Accelerated and Accurate Question Answering

Haoshen Hong **Kuangcong Liu** **Xiao Wang**
haoshen, cecilia4, xiao1105@stanford.edu

Abstract

To train a model that could have good performance on the SQuAD 2.0 dataset by developing non-PCE model, we combine GloVe embedding and char-CNN embedding and build our model based on BiDAF with Self-Attention. The stretch goal we propose is to improve the running efficiency of our model, we use SRU, an advanced RNN cell, to parallel our computations. At the time of submission, our **EM** and **F1** scores on the dev set are **68.53** and **71.46**.

1 Introduction

The task of question answering has gained a lot of popularity during the past years, and SQuAD, as a reading comprehension dataset, has also led many significant breakthrough models in building the machine comprehension system, important models like pre-trained contextual embeddings (PCE) model BERT and non-PCE model BiDAF. Our project focus on building model that could answer the question correctly on SQuAD 2.0. Our goal is to analyze methods that will explain the complicated interactions better between the context and the query, and therefore measure how well our machine comprehension system could understand the context.

In order to accomplish this goal, we use GloVe word embeddings and char-CNN as our input embeddings, build our model based on BiDAF and Self-Attention, and enhance performance of BiDAF by combining it with new variants to regularize and optimize the recurrent neural networks. We also use SRU, an advanced RNN cell, to parallel our computations, speeding up the training process while keeping the performance. Our stretch goal is to improve the running efficiency of our model, and we target to gain insights on model's robustness and sensitivity to parameters.

2 Related Work

Bi-directional Attention Flow for Machine Comprehension [5] introduces a bi-directional attention flow layer that connects the context and query information together by utilizing Context-to-query(C2Q) and Query-to-context(Q2C) attentions. Context-to-query attention indicates similarities of query words to each of context word, while Query-to-context attention explains similarities of context words to each of query word. The strength of this paper is that it doesn't use attention as a method to summary modularity as previous attention methods, instead it calculates the attention at each time step and passes the attention flow to the following model. This mechanism could reduce the loss of information caused by summarization.

One of the improvements made in *Simple and Effective Multi-Paragraph Reading Comprehension[1]* is to add residual self-attention layer after bi-attention layer to further improve performance. Inspired by this, our project will combine the mechanism of Self-Attention and BiDAF, and we will use the same structure of prediction layer as mentioned in this paper [1] to predict more precise confidence value and increase robustness of model at the same time.

On the other hand, though RNN has been proven to be effective in modeling time sequence data, it has been long criticized for being slow, as the computation must iterate from previous states in order to compute next state and loses the ability to parallel. *Simple Recurrent Units for Highly Parallelizable*

Recurrence[3], however, proposes a new RNN architecture, Simple Recurrent Unit(SRU), which both preserves the performance of RNN-like structure and enable the computation to run in parallel and thus increase the training efficiency. Through some experiments, SRU turns out having even better performance than regular LSTM and about 4x to 8x faster than that on classification benchmarks and SQuAD v1.1. SRU also increases the speed and accuracy when combined with Transformer. The performance improvement in many tasks seems marginal, but the acceleration is rather impressive.

Effective regularization methods for deep learning have been the subject of much research in recent years. *Regularizing and Optimizing LSTM Language Models [7]* focuses on specific problem of word-level language modeling and investigate strategies for regularizing and optimizing LSTM-based models. They provide a set of regularization strategies that could be used with no modification to existing RNNs such as LSTM. Our experiments also utilize the several methods suggesting by this paper. More specifically, we will use the methods of embedding dropout, activation regularization (AR), and temporal activation regularization (TAR) inspired from this paper.

3 Approach

3.1 Char-CNN

Char CNN utilizes subword information and is proven effective in modeling OOV (out-of-vocabulary) words. Similar to the Character Embedding Layer mentioned in BiDAF[5], we map each word to a high dimension vector based on Kim et al.'s work in *Character-Aware Neural Language Models* [2].

For each word $w = [c_1, c_2, ..., c_k]$, we apply 1D convolution with kernel size 5 and apply ReLU as nonlinear activation. Then maxpooling is applied to map each word into fixed length embedding space. After that we apply a highway network to increase the Char CNN complexity, and the output word vector with Char Embeddings is $x_{word\_emb} \in \mathbb{R}^{e_{word}}$, where e_{word} is the size of word embedding from GloVe.

3.2 BiDAF with self attention

BiDAF introduces a multi-stage hierarchical model that use bi-directional attention flow to generalize context representations based on query. The bi-directional attention flow layer connects the context and query information together by utilizing Context-to-query(C2Q) and Query-to-context(Q2C) attentions. Context-to-query attention indicates similarities of query words to each of context word, while Query-to-context attention explains similarities of context words to one of query words.

We will use the model that combines BiDAF with self-attention as proposed by [1], which will add residual Self-Attention layer after bi-attention layer. Self-Attention could be explained as when the model processes a word from an input sequence, Self-Attention allows the model to look at other positions in the input sequence, and find other words with strong relation of the current word and therefore could lead to a better encoding for the current word.

The Self-Attention layer is calculated really similar to BiDAF Attention. Assume we have context hidden states $\mathbf{c}_1, ..., \mathbf{c}_N \in \mathbb{R}^{2H}$. We compute the similarity matrix $\mathbf{S}' \in \mathbb{R}^{N \times N}$, which contains a similarity score $\mathbf{S}'_{ij}$ for each pair $(\mathbf{c}_i, \mathbf{c}_j)$ of context hidden states.

$$\begin{aligned}
\mathbf{S}'_{ij} &= w_{sim}^T[\mathbf{c}_i; \mathbf{c}_j; \mathbf{c}_i \circ \mathbf{c}_j] \in \mathbb{R} \\
\alpha^i &= softmax(\mathbf{S}'_{i,:}) \in \mathbb{R}^N, \forall i \in \{1, ..., N\} \\
\mathbf{a}_i &= \sum_{j=1}^{N} \alpha^i_j \mathbf{c}_j \in \mathbb{R}^{2H}, \forall i \in \{1, ..., N\}
\end{aligned}$$

We output $\mathbf{a} \in \mathbb{R}^{N \times 2H}$ as Self-Attention value of the context and pass it to the output layer. The stucture of our model is shown in Figure 1.

3.3 Prediction Layer

In order to further improve model performance, We also modify the output layer in BiDAF according to Figure 1. Instead of directly computing start index by applying FC and softmax layer, we first apply Bi-RNN and concatenate the output of RNN to attention output for end index calculation.

$$\mathbf{m}_1 = \text{Bi-RNN}(\mathbf{m}) \in \mathbb{R}^H$$
$$\mathbf{m}_2 = \text{Bi-RNN}([\mathbf{m}_1; \mathbf{m}]) \in \mathbb{R}^H$$
$$\text{start score} = softmax(\mathbf{W}_{m1}\mathbf{m}_1 + \mathbf{b}_{m1} + \mathbf{W}_{a1}\mathbf{a} + \mathbf{b}_{a1})$$
$$\text{end score} = softmax(\mathbf{W}_{m2}\mathbf{m}_2 + \mathbf{b}_{m2} + \mathbf{W}_{a2}\mathbf{a} + \mathbf{b}_{a2})$$

Here, **m** represents the output from self-attention layer.

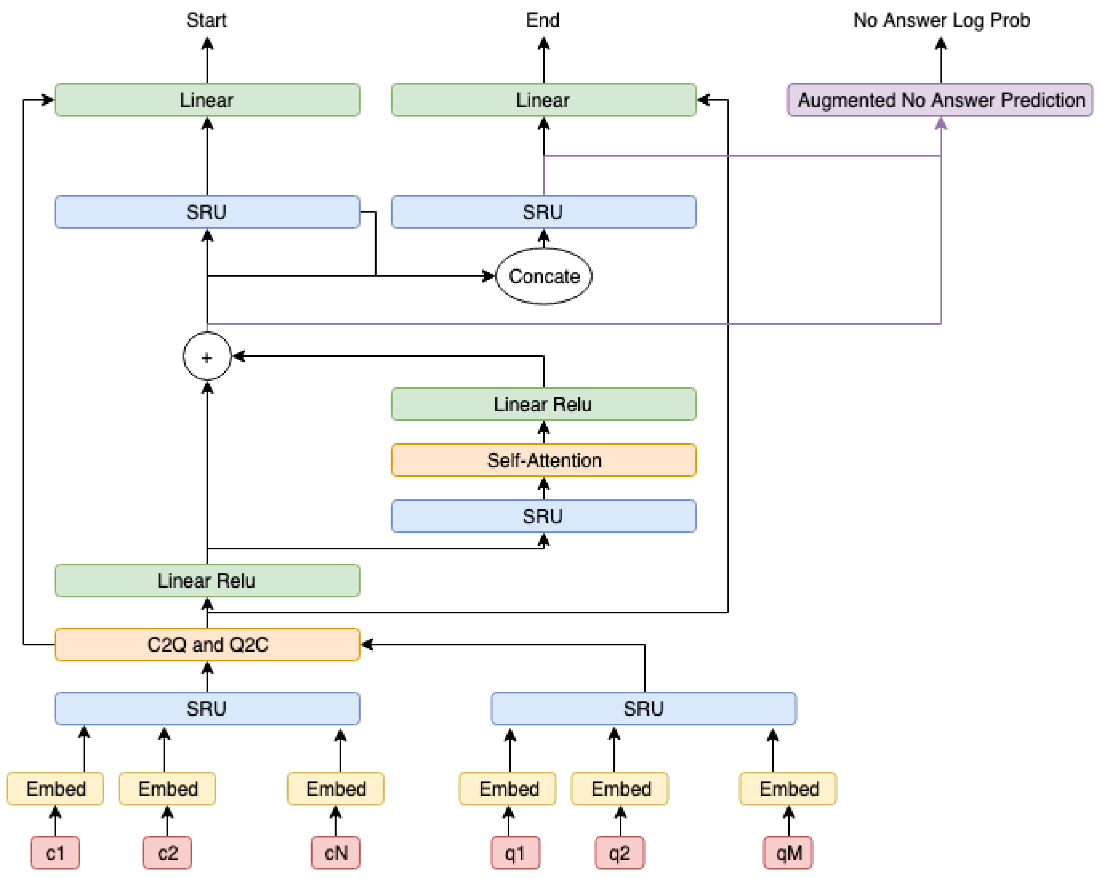

Figure 1: The model structure we developed, combining BiDAF, Self-Attention and SRU.

3.4 SRU

LSTM has been proven to be effective in modeling time sequence data. However, it has been long criticized for being slow, as the computation must iterate from previous states in order to compute next state and loses the ability to parallel. As a modified RNN cell, SRU both preserves the performance of RNN-like structure and enables the computation to run in parallel to promote the training efficiency. The key formula of SRU is:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{v}_f \odot \mathbf{c}_{t-1} + \mathbf{b}_f) \tag{1}$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot (\mathbf{W}\mathbf{x}_t) \tag{2}$$
$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{v}_r \odot \mathbf{c}_{t-1} + \mathbf{b}_r) \tag{3}$$
$$\mathbf{h}_t = \mathbf{r}_t \odot \mathbf{c}_t + (1 - \mathbf{r}_t) \odot \mathbf{x}_t \tag{4}$$

Comparing to regular LSTM formula, $\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{V}_f \mathbf{c}_{t-1} + \mathbf{b}_f)$, the formula (1) has a lighter product between **v** and **c**, which both reduces computation time and enables the model to compute every dimension of the product in parallel. The (3) and (4) comprises a highway network, which is not relied by (1) and (2), meaning we can also compute the highway network in parallel.

In experiment of [3], SRU turns out to accelerate for 2 times and get better performance on EM and F1 on SQuAD v1.1, and we will use this on our model on SQuAD v2.0. In the paper, the best performance is achieved with the number of layers is double of that of original LSTM structure. We will keep this design in our experiment.

3.5 Regularization

Inspired by *Regularizing and Optimizing LSTM Language Models [7]* , one of the mechanism we tried is embedding dropout, which will perform dropout on the embedding matrix at a word level. This method extend the original dropout technique and the dropout is broadcast across all the word vector's embedding, to prevent overfitting of the RNN model. We first sample a mask of length $num\_embeddings$ from bernoulli with probability equals (1 - dropout), and then extend this mask to be the size of ($num\_embeddings$, $embedding\_dim$) to drop all weights of a word.

Another methods they mentioned are activation regularization(AR) and temporal activation regularization(TAR). AR and TAR is defined as:

$$\text{AR regularization} = \alpha L_2(m \odot h_t)$$
$$\text{TAR regularization} = \beta L_2(h_t - h_{t+1})$$

where L_2 stands for the l2-norm, m is the dropout mask, and h_t stands for the hidden states at timestep t. AR will penalize activations that are significantly larger than 0, while TAR will penalize the model from producing large changes in the hidden state from time t to time $t+1$. Therefore, AR and TAR could control the norm of the resulting model and reduce overfitting.

3.6 Augmented No-answer Prediction

Having observed that the AvNA (accuracy of no-answer prediction) metric of our model is relative low, we seek a method that can boost model's no-answer prediction. Inspired by [4], we include an additional attention and linear layer that takes in attended contextual embedding and end of span representation, i.e., $[\mathbf{a}, \mathbf{m}_2]$, and outputs the probability of whether this question can be answered. The modified loss function is a linear combination of original loss and NA loss, the cross entropy loss of binary classification. In our experiment, we use coefficient $\lambda = 1.5$, and the correspondent loss = $\mathcal{L}_{orig} + \lambda \mathcal{L}_{NA}$.

4 Experiments

4.1 Dataset

We use SQuAD 2.0 dataset for this work [6]. SQuAD 2.0 comprises 129,941 training examples, 6078 dev examples and 5921 test examples, where each example is presented as context-question pairs. The target is whether the question is answerable from the given context, and if so, we need to predict the answer span from the given context.

4.2 Evaluation method

EM. Exact match. This metric measures the percentage of predictions that match any one of the ground truth answers exactly.

F-1 score. $F_1 = (\frac{recall^{-1} + precision^{-1}}{2})^{-1}$

4.3 Experiments and Results

Our final model (10-model-ensemble) achieves EM: 68.53 and F1: 71.46 on dev set, EM: 65.99 and F1: 68.98 on test set.

4.3.1 SRU

First we explore the performance of SRU on original baseline model. For comparison, we also use GRU, a commonly used lighter RNN cell than LSTM, as one of our baseline. We train the model for 30 epochs and all of them have converged from Tensorboard in Fig.2. To make our experiment reproducible, we run all experiments on single Nvidia K80 GPU with 8 CPUs.

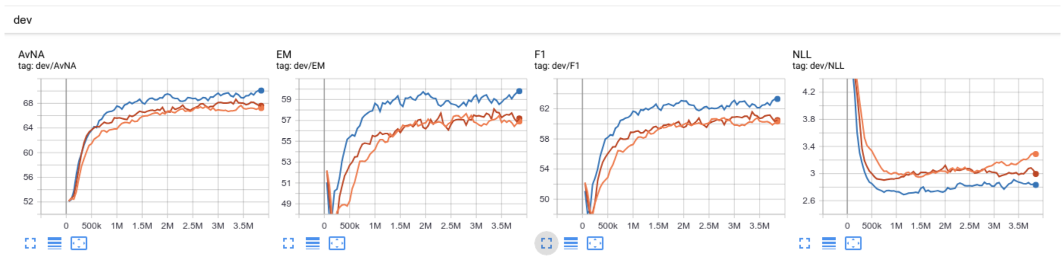

Figure 2: Performance of RNN cells on baseline model. Blue: SRU. Red: GRU. Orange: LSTM

To be specific, in our experiment, we follow the implementation similar to original paper, which doubles the number of layers when change LSTM to SRU. From Table. 1, we can see the baseline LSTM model performs worst with longest training time. Changing LSTM to GRU achieves minor increase in performance and training hours. It can also be observed that SRU could accelerate our training process by more than 2x and improve model performance by around 1.8 in EM and 2.5 in F1, which is significant comparing with GRU's performance boost.

RNN Cell	EM	F1	Time (Hours)
LSTM(Baseline)	57.64	60.80	12.98
GRU	58.09	61.59	10.15
SRU	**59.81**	**63.31**	**5.57**

Table 1: Experiments on baseline model with different RNN cells

4.3.2 Self Attention

Then we implement BiDAF with self attention model, which is proven to improve the baseline of SQuAD 1.1. Our model is very similar to original paper except the fact we add attention output and modeling layer output before they are feed into softmax layer which is consisitent with provided start code. We also observe that combining BiDAF with self attention model with SRU can achieve better performance and acceleration.

Model	EM	F1	Time (Hours)
BiDAF	57.64	60.80	12.98
BiDAF + Self Attention	59.20	62.45	13.63
BiDAF + Self Attention + SRU	**60.04**	**63.66**	**6.08**

Table 2: Experiments on BiDAF with self attention

The result, comparing with baseline model, is a significant boost both in performance and running time. In our following experiment, we will use BiDAF + Self Attention + SRU as our **base** model and explore the effectivity of each module on our base model.

4.3.3 Char-CNN

We also include Char CNN in our embedding layers as an additional embedding and concat with word embedding. Through our experiments we observe an increase in running time but a significant improve in performance. The improvement could be easily explained: if we only use the word embedding from GloVe, and if a word is not in this pre-defined vocabulary, then it is represented as the <UNK> token. With extra character embeddings, our model could overcome this problem because it will rely on the Character-based embedding as representation of the word.

Model	EM	F1
Base	60.04	63.66
Base + Char CNN	65.20	68.48

Table 3: Experiments on BiDAF with Char-CNN

4.3.4 Regularization

We introduce AR and TAR regularization in our loss, to control the norm of the resulting model and reduce overfitting of our RNN model. Similar to the parameters introduced in [7], we will use α to be twice the value of β. And from the below table 4, we could see that when $\alpha = 0.1, \beta = 0.05$, we get the best results.

Model	EM	F1
Base	60.04	63.66
Base + Embedding Dropout (= 0.1)	58.91	62.28
Base + AR(α =0.0002) + TAR(β =0.0001)	60.66	64.06
Base + AR(α =0.002) + TAR(β =0.001)	60.11	63.52
Base + AR(α =0.02) + TAR(β =0.01)	58.83	62.49
Base + AR(α =0.1) + TAR(β =0.05)	**61.12**	**64.40**

Table 4: Experiments on BiDAF with AR and TAR regularization

4.3.5 Augmented No-answer Prediction

We explore several structures of the augmented no-answer inputs by trying some combinations of attention results, self attention results and start/end raw representation and logits. The improvement seem insignificant for each structure in terms of EM and F1, but we will still decide to include the NA loss structure with attention and span end representation since the new structure improves AvNA.

Model	EM	F1	AvNA
Base	60.04	63.66	69.64
Base + NA Pred[Att, End rep]	60.09	**63.70**	**70.66**
Base + NA Pred[Att, Start rep, End rep, Max logits]	59.49	62.24	67.13
Base + NA Pred[Att, Start rep, End rep]	**60.12**	62.96	67.99
Base + NA Pred[Start rep, End rep]	60.07	62.99	68.49

Table 5: Experiments on BiDAF with Augmented No-answer Prediction

4.3.6 Fine-tuning

After working out the structure and different modules of our model, we perform an extensive parameter tuning on tune-able parameters as shown in tabel 6. The result, surprisingly, is even worse than our initial experiment. The result may be attributed to the large variance of each individual experiment and coarseness of parameters' granularity. Without massive computing power and enough time to implement robust parameter search strategy, we might not be able to find an optimal parameter via greedy search of this kind. However, this extensive analysis can at least give some insights to the model's sensitivity to each parameter and some information about variance of model's performance.

Param	Value(F1, EM)			
batch_size	32 (66.72, 63.82)	**64 (69.07, 65.72)**	128 (66.81, 63.18)	256 (Fail)
hidden_size	100 (67.52, 64.07)	150 (67.84, 64.41)	**200 (68.38, 64.54)**	250 (68.16, 64.54)
lr	0.125 (65.17, 61.87)	**0.25 (68.04, 64.53)**	0.5 (67.74, 64.07)	1.0 (63.31, 60.01)
l2_wd	**0 (68.32, 64.68)**	1e-3 (66.85, 63.23)	1e-2 (43.83, 43.30)	1e-1 (51.94, 51.92)
drop_prob	0.15 (67.15, 63.91)	0.20 (67.39, 63.75)	**0.25 (67.80, 64.46)**	0.30 (66.49, 63.08)
alpha	1e-4 (67.57, 64.26)	1e-3 (67.62, 64.24)	1e-2 (66.62, 63.20)	**1e-1 (67.33, 63.84)**
beta	**1e-4 (67.92, 64.24)**	1e-3 (67.54, 64.06)	1e-2 (67.49, 63.89)	1e-1 (66.70, 63.07)
na_loss_weight	0.5 (67.77, 64.51)	0.75 (67.87, 64.44)	**1.0 (67.98, 64.70)**	1.5 (67.33, 63.79)
drop_prob_emb	0.15 (68.23, 64.86)	0.20 (66.98, 63.82)	0.25 (67.93, 64.19)	**0.30 (68.74, 65.64)**

Table 6: Experiments of parameter tuning

5 Analysis

We summarize the modes of errors by our model and shows examples for each category of error in SQuAD 2.0. We randomly select 64 incorrect questions (based on EM) and categorize them into 5 categories: "NA (No Answer) but predict an answer", "has an answer but predict NA", "predict wrong answer with correct Part Of Speech", "predict correct answer but with extra modifiers", and "predict wrong span range", and the exact error ratio is shown in figure 3.

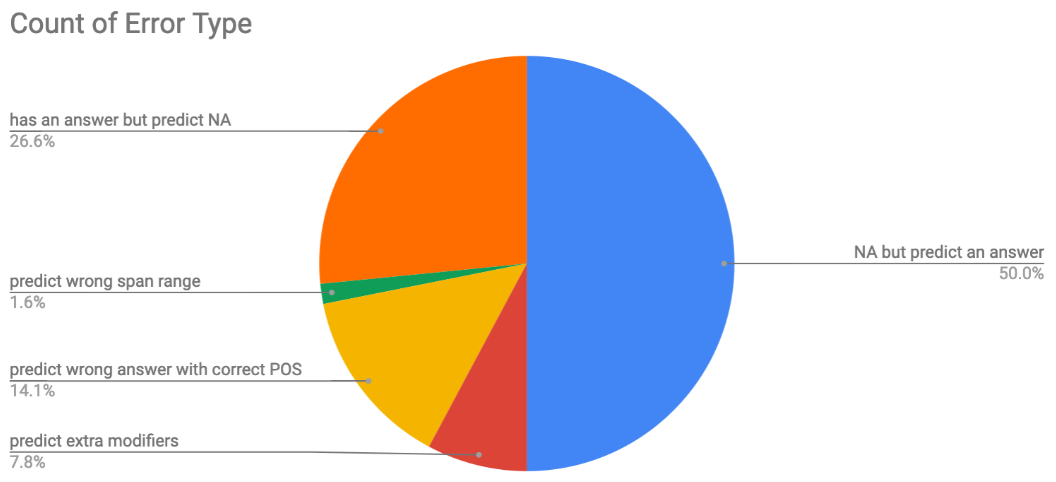

Figure 3: Ratio of 5 different error types in our prediction on SQuAD 2.0

We make the error analysis to understand when our system fails, and the below table 7 shows us examples from different error types. Among all error types, NA prediction error comprises 76.6% of total errors. In order to further improve the model, a possible option would be focus on reducing NA errors.

Error type	Ratio(%)	Example
NA (No Answer) but predict an answer	50	**Context:** One of the first Norman mercenaries to serve as a Byzantine general was Hervé in the 1050s. By then however, there were already Norman mercenaries serving as far away as Trebizond and Georgia. They were based at Malatya and Edessa, under the Byzantine duke of Antioch, Isaac Komnenos. In the 1060s, Robert Crispin led the Normans of Edessa against the Turks. Roussel de Bailleul even tried to carve out an independent state in Asia Minor with support from the local population, but he was stopped by the Byzantine general Alexius Komnenos. **Question:** Who was the first Byzantine mercenary to serve with the Normans? **Prediction:** Hervé **Answer:** []
has an answer but predict NA	26.6	**Context:** They even lent their ethnicity to the name of their castle: Afranji, meaning "Franks." **Question:** What was the name of the Norman castle? **Prediction:** [] **Answer:** ['Afranji', 'Afranji', 'Afranji']
predict wrong answer with correct POS	14.1	**Context:** The Normans were famed for their martial spirit and eventually for their Christian piety, becoming exponents of the Catholic orthodoxy into which they assimilated. **Question:** What religion were the Normans? **Prediction:** Christian **Answer:** ['Catholic', 'Catholic orthodoxy', 'Catholic']
predict correct answer but with extra modifiers	7.8	**Context:** The Duchy of Normandy, which they formed by treaty with the French crown, was a great fief of medieval France, and under Richard I of Normandy was forged into a cohesive and formidable principality in feudal tenure. **Question:** Who ruled the duchy of Normandy? **Prediction:** Richard I of Normandy **Answer:** ['Richard I', 'Richard I', 'Richard I']
predict wrong span range	1.6	**Context:** The English name "Normans" comes from the French words Normans/Normanz, plural of Normant, modern French normand, which is itself borrowed from Old Low Franconian Nortmann "Northman" or directly from Old Norse Norðmaðr, Latinized variously as Nortmannus, Normannus, or Nordmannus (recorded in Medieval Latin, 9th century) to mean "Norseman, Viking". **Question:** What is the original meaning of the word Norman? **Prediction:** Old Low Franconian Nortmann "Northman **Answer:** ['Viking', 'Norseman, Viking', 'Norseman, Viking']

Table 7: Error analysis on SQuAD 2.0. We randomly selected EM-incorrect answers and classified them into different categories. Only relevant sentence(s) from the context shown for brevity.

6 Conclusion

In this project, we work out a new model combining different modules of different Q&A models. Among our observations, Char CNN and SRU contribute most to model performace. As an additional gain, SRU could also decrease running time for approximately 2x so we can perform more experiments and parameter tuning in limited time. From our observations in parameter tuning, the model is sensitive to some parameters and the performance has relatively large variance. The model is hard to improve significantly at current state, as parameters are designated for current model and probably not working well when we include a new module which might have negative inclination of the given module.

Future work, in order to improve the performance of the model, could focus on reduce NA errors by figuring our a more robust NA classifier. Another option is to utilize learned information from general natural language: using BERT as pretrained word embedding is one of examples.

References

[1] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension, 2017.

[2] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[3] Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. Simple recurrent units for highly parallelizable recurrence, 2017.

[4] Xiaodong Liu, Wei Li, Yuwei Fang, Aerin Kim, Kevin Duh, and Jianfeng Gao. Stochastic answer networks for squad 2.0, 2018.

[5] Ali Farhadi1 Hananneh Hajishirzi Minjoon Seo, Aniruddha Kembhavi. Bi-directional attention flow for machine comprehension, 2017.

[6] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad, 2018.

[7] Richard Socher Stephen Merity, Nitish Shirish Keskar. Regularizing and optimizing lstm language models, 2017.