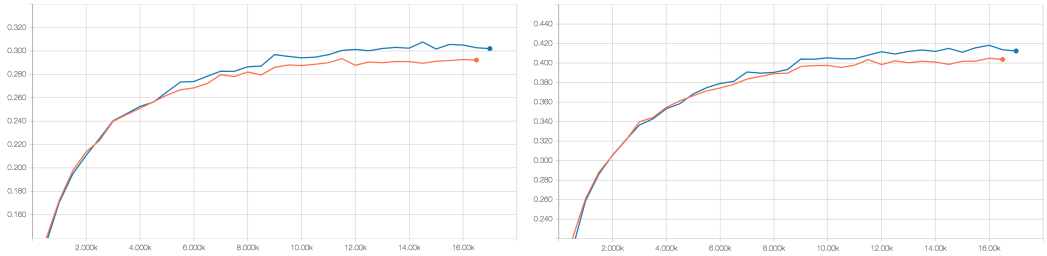

Figure 5: Improvement of LSTM (blue) over Baseline (orange). EM on the left, F1 on the right.

Our very first improvement was to change `RNNEncoder` in `modules.py` to use an LSTM instead of a GRU. This yielded a slight performance gain (0.0143 EM, 0.0132 F1).

4.1.2 BiDAF

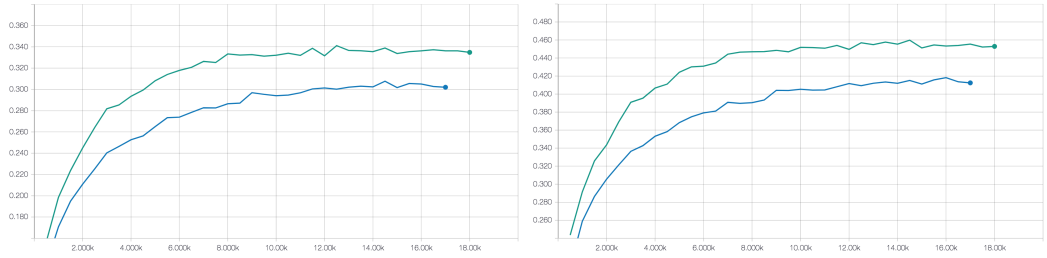

Figure 6: Improvement of BiDAF (green) over LSTM (blue). EM on the left, F1 on the right.

Next, we replaced the "Simple Attention" layer with a Bi-directional Attention Flow layer, as specified by the project handout and the paper by Seo et al. [2]. This improved our model much more significantly than LSTM alone, with an EM gain of 0.0335 and an F1 gain of 0.0416.

4.1.3 Modeling Layer

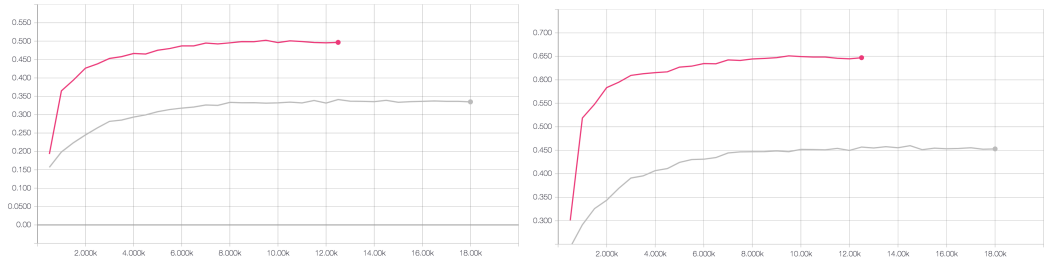

Figure 7: Improvement of BiDAF + modeling layer (pink) over BiDAF (grey). EM on the left, F1 on the right.

The modeling layer was also based off of the paper by Seo et al. [2], and provided the bulk of our improvement from the baseline. Its addition yielded gains of 0.1636 for EM and 0.1911 for F1.

Throughout our experiments, we tested multiple sizes for the modeling layer's hidden states. We started at 200 (same as the contextual embedding layer), but found that lowering as low as 20 had little effect on scores, and allowed us to add in other features (such as char-level CNN) without overflowing memory.

4.1.4 L2 Regularization

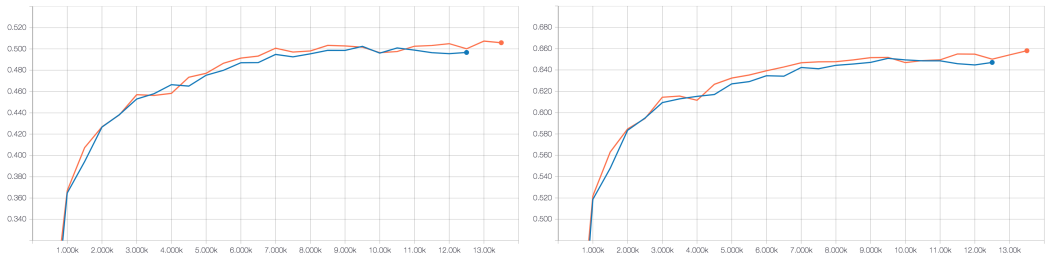

Figure 8: Improvement of L2 regularization (orange) over BiDAF + modeling layer (blue). EM on the left, F1 on the right.

We tested three different L2 regularization values: 0.001, 0.0005, and 0.0001. We found that the last one performed the best, with modest gains of 0.0049 for EM and 0.0071 for F1.

This updated model also included a simplified version of conditioning the end prediction on the start prediction. For this improvement, we simply appended the start prediction to the blended representation before predicting the end.

4.1.5 Character-level CNN + Modeling Layer GRUs

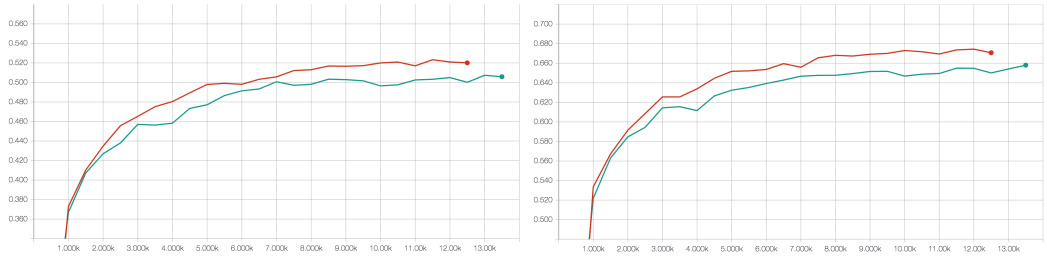

Figure 9: Improvement of Char-CNN + GRUs in modeling layer (red) over L2 Regularization (green). EM on the left, F1 on the right.

For this improvement, we implemented a character-level CNN that converted each word into a vector representation of the combination of its characters. This vector is then appended to the end of the GloVe word vector before being passed into the BiDAF layer.

In addition, we updated the modeling layer to use GRUs rather than LSTMs, which improved the performance slightly from the initial CNN test.

This update was a small improvement, with gains of 0.0158 for EM, and 0.0164 for F1.

4.1.6 Summary

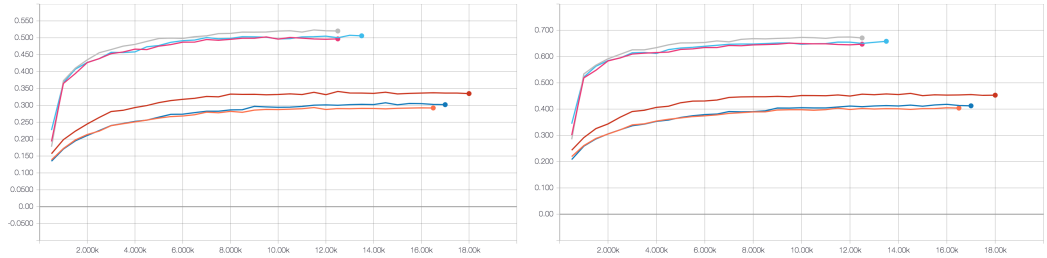

Figure 10: All successful experiments: baseline (orange), lstm (dark blue), bidaf (red), modelinglayer (pink), l2_endonstart (light blue), and cnn_grumodeling (grey). EM on the left, F1 on the right.

4.2 Additional Experiments

In addition to the successful experiments described above, we tested various hyperparameters. Here are our results for our best set:

- `dropout`: tested 0.15, 0.2, 0.3, and 0.5, settled on 0.15.
- `l2`: tested 0.0001, 0.0005, 0.001; settled on 0.0001.
- `batch_size`: tested 100, 80; settled on 80 (for memory purposes).
- `embed_size`: tested 100, 200; settled on 100.
- `context_len`: tested 300, 400, 600; settled on 400.

4.3 Qualitative Analysis

Looking at some of the predictions on the development set, it is clear that our model performs better on some types of questions over others. For example, questions that are more surface level, such as "who", "what", "when", and "where" questions, tend to have much higher F1 and EM scores than deeper questions, which are based on "why" and "how". This may also be partially related to the model's performance on longer answers. The longer the ground truth answer is, the less likely the model is to predict it correctly.

One possible source of error is the presence of very similar words with similar usage within the context. For example, for the question "`which theory states that slow geological processes are still occurring today , and have occurred throughout earth 's history ?`", the correct answer is `uniformitarianism` while the model predicts `catastrophism`. Both words describe a theory relating to geological processes, but are actually complete opposites.

Another possible source of error is the presence of multiple valid answers, while only one answer is accepted as ground truth. For example, for the question "`relegation to secondary status for abc resulted in viewership how much lower than their competitors , according to goldenson ?`", the correct answer is `five times lower viewership` while the model predicts `five times`. Even though both answers make sense, the model is still penalized.

Finally, there were a few instances in which there was no prediction at all, since the end position was predicted to be before the start position.

5 Conclusion

In conclusion, we were able to significantly improve on the baseline model, achieving a maximum F1 score of 0.6744 and an EM score of 0.5232 on the dev set. We noticed that although bidirectional attention with the modeling layer provided the majority of the improvements, the attention and modeling separately had minimal effects. We also saw that the character level CNN tended to improve models by about 2% in the F1 score.

In the future, there are many further improvements we can try. For example, it is possible to condition the end prediction on the start prediction, perhaps by setting all end position probabilities before the start position to 0. This would eliminate silly predictions by the model where the end position is before the start position. It might also be useful to take multiple models and ensemble them using a voting scheme. Moreover, we can further pursue adding additional input to the word embeddings, such as part of speech tagging, named entity recognition, exact match, and TF-IDF [1].

6 References

[1] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

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

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

[4] E. Loper and S. Bird, Nltk: The natural language toolkit, in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ser. ETMTNLP 02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 6370. [Online]. Available: http://dx.doi.org/10.3115/1118108.1118117

[5] J. Pennington, R. Socher, and C. D. Manning, Glove: Global vectors for word representation, in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 15321543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[6] Y. Kim, Convolutional neural networks for sentence classification, *arXiv preprint arXiv:1408.5882*, 2014