Question Answering with Self-Attention

Stanford CS224N Default Project

Georgios Sarmonikas
Department of Computer Science
Stanford University
gs7776@stanford.edu

Abstract

Question Answering (QA) is an increasingly important topic in NLP with the proliferation of chatbots and virtual assistants. In this project a QA system is built by exploring two end-to-end models: a) the Bi-Directional Attention Flow network (BiDAF) and b) QANet, a non-recurrent model fully based on convolution and self-attention, capable to achieve up to 13x speedup in training and inference time. Two major goals were accomplished: Firstly, the baseline BiDAF model was improved by adding a character embedding layer, an embeddings attention layer, a context-to-context self-attention layer, gated recurrent units and Swish activation. Secondly, the QANet model was re-implemented from scratch and successfully explored some hyperparameter finetunings to improve performance. The improved BiDAF model, SA-BiDAF++, achieved **67.03 EM / 70.1 F1** scores on the development set and **65.31 EM / 68.78 F1** scores on the test set of the SQuAD 2.0.

1 Key Information to include

- Mentor: Rui Yan
- External Collaborators (if you have any): None
- Sharing project: N/A

2 Introduction

Machine reading comprehension and Question Answering (QA) systems are two domains of Natural Language Processing (NLP) which over the past years have made significant progress. Reading comprehension is the ability to process text and understand its meaning, while QA systems aim to understand a given context or text passage and predict the correct answer for a corresponding query, by extracting a short span of text from a corresponding context paragraph and present it as the answer. This is the objective of SQuAD (the Stanford Question Answering Dataset) models [1]. In SQuAD 2.0, an additional challenge was introduced: The model has to indicate when a question is unanswerable given the corresponding context paragraph. An accurate execution of this task has many important and practical applications such as in Search Engines, chat-bots, virtual assistants and more, and therefore is an active area of research.

Recently a variety of neural architectures exceeded human performance in the QA task according to the SQuAD leaderboard [2]. These advances can be broadly categorised into a) Pre-trained Contextual Embedding (PCE) methods which are usually "off-the-shelf" models such as BERT [3] that could be employed for specific tasks, and in many cases using ensemble methods to achieve very high performance, and b) Non Pre-trained (Non-PCE) methods, which are not achieving state-of-the-art performance but allow for deep learning practitioners to explore different techniques, develop new models or enhance model implementations.

This project focuses on the latter approach using the SQuAD 2.0 dataset and has experimented with some of the most successful non-PCE methods like the Bi-Directional Attention Flow (BiDAF) [4] model and the QANet model [5], with the overall goal to achieve the highest performance possible on the Non-PCE SQuAD 2.0 leaderboard.

A basic baseline implementation of BiDAF is provided by the course, and so my experiments are related to proposing improvements to the architecture to achieve higher performance than the baseline. Being inspired by [5] and [6], Self-Attention layers are added to the model structure. Also, a character embedding layer is added, changed the recurrent units from LSTM to Gated Recurrent Units (GRU) and replaced ReLU with Swish [7] activation functions. The improved BiDAF model added a gain of 11 EM points and 10 F1 points compared to the basic baseline BiDAF model, when measured on the dev set of the given SQuAD 2.0 set.

In addition, the QANet model is re-implemented in Pytorch and re-evaluated using the SQuAD 2.0 set. It adapts ideas from the Transformer [6] by replacing the RNNs with convolutions and self-attention. The main component of QANet is the Encoder Block which is similar to the Transformer but differs by its use of stacked convolution sublayers which use depthwise-separable convolution to capture local dependencies in the input sequence. While it was difficult to reproduce the performance, especially the training boost reported in the original paper, a few simple hyperparameter modifications are proposed that can improve the performance, as described in the Experiments section. Lastly, the model is compared against the baseline BiDAF and the new improved BiDAF model.

3 Related Work

Machine question-answering has been an active area of research in the past few years. Datasets like SQuAD 2.0 [8] are very popular benchmarks for NLP research, especially for Q&A or NMT tasks. This has allowed the development of many models (non-PCE) in the recent years that have achieved significant progress in achieving close to human performance.

The BiDAF [4] model is one such model. It employs both recurrent neural network units such as LSTM for capturing the sequential input, and also exploits the use of attention mechanism for capturing long-term interactions. Its architecture is based on a hierarchical multi-stage end-to-end network which takes inputs such as character, word and phrase to obtain a query-aware context representation using memory-less context-to-query (C2Q) and query-to-context (Q2C) attention. This representation can then be used for different final tasks, such as question answering.
In BiDAF, the attention at each time-step is a function of the question and the context and does not depend on the attention of the previous time-step. This allows the model to focus on the interaction between Query and Context at each time-step. One major disadvantage of this model is that it is heavily reliant on RNNs (Recurrent Neural Networks), which are resource intensive and slow in training. Also, the sequential nature of RNNs prevent parallel computation, since tokens must be fed into the RNNs in order.

To improve upon these disadvantages, a breakthrough was made by Vaswani et.al in "Attention Is All You Need" [6], where the Transformer architecture was proposed. This architecture relies solely on attentions and compute representations of its input and output without using sequence-aligned recurrent layers nor convolution layers. It achieved great results in neural machine translation, and since then has shown advantages over RNNs in many other tasks.

QANet [5] was published about a year later, in 2018, which increased the training speed by discarding the recurrent architecture. It borrowed ideas proposed in the Transformer [6] architecture and exclusively used convolutions and self-attention as the building blocks which encode the question and context separately. Standard attentions are used to learn the interactions between context and questions and then the final answers are predicted based on RNN-free encoded representations. The authors estimated that QANet was training 4.3 times faster than BiDAF, and had 7 times faster evaluation, which means a significant advantage as they could use data augmentation techniques to increase the number of training examples available, and process more training data than the BiDAF model. Prior to PCE methods such as BERT, QANet had state-of-the-art performance for SQuAD 1.1 significantly outperforming the previous best model on the official SQuAD leaderboard.

4 Approach

This project is addressed in two parts: 1) Improving BiDAF model and 2) Re-implementing QANet.

4.1 BiDAF

BiDirectional Attention Flow Model (BiDAF) consists of five layers, which are: Embedding layer, Encoder layer, Bi-direction Attention layer, Modeling layer and Output layer, as shown in the model architecture in Figure 1.A.

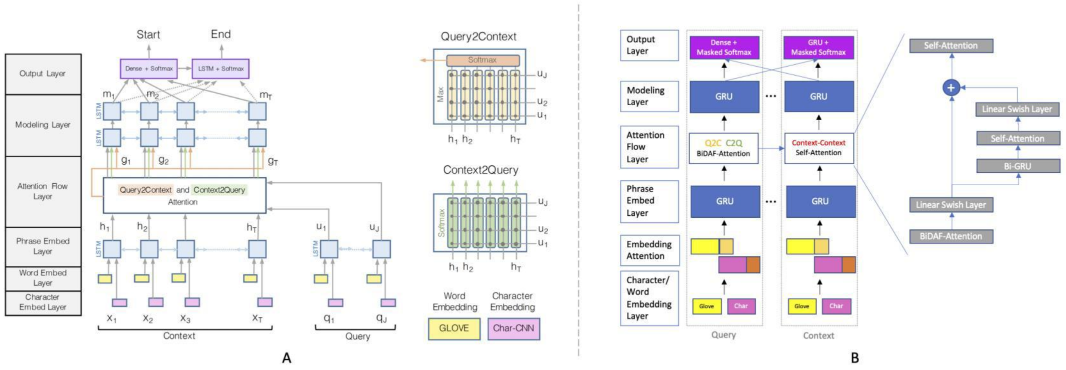


Figure 1: BiDAF architecture: A) original architecture, B) SA-BiDAF++ (with Self-Attention, Bi-GRU and Swish Activations)

4.1.1 Baseline: without character embedding

The baseline of the BiDAF model is provided by the teaching course, and it only used pretrained GloVe word embeddings with 300 dimensions. That is the BiDAF baseline in this project.

4.1.2 BiDAF with character embedding

The character-level embedding was not implemented in the above mentioned baseline, as when compared to the implementation of the original BiDAF paper. This embedding can be an extra feature for the questions and contexts and can be useful for improving the model performance.

To match therefore the original BiDAF model, a character-level embedding block is added which can extract information from the internal structure of words and provide benefit for typos or new words. It first embeds with the pretrained 300 dimension GloVe word-embedding for both context and questions and then concatenate with 64 dimension character-level embedding. The embedding block contains two convolutional networks with kernel sizes 5 and 3 respectively, extracting the char-level information. The output, word-character embedding, then follows the original procedure for the downstream blocks.

4.1.3 Embeddings Attention

Inspired by the use of attention in [6] and how it can improve performance, an extra layer of attention is added for the word-character embedding block which summarizes or calculates the "mean" of each one of the char and word vectors. Instead of just averaging or using max pooling, a learning function learns to compute the weights for each of the character and word vectors before summing them together. Weighting the vectors by the value this attention module learns, can be seen as computing the expection of the char and word vectors. Each learned attention weight is then appended back to the original (char and word) vectors, prior to following the procedure to the RNN Encoder layer. That helps the model pay attention to different parts of the char-word vector, further improving the performance.

4.1.4 Context to Context Self-Attention

The baseline model already included the bi-directional attention flow which can capture the context-query and query-context attentions. However, to help improve the coreference resolution, as mentioned in [9], an extra layer of self-attention is added which computes the attention scores between contexts. This is implemented by taking the BiDAF attention score tensor into a linear layer to change its dimensions. Then it passes through non-linear activation function and dropout, then to a RNN encoder to encode the scores and finally mix with the original input after computing the similarity between the two contexts and normalising them with a masked softmax function. The modified architecture, including the C2C Self-Attention is shown in Figure 1.B.

4.1.5 Fine-tuning

Several training approaches have also been explored. The modeling layer was changed from using LSTM to using GRU and increased the layers to two. That has reduced the time to train the model. In addition several non-linear activation functions were explored beyond ReLU; initially used SELU [10] and lastly, inspired by [7], used Swish which achieved the best BiDAF performance on the SQuAD 2.0 dev set.

4.2 QANet

In contrast to BiDAF, which applies the bidirectional Recurrent Neural Networks to capture the contextual information, QANet applies the self-attention with convolutions to reach the goal. The model consists of 5 main blocks, illustrated in the left part of Figure 2: An input embedding layer, an embedding encoding layer, a context-query bi-attention layer, 3 repeated model encoding layers, and an output layer.

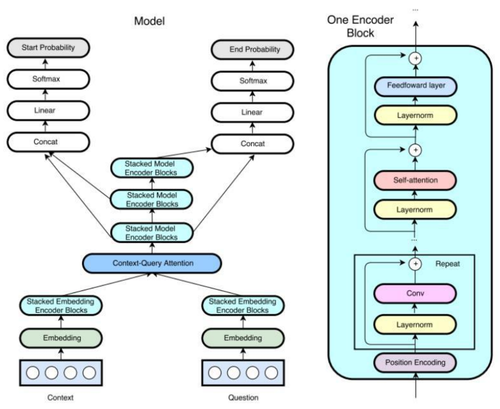


Figure 2: QANet model architecture and its Encoder block from the paper by Yu et al [5]

The **input embedding layer** is very similar to the one used in the BiDAF implementation. It consists of the GloVe embeddings of each word concatenated with the output of a convolutional layer that processes the embeddings of each character that makes up a word up to a maximum length of 16 characters per word. The only difference is that instead of using the pre-trained character embeddings of dimension 64, each character is represented as a trainable vector of dimension 200. The concatenated vectors are then passed through a two-layer highway network, similar to BiDAF.

Then the input is passed through the **embedding encoder layer**. Each encoder block consists of a positional encoding layer, several convolution layers, a self-attention layer, and a feed-forward layer. The positional encoding is added, since otherwise the convolutional structure of the network would lose track of that information. The rest of this block consists of several depthwise separable convolutions, followed by a multi-head self-attention layer and finally a feed-forward layer. The idea is that "convolution captures the local structure of the text, while the self-attention learns the global interaction between each pair of words.". The positional encoding sublayer and self-attention sublayer are the same as those in the Transformer model [6].

A **context-query attention layer** is then followed which is the same as the one used by the BiDAF model, and was reused from the BiDAF implementation. The attention output is expressed as $g_i = [c_i; a_i; c_i \circ a_i; c_i \circ b_i] \in \mathbb{R}^{8d}$ where d is the hidden dimension, $c_1, ..., c_N$ the context input, $a_1, ..., a_N$ and $b_1, ..., b_N$ the context-to-query and query-to-context attention.

The **output** of the attention layer is then fed through the model Encoder layer, which consists of several Encoder blocks chained together. The number of convolutions in each block can be different from the number we had in the encoder blocks of the embedding encoder layer. This step is repeated three times to get three output matrices M_0,M_1,M_2,in which the weights are shared between each repetition. Then these matrices are fed through the output layer, which follow a similar strategy as the one used in BiDAF. The probabilitis of starting and ending position are modeled as: $p_{start} = softmax(W_1[M_0, M_1])$, $p_{end} = softmax(W_2[M_0, M_2])$ and are computed independent from one another.

The original paper also describes a data augmentation process. This was not implemented due to lack of time. That may have caused though the implemented model to train slower than the original model which claim to have achieved 3x-13x faster training than RNN-based models.

5 Experiments

5.1 Data

For this project we are using the updated Stanford Question Answering Dataset [8], named SQuaD 2.0, which extends the original dataset with unanswerable questions. It consists of 100k+ examples from Wikipedia articles and the questions and answers were crowdsourced using Amazon Mechanical Turk. Each question is either unanswerable using the provided paragraph or has an answer that is a chunk of text taken directly from the paragraph. This means that the model has to decide whether a question is answerable, and if so, select a span of text in the paragraph that answers the question. About half of the questions are unanswerable.

The dataset that is used in this project is a slightly modified version of the SQuAD 2.0 dataset, where our training data is identical, but our dev and test sets are drawn from the official dev data. The data consists of (context, question, answer) triples, and each paragraph has an average length of around 250 words and the questions are usually around 15 words.

The dataset is split into three subsets: train, dev and test examples. The models are trained on the train set and the hyperparameters are tuned on the dev set. The test set is used for the evaluation against the SQuAD leaderboard. The training set used has 87k query-answer pairs, 10.1K for validation, and another 10.1K for testing.

5.2 Evaluation method

Evaluation of performance of the QA models is based on a) Exact Match (EM) score, measuring whether the answer span matches exactly with the ground truth answer, b) F1 score, computed as the harmonic mean of precision and recall, where precision is calculated as the number of correct words divided by the length of the predicted answer, and recall is calculated as the number of correct words divided by the length of the ground truth answer $F1 = 2 * precision * recall/(precision + recall)$, c) Answer vs. No Answer (AvNA), during the training process, which is the percentage of correct predictions of whether or not a question is answerable.

5.3 Experimental details

5.3.1 BIDAF experiments

The BiDAF experiments started with the baseline model. To run the experiments, an Azure VM instance with GPU acceleration was used as well as my local machine. Training time was about 50 minutes per epoch initially. To speed up the development iteration the model was trained for 20 epochs instead of 30. Also, Google Colab was used with GPU acceleration and that reduced the training time to about 25mins per epoch on the baseline model, since the P100 GPU in Colab is better than the one in Azure VM. That helped a lot with iterating and testing quickly different ideas and experiments.

To increase the model performance a character-level embedding was added to the baseline model. One Conv1D layer was used using kernel size = 3 and ReLU activation. That did not improve the performance much (only by 0.5 EM/F1 points) and slowed down the training process.

The learnable Embeddings attention as described in Section 4.1.3 was added to the embedding layer to further boost the performance. To decrease the training time, LSTM was replaced by GRU. That has led to simplified training as GRU require to do less calculations than LSTM. Also the training process was further reduced to about 15mins when using the GPU-accelerated Google Colab instance. In addition and as described in Section 4.1.4, Context-to-Context Self-Attention layer was added to the model. This has led to have more total layers and made the model more complicated and training time was increased to about 30mins per epoch.

To further improve the performance, two convolution layers were added in the character embeddings layer, one after the other to be able to learn more hidden structure from the embeddings. In the first one kernel size = 5 was used and the second used had kernel size = 3. Also, batch normalization was added. Batch normalization ensures that the mean and standard deviation of the layer inputs will always remain the same, thus, the amount of change in the distribution of the input of layers is reduced. That helps with the learning process. An additional benefit is that batch normalization has a regularization effect, which can help overcome overfitting and also help learn better. In our experiments that regularization effect was not visible, however it helped to achieve better performance in the evaluation metrics.

The activation function was also changed from ReLU to SELU [10]. ReLU suffers from the dying ReLU problem, in which the weight can get trapped in a dead state - the activation function stuck at the left side of zero. Scaled Exponential Linear Unit (SELU) is similar to ReLU, but provides benefits such as internal normalization, unlike ReLU cannot die and can learn faster and better, as mentioned in [10]. The change from ReLU to SELU reduced the training time by about 6 mins (in the Google Colab environment) and improved the model performance by about 1 points for both EM and F1 metrics.

Having as reference the extensive experiments done recently on activation functions by Ramachandran et.al in [7], I started to experiment further and applied the Swish activation function. Swish is given by $f(x) = x \cdot \sigma(\beta x)$ where $\sigma(x)$ is the sigmoid function and β is either constant or a trainable parameter depending on the model. Extensive experiments in [7] show that Swish consistently matches or outperforms ReLU and SELU in a variety of challenging domains such as image classification and machine translation. Swish, like SELU, also solve the dead ReLU problem because the derivative is non-zero for all x. Initially Swish was used in the Embedding layer after the convolutions and it achieved better results than SELU. Then all activation function in the BiDAF network were replaced by Swish. That proved to be successful given a jump to the performance by 1.74 and 1.6 points for EM and F1 respectively. It is important to notice that β was set to one ($\beta = 1$) for simplicity. In that case, Swish becomes same as the SiLU activation, as mentioned in [11].

All the changes to the baseline BiDAF, including Embeddings Attention, Context-to-Context Self-Attention layers, GRU, as well as the change from ReLU to Swish, increased the model performance by a total of 10 points for both EM and F1 when compared to the baseline performance. It is important to notice than this performance is reported at half the epochs (15 epochs) to the baseline performance values (30 epochs). The hyperparameters used to train the improved BiDAF model were: learning rate=0.5, dropout rate=0.2, batch size=64, hidden size=100. These remained the same to be able to compare the outcomes of each iteration. See Appendix Figure 4.

5.3.2 QANet

For the implementation of the QANet model, an exact reproduction of the parameters described by Yu et al. in [5] was initially attempted. Adam optimizer was used with β_1=0.8, β_2=0.999, ϵ=10^{-7} and L2 weight decay=$3{\cdot}10^{-7}$; 200-dimensional pretrained GloVe vectors were used for word-level embedding. The hidden size was 128 like the original paper, but I also experimented with hidden size 96 and found that it decreased both F1 and EM scores by around 1.0 points, while not giving any boost in training speed. This was tested with a quick training iteration of only 10 epochs.

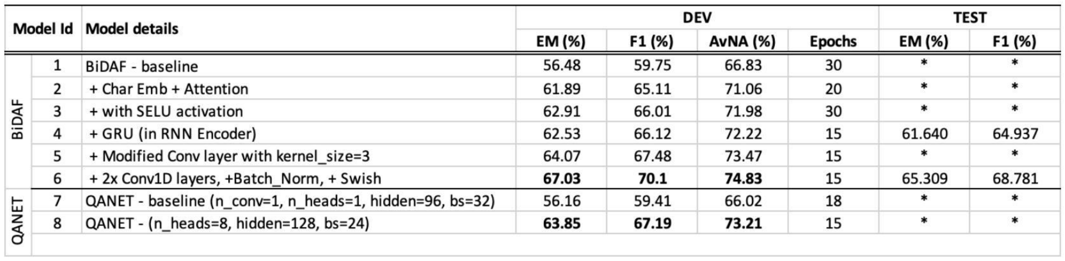
Also, like in the original paper, I used 8 heads in the self-attention layer, 7 encoder blocks in the model encoder layer, with 2 convolutions and kernel size of 5. Also for the embeddings encoder used

4 convolutions with kernel size of 7 and one encoder blocks. The dropout rates were set same as the original paper, 0.1 for word and between every two layers and 0.5 for characters.

When developing the QANet model, lots of trial and error iterations were necessary until the model train without memory issues. The QANet model seemed to require much more memory for the multi-head attention. Initially the model was run on the Azure VM. However, with hidden size = 96, multi-head attention = 1, batch size = 32, the model achieved lower performance than the baseline BiDAF model, see Model-id 7 in Figure 3. The use of the original paper parameters meant more memory requirements and faster GPU. Therefore, Google Colab was used once again in which the GPU acceleration (P100) was much better than the one provided by the Azure VM. However, and despite the better GPU, the QANet model took much longer to train (almost 90% longer) when compared to the improved BiDAF model. The experiment which provided best results was when batch size is 24 , 128 for hidden size and 8 for multi-head attention.

5.4 Results

The best BiDAF model achieved a **EM: 67.03** and **F1: 70.1** on the dev set and **EM: 65.31** and **F1: 68.78** on the course test set, ranked 4th in the leaderboard at the time of writing. I have named Self-Attention BiDAF++ (or SA-BiDAF++), due to the improvements with Self-Attention and other modifications in the architecture mentioned above. See Model-id 6, in Figure 3.
To compare, the re-implemented QANet model achieved **EM: 63.85** and **F1: 67.19** on the dev set. It took almost double the time to train it, averaging about 70 minutes/epoch, while BiDAF with all the modifications took on average 30 minutes/epoch in Google Colab with GPU being P100. Therefore, the training performance boost that is claimed in [5] was not validated.
I expected QANet to perform much better in both training performance and EM/F1 score evaluation than BiDAF. This is in big contrast with the finding of the original paper, where QANet equals the peak F1 score of BiDAF after one-fifth of the training time and trained at least 3x faster. I believe this may be due to two reasons: a) my implementation is sub-optimal and/or b) data augmentation which was not implemented due to lack of time.



	Model Id	Model details	DEV				TEST	
			EM (%)	F1 (%)	AvNA (%)	Epochs	EM (%)	F1 (%)
BiDAF	1	BiDAF - baseline	56.48	59.75	66.83	30	*	*
	2	+ Char Emb + Attention	61.89	65.11	71.06	20	*	*
	3	+ with SELU activation	62.91	66.01	71.98	30	*	*
	4	+ GRU (in RNN Encoder)	62.53	66.12	72.22	15	61.640	64.937
	5	+ Modified Conv layer with kernel_size=3	64.07	67.48	73.47	15	*	*
	6	+ 2x Conv1D layers, +Batch_Norm, + Swish	**67.03**	**70.1**	**74.83**	15	65.309	68.781
QANET	7	QANET - baseline (n_conv=1, n_heads=1, hidden=96, bs=32)	56.16	59.41	66.02	18	*	*
	8	QANET - (n_heads=8, hidden=128, bs=24)	**63.85**	**67.19**	**73.21**	15	*	*

Figure 3: Model results at each implementation level

6 Analysis

Here are some randomly picked questions where the SA-BiDAF++ model did not answer correctly:

Question: What natives were displaced by British takeover in Florida?
Context: For many native populations, the elimination of French power in North America meant the disappearance of a strong ally and counterweight to British expansion, leading to their ultimate dispossession. The Ohio Country was particularly vulnerable to legal and illegal settlement due to the construction of military roads to the area by Braddock and Forbes. Although the Spanish takeover of the Louisiana territory (which was not completed until 1769) had modest repercussions, the British takeover of Spanish Florida resulted in the westward migration of tribes that did not want to do business with the British, and a rise in tensions between the Choctaw and the Creek, historic enemies whose divisions the British at times exploited. The change of control in Florida also prompted most of its Spanish Catholic population to leave. Most went to Cuba, including the entire governmental records from St. Augustine, although some Christianized Yamasee were resettled to the coast of Mexico.
Answer: Choctaw and the Creek

Prediction: N/A
Comment: Maybe the model could not answer this question because the "rise in tensions" does not have the direct meaning of being displaced. Moreover, the "westward migration of tribes" which means displacement, does not infer to a particular tribe. One could argue that this is a bit hard even for humans unless someone reads the question a couple of times.

Another interesting example is the following:

Question: What is the largest city in Carpathia?
Context: Warsaw (Polish: Warszawa [var´ava] (listen); see also other names) is the capital and largest city of Poland. It stands on the Vistula River in east-central Poland, roughly 260 kilometres (160 mi) from the Baltic Sea and 300 kilometres (190 mi) from the Carpathian Mountains. Its population is estimated at 1.740 million residents within a greater metropolitan area of 2.666 million residents, which makes Warsaw the 9th most-populous capital city in the European Union. The city limits cover 516.9 square kilometres (199.6 sq mi), while the metropolitan area covers 6,100.43 square kilometres (2,355.39 sq mi).
Answer: N/A
Prediction: Warsaw
Comment: Here the model makes the inference that Carpathia is a region or are in Poland and therefore predicts that Warsaw is the largest city, as indicated in the context. It is clear that the model does not have common sense logic to reason that Warsaw is located 300 kilometer from Carpathian Mountains and also that Carpathia is a mountain region covering many countries and cities. This is an example in which the model should have answered "Non Answer" since the context does not provide the answer. Maybe by embedding additional knowledge e.g. via a knowledge base the problem in this example could be addressed.

Another interesting example from the QANet model implementation is the following:
Question: What are the secondary sources of primary law?
Context: European Union law is a body of treaties and legislation, such as Regulations and Directives, which have direct effect or indirect effect on the laws of European Union member states. The three sources of European Union law are primary law, secondary law and supplementary law. The main sources of primary law are the Treaties establishing the European Union. Secondary sources include regulations and directives which are based on the Treaties. The legislature of the European Union is principally composed of the European Parliament and the Council of the European Union, which under the Treaties may establish secondary law to pursue the objective set out in the Treaties.
Answer: regulations and directives
Prediction: N/A
Comment: This is an easy case in which the model did not perform well. The answer is in the same sentence as part of the question and therefore it should have predicted the answer. Maybe by capturing a longer context of the embedding could improve this problem.

7 Conclusion

This project explores a variety of methods for Q&A task on SQuAD 2.0. It gave the opportunity to gain hands-on experience on implementing advanced deep learning architectures from scratch.
The first experiment was performed with the baseline BiDAF model and improved its performance by adding character-level embedding, Attention on the embedding, context-to-context self-attention, GRU and Swish activation function. Second, QANet achieved higher performance than the BiDAF baseline. However it has not achieved the training boost and prediction of the original paper, maybe due to not implementing data augmentation.
Overall, the experiments and analysis demonstrated the power of self-attention mechanisms in deep learning models for question answering systems. These improved significantly the baseline BiDAF model giving the best F1 (70.1%) and EM(67.03%) scores on the dev set and F1 (68.78%) and EM(65.31%) scores on the test set, coming 4th at the leaderboard. That is a clear indication that architecture finetunings and optimizations on non-PCE models can achieve a great performance, but maybe not easy to reach the performance of PCE and/or Ensemble models. Future work can involve the implementation of the data augmentation in QANet and ensembling both SA-BiDAF++ with QANet to achieve higher EM/F1 scores.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[2] https://rajpurkar.github.io/SQuAD explorer/. The stanford question answering dataset.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

[5] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 2018.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[7] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[8] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

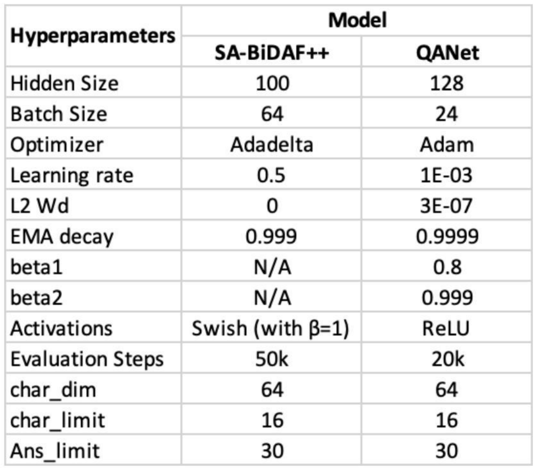
[9] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. *arXiv preprint arXiv:1710.10723*, 2017.

[10] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

[11] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

A Appendix

A.1 Hyperparameters



Hyperparameters	**Model**	
	SA-BiDAF++	**QANet**
Hidden Size	100	128
Batch Size	64	24
Optimizer	Adadelta	Adam
Learning rate	0.5	1E-03
L2 Wd	0	3E-07
EMA decay	0.999	0.9999
beta1	N/A	0.8
beta2	N/A	0.999
Activations	Swish (with β=1)	ReLU
Evaluation Steps	50k	20k
char_dim	64	64
char_limit	16	16
Ans_limit	30	30

Figure 4: Hyperparameters of SA-BiDAF++ and QANet

Note: QANet model refers to Model-id 8 as per Figure 3.
SA-BiDAF++ model referes to Model-id 6 as per Figure 3.

A.2 Training Performance

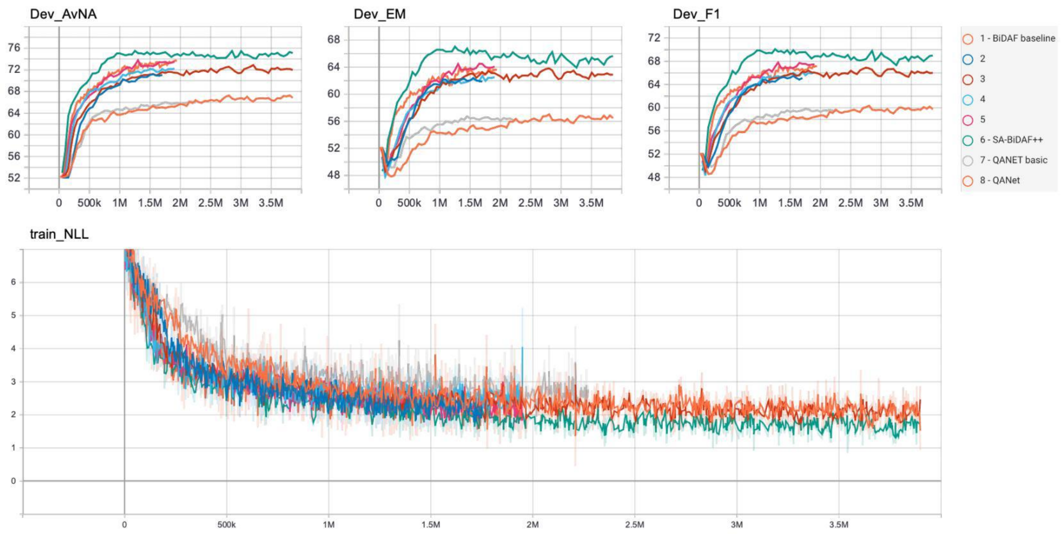


Figure 5: Training performance graphs of all Model-Ids