R-Net Prime

Stanford CS224N Default Project

Nik Caryotakis
Department of Computer Science
Stanford University
nikcaryo@stanford.edu

Parker Killion
Department of Computer Science
Stanford University
pkillion@stanford.edu

JP Reilly
Department of Computer Science
Stanford University
jpreilly@stanford.edu

Abstract

Building upon the Bidirectional Attention Flow model (BiDAF) and inspired by R-NET we implemented R-NET Prime. Featuring character level embeddings and a pointer network for an output layer, R-NET Prime improves the performance of the baseline model by using self attention in addition to the bidirectional attention layer in BiDAF. Our version of self attention is more easily parallelizable as opposed to R-NET's, allowing for faster training. This faster training allowed us to perform an ablation study to isolate the performance contribution of each R-NET Prime component, as well as thorough hyper-parameter tuning. In the end our best model improved upon the baseline F1 and EM scores by almost 6% in both categories.

1 Key Information to include

- TA mentor: Michihiro Yasunaga
- External collaborators: No
- External mentor: No
- Sharing project: No

2 Introduction

How do we evaluate reading comprehension? From a young age we (humans) have been evaluated on our reading comprehension by being given a passage and a question and forming an answer. How do you code a computer to understand multiple clauses, antecedent referral, or simply that the answer is not in the text provided? This reading comprehension problem for natural language processing (NLP) lead researchers to form question answer data sets in order to build a framework for training NLP models. These question answer data sets are not easy for NLP models because they have to return the section of the text where the answer is or say that no answer exists. A vital piece of search engines like Google, Bing, & Yahoo is their ability to answer questions. As more and more data has become readily accessible, the need for efficient ways of dissecting important information from this data is paramount. Making it a necessity that the model begins to understand the grammar, and conventions of English writing in order to confidently and correctly find the answer.

The SQuAD dataset was developed by Stanford researchers and houses a leader board for models to be compared [1]. In this paper we investigate R-NET, a model that topped the leader board when it was released [2]. We were interested in R-NET because of its usage of self-attention in particular.

We were interested self-attention as it is still an active field of research. Furthermore, the usage of both word and character embeddings from the BiDAF model allows for more encoding information in the model. R-NET utilizes self-attention and we wanted to investigate how effective self-attention is in their model and see if we could improve upon it in our own. Our implementation, R-NET Prime, utilizes character embeddings, self-attention, and a pointer-network output layer in this endeavor for knowledge.

After running R-NET Prime through a myriad of experiments, we found the R-NET Prime model with a hidden layer size of 75 and a dropout of 0.3, obtained the best result. This combination lead to a 5.92% increase in F1 score over the baseline and a 5.96% increase over the baseline EM score. Moreover, after conducting an ablation study we found that the character embeddings typically led to the greatest increase in performance when compared to the pointer network and self-attention, but when the dropout was 0.5 and a hidden layer size of 100 self-attention was the greatest increase.

3 Related Work

BiDAF (sans the character level embeddings) acts as our baseline model [3]. Its key component, which is also used in R-NET Prime, is its bidirectional attention flow layer, which allows the question to attend to the context, and the context to attend to the question. Its output layer produces two independent probability vectors of probabilities for the start and end tokens (i.e. the end token is not conditioned on the start token).

R-NET is designed to overcome many of the known roadblocks of past QA systems, such as limited attention and representation passing. At the time of its release in 2017, R-NET was placed at the top of the SQuAD leaderboard. R-NET itself largely expands upon a model built by Wang & Jiang [4] for the same Stanford Question Answering Dataset (SQuAD) [1]. The Wang & Jiang model utilizes match-LSTM to check if each premise (token in the passage) is entailed by the hypothesis (question), and then applies an Answer Pointer layer (Ptr-Net [2]) to predict the start and end tokens. R-NET expands upon this by adding self attention and an additional gate to the the modeling layer.

4 Approach

We started by running the baseline model given to us in the SQuAD starter code which is a Bidirectional Attention Flow model [3]. We extend upon this model in three ways.

First, we add character-based embeddings. Inspired by the original R-net paper (see figure below for reference), the character-level embeddings are found by taking final forward and backward hidden states of a bi-directional recurrent neural network (RNN), which accepts pretrained letter embeddings one at a time. These two states are concatenated together to form a character-level word embedding, that is then concatenated with the 300 dimensional GloVE word vectors given [5]. This approach should, in theory, give us better performance on out of vocabulary words. We chose to use a Gated Recurrent Unit (GRU) for our RNN since as described in the paper its comparable to LSTM, but computationally cheaper.

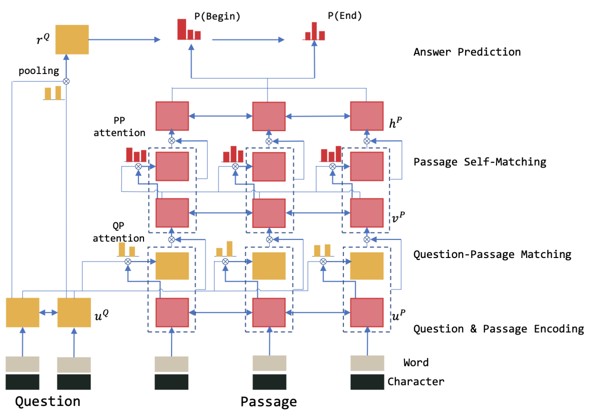

Figure 1: R-NET structure overview.

Formally, given a word w_i, either in a question Q or a passage P, we lookup its word embedding e_i and its character embeddings $\{c_j\}_{j=1}^{m}$. We feed each character embedding into the bidirectional RNN, and use the final hidden states from the forward and backward direction of the last layer, h_{cf} and h_{cb} respectively. Our final word embedding is the concatenation $[e_i, h_{cf}, h_{cb}]$. These embeddings are then fed into a GRU to complete the encoding of both the questions and context.

Second, we add a self attention layer and an additional modeling layer to BiDAF. For self attention, different from R-NET's paper, we do not use the output of one RNN cell as part of a gated input into the next. Instead, we use the existing implementation from BiDAF and modify it to return the context representations after attending to itself. The output from this layer is the concatenation of the original context representation, the attention weights, and the two multiplied together. This is then fed into a modeling layer, just as the BiDAF attention does before it. This section of our model primarily reuses code from the existing BiDAF implementations of the attention and modeling layers.

Third, we replace the BiDAF output layer with a Pointer Network (Ptr-Net) [6], continuing to follow the R-NET paper. This network uses attention as a way to select input tokens that work best as start/end indices for our question answering system. R-NET's version of Ptr-Net additionally uses attention over the question representation as the initial hidden state for the Ptr-Net RNN whose output is the answer sequence.

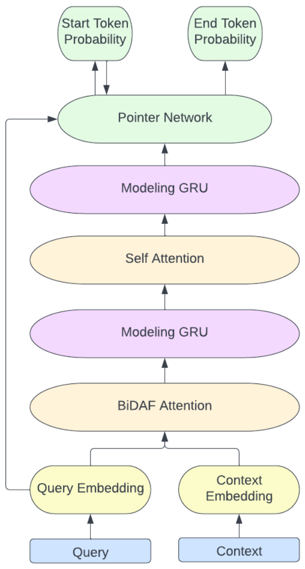

Figure 2: R-NET Prime structure overview.

Notably, our codebase maintains backward compatibility with the original starter code, meaning we can easily perform an ablation study. For the embedding layer, we have the choice of the baseline implementation (B) or character level embeddings (C). For the attention layer, we either solely use the baseline bidirectional attention (B) or self attention in addition to bidirectional attention (S). And for the output layer, we either use the baseline output layer or the R-NET style output layer (R). This lets us describe the baseline model with the notation BBB, and the full R-NET Prime model as CSR. Ablated versions tested include CBB, BSB, and BBR.

5 Experiments

5.1 Data

We used the SQuAD 2.0 training and development sets provided to us for the default project. This consists of a training set with 129,941 examples, a dev set of 6078 examples, and a hidden test set of 5915 examples. To increase development speed, we made several smaller datasets, ranging in sizes of 3 examples to 1000 examples, from a subset of the SQuAD 2.0 training and development sets provided to us, which we named MiniSQuAD datasets.

5.2 Evaluation method

For our evaluation methods, we used almost all of the Tensorboard outputs: F1, EM, and NLL. We did not regard AvNA as importantly, as all of the experiments performed similarly in this metric. As the leaderboard was judged off of F1 & EM, we used those as performance benchmarks and tracked the NLL to ensure our models were training properly and not overfitting.

5.3 Experimental details

We ran all of our experiments on either Google Cloud Platform or Azure, using a NVIDIA K80 and A100 respectively. Due to the added RNNs included in the character embedding step, models which included that layer would train for around 12-14 hours, while models without would take closer to 4-6 hours. In all of our initial research, we used a dropout rate of 0.2 and a hidden layer size of 100. Our learning rate starts at 0.5, and we would use a Adadelta optimizer and a LambdaLR scheduler to accompany this. The Adadelta optimizer allows us to not have to worry about the starting learning rate, and focus on other hyper-parameters. During our parameter search, we also ran experiments using a dropout of 0.5 and a hidden layer size of 75. We used Tensorboard to visualize and track experiments. Initially, we attempted to use RayTune to help with our parameter search, but with our limited time and VM resources opted to do a smaller manual parameter search by running multiple models in the ablation study.

5.4 Results

Our best result was from our CSR (full R-NET Prime) model with a hidden layer size of 100 and a dropout of 0.3. Obtained from the test leaderboard, we were able to achieve a F1 score of 63.67 and a EM score of 60.372.

We were able to achieve the metrics above after doing the following hyper-parameter search:

Dropout	Hidden Layers	Learning Rate	Dev F1	Dev EM
0.2	75	0.5	63.69	60.38
0.3	75	0.5	**64.55**	**61.42**
0.25	85	0.5	63.23	60.07
0.2	100	0.5	63.8	60.63
0.3	100	0.5	64.12	60.66
0.5	100	0.5	58.25	55.3
0.4	130	0.5	61.03	57.75

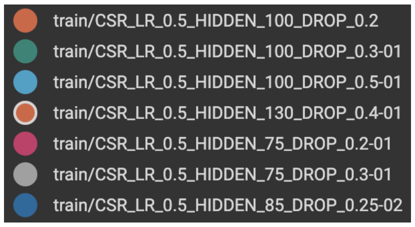
● train/CSR_LR_0.5_HIDDEN_100_DROP_0.2
● train/CSR_LR_0.5_HIDDEN_100_DROP_0.3-01
● train/CSR_LR_0.5_HIDDEN_100_DROP_0.5-01
● train/CSR_LR_0.5_HIDDEN_130_DROP_0.4-01
● train/CSR_LR_0.5_HIDDEN_75_DROP_0.2-01
● train/CSR_LR_0.5_HIDDEN_75_DROP_0.3-01
● train/CSR_LR_0.5_HIDDEN_85_DROP_0.25-02

Figure 3: Color key for charts below.

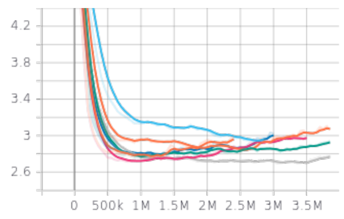

Figure 4: Dev NLL

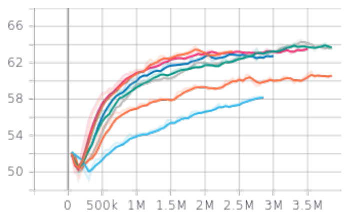

Figure 5: Dev F1

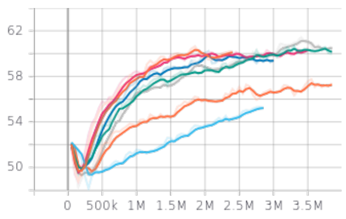

Figure 6: Dev EM

These results are slightly worse than expected, as we were not able to match the performance of the R-NET model in the paper. However, when taking into account that the original paper used SQuAD 1.0 and we were using SQuAD 2.0, a substantially harder dataset, it is not that surprising we weren't able to match the original metrics. The best combination of hyperparameters lead to a 5.92% increase in F1 score over the baseline and a 5.96% increase over the baseline EM score.

6 Analysis

As stated in our approach, we performed an ablation study to analyze the unique effects and impacts separate pieces of our R-NET Prime model, and then to finally determine the effect our faster, modified attention mechanism had in comparison to other standard R-NET layers. We created a naming system for the models where (B) stands for baseline, (S) stands for self attention, (C) stands for character embeddings, and (R) stands for R-NET output pointer-network. The table below shows our results from these ablation studies. Recall that throughout, B = Baseline, C = Character embeddings, S = Self Attention, and R = R-NET output layer.

Model Type	**Dropout**	**Hidden Layers**	**Learning Rate**	**F1**	**EM**
BBB	0.2	100	0.5	60.94	57.96
CBB	0.2	100	0.5	62.74	59.57
BSB	0.2	100	0.5	62.57	59.2
BBR	0.2	100	0.5	60.92	57.3
CSR	0.2	100	0.5	**63.8**	60.63
CBB	0.5	100	0.5	56.76	53.71
BSB	0.5	100	0.5	56.38	53.54
BBR	0.5	100	0.5	56.34	53.96
CSR	0.5	100	0.5	58.25	55.3
CBB	0.2	75	0.5	62.86	59.6
BSB	0.2	75	0.5	61.2	57.84
BBR	0.2	75	0.5	60.31	57.13
CSR	0.2	75	0.5	63.69	60.38

We see from the different experiments ran that typically character level embeddings outperforms both self-attention and the pointer network when ran in isolation. In the case where hidden layer size is 100 and the dropout is 0.5, self-attention proved to be the greatest help. In fact, the self-attention model outperfromed the R-NET Prime model with these hyperparameters leading us to believe that the pointer network and character embeddings suffer greatly from the higher drop out. The pointer network appears to only have marginal increase in our model. In essence the R-NET Prime framework gains the most from the character level embeddings as it encodes more information for the rest of the model to gather from.

When initially running experiments, we experienced overfitting when our dropout probability was set to the standard 0.2. Thus, we ran all 4 combinations of our model at an extremely high dropout rate of 0.5 to see if this could address our overfitting woes. As expected, our loss curves flattened with this higher dropout rate (as opposed to a rising loss in later epochs with a dropout of 0.2). However, we experienced significant performances drops from all of our models in this catagory. Thus, when running experiments on our final model configuration (CSR), we made sure to work with a variety of dropout rates between 0.2 and 0.5, searching for a balance of robustness and power.

Interesting enough, our final ablation study of decreasing the hidden layer size from 100 to 75 had an almost negligible difference when compared to the original size of 100. Beneficially, this meant we could train models longer and faster with these reduced hidden layer sizes, and as a result 2 of our top 3 performing models had hidden layer sizes less than 100.

When comparing the CSR model to the original R-NET model from the paper, we experienced significantly worse results. While it is still impossible to compare directly as the original R-NET used SQuAD 1.0 and we used SQuAD 2.0, we still could reasonably infer that our model would have performed worse than the original R-NET architecture. However, this architecture was so performance intensive that with our constraints on both time and VM resources, we couldn't complete a single training session of the original architecture. Thus, while our attention mechanism was less robust, it allowed us to run as many experiments as we did.

7 Conclusion

In this project, we explored which components contribute most to the success of R-NET Prime, a model based off of a combination of BiDAF and R-NET. R-NET Prime uses character-level embeddings, self-attention, and a pointer network output layer (all from R-NET) along with bidirectional attention and modeling layers (from BiDAF) to increase performance while still training quickly.

Though we did not implement self-attention exactly as described in R-NET, we still show some gain in performance with its inclusion. This leads us to believe that the intuition behind self attention is correct, regardless of how exactly it is implemented. This is also a limitation of our work, as the results found here may not extend to a full R-NET implementation.

Future work includes the analysis of the gated attention mechanism used in R-NET, which we did not implement for R-NET Prime. This gate is used before the 'modeling' layer and consists of the following operation, where u_t^P is the passage representation, c_t is the previous output from the RNN, and W_g is a learned parameter.

$$g_t = sigmoid(W_g[u_t^P, c_t])$$

$$[u_t^P, c_t]* = g_t \odot [u_t^P, c_t]$$

Since this gate introduces a dependency between the input and output of the RNN, when we implemented it as written, we saw a great increase in training time required, to the point where further exploration was infeasible and it made more sense to stick to BiDAF's modeling layer.

This project was a learning experience for all of our team members. Each of starting the class with various levels of artificial intelligence knowledge and through this project can now say that we have a basis of understanding for natural language processing. More importantly, we learned the steps for design, implementation, training and testing and how to tackle the challenges in this process. From combating annoying Microsoft Azure VM shutdowns, to simulations getting stuck on virtual machines we lost access to we learned that natural language processing research takes patience.

References

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

[2] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[3] Seo Minjoon, Aniruddha Kembhavi, Ali Farhadi, and Hananneh Hajischrizi. Bi-directional attention flow for machine comprehension. In *International Conference on Learning Representations*, 2017.

[4] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. 2016.

[5] Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. May 2017.

[6] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2017.