Effect of Character- and Subword-Embeddings on BiDAF Performance

Stanford CS224N Default Project

Andy Kim
Department of Electrical Engineering
Stanford University
andydhk@stanford.edu

Abstract

Systems trained end-to-end have achieved promising results in question answering the past couple of years. Many of the deep-learning based question answering systems are trained and evaluated on the Stanford Question Answering Dataset (SQuAD), where the answer to every question is either unanswerable or a segment of text from the corresponding reading passage [4]. In this work, we investigate the effectiveness of different embeddings in improving the performance of the baseline Bi-Directional Attention Flow model on solving SQuAD 2.0. The first model improves upon the baseline with character-level embeddings; the second model improves with subword-level embeddings; the third improves with both character-level and subword-level embeddings. Our best model, which incorporates word-level and subword-level embeddings, achieves an EM score of 57.70 and F1 score of 61.26 on the test set.

1 Introduction

The past couple of years, systems trained end-to-end have achieved promising results in both machine comprehension (MC) and question answering (QA). A key factor to this advancement has been the discovery and use of neural attention mechanism, which allows the system to focus on a specific area within the context paragraph that is most relevant to answering the question. However, attention mechanisms are typically limited with the following characteristics. First, the attention weights are used to extract and summarize the context into a fixed-size vector. Second, the attention weights at the current time step are a function of attended vector at the previous time step. Third, they are uni-directional, where the query attends on the context paragraph.

The Bi-Directional Attention Flow (BiDAF) network, "a hierarchical multi-stage architecture for modeling representations of context paragraph at different levels of granularity", was introduced by Seo et al. as a way to tackle each limitation listed above [5]. First, the attention layer is computed for every time step, and the attended vector is allow to flow through the subsequent modeling layer. This reduces the information loss caused by early summarization. Second, the attention mechanism is memory-less and is a function of only the query and context paragraph at the current time step. Third, the attention mechanism is used in both directions, query-to-context and context-to-query, which can provide complimentary information.

In this work, we seek to improve upon the provided baseline BiDAF network with character and subword embeddings. This leads to a unique model that includes 7 distinct layers. The character, subword, and word embedding layer maps each word to a vector space using character-level Convolutional Neural Networks (CNNs), subword-level CNNs, and word embedding models. The Contextual embedding layer then uses contextual cues from surrounded words to refine the embedding of the words. The attention flow layers and modeling layers are then used to fuse the information contained in the Query and Context. Finally, the output layer is fed in this Query-aware context representation to transform the context into probability values to determine where the answer starts and ends.

Analysis was carried out with quantitative measurements of EM and F1 scores, as well as qualitative measurements of answer predictions and corresponding question/context paragraphs. Our results indicate that subword-level embeddings alone are more effective than a mix of character- and subword-level embeddings.

2 Related Work

2.1 Dynamic Attention Mechanism

Dynamic attention mechanism refers to the process where attention weights are updated dynamically given the query, context, and previous attention [1]. It has been argued that the dynamic nature helps attend on context words on CNN and DailyMail datasets [2]. Other work has shown that reversing the direction of attention such that the model attends on query words as the RNN moves through the context paragraph, can help improve accuracy on SQuAD [7]. On the other hand, BiDAF uses a memory-less attention mechanism that encourages a division of labor between the attention layer and modeling layer.

2.2 Single Attention Weight Calculations

Some work uses the attention layer such that the weights are computed once and fed into the output player for final prediction [8]. For instance, the attention-over-attention model incorporates a similarity matrix between query and context words to compute a weighted average of query-to-context attention [9]. BiDAF instead lets attention flow through the subsequent modeling layer, reducing the potetntial information loss caused by early summarization.

2.3 Memory Network

Other work repeats the computation of attention vectors between the query and context through multiple layers[10]. These types of networks are considered variants of the Memory Network [3], and some systems try to dynamically control the number of hops with Reinforcement Learning [11]. The BiDAF model can also incorporate multiple hops if desired, displaying its flexibility.

3 Approach

3.1 Baseline

The baseline is provided as the BiDAF architecture without character-level embeddings.

3.2 Model Overview

The most complicated model extends the provided baseline to include character-level and subword-level embeddings. In total, there are seven total layers, each with a brief description below:

- Character Embedding Layer: maps each individual word to a higher dimensional vector space using 1d Convolutional Neural Networks (CNNs) with character-level embeddings.
- Subword Embedding Layer: maps each indiivudal word to a higher dimensional vector space using 1d Convolutional Neural Networks (CNNs) with byte pair embeddings.
- Word Embedding Layer: also maps each individual word to a higher dimensional vector, but uses pre-trained word vectors from GloVe [6].
- Contextual Embedding Layer: uses bidirectional LSTMs on top of the embeddings from the previous 2 layers to model temporal interaction between words.
- Attention Flow Layer: links and fuses information from the query and context vectors to produce a set of query aware feature vectors for each word in the context.
- Modeling Layer: employs a Recurrent Neural Network to capture the interaction among the context words conditioned on the query.
- Output Layer: provides final answer by creating a probability distribution of starting index and end index.

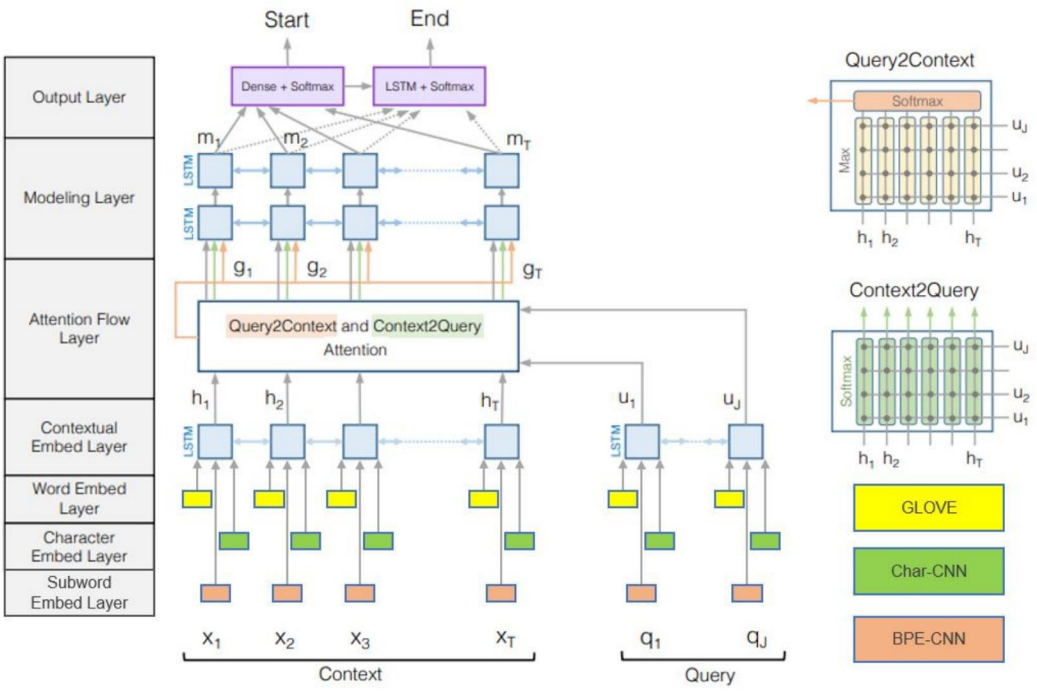

Figure 1: BiDAF with Character-Level Embedding and Byte-Pair Subword-Level Embedding. Adapted from [5]

3.3 Character Embedding Layer

The baseline BiDAF model only included the word embedding layer. This meant a simple lookup method was used to obtain a vector representation of a word; in particular, each word was mapped to a GLoVe [12] vector of length 300. Word-level embeddings are widely used in NLP tasks and tend to be computationally cheaper compared to lower-level embeddings. However, a key limitation with word-level embeddings is that they are unable to create a vector representation for words that are not in its predefined vocabulary. That is, any words that have not been seen before are given the same token, creating potential issues with systems that are used on text that was not seen during word-embedding-training time.

This limitation can be alleviated by concatenating word-level embeddings with character-level embeddings. To calculate character-level embeddings, we first split every word in a given paragraph into its respective characters. Each character is then assigned an index, and the list of characters are padded appropriately with zeros such that every batch of data has the same "para_limit=400" and "char_limit=16" dimensions. We then use a lookup table to find the embeddings of length 64 that is associated with each character. The embedding of each character is then combined with a 1D-CNN, ReLU, and max-pool to create a embedding vector of uniform size for each word.

Afterwards, it is ensured that the rest of the model has twice the number of input layers as before in order to accommodate the new embedding size that includes both the word and character layer embeddings.

To perform character-level embedding, we had to perform a number of steps in the PyTorch code that may not be straightforward with the aforementioned explanation. One of the main steps was permuting the character embeddings such that it went from the shape of (batch_size * seq_len, max_word_len, input_emb_size) to (batch_size * seq_len, input_emb_size, max_word_len); this step was necessary since PyTorch only performs convolution on the last dimension.

3.4 Subword Embedding Layer

Another way to alleviate the limitation of unseen words is to use subword embeddings. Subword approaches try to solve the unknown word problem by assuming it is possible to reconstruct a word's meaning from its parts. For instance, the suffix "-an" in "Korean" may help us guess that the word refers to a nationality. There are many ways of splitting words into subwords; in fact, the character embedding layer above is a form of subword embedding in which a character sequence is transformed into a vector representation for each word. However, the character embedding layer is most granular, and one may wonder the effectiveness of subword embeddings that explicitly captures larger parts of the word.

This motivates the use of Byte-Pair Encoding (BPE) [13], an unsupervised subword segmentation model. BPE begins with a sequence of symbols and iteratively merges the most frequent pair into a new symbol. The BPE used in this work was trained on Wikipedia and had an embedding size of 100.

The process of calculating subword-level embeddings with BPE was similar to that of character level embeddings. We first split every word in a given paragraph into its respective subwords with BPE. Each subword is then assigned an index, and the list of subwords are padded appropriately with zeros such that every batch of data has the same "para_limit=400" and "bp_limit = 10" dimensions. We then use a lookup table to find the embeddings of length 100 that is associated with each subword. The embedding of each character is then combined with a 1D-CNN, ReLU, and max-pool to create a embedding vector of uniform size for each word.

As before, it is ensured that the rest of the model has three times the number of input layers as the baseline in order to accommodate the new embedding size that includes the word, character, and subword layer embeddings. Moreover, the subword embeddings had to be permuted such that the last dimension was of size "max_bp_len".

3.5 Hyperparameter Tuning

The primary hyperparamter that was tuned was the learning rate. Every model was tested on the learning rate of 0.5 and 0.7 to see whether the performance trends were consistent across learning rates.

4 Experiments

4.1 Data

The dataset used is divided in train, dev, and test splits. The train split is taken directly from the official SQuAD 2.0 training set with a total of 129,941 examples. The dev and test splits are roughly half of the official SQuAD 2.0 dev sets, with a total of 6078 and 5915 examples, respectively.

4.2 Evaluation Method

Performance is measured the same way as in the official SQuAD 2.0 competition via two metrics: Exact Match (EM) score and F1 score. We also include the Dev NLL and AvNA scores.

- The Exact Match (EM) score is a binary measure of whether the system output matches the ground truth answer exactly.
- F1 is the harmonic mean of precision and recall: $F1 = \frac{2}{precision^{-1}+recall^{-1}}$

4.3 Experimental Details

Seven models are involved in the project.

- Basline: BiDAF architecture with only word-level embeddings.
- CharEmb: BiDAF architecture with word-level and character-level embeddings.
- SubEmb: BiDAF architecture with word-level and subword-level embeddings.

- CharSubEmb: BiDAF architecture with word-level, character-level and subword-level embeddings.

The following hyperparameters were used in the experiment. Learning Rate: 0.5, 0.7. L2 Weight Decay: 0. Number of Epochs: 30. Dropout Probability: 0.2. Maximum Gradient Norm: 5.0. Decay rate for exponential moving averages: 0.999.

4.4 Results

We notice that the SubEmb model with a learning rate of 0.5 provides the best performance, with a EM score of **57.70** and F1 score of **61.26** on the **IID SQuAD** test set. For each learning rate, the SubEmb model performs the best, followed by the CharSubEmb, CharEmb, and baseline model. As expected, adding different embedding layers does indeed help improve the performance of the model on SQuAD.

Model	LR	EM	F1	Dev NLL	AvNA
Baseline	0.5	54.83	58.25	3.27	65.08
CharEmb	0.5	57.42	60.67	3.28	67.52
CharEmb	0.7	57.97	61.53	3.42	68.90
SubEmb	**0.5**	**59.50**	**62.96**	**3.01**	**69.15**
SubEmb	0.7	58.38	61.69	2.94	67.90
CharSubEmb	0.5	58.53	62.09	3.48	68.96
CharSubEmb	0.7	58.12	61.32	3.17	67.79

Table 1: Performance of models and various hyperparameters on dev split

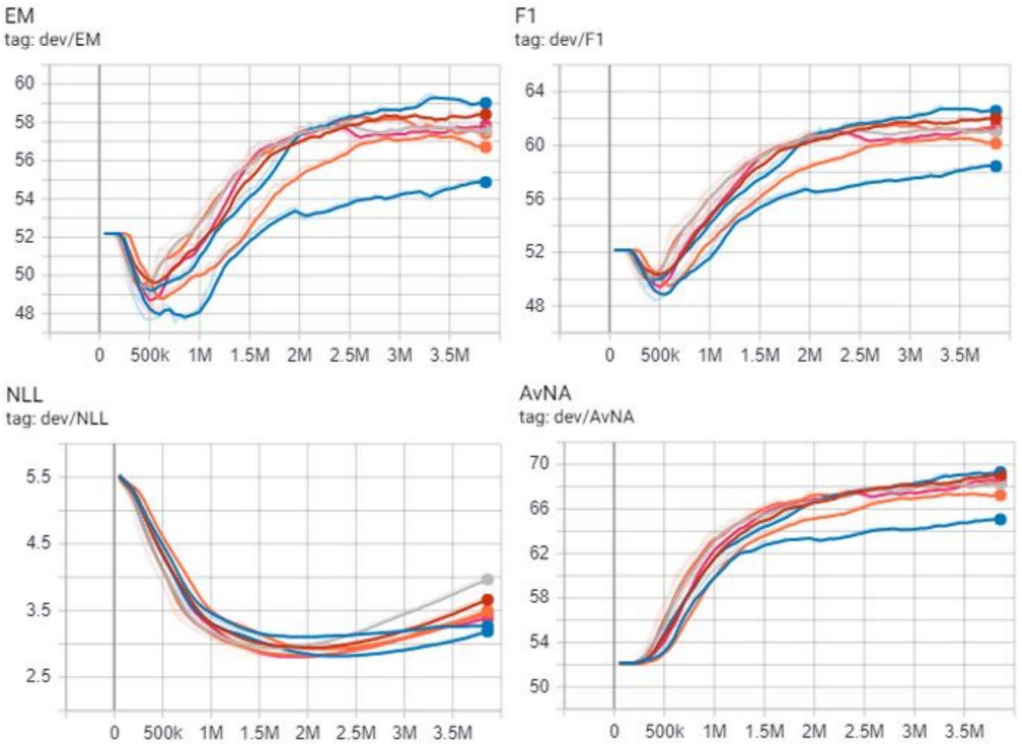

Figure 2: Dev Set Evaluation Curves. Top Blue: SubEmb-0.5, Red: CharSubEmb-0.5, Pink: CharEmb-0.7, Gray: CharSubEmb-0.7, Top Orange: SubEmb-0.7, Bottom Orange: CharEmb-0.5, Bottom Blue: Baseline

5 Analysis

5.1 Improved Performance

The SubEmb model with a learning rate of 0.5 provided the best performance, with an EM of 59.50 and F1 of 62.96. In fact, for each learning rate, we had that that the SubEmb model performs the best, followed by the CharSubEmb, CharEmb, and the baseline model. The second best performance is provided by the CharSubEmb model with a learning rate of 0.5.

As expected, adding a new form of embedding model helps boost the performance of the baseline BiDAF model since we are able to deal with words that are not present in GloVe's vocabulary; in the Baseline, we simply assign a random vector value to these words which can end up confusing the model.

The character- and subword-level embeddings, on the other hand, are able to find a vector representation for any type of word by breaking it down into smaller components. In the character-level embeddings, each word is broken down into the character level and character-specific embeddings are input into a 1D-CNN to output a vector representation of a word. Similarly, in a subword-level embeddings, each word is broken down into the subword level and Byte-Pair embeddings are input into a 1D-CNN to output a vector representation of a word.

5.2 Subword Embedding vs Character+Subword Embedding

An interesting observation is that using both the character and subword embeddings caused a worse performance compared to using only the subword embeddings. This can most likely be explained by overfitting. When we look at the NLL during training, we notice that the two models that were able to achieve the lowest NLL were the CharSubEmb models with an NLL of around 1.2. On the other hand, the SubEmb and CharEmb models were both displaying an NLL of around 1.5 and 1.8, repsecitively. Thus, we can see that while the more complicated and expressive CharSubEmb models were able to lower the NLL loss during training, it performed worse on the development set since it was overfitting on the training set.

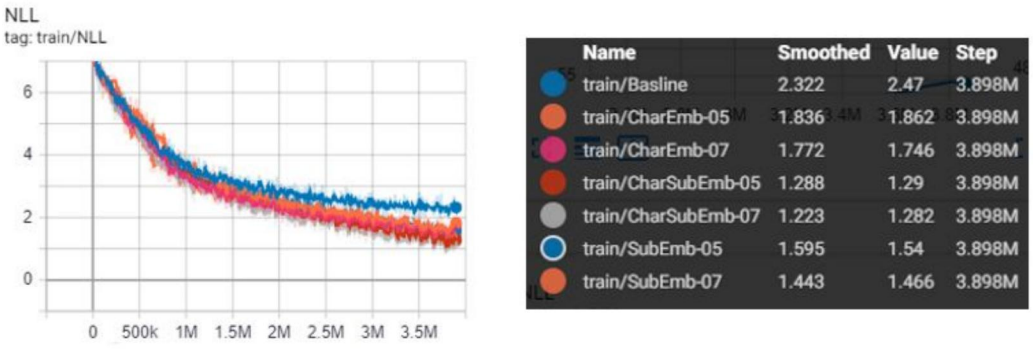

Figure 3: Training Set Evaluation Curve

5.3 Error Analysis

Below are some examples of errors from the SubEmb-0.5 model.

- **Question:** What did the Vilnius formally establish in 1573?
- **Context:** In 1529, Warsaw for the first time became the seat of the General Sejm, permanent from 1569. In 1573 the city gave its name to the Warsaw Confederation, formally establishing religious freedom in the Polish–Lithuanian Commonwealth. Due to its central location between the Commonwealth's capitals of Kraków and Vilnius, Warsaw became the capital of the Commonwealth and the Crown of the Kingdom of Poland when King Sigismund III Vasa moved his court from Kraków to Warsaw in 1596. In the following years the town expanded towards the suburbs. Several private independent districts were established, the property of aristocrats and the gentry, which were ruled by their own laws. Three times between 1655–1658 the city was under siege and three times it was taken and pillaged by the Swedish, Brandenburgian and Transylvanian forces.
- **Answer:** N/A
- **Prediction:** religious freedom

- **Question:** How many clergymen were there in the Dutch Republic before the influx of Huguenots?
- **Context:** After the revocation of the Edict of Nantes, the Dutch Republic received the largest group of Huguenot refugees, an estimated total of 75,000 to 100,000 people. Amongst them were 200 clergy. Many came from the region of the Cévennes, for instance, the village of Fraissinet-de-Lozère. This was a huge influx as the entire population of the Dutch Republic amounted to ca. 2 million at that time. Around 1700, it is estimated that nearly 25% of the Amsterdam population was Huguenot.[citation needed] In 1705, Amsterdam and the area of West Frisia were the first areas to provide full citizens rights to Huguenot immigrants, followed by the Dutch Republic in 1715. Huguenots intermarried with Dutch from the outset.
- **Answer:** N/A
- **Prediction:** 200

Figure 4: Example of Error from SubEmb-0.5 model

The first type of error are cases in which the model was not able to comprehend numbers in the question and context properly. In both cases, the model concluded that there was no answer rather then trying to connection numbers with dates or amount.

- **Question:** Why has the Muslim Brotherhood facilitated inexpensive mass marriage ceremonies?
- **Context:** Islamist movements such as the Muslim Brotherhood, "are well known for providing shelters, educational assistance, free or low cost medical clinics, housing assistance to students from out of town, student advisory groups, facilitation of inexpensive mass marriage ceremonies to avoid prohibitively costly dowry demands, legal assistance, sports facilities, and women's groups." All this compares very favourably against incompetent, inefficient, or neglectful governments whose commitment to social justice is limited to rhetoric.
- **Answer:** avoid prohibitively costly dowry demands
- **Prediction:** to avoid prohibitively costly dowry demands

- **Question:** What was the purpose of CSNET
- **Context:** The Computer Science Network (CSNET) was a computer network funded by the U.S. National Science Foundation (NSF) that began operation in 1981. Its purpose was to extend networking benefits, for computer science departments at academic and research institutions that could not be directly connected to ARPANET, due to funding or authorization limitations. It played a significant role in spreading awareness of, and access to, national networking and was a major milestone on the path to development of the global Internet.
- **Answer:** to extend networking benefits, for computer science departments at academic and research institutions that could not be directly connected to ARPANET
- **Prediction:** to extend networking benefits

Figure 5: Example of Error from SubEmb-0.5 model

The second type of error are cases in which the answer was essentially correct, but the model extended its explanation longer than the SQuAD answer. Imprecise answer boundaries should be alleviated by finding more expressive models, but as we have discussed before this may lead to overfitting and poorer overall generalization.

6 Conclusion

In this project, we experimented with different embedding layers to improve upon the BiDAF baseline. There were three different model types tested: CharEmb, which only added a character embedding layer; SubEmb, which only added a subword embedding layer; and CharSubEmb, which added both character and subword embedding layers. In terms of F1 and EM, the SubEmb model with a learning rate of 0.5 was able to achieve highest performance. It was intersting to notice that adding both the character and subword embedding layers did not lead to the highest performance; this could most likely be explained with overfitting.

Future avenues of work could address the computational and time limitations of this work by experimenting with a vaster array of learning rates and hyperparameters to establish better consistency. It would also be interesting to see if the more expressive CharSubEmb could outperform the SubEmb if it was subject to more regularization, i.e. dropout with a higher probability.

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.
[2] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *NIPS*, 2015.
[3] Alessandro Sordoni, Phillip Bachman, and Yoshua Bengio. Iterative alternating neural attention for machine reading. *arXiv preprint* arXiv:1606.02245, 2016.
[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. CoRR, abs/1606.05250, 2016.
[5] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603, 2016.
[6] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In EMNLP, 2014.
[7] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint* arXiv:1608.07905, 2016.
[8] Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text understanding with the attention sum reader network. In *ACL*, 2016.
[9] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-overattention neural networks for reading comprehension. *arXiv preprint* arXiv:1607.04423, 2016.
[10] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *ICLR*, 2015.
[11] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. *arXiv preprint* arXiv:1609.05284, 2016.
[12] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014. [13] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, 2015.