Question Answering on SQuAD 2.0

Boxiao Pan
Electrical Engineering department
Stanford University
bxpan@stanford.edu

Gael Colas
Aeronautics and Astronautics department
Stanford University
colasg@stanford.edu

Shervine Amidi
Computational and Mathematical Engineering department
Stanford University
shervine@stanford.edu

Abstract

The task of Reading Comprehension can be described as follows: from a context (passage / paragraph) and question pair, the goal is to answer the question using the information from the context. This paper aims at solving the sub-task of Extractive Question Answering. In this framework, the answer is a segment of the context (span). Our goal is to achieve good performance on the updated version of the Stanford Question Answering Dataset (SQuAD 2.0) without the use of Pretrained Context Embedding (PCE). In this paper, we built our QA system on top of the Bi-Directional Attention Flow (BiDAF) [1] model, and further explored several ideas to improve on this baseline. We specifically considered existing approaches to enrich the word embedding (including character embedding and tag features) and incorporated a co-attention mechanism. Moreover, we explored several novel ideas such as a distance-aware loss and a sliding-window based segment aggregation. We were able to improve the baseline EM, F1 score from 55, 58 to 61, 64, respectively. For the methods that did not work well, we carry out in-depth analysis and discuss ways to improve these approaches if we had more time.

1 Introduction

In 2016, Rajpurkar et al. built a dataset tailored for the task of Extractive Question Answering: the Stanford Question Answering Dataset (SQuAD 1.1) [2]. State of the art algorithms using the pretrained language representation model called BERT (which stands for Bidirectional Encoder Representations from Transformers [3]) now achieve past human performances on this dataset. However, one assumption that made this dataset simpler was that all the questions could be answered using the context paragraph. Our model is trained on the updated version of the SQuAD dataset, namely SQuAD 2.0 [4]. This version includes unanswerable questions, which make the problem more challenging. To achieve good performance on SQuAD 2.0, our model must not only answer questions correctly when possible, it also has to abstain from answering when no answer is provided in the passage. All the 20 best performing models on the SQuAD 2.0 leaderboard use BERT in one form or another.Pretrained Contextual Embeddings (PCE) in general provide an indisputable performing edge. Thus in this paper, we followed Herb Simon's recommendation to "go somewhere else" and investigate other creative avenues of improvement. We first tried several published ideas to boost the sentence encoder, including character embedding, word-level tag features as well as co-attention mechanism. Then we explored some novel ideas. We designed a new distance-aware loss to penalize more predictions that are further from the ground-truth than those close to the true locations. Finally, motivated by a similar idea in the computer vision community [5] and another

coarse-to-fine QA [6] approach, we devised a novel sliding-window based segment aggregation method to tackle the long-term dependency problem.

With all our workable methods, we are able to improve the baseline EM, F1 score from 55, 58 to 61, 65, respectively. During the experiments, we were also surprised to find that several intuitively sound ideas did not quite work out. For these methods, we provide our analysis as well as steps and thoughts on how we can improve them in future works.

2 Related work

Our baseline is based on the Bi-Directional Attention Flow (BiDAF) [1] architecture. This model was the previous state-of-the-art on the original SQuAD 1.0 dataset before the upcoming of PCE. The key insight of this model was to use a bi-directional attention flow mechanism to obtain a query-aware context representation. The encoding layer uses both GloVe word embedding and a character-level embedding [1] to represent words.

To extend on this baseline, we considered ways to enrich this word embedding. This idea was inspired by the work of Chen et al. [7] on the Attentive Stanford Reader++ model. They showed that using clever features can drastically improve performance. Their model added among others Part Of Speech (POS) and Named Entity Recognition (NER) tags to the previous word embeddings.

An important notion in the QA community is the co-attention idea [8], which calculates a co-attention score between input context and question by looking at its counterpart. In practice, it first leverages a co-attention encoder to produce a question-aware context representation, and then sends it into a dynamic point decoder to iteratively predict the start and end index of the answer span. We adopted this idea and developed a co-attention QA model.

In the video understanding field, one successful approach [5] is to first segment an input video into several clips, then model each clip and finally aggregate the scores for all the clips. This motivated us to try to adapt this idea to QA. We first slide a window over the input context and generate a start and end index probability distribution for each window and question pair, then we do an aggregation over all these distributions to obtain the final output. There is a similar idea [6] in the QA domain that leverages a network to first extract key sentences from the input context, then uses traditional LSTM to process them. The main difference between this approach and ours is that we don't depend on another separate network. Instead, we directly operate on the raw context, thus preserving more information.

3 Approach

3.1 Baseline

Our baseline is the adapted BiDAF [1] implementation from Chris Chute. It was provided by CS224N teaching staff in the starter code. Starter code repository: `https://github.com/chrischute/squad`.

3.2 Character-level word embedding

Our first modification to the baseline was to reproduce the original BiDAF architecture: each word embedding is built by concatenating a character-based word embedding to the pretrained GloVe word embedding.

Our Character Embedding layer follows the specification from [1]. Each word is represented as a list of character indices. Then characters' indices are embedded into 50-dimensional vectors. These are fed into a 1D Convolutional layer, the character embedding size being the number of input channels and the word embedding size being the number of output channels. The outputs of the CNN are max-pooled over the entire width to obtain a fixed-size vector for each word.

The character-level word embedding is then concatenated to the GloVe word embedding to produce a more powerful word embedding.

3.3 Tag features

Chen et al. showed that using clever features can drastically improve performance [7]. Like they did in their model, we leveraged tag features that capture some properties of the word in the context of the sentence. Following their recommendations, we used both Part Of Speech (POS) and Named Entity Recognition (NER) tags as new input features. The tag classes were first embedded as one-hot vectors. Then after training the initial model, we unfroze the one-hot encoding to learnable tag classes' embeddings. This further improved performance.

We also added Term Frequency as a new word feature. But contrary to Chen et al. who directly used the (normalized) Term Frequency (TF), we used the (normalized) Term Frequency-Inverse Document Frequency (TF*IDF). For a word w in a context c, the corresponding weight $W_{w,c}$ is defined as:

$$W_{w,c} = TF_{w,c} \ln(N_{train}/DF_w)$$

Where $TF_{w,c}$ is the (normalized) term frequency of word w in the context c, N_{train} is the size of the train corpus (number of train examples), and DF_w is the number of documents containing the word w. The idea is that words such as the article "the" can have a very high term frequency while being meaningless for the prediction. But such words will likely be present in every document, leading to N_{train}/DF_w 1 and the inverse document frequency. The TF*IDF will be small in the end. That's why TF*IDF weight is a better estimate of the potential usefulness of a word than the standard TF.

Finally we also added the Exact Match (EM) indicator that indicates whether or not a context word appears in the question and vice-versa for a question word. All these features were concatenated to the previous embedding.

3.4 Index distance-aware loss

Here, we also explore a new way of quantifying our happiness regarding the predicted output of questions that have an answer. From a qualitative perspective, we see that predicting an index p_{pred} that is 10 words away from the true index p_{true} is worse than if it is only 2 words away. We would like to include a loss term $\mathcal{L}_{\text{ind}}$ penalizing the distance between predicted and true indices in the total loss function.

One might be tempted to include a term of the form :

$$\mathcal{L}_{\text{ind}} = ||\arg\max(p_{\text{pred}}) - y_{\text{true}}|| \tag{1}$$

where $||\cdot||$ is a given norm. However, we see that this operation will not work for us since indices are just discrete values that do not have a differentiable expression with respect to the model parameters.

We alleviate this issue by substituting the raw index comparison with a series of operations that emulate the property we want to highlight. Distributions of probability for p_{pred} that are spread out with respect to the target value y_{true} are to be more penalized than those that display high values of probability around the latter index. With that observation in mind, we introduce for each sample a mask $m(y_{\text{true}})$, function of y_{true}. The mask is of same dimension as p_{pred}. It has one key property: the component of m at position y_{true} is zero, and the remaining components of the vector increase as their indices get further from y_{true}. As a result, a measure of the *penalized dispersion* of values around y_{true} can be found through the vector:

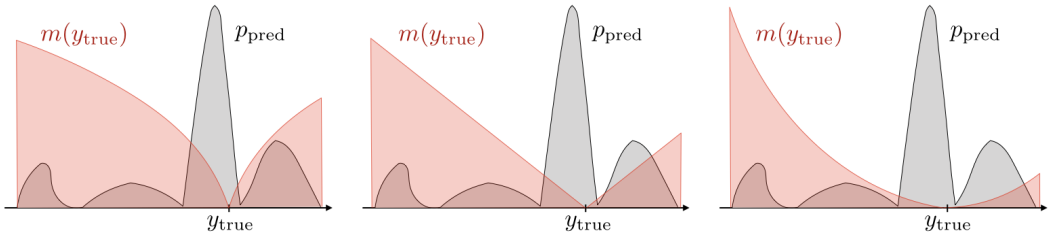
$$m(y_{\text{true}}) \circ p_{\text{pred}} \tag{2}$$

We can choose our additional loss term to be of the form

$$\mathcal{L}_{\text{ind}} = \lambda \cdot \sum_i [m(y_{\text{true}}) \circ p_{\text{pred}}]_i \tag{3}$$

where $\circ$ denotes the element-wise multiplication operation and λ is a hyper-parameter we introduce to control the importance of this penalization with respect to the other loss terms.

Figure 1 shows the shape that $m(y_{\text{true}})$ can take respectively in the 'square root', 'linear' and 'quadratic' modes.



(a) Square root (b) Linear (c) Quadratic

Figure 1: Representation of the penalization loss for the three modes considered

3.5 Segment-Based Aggregation

Here we propose a segment-based aggregation approach for question answering. This strategy is motivated by the coarse-to-fine approach in Question Answering [6] as well as the aggregation over segments method in Video Understanding [9, 5]. We slide a window over the input context with a predefined size and stride. Then for each of the selected windows, we calculate a corresponding p_{start} and p_{end}. During the aggregation stage, we select out the p_{start} and p_{end} with the highest predicted probability or the one within the distribution with the lowest entropy. We use these as the final predictions for this QA pair. This approach should be helpful in handling the problem of long-term dependency, thus include more information.

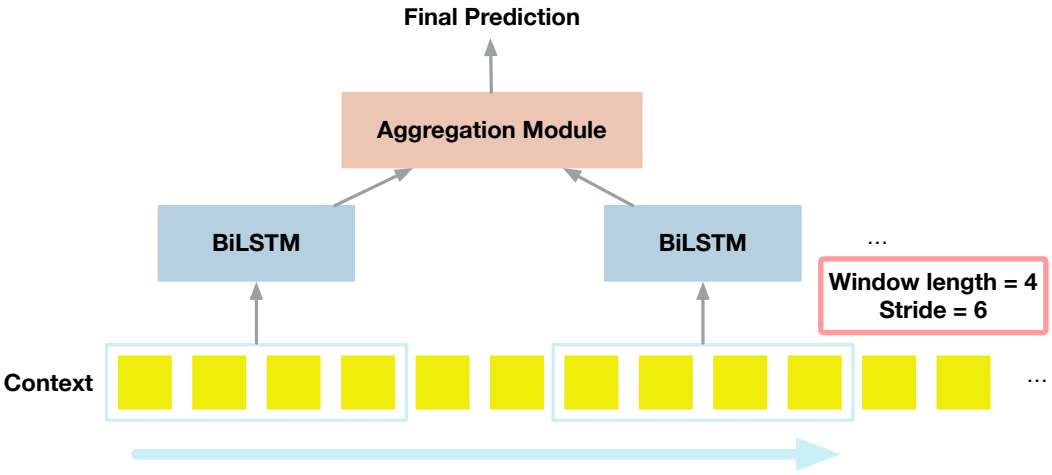


Figure 2: Illustration of our segment-based aggregation approach. For demonstration purpose, we use a window length 4 and stride 6.

3.6 Deep Residual Co-attention

We adopted the idea of co-attention from [8]. It basically computes a co-attention score between question and context in order to focus on the relevant parts of both. [10] further improves on this idea by constructing a multi-layer co-attention encoder, thus enhancing the model's ability to compose complex input representations. On the decoder side, it designs a dynamic point decoder which adopts an iterative approach to alternately predict the start and end index of the answer span. This iterative approach is proven to be able to recover from initial local optima. Next, we illustrate this approach step by step.

3.6.1 Deep Residual Co-attention Encoder

We first compute feature encodings for document (E_1^D) and question (E_1^Q) using a bidirectional LSTM as follows:

$$E_1^D = biLSTM_1(L^D) \in \mathbb{R}^{h\times(m+1)} \quad (4)$$
$$E_1^Q = tanh(W \cdot biLSTM_1(L^Q) + b) \in \mathbb{R}^{h\times(n+1)} \quad (5)$$

Where $L^D \in \mathbb{R}^{e\times m}$ and $L^Q \in \mathbb{R}^{e\times n}$ are the word embeddings for the document and the question.

Then we compute the affinity matrix between the document and question as $A = (E_1^Q)^T E_1^D \in \mathbb{R}^{(m+1)\times(n+1)}$. We apply this affinity matrix to document and question encodings to get the question and document summary vectors separately:

$$S_1^D = E_1^Q softmax(A^T) \in \mathbb{R}^{h\times(m+1)} \quad (6)$$
$$S_1^Q = E_1^D softmax(A) \in \mathbb{R}^{h\times(n+1)} \quad (7)$$

Where $softmax(A)$ computes the column-wise softmax of A.

After this, we further compute a document co-attention context as the highest level (within this layer) representation of context:

$$C_1^D = S_1^Q softmax(A^T) \in \mathbb{R}^{h\times m} \quad (8)$$

Up to this point, we have finished the calculations for a single layer. In the next layer, we first compute E_2^D and E_2^Q, taking S_1^D and S_1^Q as input, then S_2^D, S_2^Q. Finally we produce the output of the encoder like so:

$$U = biLSTM(concat(E_1^D; E_2^D; S_1^D; S_2^D; C_1^D; C_2^D)) \in \mathbb{R}^{2h\times m} \quad (9)$$

Where $concat()$ concats the inputs along the first dimension.

3.6.2 Dynamic Point Decoder

The decoder side mainly takes advantage of a bi-directional LSTM to produce hidden representations and a Highway Maxout Network (HMN) to estimate the start and end position of an answer span.

It first computes the new hidden state by taking the previous hidden state, estimation of the start and end position as inputs:

$$h_i = \text{LSTM}_{dec}(h_{i-1}, [u_{s_{i-1}}; u_{e_{i-1}}]) \quad (10)$$

Where $u_{s_{i-1}}, u_{e_{i-1}}$ are the co-attention encoding of the estimated start and end positions from U calculated by the encoder.

Then a HMN is used to compute the start (α_t) and end score (β_t):

$$\alpha_t = \text{HMN}_{start}(u_t, h_i, u_{s_{i-1}}, u_{e_{i-1}}) \quad (11)$$
$$\beta_t = \text{HMN}_{end}(u_t, h_i, u_{s_{i-1}}, u_{e_{i-1}}) \quad (12)$$

Here, t denotes the tth word in the context, and u_t is the co-attention encoding of the tth word.

Finally, we get the estimate of the start and end position for the current time step:

$$s_i = \underset{t}{\text{argmax}}(\alpha_1, \ldots, \alpha_m) \quad (13)$$
$$e_i = \underset{t}{\text{argmax}}(\beta_1, \ldots, \beta_m) \quad (14)$$

4 Experiments

4.1 Data

This paper uses the SQuAD 2.0 dataset [4]. Contrary to SQuAD 1.1, this version also includes unanswerable questions. An example input is a pair (context, question) and an output is the corresponding answer or 'No Answer' if there is none. Both the context and the question can be represented as strings of variable lengths. The answer, if there is one, can be represented as a substring of the context or equivalently as a pair of start and end word indices of the context. Figure 3 shows examples of both an answerable question and an unanswerable one with respect to a given context.

Because we added new features that were not computed for the baseline, we had to rewrite the pre-processing files. In particular, we modified the 'setup.py' file to compute the POS and NER tags offline and initialize the corresponding tag embeddings. We used spacy Language Model "en_core_web_sm" to perform this tagging. The files 'util.py', 'train.py' and 'test.py' were modified accordingly to handle these new inputs.

Context paragraph: The principle of inclusions and components states that, with sedimentary rocks, if inclusions (or clasts) are found in a formation, then the inclusions must be older than the formation that contains them. For example, in sedimentary rocks, it is common for gravel from an older formation to be ripped up and included in a newer layer. A similar situation with igneous rocks occurs when xenoliths are found. These foreign bodies are picked up as magma or lava flows, and are incorporated, later to cool in the matrix. As a result, xenoliths are older than the rock which contains them.
Question: What is something that is often torn up and included in sedimentary rock?
Ground Truth Answers: gravel ; gravel ; gravel ; gravel
Question: What do matrix components show about how magma flows?
Ground Truth Answers: <No Answer>

Figure 3: SQuAD 2.0 dev set example: a context with two associated questions, one with an answer and one without

4.2 Evaluation method

Our model is evaluated using the dev leaderboard. The two SQuAD evaluation metrics are: Exact Match (EM) score and F1 score.

- Exact Match is a binary measure (0 or 1) checking if the model outputs match one of the ground truth answer exactly ;
- F1 score is the harmonic mean of precision and recall for the answer words.

See Table 1 for the model dev performances.

4.3 Experimental details

All the models were trained for 30 epochs. The training hyperparameters are unchanged from the baseline: Adadelta optimizer with the same learning rate and exponentially weighted moving average of the model parameters during evaluation.

For the distance-aware loss, a hyperparameter search has been conducted on the penalization weight λ. We selected a value of $\lambda = 0.5$ as it made the additional loss about 1 order of magnitude smaller than cross-entropy. The tuning has been conducted this way in order to reflect the fact that our new loss is a correction term.

4.4 Results

Our model does not use Pretrained Contextual Embeddings (PCE). **We will be competing in the non-PCE division**.

Distance-aware loss. Our strategy was to apply an additional term to the loss to penalize distributions of probability that have components far from the true index. It mildly improves baseline performance.

Table 1: SQuAD 2.0 Dev Leaderboard performances

Model	EM	F1
Baseline (BiDAF)	55.1	58.2
BiDAF + square root distance loss	55.7	59.8
BiDAF + linear distance loss	56.0	59.7
BiDAF + quadratic distance loss	56.6	60.3
BiDAF + Deep Dynamic Co-attention	57.3	61.7
BiDAF + Segment-Based Aggregation	58.4	62.1
BiDAF + character embedding (Char-BiDAF)	60.6	64.0
Char-BiDAF + POS and NER tags (Tag-BiDAF)	61.4	64.9
Tag-BiDAF + Deep Dynamic Co-attention (Coattn-BiDAF)	60.7	62.5
Tag-BiDAF + EM and TF*IDF tags (Tag-ext-BiDAF)	60.9	64.0
Tag-BiDAF + Segment-Based Aggregation (Segmented-Tag-BiDAF)	**62.0**	**65.4**

We observe that the quadratic penalization performs better than the linear penalization, which in turn works better than the square root one. We can interpret this by the fact the variance of penalization coefficients is the highest when $m(y_{\text{true}}) \propto x^{\alpha}$ with α large.

Model 'Char-BiDAF'. Adding a character-level word embedding in addition to the pretrained GloVe embedding improved performance compared to the baseline. This is no surprise as using character-level embeddings allows to handle out-of-vocabulary words better: each of these words is given a unique embedding based on its morphology. Moreover, using a CNN to build the character-level embeddings allows to work implicitly at the level of morphenes.

Model 'Tag-BiDAF'. Adding hand-engineered features as recommended by Chen et al. [7] further improved the performance. Intuitively this makes sense: Part Of Speech tags allow the model to get a better understanding of the sentence structure. Name Entity Recognition makes it easier for the model to be attentive to named entities. This is especially useful for question of the form: "Who is this person?" or "What happened at this date?".

Model 'Tag-ext-BiDAF'. We tried adding a last set of features to the word embedding: word Exact Match and TF*IDF term frequency. But the performance was worse than with the previous model. Looking at the training curves on Tensorboard, we observed that the model was overfitting quickly: after 15 epochs, while the train NLL was still decreasing the dev metrics kept decreasing. We tried increasing the dropout rate to regularize but it did not help.

Dynamic Co-attention As shown in Table. 1, our co-attention technique successfully improved on the baseline. When we tried to incorporate this to our best performing word embedding, namely Tag-BiDAF, the resulting Coattn-BiDAF did not improve on the base model.

Segment-Based Aggregation With the segment-based aggregation technique we were able to improve on the baseline after tuning the window length and stride, and this further helped with our best-performing model. This demonstrates the efficacy of the segment-based aggregation approach.
The final results on the test set leaderboard are: EM = 61.2 and F1 = 65.0

5 Analysis

Dynamic Co-attention Analysis In order to gain insight into why the co-attention mechanism does not further improve on Tag-BiDAF and how much each component (in the co-attention module) affects the final performance, we carried out ablation study for each component, as shown in Table 2. Where (1) **coattn - feature** stands for the variant without concating feature encodings E into encoder output U; (2) **coattn - summary** stands for the variant without concating summary vectors S into encoder output U; (3) **coattn - context** stands for the variant without concatenating document co-attention context C into encoder output U.

From Table. 2, we can see that each part of the co-attention encoder output possesses some importance w.r.t the final performance. The document co-attention context C makes the biggest difference, which makes sense since it's the highest level of co-attention representation. To some extent, this verifies

Table 2: Ablation study on the co-attention module of Coattn-BiDAF model.

Variant	EM	F1
coattn - feature	60.4	62.4
coattn - summary	60.2	62.4
coattn - context	59.3	61.2

our implementation of the co-attention idea. Future work could try incorporating more features or using the PCE.

Segment-Based Aggregation We tuned the window length and stride hyper-parameters for the Segmented-Tag-BiDAF model, with stride fixed at three values: 10, 20, 30 (words) and varied the window length from 30 to 60, with a step of 5. We show the F1 - window length plot under these three strides in Fig. 4.

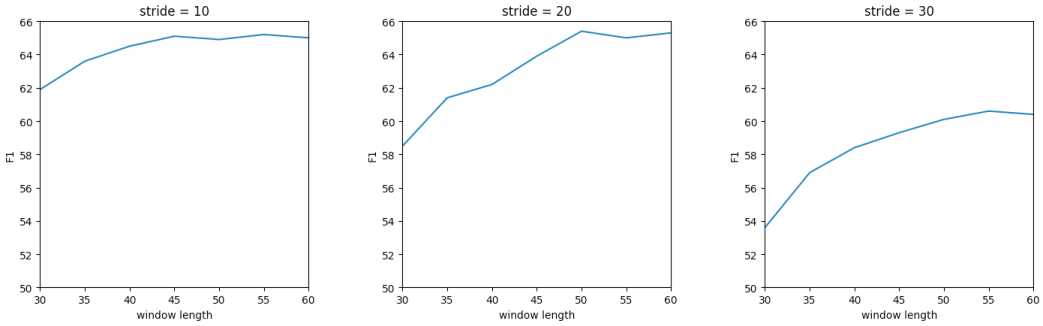


Figure 4: F1 - window length plots under different stride values using Segmented-Tag-BiDAF model.

It is not surprising to find a climbing trend in all three sub-figures, since including more information tends to always help with prediction accuracy. We also see that under a stride 10, the F1 score doesn't change much with different window lengths, while for the other two cases the curve displays a steeper slope. For stride 10 and 20, the optimal F1 scores are very close (65.2 for stride 10 and 65.4 for stride 20), and both these curves are consistently above the stride 30 case. This indicates that using a stride 10 or 20 includes a similar amount of information, while 30 is too big so it loses relevant insight. Finally, considering both accuracy and training speed, we chose the combination of window length 50 and stride 20.

6 Conclusion and future work

Our new loss improves performance metrics and shows that it can be meaningful to penalize a language model by the quality of its probability distribution. It is however important to note that performance was not improved by a statistically significant margin. One of the limitations of our approach has been the limited exploration of which penalization shape discriminates best against undesired probability distributions. With more time and computational resources, the next step would be to implement a module that learns the optimal penalization shape on the fly.

Augmenting the baseline BiDAF architecture with a more powerful word embedding significantly improved the model performance. Adding tag features and a novel segment-based aggregation approach further improves the performance. For all other methods that don't work as expected, we performed extensive analysis and gained a deeper insight into how we can refine them.

If we had more time we would like to try other regularization techniques (L1 and L2 among others) as well as playing with the projected embedding dimensionality H to be able to incorporate the last set of word features in the embedding without suffering from overfitting. If that worked, the next step would be to leverage the parsing information. By comparing the dependencies of the question and the context, we could identify which word groups are the most likely answers. Also, we would like to try the co-attention idea with PCE.

Our GitHub repository: https://github.com/ColasGael/squad-QA

Acknowledgments

We would like to thank the CS224N teaching staff for their continuing help throughout the project. We would especially like to acknowledge the hard work of Chris Chute who provided the adapted implementation of the BiDAF network (our baseline in this paper).

References

[1] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[2] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[4] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018.

[5] Bolei Zhou, Alex Andonian, Aude Oliva, and Antonio Torralba. Temporal relational reasoning in videos. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 803–818, 2018.

[6] Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. Coarse-to-fine question answering for long documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 209–220, 2017.

[7] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *CoRR*, abs/1704.00051, 2017.

[8] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

[9] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer, 2016.

[10] Caiming Xiong, Victor Zhong, and Richard Socher. Dcn+: Mixed objective and deep residual coattention for question answering. *arXiv preprint arXiv:1711.00106*, 2017.