A Variation on the Theme: SQuAD

Gawan Fiore
Department of Computer Science
Stanford University
gfiore@cs.stanford.edu

Abstract

Question answering is becoming ever more relevant as an increasing portion of human search queries are performed via digital assistant, and thus require a single specific answer rather than a page of links. The SQuAD dataset provides a benchmark for computer reading comprehension. Work in this field falls into two categories: using pre-trained contextual embeddings or relying solely on default word embeddings and a combination of attention and pointer networks. In this project I forgo the use of pre-trained contextual embeddings and instead explore combinations of different kinds of attention and character-level subword models. Embedding enhancements resulted in the greatest performance increase and self-attention methods were modestly successful. A novel full attention model may merit further development as it showed success in certain areas.

1 Introduction

Question answering is an active field of research at this time and its applications are clear, especially as more human interaction with computers occurs via spoken interfaces. With a vocal agent like Siri or Alexa, or a written chatbot, returning a single, correct or relevant answer is more important than in traditional search where the user has the ability to easily evaluate the results. Even in traditional search queries, engines like Google strive to return exactly what the user is looking for in a single result. Since people increasingly expect such precise answers to their questions, computer reading comprehension, as highlighted by the SQuAD challenge, continues to grow in prominence. Given the widespread research in this area, current methods perform quite well, nearly matching the human benchmark [7]. However, there are still improvements to be made before computer comprehension reaches a level where error rates are imperceptible to human users.

The approach taken in this work emphasizes exploration of language mechanics with a focus on enhancing the information captured in embeddings and various uses of self-attention. Throughout the experiments, embeddings had the greatest positive impact on performance, notably the inclusion of subword modeling and continued training on word vectors. I also explored using a novel full attention model that attends the question and context to itself in order to make a prediction over the context. While this showed promise for future development, it was not immediately as successful as other methods.

1.1 Prior Work

State-of-the-art approaches to SQuAD all use pre-trained contextual embeddings [7]. In this work I chose not to use PCEs as I wanted to explore the mechanics of question answering more closely.

Non-PCE approaches focus on the use of subword modeling, various forms of attention, and pointers to locate the answer within a context [2,5]. As the starter code implements most of the architecture for BiDirectional Attention Flow [2], that was a clear place to begin. This paper used subword modeling in the form of a character-level convolutional network, which the default project baseline did not implement.

Similar papers include RNet and Match-LSTM which share the same foundation, but approach the problem slightly differently. RNet uses self attention by attending the context against itself in addition to the question attending to the context [5] which helps achieve an improvement from EM 68.0 and F1 77.3 in BiDaf to EM 72.3 F1 80.6 with R-Net. Match-LSTM is a more foundational project, upon which the others built. It did not use attention, but demonstrated the utility of RNNs and pointer networks to this task [4], achieving somewhat lower scores of EM 64.1 and F1 73.9. In an early experiment I removed attention layers in an attempt to reproduce this model.

For specific design decisions, Empirical Evaluation of Gated Recurrent Neural Networks and Attention is All You Need both informed my approach. GRUs have been shown to achieve comparable results to LSTMs while training slightly faster [3], which was attractive given the shorter time-frame of this project. Since attention takes various forms, as described in [1], I experimented with both additive and multiplicative attention in this project.

2 Approach

Embeddings The baseline model is implemented based on the original BiDirectional Attention Flow [2] paper, but omits sub-word modeling. Therefore, I first added the missing character-level embeddings to the baseline model to supplement the word vector-based embeddings. An improvement on subword modeling from learned character embeddings helps handle words that occur in different contexts than they do in the Word2Vec training data or word not found in the vocabulary, plus simply adding additional information for the model to use in making predictions.

For the character embeddings, as in [2], I used a simple convolutional subnet (one convolutional layer and one max pooling layer) over the character input with an embedding size of 16, kernel size of 5, and depth of 64. I experimented with an additional convolutional layer before the max pool, as well as modifications to the embedding, kernel and depth, but the simple architecture performed the best.

I also changed the usage pattern of the pre-trained word vectors by adding backpropagation into them (thus making them a learnable parameter), which produced a slight performance improvement (see experiments section below).

The character embeddings and word vector embeddings were combined via simple concatenation. I experimented with running the resulting [word_embedding_size + character_embedding_size] tensor through the Highway Network together or concatenating the character embeddings to the output of the Highway Network when only the word embeddings were given as input. Running both embeddings together through the Highway Network shows improved performance, which makes sense given the purpose of the Highway Network as part of the input encoding. In Table 2, "Character embeddings v1" had a 5x64 kernel and 16 dimensional embedding and "Character embeddings v2" has 3x50 kernel and 24 dimension embedding. The two conv + max pool character embeddings trained slowly and used so much memory that I did not pursue it beyond 100k iterations and therefore I do not present the performance of that model.

Architecture My model skeleton is based on that presented in [2], and in summary consists of:

- a Highway Network for encoding the word vectors
- an RNN encoder over all embeddings
- a bidirectional RNN attention layer based on a similarity matrix that attends question to context and context to question
- an RNN encoder modeling layer
- a pointer network for predicting the beginning and ending index of the answer within the context

See [2] for further details on the basics of this network. I added an additional self-attention layer and modified the extant modules.

I experimented with modifications to the overall architecture, specifically the addition of layers to the RNNs to capture more complex function mappings. I ran different experiments to add a layer to the Highway Network and RNN encoder, yielding a total of two LSTM layers in each, and a layer to the RNN encoder Modeling Layer, yielding a total of three LSTM layers. The motivation for this was

that perhaps matching the total number of LSTM layers in the encoding step (Highway Network + RNN encoder) and the modeling step would improve performance, but it did not. These all slowed down training considerably, as expected, and did not yield any improvement in performance in initial experiments, so it was not explored further. In fact, the highest capacity model (2 in the Highway Network, 2 in the RNN encoder, and 4 in the Modeling Layer) did not seem to learn at all over 1M iterations.

Attention The next modification was self attention integrated into the existing network. I implemented most of the necessary changes to completely replace the BiDirectional Attention network with R-Net [5], but found after some experimentation that inserting an additional self-attention bidirectional attention layer into the extant BiDirectional Attention network performed better than the architecture that was more true to R-Net.

I implemented a distinct gated self attention layer modeled after that in the R-Net like the following, where *x* is the context attending back to itself:

$$s_1 = \tanh(W1x)$$

$$s_2 = softmax(Ws_2 + b)$$

$$s_3 = \sigma(s_2)$$

$$gated = s_3 * x$$

Where * denotes elementwise multiplication.

I found more success, however, with simply repurposing the attention layer from the BiDaf paper with both inputs as the context, via a trainable parameter a, the embeddings for the context C, and the similarity matrix *S* where S_{ij} is the similarity between ith and jth words in the context:

$$S_{ij} = a(C_i C_j^T)$$

In this case, computation proceeded as follows: attention with input [context, question]; attention with with input [context, context]; elementwise addition, multiplication, or average of the two (depending on the experiment). In one experiment, I replaced traditional [context, question] with [context, context] self attention only, instead of performing both of them. The best performance resulted from using both forms of attention and essentially ensembling the pointer layers with each form to choose the start and end positions.

Novel contribution Finally, I attempted a novel "full attention" network in which the self attention operation was applied over a tensor formed by combining the question and the context, followed by an RNN encoder that reshaped the result into the correct context length for the pointer network. This showed improvement over the baseline that had additional attention (besides the simple attention in the partial BiDaf starter code) but was not as performant as the basic self attention module inspired by R-Net.

$$X = C + Q$$

$$X_{mask} = C_{mask} + Q_{mask}$$

then BiDaf self attention as described above with the similarity matrix as follows:

$$S_{ij} = a(X_i X_j^T)$$

And finally reshaping using a mapping linear layer with $W \in^{[c+q],c}$ to create a full input attention matrix of the same shape as C for the attention.

Alternately, pass the full attention result with $c + q$ dimensions to the model and output pointer networks but reject all predicted dimensions that are outside the context.

Table 1: Models Compared

Description	iteration	EM	F1	AvNA
Baseline	1M	51.83	54.99	62.98
Embeddings	1.4M	57.92	61.69	68.22
Novel Full Attention	2M	57.46	60.82	67.95
Embeddings Self Attention	850k	56.54	59.82	65.61
Published R-Net	unknown	77.3	68.4	N/A
Publisged BiDaf	unknown	77.5	68.0	N/A
Published Match-LSTM	unknown	73.7	64.7	N/A

Table 2: Ablation Studies

Experiment	iteration	EM	F1	AvNA
Character embeddings v1	1.2M	57.92	61.56	67.27
Character embeddings v2	1.2M	56.78	59.94	65.92
All recurrent plus self attention	1.1M	49.24	53.02	64.21
Self attention in BiDaf	1.4M	53.91	57.13	64.73
Increased layers	950k	56.63	60.61	62.82
Hidden size 150	900k	53.84	56.65	63.88
Hidden size 200	900k	54.81	57.95	64.34

3 Experiments

3.1 Background

Data The SQuAD dataset is a standard benchmark dataset in the field of computer reading comprehension that includes paragraphs, questions, and expected answers. I used the subset of this dataset as prescribed in the default final project instructions.

Evaluation For evaluation I used the official SQuAD metrics, F1 and EM. I additionally tracked the AvNA score, which is the classification accuracy over all questions for which the model provided an answer, and the NLL to monitor overfitting.

3.2 Details

Experiments ran for between 700k and 2M iterations, depending on whether they began to overfit and how they appeared to be doing at benchmarks where I chose to continue or restart them with various modifications. I ran a total of sixteen complete experiments to explore the architecture and embedding changes I made and several more hyperparameter tuning experiments.

I began with empirically-driven hyperparameter improvements to 1) improve the model capacity, 2) reduce overfitting, 3) fine-tune word vectors, and 4) improve model speed. To do this, I increased hidden size universally from 100 to 200, increased the dropout probability from 0.2 to 0.3, introduced backpropagation into the word vectors, and switched all LSTM cells to GRU cells.

Since the baseline showed overfitting, I increased the dropout probability and the model capacity via increased hidden size. These changes alone yielded a performance increase (see Ablation Studies below). I experimented with hidden sizes of 150 and 200 and dropout probabilities of 0.3 and 0.4. The best combination was a hidden size of 200 and a dropout probability of 0.3.

The addition of character embeddings and learned word vectors further improved performance significantly.

Results As the ablation studies summarized above make clear, each of the modifications applied alone yielded improved performance at 750k iterations. However, simply combining all of them did not result in the best model. Tuned hyperparameters plus improved embeddings performed better without the addition of self-attention than they did with self-attention.

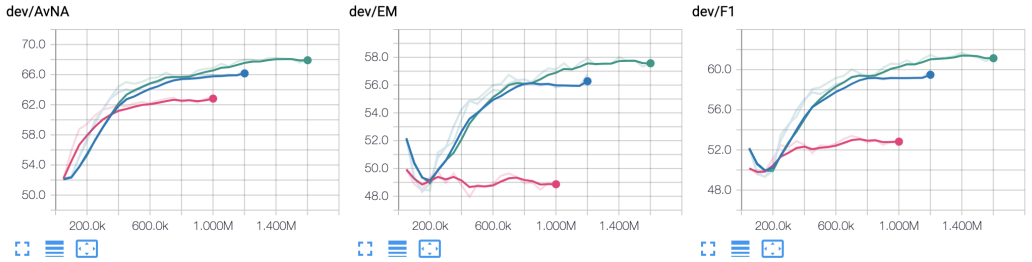

Figure 1: Effect of different character embeddings. Pink is an additional convolutional layer, green is v2 as described above, and blue is v2 as described above and used for other models.

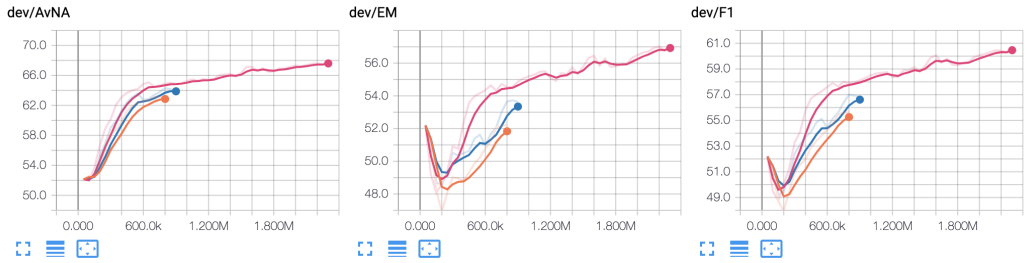

Figure 2: Effect of hyperparameter tuning. Orange is the untuned BiDaf starter code baseline, blue had backpropagation into the word vectors and dropout probabiliy of 0.3 instead of 0.2, and pink has increased hidden size plus convolutional character embeddings.

My intended novel addition of full self-attention did not improve performance. Intuitively, it seems appealing that allowing the model to attend to both the question and the context in one module would help it to find similarities within the entire space, but whatever improvement this could have yielded was likely lost in the necessary reshaping step that allows the pointer network to only select through the context for the actual answer.

Interestingly, when I removed the pointer constraint by eliminating this reshaping step and only generating an answer if the pointer network chose indexes in the context (instead of the question), performance seemed to improve. However, since I treated pointers to indexes in the question as "no answer", there were many more prompts which the network did not answer because it attempted to answer the question with itself. Given more time I would explore full attention like this further.

The best model I submitted to the non-PCE leaderboard was increased hidden size with convolutional character embeddings and baseline attention, which scored F1 61.685 and EM 58.007 on the dev set and F1 61.823 and EM 58.022 on the test set.

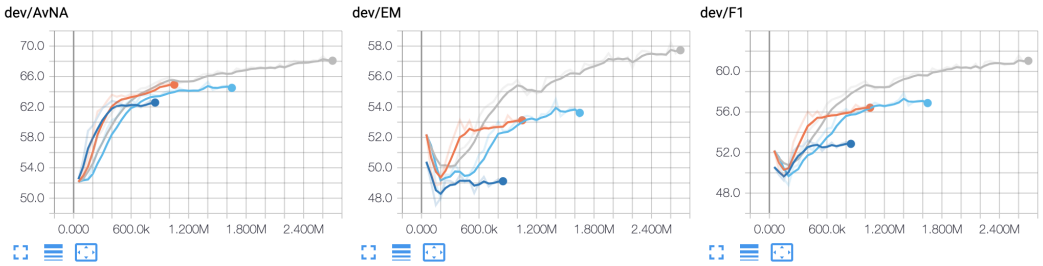

Figure 3: Effect of additional pieces of the model. Grey is the best model, with character embeddings and repurposed BiDaf self attention. Dark blue is the attempted novel full attention with character embeddings. Light blue is just self attention and orange is just character embeddings.

Table 3: OOV Examples Where Character-Level Embeddings Help

Dataset Portion	Contents	Problem
Question	Governon Robert Dinwiddie	typo, proper noun
Question	significan company	typo
Question	Henry IV	proper noun
Answer	il milione	not English

4 Discussion

I expected the additional layers to improve performance or at the very least to have no effect, but instead they degraded performance significantly on all measures. This could be due to incorrect hyperparameters for the new and different architecture or a mismatch between the problem space and the new functions modeled by the more complex geometry.

The addition of the character embeddings had the single largest impact on model performance with very minimal impact on training speed, which yielded a more significant improvement than expected. Clearly, subword and out-of-vocabulary modeling effects are significant on this task, likely because questions often have proper nouns which are not represented in the Word2Vec vocabulary. See Table 3 for a presentation of examples in the dataset that intuitive and empirically are improved by the addition of character embeddings.

Neither of my implementations of R-Net-style self-attention approached the published results, though it did yield a modest improvement. It may be that my attempts to merge BiDaf and R-Net combined the wrong portions of each or that I did not sufficiently explore the hyperparameter space. Combining multiple extant models often does not result in immediate performance gains and requires experimentation to determine which portions of each are compatible.

5 Conclusion

The largest portion of my model's improvement came from modifications to embeddings, including the addition of character-level embeddings and making word embeddings trainable. Integrating the context self-attention module from R-Net with an overall BiDaf-based architecture showed some improvement but did not yield results similar to either of the published BiDaf or R-Net models. The novel full attention layer I attempted, in which the context and question are combined and attend back to themselves, did not show marked improvement overall, but did show improvement in some contexts and therefore merits further investigation. Folding full self-attention into an ensemble network could decrease the rate of non-answers and would be a promising first step considering the additional performance benefits ensemble networks provide.

References

[1] Vaswani, et al. *Attention is All You Need.* 2017. arXiv:1706.03762 [cs.CL], https://arxiv.org/abs/1706.03762

[2] Seo, et al. *BiDirectional Attention Flow for Machine Comprehension*. 2016, rev. 2018. arXiv:1611.01603 [cs.CL], https://arxiv.org/abs/1611.01603

[3] Chung, et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv:1412.3555 [cs.NE], https://arxiv.org/abs/1412.3555

[4] Wang, Shuohang & Jiang, Jing. *Machine Comprehension Using Match-LSTM and Answer Pointer*. 2016. arXiv:1608.07905 [cs.CL], https://arxiv.org/abs/1608.07905

[5] Group, Natural Languange Computing. *R-Net: Machine Reading Comprehension with Self-Matching Networks*. 2017. https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf

[6] HKUST, Knowledge Computation Group. *Tensorflow Implementation of R-Net*. https://github.com/HKUST-KnowComp/R-Net

[7] Purkar, Raj. *SQuAD Leaderboard*. https://rajpurkar.github.io/SQuAD-explorer/