QANet+: Improving QANet for Question Answering

Stanford CS224N Default Project Report

Moo Jin Kim
Department of Computer Science
Stanford University
moojink@stanford.edu

Christopher Wolff
Department of Computer Science
Stanford University
cw0@stanford.edu

CS224N Staff Mentor: Prerna Khullar

Abstract

In this work, we build a question answering (QA) system and apply it on the Stanford Question Answering Dataset (SQuAD) version 2.0. Our goal is to achieve strong performance on this task without using pre-trained language models. Our primary contribution is a highly performant implementation of the QANet model. Additionally, we experiment with various modifications to this architecture. Most notably, we show that modifying the output layer, such that the answer span's ending position prediction is a function of the starting position prediction, yields significant improvements over the original design. Using a QANet ensemble, we achieve an F1 score of 71.87 and an EM score of 68.89 on the unseen SQuAD 2.0 test set (rank #1 out of 100+ submissions to the test leaderboard for the IID SQuAD Track of CS224N at Stanford University, Winter 2021).

1 Introduction

QA systems aim to answer questions about a document or passage which are posed in natural language. More specifically, given a context paragraph and a question, we would like to build a system that can output a contiguous span from the context paragraph that answers the question, or predict that the question is unanswerable. Clearly, such systems have a wide array of important applications: they are used in search engines, dialogue systems, and medical applications [1].

QA is a challenging task for many reasons. For one, answering questions can require reasoning over long time horizons, as it may be necessary to combine information from different parts of the context passage. Furthermore, a QA system needs to learn subtle nuances in wording and lexical ambiguities. In the English language, small changes in wording can often change the meaning of a statement entirely. Finally, consider the passage in Figure 1. Here, the model has to correctly predict that the posed question is unanswerable given the information in the passage. To do this, the model not only has to develop an understanding of the individual facts presented, but also needs to determine that none of these facts can be used to infer an answer. Clearly, this requires a deep understanding of natural language.

Question: What title did Henry II take in the Canary Island?
Context: Bethencourt took the title of King of the Canary Islands, as vassal to Henry III of Castile. In 1418, Jean's nephew Maciot de Bethencourt sold the rights to the islands to Enrique Pérez de Guzmán, 2nd Count de Niebla.
Answer: N/A

Figure 1: An example of an unanswerable question from the SQuAD 2.0 dataset.

Currently, the field of QA is dominated by the use pre-trained language models (PLMs). PLMs such as ALBERT [2] and ELECTRA [3] are used by many of the top entries on the SQuAD 2.0 leaderboard. However, in this work, we restrict ourselves from using a PLM, with the intent of focusing on the model architecture design. Furthermore, recent years have shown a clear trend away

from recurrent models and towards attention-based models for natural language processing tasks. Hence, we chose to implement a model based on attention as well.

Among the leaderboard entries that did not use PLMs, QANet [4] was the top performer for version 1.1 of the SQuAD dataset. This motivated us to implement QANet for ourselves and explore its performance for SQuAD 2.0. The main difference between these two versions of SQuAD is that version 2.0 contains unanswerable questions whereas version 1.1 does not. Hence, the model has to learn whether it can even infer the answer from the passage. With this in mind, our contributions are as follows:

1. We re-implement the original QANet architecture from scratch.
2. We explore various modifications and extensions to this architecture.
3. We evaluate QANet's performance for SQuAD 2.0 and analyze its limitations.

To improve on the original architecture, we experiment with different model sizes and architectural changes. Interestingly, we find that increasing the size and/or depth of the network does not improve performance. Instead, our most significant improvement comes from modifying the output layer such that the prediction of the end token is conditioned on the prediction of the start token.

2 Related Work

In order to infer an answer, QA systems need to compute interactions between the context and the question. Bi-Directional Attention Flow (BiDAF) [5] introduced the idea of *bi-directional attention*. The key idea is that attention should not only flow from the question to the context, but also from the context to the question. This allows the model to build question-aware representations of the context. The BiDAF model then uses a long short-term memory network (LSTM) [6] to transform this representation into a span prediction.

Then, after the Transformer [7] popularized the use of deep attention-based networks, the QANet model attempted to do without recurrence entirely, and proposed to use only self-attention and convolution operations. The advantage of this approach is that the output no longer needs to be generated sequentially, and so forward computations could be parallelized much more easily. This resulted in training speedups of 3x to 13x compared to state-of-the-art recurrent models at the time, which allowed the authors to train bigger models and effectively train on more data by using back-translation. As a result, the QANet paper surpassed all existing approaches to QA at the time. Since the paper does not include an implementation by the authors, we believe that our re-implementation, which has a strong performance for SQuAD 2.0, will be a useful resource to researchers that would like to build on this work in the future.

Typically, the start and end tokens are predicted independently. However, it seems reasonable to first predict the start token, and only then predict the end token, taking our own start token prediction into account. The Match-LSTM with Answer Pointer [8] model does this. It uses an LSTM that attends to a question-aware context representation, and runs for exactly two time steps. The attention weights for the first time step are used to predict the start token, and the attention weights for the second time step are used to predict the end token. We do not make use of this idea directly, but we use it as inspiration to design our own output layer.

3 Approach

3.1 BiDAF

Baseline We use the BiDAF model with pre-trained 300-D GloVE word embeddings [9], which is provided by CS224N staff, as the baseline for our project. Its architecture is described in the BiDAF paper [5].

BiDAF with character embeddings We improve on the baseline model by incorporating learnable 200-D character-level embeddings, which are concatenated with the word vectors and fed through a fully-connected layer. Aside from this addition, the model is identical to the baseline BiDAF model.

3.2 QANet

We built the original QANet model *f*rom scratch in PyTorch [10]. We only make use of the provided project skeleton's context-query attention layer and an open-source implementation of standard building blocks, namely sinusoidal positional encoding and depth-wise separable convolutions[1]. We implemented everything else using components from the PyTorch standard library.

Input embedding layer We first process the strings for the question and context into sequences of tokens representing the words and characters. Then, we combine a pre-trained GloVe embedding [9] for each word with the learnable character embeddings for each character in the word as follows: We concatenate the embeddings of the first 16 characters of the word (padded with zeros if the word is shorter than 16 characters), resize each embedding to size 128 (the hidden state size) by convolving them with 1-D kernels of size 5, and take the max across the characters dimension to get a vector representation of each word. We also resize the word embedding to size 128 by convolving them with 1-D kernels of size 1, then concatenate the result with the character embeddings, and finally resize the resulting 256-D output to 128-D via another 1-D convolution layer. Lastly, we apply a two-layer highway network [11] on the embedding vector. This computation is performed for every word in the context and the question.

Encoder blocks The core component of QANet is the encoder block, illustrated on the right side of Figure 2. It consists of a sinusoidal positional input encoding [7], followed by several convolution layers, a self-attention layer with 8 heads, and a feed-forward layer. Each of these layers uses layer normalization [12] for its input to stabilize the hidden state dynamics, and a residual connection to its input to allow for better gradient flow through the network. The encoder block is used in two places throughout QANet: once to process the input embeddings, and a second time to process the bi-directional attention outputs at the core of the model, as shown in Figure 2. To process the input embeddings, we use a single embedding encoder block containing 4 convolutions with a kernel size of 7, and 8 multi-head attention heads. For the core of the model, we use 3 stacks of 7 model encoder blocks each, with 2 convolutions per block and a kernel size of 5. We share weights between the input encoder of the question and context, and between each of the 3 encoder block stacks at the model core.

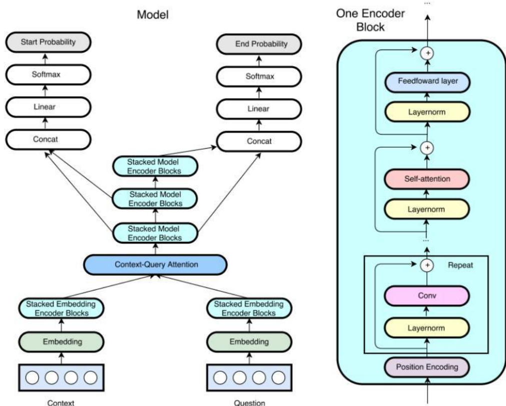

Figure 2: The QANet architecture. Source: [4]

Context-Query attention QANet uses the same bi-directional context-query attention mechanism as BiDAF. For brevity's sake, we refer you to the BiDAF paper [5] for a detailed description.

Output layer Let M_0, M_1, and M_2 denote the outputs of the three model encoder stacks, respectively. QANet computes the probability of each position in the context being the start and end position of the answer span as

$$p_{\text{start}} = \text{softmax}(W_0[M_0; M_1]), \qquad p_{\text{end}} = \text{softmax}(W_1[M_0; M_2]), \tag{1}$$

where W_0 and W_1 are learnable parameters.

[1] Implementations of the positional encoding and depth-wise separable convolutions are taken from https://github.com/BangLiu/QANet-PyTorch.

Training The model maps from a (query, context) pair to two probabilities distributions: one for the predicted start of the answer span and one for the predicted end. Inputs with no answer have the label (0, 0). Our loss function is the sum of the negative likelihood of the predicted start and end position distributions. That is, given a gold answer span of (i, j) and model parameters θ, we have

$$\mathcal{L}(\theta) = -\log p_{\text{start}}(i;\theta) - \log p_{\text{end}}(j;\theta). \tag{2}$$

Similar to the QANet paper, we use dropout [13] to stochastically thin the width of layers, and stochastic depth layer dropout [14] within each encoder layer, where each sublayer l drops out with a probability of $\frac{l}{L}(p_{drop})$, where L is the last sublayer in the encoder layer and p_{drop} is a hyperparameter. At inference time, we use the full network, but adjust for the larger expected value of the activations by scaling them by the survival probabilities.

Inference At inference time, we first use the model to generate the two probability distributions p_{start} and p_{end}. Then, we search for the span that maximizes $p_{\text{start}}(i) \cdot p_{\text{end}}(j)$ subject to $i \leq j$ and $j-i+1 \leq L_{\text{max}}$, where L_{max} controls the maximum length of a predicted answer. In our experiments, we use $L_{\text{max}} = 15$. The idea behind this is that we would generally like to predict reasonably short spans, and not just blindly use argmax(p_{start}) and argmax(p_{start}) as our predictions.

3.3 QANet Extensions

After implementing and testing the original QANet architecture, we also explore several model variations in isolation.

Parameterized positional encodings Instead of using fixed sinusoidal positional encodings, we allow the model to learn the encoding parameters for itself. To do that, we simply initialize a parameter matrix of size $L \times h$, where L is the maximum sequence length of a context passage, and h is the hidden size of the model. In the forward pass, for an input sequence of length l, we simply index the first l rows of the parameter matrix, and add them to the input sequence representation.

Conditional output layer (1) Instead of computing p_{start} and p_{end} independently, we compute p_{end} as a function of p_{start}. We explore two **original** lightweight implementations of this conditioning, which we label as "Conditional output layer (1)" and "Conditional output layer (2)." For the first version, we combine the hidden states of the answer span's starting position distribution with the hidden states of the ending position distribution, modifying the output layer computations described previously as follows:

$$A = W_0[M_0; M_1], \tag{3}$$

$$B = W_1[M_0; M_2], \tag{4}$$

$$p_{\text{start}} = \text{softmax}(W_2 A), \qquad p_{\text{end}} = \text{softmax}(W_3[A; B]), \tag{5}$$

where $M_0, M_1, M_2 \in \mathbb{R}^{\text{h}\times l}$ are the model encoder stack outputs, and $W_0, W_1 \in \mathbb{R}^{h\times 2h}$, $W_2 \in \mathbb{R}^{1\times h}$, and $W_3 \in \mathbb{R}^{1\times 2h}$ are learnable parameters. The motivation is to make information contained in the hidden states of the starting position's distribution available to the model when predicting the ending position's distribution.

Conditional output layer (2) In our first version of conditional output, we realized that neither the logits distribution $W_2 A$ nor the final probabilities distribution p_{start} after applying the softmax is used in the calculation of p_{end}; instead, the only additional information comes from the hidden state A, which means that the model does not incorporate information learned from the linear projection via W_2. We therefore propose our second original implementation of conditional output, which *does* fully incorporate this information. The calculations are as follows:

$$L = W_0[M_0; M_1] \tag{6}$$

$$A = W_1(L \odot [M_0; M_1]) \tag{7}$$

$$B = \text{ReLU}(W_2[M_0; M_2]) \tag{8}$$

$$p_{\text{start}} = \text{softmax}(L), \qquad p_{\text{end}} = \text{softmax}(W_3[A; B]), \tag{9}$$

where $M_0, M_1, M_2 \in \mathbb{R}^{\text{h}\times l}$ are the model encoder stack outputs, and $W_0 \in \mathbb{R}^{1\times 2h}$, $W_1, W_2 \in \mathbb{R}^{h\times 2h}$, and $W_3 \in \mathbb{R}^{1\times 2h}$ are learnable parameters. The starting position's logits distribution L is

used to weight the hidden states representing the span's starting position distribution, incorporating the information learned by the projection via W_0 (the use of logits rather than probabilities preserves negative values in the hidden states). Positions in the context with high probability of being the start of the span will be more highly activated than others, and the model can utilize this information when predicting the end of the span. This layer is also illustrated in Figure 3.

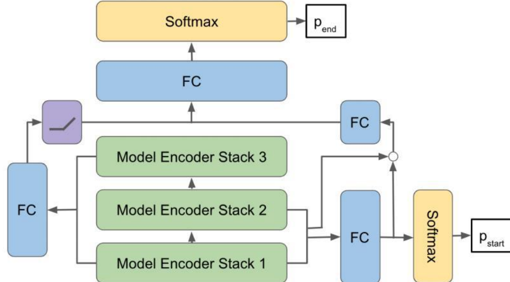

Figure 3: Our conditional output layer design.

Upsized QANets We explore five larger variants of the QANet to observe how model complexity affects performance. The first variant, "QANet + Deeper Model Encoder (1)," contains 9 encoder blocks instead of 7 for each of the 3 model encoder layers. "QANet + Deeper Model Encoder (2)" is the same, except that it contains 11 model encoder blocks per layer instead of 7. "QANet, 1.5x Larger (1)" increases the hidden size by 1.5x to 192 instead of 128 and uses 2 encoder blocks instead of 1 in the embedding encoder. "QANet, 1.5x Larger (2)" uses the exact same configuration but with increased regularization; we discuss details in the Experiments section. Finally, "QANet, 2x Larger" uses a hidden size of 200, 2 embedding encoder blocks, and 11 model encoder blocks per layer.

Ensemble We assemble a final ensemble of 9 QANet models: 6 copies of QANet with "conditional output layer (2)" (each trained with a different random seed), 1 base QANet model, 1 QANet with "conditional output layer (1)", and 1 QANet with "deeper model encoder (1)." These are the models that achieve the highest F1 scores on the SQuAD 2.0 development set, as we will show in the Results section. For each question, we choose the answer span with the largest number of votes, where each of the 9 models gets 1 vote. We weight each vote to break ties: the model with highest development F1 score has a weight of 1.00, the second-best has weight 0.99, the third-best has weight 0.98, and so on.

4 Experiments

4.1 Data

We use version 2.0 of the Stanford Question Answering Dataset [15], which contains both answerable and unanswerable questions, to train and test our models. The training set consists of the roughly 130K examples in the official SQuAD 2.0 training set. The development set and test set we use are roughly equally sized halves of the official SQuAD 2.0 development set, with about 6K examples each, since the official test set is hidden from the public.

4.2 Evaluation method

We primarily use the F1 score for evaluation, using this to compare the performance of different models. We also note the exact match (EM) and answer vs. no-answer (AvNA) scores.

4.3 Experimental details

BiDAF We run our baseline BiDAF model with a hidden state size of 100, batch size of 64, EMA decay of 0.999, dropout rate of 0.2, and learning rate of 0.5, as in the original BiDAF paper [5]. We run our new BiDAF model with 200-D character embeddings using the same configuration. We train the baseline model for about 30 epochs in about 11 hours on a Tesla K80 GPU; we train the model with character embeddings in about 5 hours on a Tesla V100 GPU.

QANet We run our implementations of the base QANet, QANet with parameterized position encodings, and QANet with a conditional output layer with the hyperparameter configuration discussed in the Approach section. For regularization during training, we use a L2 weight decay of 3×10^{-7}, a general dropout rate of 0.1 after every 2 layers, and stochastic depth layer dropout [14] within each encoder layer, as discussed previously. This model configuration is used in the original QANet paper [4]. We train each QANet model variant until convergence (roughly 25 epochs) in about 35 hours on a Tesla K80 GPU, or about 10 hours on a Tesla V100 GPU.

Upsized QANets For "QANet + Deeper Model Encoder (1)," "QANet + Deeper Model Encoder (2)," and "QANet, 1.5x Larger (1)" we use the same hyperparameter setup as the base QANet. We realize that "QANet, 1.5x Larger (1)" achieves lower training negative log-likelihood (NLL) but higher development NLL than the base QANet, so we run "QANet, 1.5x Larger (2)," which is an identical copy except with more regularization: an L2 weight decay of 3×10^{-6} instead of 3×10^{-7}, and a general dropout rate of 0.2 instead of 0.1 after every 2 layers. After seeing that this increased regularization hindered both training and development NLL, we run the final upsized QANet, "QANet, 2x Larger (2)," with an L2 weight decay of 6×10^{-7} and a general dropout rate of 0.15.

4.4 Results

Table 1: Performances of various models on the SQuAD 2.0 development set.

Model	F1	EM	AvNA
BiDAF (Baseline)	61.72	58.43	68.53
BiDAF + Char. Embeddings	**64.62**	**61.12**	**71.25**
QANet	70.01	66.26	76.27
QANet + Conditional Output Layer (1)	70.15	66.41	76.52
QANet + Conditional Output Layer (2)	**71.54**	**67.67**	**77.63**
QANet + Deeper Model Encoder (1)	70.03	66.31	76.24
QANet + Deeper Model Encoder (2)	69.02	65.23	74.88
QANet, 1.5x Larger (1)	69.63	65.90	76.00
QANet, 1.5x Larger (2)	69.80	66.21	75.58
QANet, 2x Larger	69.28	65.57	75.28
QANet, Parametric Position Encodings (stopped early)	63.94	60.66	70.16
QANet Ensemble	**74.17**	**71.03**	**79.06**

The results of all of our individual model experiments on the SQuAD 2.0 development set are shown in Table 1. The addition of learnable character embeddings to BiDAF improves performance significantly upon the baseline model, as the augmented model can use the finer-grained, character-level representations of words and subwords to better learn the meaning of different words and thus more accurately predict which words belong in the answer spans. Better yet, our implementation of the base QANet model far outperforms even this improved version of BiDAF, achieving 70.01 F1 and 66.26 EM.

Among the individual models, the QANet with our second version of conditional output performs best, which is sensible since the end of an answer span depends on the beginning, and it is more reasonable to condition the end based on the start rather than predicting them entirely independently. We see that the first version of conditional output does not improve performance as significantly, suggesting that the information contained within the starting position's logits/probabilities distribution is significant when predicting the ending position, which is reasonable since the final linear projection that generates this logits distribution is the decision-maker that maps model encoder outputs to starting position probabilities.

Surprisingly, the five upsized QANet variants did not achieve expected performance, suggesting that adding more helpful features such as conditional output to the model architecture is much more significant than increasing the QANet's complexity.

Further, the parametric position encodings perform poorly. The F1 score appears to plateau early at around 64, so we decided to discontinue the experiment. Our intuition is that perhaps the context

sequences are too long to learn meaningful positional encodings that generalize well. For the sake of time, we did not explore this idea any further.

Lastly, we see that the final ensemble model of nine QANet models, which we discussed in the Approach section, achieves significant gains in development F1, EM, and AvNA scores. The ensemble also achieves high performance with an **F1 score of 71.87** and **EM score of 68.89** on the test set, ranking #1 on the test leaderboard. We attribute the difference between the development and test performance to possible minor distributional shift, as well as slight overfitting to the development set when we assemble the ensemble based on highest F1 scores on the development set.

5 Analysis

AvNA Figure 4 below reveals how well our QANet ensemble learns to predict an answer for answerable questions and no answer for unanswerable questions. The ensemble predicts an answer more often than no answer even though the development set has more unanswerable questions than answerable ones, and its true positive rate is higher than its true negative rate. This indicates one weakness of the model: it has some difficulty with learning when there is no answer to a question and often forces an answer to an unanswerable question. Having a separate head that is trained to predict the presence or lack of an answer might help mitigate this issue.

		Ground Truths		
		Answer	No Answer	Total
Predictions	Answer	2386	784	3170
	No Answer	462	2319	2781
	Total	2848	3103	5951

TPR	TNR	FPR	FNR
83.78%	74.73%	25.27%	16.22%

Figure 4: Confusion matrix of the ensemble model's predictions on the SQuAD 2.0 development set, followed by rates derived from the matrix (e.g., true positive rate, true negative rate, etc.).

Performance breakdown Table 2 below shows how the ensemble model performs on different types of questions. (The "Other" category includes questions that contain more than one of the six question words.) The ensemble performs best for "When" questions, most likely due to the relative ease in predicting which numbers/words represent a point in time. It performs worst for "Why" questions, which is sensible since these questions are more difficult to answer compared to questions such as "Who" and "Where"; proper nouns, for instance, provide significant hints for the latter but not the former, which involves the more complex learning of intent rather than rote fetching of entities or locations.

Table 2: Performance of the ensemble model on different types of questions in the development set.

Question Type	Who	What	When	Where	Why	How	Other	Overall
Count	608	3522	434	251	87	556	493	5951
F1	73.76	74.47	81.49	69.93	68.06	74.92	68.47	74.17
EM	71.38	71.32	80.87	65.73	58.62	70.86	64.90	71.03
AvNA	76.97	79.44	85.25	76.49	74.71	79.13	75.45	79.06

Exploration of individual questions To get a better sense of where our model is making mistakes, we look through individual questions ourselves and try to identify common sources of error. For one, we noticed that our model tends to be biased towards giving numeric answers when the question starts with "How many [...]" and one or more numbers appear in the context. The model often predicts one of those numbers to be the answer, even if the question is unanswerable. Similarly, for a question that begins with "What percentage [...]," our model predicted an incorrect answer with a percentage sign. This suggests that the model learns to do high level pattern matching for some questions,

without actually understanding the question or context. We also noticed that our model struggles with questions that have complicated grammatical structure. For instance, one question asks: "Issues dealt with at Westminster are not ones who is able to deal with?" This question uses passive form and negation and is thus quite complex. Most likely, the training set does not contain many such questions, and thus it is difficult for the model to interpret.

Conditional output layer Next, we attempt to answer why adding the conditional output layer improved the performance of the model. One of our hypotheses is that it helps the model output more coherent span predictions than the independent approach, in which p_{start} and p_{end} are predicted individually. To see whether this is true, we compare our base QANet implementation to our model with the conditional output layer. For each, compute the predictions for every sample in the development set, look at the highest probability start token and the highest probability end token, and see whether these form a "reasonable" span. We define a span to be reasonable if the end token occurs no more than 15 tokens away from the start token. If our hypothesis was true, we would expect the base model to have more unreasonable predictions, in which the predicted end token comes before the start token or the two are far away from each other. It turns out that this is not the case. For the base model, about **7.1%** of predictions are unreasonable, and for the conditional model, about **7.7%** are. We believe that the difference between these two results is not significant enough to confirm or deny our initial hypothesis.

Table 3: Performance comparison between base QANet and QANet with conditional output layer on different types of questions in the development set.

Question Type	Who	What	When	Where	Why	How	Other	Overall
Base QANet F1	71.56	70.30	76.26	66.62	67.71	68.80	63.93	70.01
Cond. Out. QANet F1	70.73	71.84	78.23	66.19	68.94	73.81	65.05	71.54

Another hypothesis we investigate is whether the conditional output layer allows the model to learn more complex interactions between words within the context. We turn to the relative performance of the base QANet and the conditional output QANet on different subsets of the development set, as shown in Table 3. For most of the question types, the difference in F1 score between these two models is minor and could be attributed to random noise. However, we see that the conditioned QANet performs significantly better on "How" questions than the base QANet, with a 5-point increase in F1 score in this category. This finding suggests that the conditioned QANet indeed can better learn complex interactions among words in the context, that are useful in predicting answers to more difficult questions.

6 Conclusion

In summary, we present an implementation of QANet that achieves strong results on the SQuAD 2.0 dataset. We show that increasing the model size and adding parameterized positional encodings does not improve the model's performance. Additionally, we are able to improve the model by adding a novel conditional output layer and using an ensemble. So far, we have only explored the viability of our output layer for a single dataset. It remains to be seen whether our the performance improvement carries over to other tasks, as well. In the future, we are interested to see how QANet synergizes with embeddings from pre-trained language models. Furthermore, we would like to see whether our output layer architecture works well for other QA datasets, such as NewsQA [16], as well. Finally, we are curious to explore the viability of a different label space and loss function. Instead of predicting a distribution for just the start and end token, one could learn a probability for each word in the context, that indicates the likelihood that the word is part of the answer span.

References

[1] Roser Morante, Martin Krallinger, Alfonso Valencia, and Walter Daelemans. Machine reading of biomedical texts about alzheimer's disease 1.

[2] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[3] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators, 2020.

[4] 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. *CoRR*, abs/1804.09541, 2018.

[5] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

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

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

[9] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[11] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks, 2015.

[12] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[14] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

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

[16] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. Newsqa: A machine comprehension dataset, 2017.