QAN-et al.: Exploring Extensions on QANet

Stanford CS224N Default Project

Timothy Dai
Stanford University
timdai@stanford.edu

Michelle Qin
Stanford University
mdqin@stanford.edu

Jessica Yu
Stanford University
jmyu@stanford.edu

Abstract

In this project, we implement a question answering model for the SQuAD 2.0 dataset that improves upon the performance of Seo et al.'s BiDAF baseline. We implement from scratch an improved BiDAF baseline using character-level embeddings, a high performing QANet-based architecture, and the following variations on that model: an additional module that predicts answerability, conditioned end predictions, learnable positional encodings, and relative positional encodings. With a weighted BiDAF and QANet ensemble (using learnable position encodings, conditioned end prediction variatons, and more) we achieve a significant performance increase over the baseline, reaching 3rd place on both the development set leaderboard and the test set leaderboard.

1 Key Information

Our mentor is Michi Y. We have no external collaborators nor are we sharing projects.

2 Introduction

In this project, we address the NLP challenge of question answering, one of the "strongest possible [demonstrations] of text comprehension" [1]. Specifically, in this project, we address a model's ability to produce the answer span of text in a context paragraph from the SQuAD 2.0 dataset. Recent years have seen much progress made on end-to-end models for this task, such as the Bidirectional Attention Flow (BiDAF) model by Seo et al. [2], and the main aim of this project is to implement a successful model that beats this baseline.

We first attempt to improve upon this baseline by restoring the character-level embeddings removed by CS 224N staff, which encode lexical information from morphemes and prefixes. While this achieves an improvement over the baseline, it's quickly exposed by Yu et al. [3] that recurrent neural networks suffer from slow training speeds and difficulties in learning meanings of sentences as a whole. As a response to some of recurrent neural networks' weaknesses, we implement the QANet architecture described in Yu et al., composing almost all of the described layers from scratch. As expected, we achieve significantly improved performance from the baseline.

We experiment with the following techniques with the intention of further improving upon our vanilla QANet model. The second and third item below include original contributions:

1. QANet with learnable positional encodings, in which we add a learnable positional tensor to the input of each encoder block in order to preserve positional representations.
2. QANet with an AvNA module, in which we extend the QANet architecture and loss function to include a branch that separately predicts a question's answerability. This acts as an alternative to the provided method of predicting no-answer whenever the leading OOV token maximizes joint start and end probabilities. This AvNA module is first introduced in Aubet et al. [4] but we make original contributions, specified in the Approach section.

3. QANet with conditioned end predictions, in which we condition the prediction of the end position on the prediction of the start position. We make original contributions in this section also, specified in the Approach section.
4. QANet with relative positional encodings, in which we incorporate the clipped relative positional encodings of each word's two neighboring elements into QANet's multiheaded attention module, introduced in Shaw et al. [5].

Of these modifications, adding relative positional encodings to the self-attention layer leads to the best improvement when tested in isolation. Finally, we assemble an ensemble by selecting high performing models from our repertoire of BiDAF and QANet variants to achieve our highest performing model.

3 Related Work

The Bidirectional Attention Flow (BiDAF) model by Seo et al. was significant in making progress on end-to-end models for machine reading comprehension and question answering [2]. BiDAF is notable for introducing the concept of context-query attention. When applied to a recurrent neural network (RNN), the model is able to achieve strong results on SQuAD. However, because of the recurrent nature of BiDAF (ie., BiDAF is a recurrent model that processes inputs with an added attention component to handle long-term interactions), the model is slow and expensive to train thereby resulting in high turnaround time for experiments and deploying applications.

The QANet model by Yu et al. aimed to remove the recurrent nature of the BiDAF model in order to make machine comprehension faster [3]. Instead of using RNNs, QANet uses convolutional layers to capture the local structure of the text and self-attention to capture the longer term interaction between words, borrowing heavily from Vaswani et al.'s seminal work on transformers [6]. QANet proved to be a faster model, allowing more data to be trained and greater scale to be employed. While both Vaswani et al. and Yu et al. use absolute positional encodings, specifically sinusoidal positional encodings, further work has explored relative positional encodings, introduced in Shaw et al. [5] and developed further in Dai et al. [7], which encode distances between sequence elements (ie., relative positions) in the self-attention layers of the transformer model.

One of the first attempts to handle the no-answer examples of SQuAD 2.0 were produced in Aubet et al.'s 2019 model called EQuANt [4], which adds an AvNA module to specifically predict answerability. Levy et al. [8] introduce another such technique to handle the no-answer examples of SQuAD 2.0 by prepending an out-of-vocabulary (OOV) token to the contexts, and giving no-answer labels to predictions that maximize probabilities at the OOV token.

These works show that improving model performance is an iterative process. We adopt a similar mindset in this project as we replicate each of our chosen techniques and make original contributions, testing each one in isolation.

4 Approach

4.1 BiDAF

Our baseline is Seo et al.'s BiDAF [9] with slight modifications by the CS 224N staff. Among the biggest changes made by the CS 224N staff are the removal of character-level embeddings and the addition of an out-of-vocabulary token to all contexts to accommodate SQuAD 2.0's no-answer examples, a technique introduced in Levy et al. [8]. For more details, we refer the reader to the default project handout.

4.2 BiDAF extensions

We implement from scratch two techniques to improve our BiDAF baseline.

Character-level embeddings. We implement character-level embeddings which maps each word to a high dimensional character space. In detail, we apply a 2D convolution with kernel size 3 to learnable character embeddings and max pool across the character dimension to obtain a character-level embedding for each word. We then concatenate the character-level embeddings to the word

embeddings, and pass the output through the remainder of the original model. Explained in Kim 2014 [10], running kernels over character-level embeddings helps extract more granular information about words, including morpheme information and subword information.

Coattention. We also attempt to improve our BiDAF baseline by implementing a coattention layer introduced in Xiong et al.'s 2016 paper [11] to produce second-level attention outputs, which are concatenated with the first-level context-to-query attention and passed to the model encoder layer thereafter. For the sake of brevity, we refer the reader to the default project handout Section 5.2 for the exact sequence of operations applied in this coattention layer.

4.3 QANet

We implement the QANet model architecture [3] from scratch, barring several instances where we reuse code from the BiDAF model or reference short snippets of open-source code, indicated where applicable. Given the importance of the encoder block to the QANet model, we define it first.

Encoder block. QANet's encoder block (Figure 1) borrows ideas from the transformer model [6] and is the central component of QANet.

The input representation to the encoder block is first summed with a sinusoidal positional encoding [6], which encodes a cyclical float between -1 and 1 that allows the model to retain positional information. We then feed the summed vector through a convolutional network consisting of a predetermined number of depthwise separable convolutional sublayers, followed by a multiheaded attention sublayer of 8 heads, and a final feed-forward network. We cite open-source notebooks[1] as helpful resources in the implementation of the multiheaded attention module. The multiheaded attention and depthwise separable convolutional sublayers are the key modules for language understanding.

Finally, we refer the reader to the Appendix for all remaining configurations for the encoder block. We will see below how the encoder block is used in the embedding encoder and the model encoder layers.

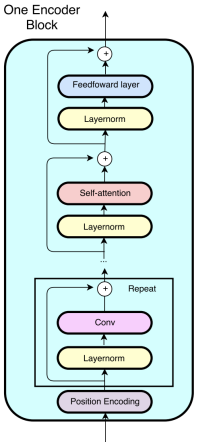

Figure 1: An encoder block, the central component of the QANet model. Source: [3]

Embedding layer. The embedding layer processes the word and character indices fed as input to the model. Like BiDAF, the QANet embedding layer uses pretrained GLoVe vectors [12], sized at $p_1 = 300$ for word representations. For character embeddings, we use randomly initialized, trainable embeddings of size $p_2 = 200$. As recommended by the QANet paper, we apply a different dropout rate to word and character-level embeddings: 0.1 for word and 0.05 for character-level. We then pass the character-level embeddings through a 2D convolutional layer followed by a max over the character dimension, and concatenate the word and character-level embeddings together. Finally, we apply a fully-connected layer to reshape our embeddings from $p_1 + p_2$ to $d_{model} = 128$, and pass this resized representation through a 2-layer highway network [13], designed to ease training for deep neural networks.

Embedding encoder layer. The embedding encoder layer processes the size d_{model} output of the embedding layer. It consists of 1 encoder block (defined above), configured with 4 depthwise convolutional sublayers each with kernel size 7. The output to this layer is fed to the subsequent context-query attention layer.

Context-query attention layer. We reuse BiDAF's context-query attention layer, with one addition. We add a fully-connected layer to project the attention layer output of hidden size $4 \cdot d_{model}$ to hidden size d_{model} for size compatibility with proceeding blocks.

[1] https://tinyurl.com/uvadlc-notebooks and https://tinyurl.com/multihead

Model encoder layer. The model encoder layer consists of a stack of 7 encoder blocks (defined above), each with 2 convolutional sublayers and a kernel size of 5. We pass the context-query attention layer output through the model encoder layer 3 times, effectively sharing weights between 3 model encoder layers. We store the 2 intermediate results and final output in 3 separate matrices, M_0, M_1, M_2.

Output layer. We concatenate M_0 and M_1 from the model encoder layer's output, project the concatenated vector to a size of 1, and softmax over its sequence-length dimension to obtain a probability distribution of start positions, p_1. We perform the same operations on M_0 and M_2 to obtain a probability distribution of end positions, p_2. We refer the reader to the Appendix for details on how loss is calculated for backpropagation and how p_1 and p_2 are discretized for predictions.

4.4 QANet extensions

We extend our vanilla QANet model with several techniques in an attempt to improve performance.

Learnable positional encodings. We try replacing the sinusoidal positional encodings with learnable positional encodings. To do so, we initialize a parameter matrix of size $(1000, d_{model})$, given that the maximum paragraph length of the model is 1000. At the start of each encoding block, we simply truncate the first dimension of the learnable matrix to match the sentence-length dimension of the input, and sum the truncated positional encodings with the input representation.

AvNA module. We explore a module introduced in Aubet et al.'s 2019 paper [4] that exclusively learns answerability. Currently, to handle unanswerable questions in SQuAD 2.0, CS 224N staff implemented a technique introduced in Levy et al. [8] that predicts no-answer when the joint probabilities of the start and end predictions are maximized at 0. However, we wish to explore another way of predicting no-answer.

Briefly, the AvNA module described in Aubet et al. takes as input the first of our 3 model encoder layers' output, and returns a float that represents the 'answerability' score of an example, with an answerable example having a ground-truth score of 1. We refer the reader to the Appendix for a more detailed outline of the AvNA module's architecture.

In order to train this module, we set the loss function according to Aubet et al., with an **original** modification:

$$l(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left[\mathcal{L}_0^{(i)}(p_0^{(i)}) + \mathcal{L}_1^{(i)}(p_1^{(i)}) + \mathcal{L}_2^{(i)}(p_2^{(i)}) \right], \tag{1}$$

where $\mathcal{L}_0^{(i)}(p_0^{(i)})$ is the answerability cross-entropy loss, $\mathcal{L}_1^{(i)}(p_1^{(i)})$ is the start prediction cross-entropy loss, and $\mathcal{L}_2^{(i)}(p_2^{(i)})$ is the end prediction cross-entropy loss. This loss function (Equation 1) allows the model to train both the traditional output layer and the new AvNA module. We refer the reader to the Appendix for more information about how this original loss equation differs from Aubet et al.'s loss and our motivation for such a change.

Furthermore, Aubet et al. do not specify how exactly to use the AvNA module in discretizing predictions, so we made further **original** contributions by experimenting with various ways to discretize the model's output using the new module's output:

Discretization Method 1. We set predictions to no-answer when the answerability probability is < 0.5.

Discretization Method 2. We set predictions to no-answer when the answerability probability is < 0.5 and the joint probability of the start and end predictions is maximized at 0.

Discretization Method 3. We set predictions to no-answer when the answerability probability is < 0.5 or the joint probability of the start and end predictions is maximized at 0.

We report the highest performing configuration in the Experiments section and compare with the results from other configurations in the Analysis section.

Conditioned end predictions. We investigate 2 **original** ways of conditioning end predictions on start predictions by modifying the output layer:

Conditioning Method 1. When predicting end positions in the output layer, we concatenate M_0 and M_2, as before, but now we also concatenate the predicted start positions p_1 to obtain the result $[M_0^{(i)}; M_2^{(i)}; p_1^{(i)}] \in \mathbb{R}^{2d_{model}+1}$. Following, we apply a fully-connected linear layer to project the concatenated matrix to a representation of end predictions p_2 with a hidden size of 1, and proceed as before.

Conditioning Method 2. We fear that the previous technique may dilute the information about start predictions in the end prediction network, as it concatenates p_1 with two large dimension matrices, M_0 and M_2. Therefore, we attempt another method where information about the start predictions are comparable in dimension to end prediction inputs. This second model obtains the end position predictions $p_{2,revised}$ as follows:

$$p_{2,revised} = W[p_1; p_2] + p_2.$$

In other words, we allow the model to first predict start and end predictions separately, obtaining prediction representations p_1 and p_2, as before. However, we give p_2 a chance to 'revise' itself after looking at start predictions p_1, by feeding $[p_1; p_2]$ through another fully-connected layer. We grant the network a residual skip to 'revert' its revision if beneficial to learning.

We report results from Conditioning Method 2 in the Experiments section as it is the higher performing model between the two.

Relative positional encodings. We implement relative positional encoding to capture information about local word contexts as described in Shaw et al.'s 2018 paper [5]. To do this, we extend the self-attention mechanism to incorporate pairwise positional information between input elements following an open-source implementation [2]. We set the max relative position to 2 as recommended by Dai et al.'s Transformer-XL paper [7], considering a window of 2 input elements before and after each word, which can be thought of as modeling the input as a directed, connected graph.

In our implementation, we extend the computation of each output element (previously a weighted sum of a linearly transformed input elements) to learn distinct edge representations between 2 input elements x_i and x_j, represented as vectors a_{ij}^V and a_{ij}^K. To derive these edge representations, we instantiate an embeddings table of size 5 which represents a window of 5 relative positions. For a given key and query, we create a matrix of the distances between each input element, clipping values to a maximum absolute value of 2. The embedding lookup of this distance matrix is used in a modified self-attention operation:

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V + a_{ij}^V) \tag{2}$$

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}} \tag{3}$$

This new output substitutes the traditional self-attention output in the encoder block.

Ensembling. For our final model ensemble, we combine 4 vanilla QANets, each trained at a different seed, 1 QANet with learnable positional encodings, 1 QANet with relative positional encodings, both QANets with conditioned end predictions, 1 BiDAF baseline, and 1 BiDAF with character-level embeddings. All models with QANet backbones were chosen based on their improvement upon the vanilla QANet performance. We aggregate the predictions using majority vote, with ties broken by each model's F1 score on the development set.

5 Experiments

5.1 Data

We use the modified SQuAD 2.0 dataset provided by CS224N. It contains 130K training examples, 6K validation examples (randomly selected from half of the official dev set), and 6K test examples (the remainder of the official dev set with additional hand-labeled samples).

[2] https://tinyurl.com/relativeposenc

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

Model description	dev F1	dev EM	dev AvNA
BiDAF (baseline)	61.29	57.86	67.72
BiDAF with character-level embeddings	63.46	60.14	69.82
BiDAF with coattention	56.15	52.60	61.24
QANet	68.95	65.15	75.40
QANet with learnable positional encodings	69.83	66.21	76.07
QANet with relative positional encodings	**69.98**	**66.26**	**76.39**
QANet with AvNA module	68.57	64.95	74.76
QANet with conditioned end predictions	69.08	65.55	75.40
QANet + BiDAF ensemble	**72.35**	**69.43**	**77.31**

Table 2: Performance of ensemble on SQuAD 2.0 test set.

Model description	test F1	test EM
QANet + BiDAF ensemble	**70.23**	**67.29**

5.2 Evaluation method

We use the exact match (EM) score, a binary measure of whether the answer matches the ground truth exactly, and F1 score, a harmonic mean of precision and recall, to evaluate the performance of our models. We also record the answer vs. no answer accuracy (AvNA).

5.3 Experimental details

BiDAF-based models. For all models with a BiDAF backbone, we run experiments with default training parameters. Specifically, we train with the Adadelta optimizer [14], for 30 epochs with a learning rate of 0.5 and $\epsilon = 10^{-6}$, no weight decay, and an exponential moving average (EMA) with a decay rate of 0.999. We use batch size of 64.

QANet-based models. For all models with a QANet backbone, we run experiments according to the Training Details section of the original QANet paper [3]. Specifically, we use an Adam optimizer [15], a learning rate of 0.001 with an exponential learning rate warm-up scheme for the first 1000 steps, $\epsilon = 10^{-7}$, $\beta_1 = 0.8$ and $\beta_2 = 0.999$, weight decay of 3×10^{-7}, and an EMA with a decay rate of 0.9999.

Note that to accommodate memory limits, we use batch size of 24 for QANet-based models, although the original QANet paper recommends 32. For the same reason, we use even further reduced batch size of 16 for our QANet with relative positional encodings.

5.4 Results

We present the compiled results of our models on the development set in Table 1. Apart from our significant increase in F1 and EM compared to the baseline, we observe that, among all our models with a QANet backbone (lines 4-9 of Table 1), changes to architecture produce only small changes in performance. This observation aligns well with guest speaker Jared Kaplan's March 1, 2022 lecture and accompanying 2020 paper [16] on scaling laws in which he alludes to the fact that architecture is not crucial in improving performance unless when addressing a bottleneck.

We present our ensemble model's performance on the test set in Table 2. It is not surprising that our ensemble performs slightly worse on the test set than on the development set, since the test set is completely unseen data. This exposes a weakness in our training algorithm: we are saving and testing with models that achieve the highest F1 score, which may cause slight overfitting on the development set.

6 Analysis

In this section, we offer analysis for three of our most consequential extensions of QANet: learnable positional encodings, the AvNA module, and relative positional encodings.

6.1 Relative positional encodings

Because QANet is a non-recurrent model, it does not contain sequential information in its encodings and it is common to use various forms of positional encodings to provide this information to the model. In this work, we implement relative positional encodings, which encodes information about the relative position between each pair of words within a clipped distance of 2. Providing the model with relative positions enables it to learn generalizable information about the different meanings of each word in specific contexts.

In Table 3, we see that QANet with relative positional encodings outperforms a vanilla QANet on all question types except for 'Who' questions. Most notably, QANet with relative encodings performs significantly better than vanilla QANet on 'How' questions. This jump in performance makes sense, as 'How' questions require a higher understanding of language than, for example, 'Where' or 'When' questions, where the model could simply learn to look for place names or 4-digit years as answers. We infer that the relative positional encodings provide that nuanced understanding of the relationships between nearby words required to answer 'How' questions correctly. We refer the reader to the Appendix where we present several examples that showcase the heightened performance of the QANet model with relative positional encodings on 'How' questions, among other question types.

Table 3: Comparing F1 score of a vanilla QANet against our QANet with relative positional encodings, by question type.

Model	Who	What	When	Where	How	Why	Other
QANet	**71.70**	68.76	73.44	66.02	64.90	62.79	64.76
QANet w/ relative positional encs.	71.56	**69.71**	**73.89**	**66.26**	**68.69**	**64.64**	**66.26**

6.2 AvNA module

In Table 4, we discuss more granular results for all 3 of our earlier presented methods of discretizing no-answer examples using the AvNA module; we also include a 'baseline' where we do not use the AvNA output at all. We consider Method 2 to be our most 'conservative' method in predicting no-answer examples, because examples must be assigned a no-answer prediction from both the AvNA module as well as the traditional QANet output layer before being labeled as no-answer. Contrastingly, we consider Method 3 to be our most 'generous' method in predicting no-answer examples, because examples have two avenues by which they can obtain a no-answer label, either through the AvNA module or the traditional QANet output layer.

Table 4: Performance of various discretizing techniques of the AvNA module. Each method is described in full in the Approach section for the AvNA module.

Method	dev F1	dev EM	dev AvNA
Ignore AvNA output (joint start-end only)	68.22	64.59	74.42
Discretization Method 1 (AvNA only)	66.14	62.09	73.45
Discretization Method 2 (AvNA && joint start-end)	65.81	61.75	73.13
Discretization Method 3 (AvNA ‖ joint start-end)	**68.57**	**64.95**	**74.76**

When we compare our most conservative method and our most generous method, the generous method outperforms other methods on all fronts, suggesting that, in general, this iteration of QANet suffers from overly cautious no-answer labels. Furthermore, using the AvNA module in Method 3 outperforms the 'baseline' model which ignores the AvNA module altogether. This indicates to us that additional focus on developing discretization methods that more generously assign no-answer labels may be fruitful in future improvements in performance.

6.3 Learnable positional encodings

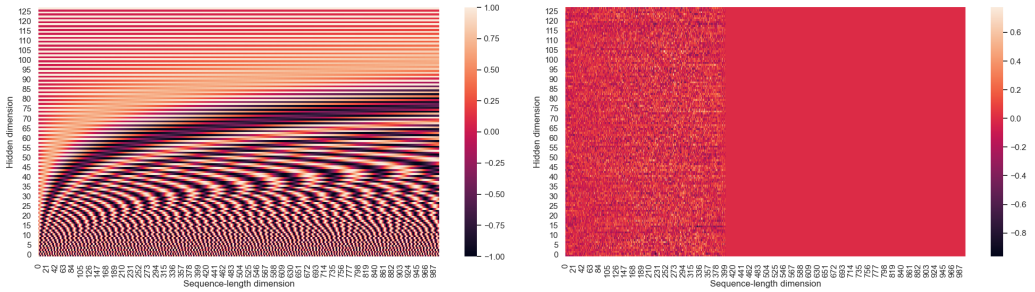

Figure 2: Left: sinusoidal positional encodings used in vanilla QANet. Right: learned positional encodings.

We observe in Figure 2 an immediate shortcoming of learned positional encodings: unlike the sinusoidal positional encodings, the learned positional encodings are not sequence-length invariant. This is because maximum sequence length for all training examples is 400, which means positions 401-1000 are not backpropagated through during training. Thus, during test time, since test paragraphs can have a maximum sequence length of 1000, the learned positional encodings will not contribute useful positional information to words in positions 401-1000.

Despite this major shortcoming, we also notice several similarities between the two encoding visualizations. Firstly, both encodings use floats between -1 and 1, which tells us having normalized values in the [-1, 1] range is most useful for models to learn parameterized encodings. More notably, we see that, like the sinusoidal positional encodings for early hidden dimensions, the learned positional encodings do not exhibit long stretches of constant values. This suggests that the model does, indeed, use the learnable parameter to somewhat encode positional information; with frequent variations in float values, the model is able to distinguish between positions moreso than if the positional encodings were all of similar values.

7 Conclusion

In this project, we implement a QANet model and explore extensions of it, in particular relative positional encodings, learnable positional encodings, and conditioned end predictions. We show that adding relative positional encodings is the most influential extension in improving the model's performance. In addition, we create an ensemble of our highest performing models which achieves strong results on SQuAD 2.0: F1/EM 72.53/69.43 on dev leaderboard and 70.23/67.29 on test leaderboard. We observe that changes in architecture only lead to small changes in performance; to further improve performance, we believe scaling up QANet, using larger hidden sizes and more encoder blocks, may help improve performance more than small architecture changes. In future work, we are interested in scaling up QANet as such, as well as learning how we can integrate state-of-the-art components, for example, pre-trained models such as BERT.

References

[1] Wendy Lehnert. Human and computational question answering*. *Cognitive Science*, 1(1):47–73, 1977.

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

[3] 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, 2018.

[4] François-Xavier Aubet, Dominic Danks, and Yuchen Zhu. Equant (enhanced question answer network). *CoRR*, abs/1907.00708, 2019.

[5] Ashish Vaswani Peter Shaw, Jakob Uszkoreit. Self-attention with relative position representations, 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. *CoRR*, abs/1706.03762, 2017.

[7] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

[8] Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension, 2017.

[9] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2016.

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

[11] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016.

[12] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[13] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[14] Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2015.

[16] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

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

A Appendix

A.1 QANet encoder block and other training details

There are several remaining details in the implementation of the encoder block: wrapping each sublayer is a residual skip connection which accommodates deep learning by allowing the model to resort to the identity function when beneficial to learning. We also normalize inputs for each sublayer using layer normalizations. A final detail is that, during training, we employ a technique called layer dropout [17] which randomly drops sublayers by forcing residual skips at a rate proportional to layer depth, with deeper layers dropping out at higher probabilities.

For backpropagation, we compute the loss by summing the negative log likelihood of p_1 and p_2 with their respective ground truth start and end positions. For making a prediction, we discretize the predictions by selecting the start and end indices that maximize the joint probabilities of the start and end predictions. A prediction of (0, 0) is considered a no-answer prediction, because we preprocess examples by appending an out-of-vocabulary token to the very beginning of each context.

A.2 AvNA module details

Here, we provide more details about the AvNA module. Firstly, we will outline specifically the architecture of the AvNA module. The AvNA module (Figure 3) takes as input the output of the first of our 3 model encoder layers. This is then fed through 2 independently initialized encoder blocks, each configured with 2 convolutional sublayers and a kernel size of 5, 3 feed-forward layers which reduce the hidden dimension to 1 with rectified linear units as activation functions, and a mean-pool along the sequence-length dimension to obtain a single float for each batch. Each float represents the 'answerability' score of the example, and are outputted by the model alongside the start and end position predictions.

Next, we discuss our original contributions to the loss function. Aubet et al. recommend the following loss function:

$$l(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\mathcal{L}_0^{(i)}(p_0^{(i)}) + \delta^{(i)}\left(\mathcal{L}_1^{(i)}(p_1^{(i)}) + \mathcal{L}_2^{(i)}(p_2^{(i)})\right)\right], \quad (4)$$

where δ is the gold answerability, $\mathcal{L}_0^{(i)}(p_0^{(i)})$ is the answerability cross-entropy loss, $\mathcal{L}_1^{(i)}(p_1^{(i)})$ is the start prediction cross-entropy loss, and $\mathcal{L}_2^{(i)}(p_2^{(i)})$ is the end prediction cross-entropy loss. The issue with this loss function, we found, was that, when examples are unanswerable, loss (Equation 4) for start and end predictions are zeroed out, and only the AvNA module is trained, leaving the original output layer and the final two model encoder layers untouched. Thus, our original contribution on the loss function simply involves removing the δ, as such:

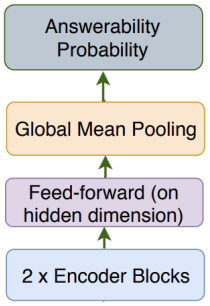

Figure 3: The AvNA module. Source: [4]

$$l(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\mathcal{L}_0^{(i)}(p_0^{(i)}) + \mathcal{L}_1^{(i)}(p_1^{(i)}) + \mathcal{L}_2^{(i)}(p_2^{(i)})\right], \quad (5)$$

allowing the original QANet's start and end prediction pipeline as well as the additional AvNA module to be trained for all examples. This new loss function (Equation 5) also gives the model the ability to continue to attend to the prepended OOV token which indicates a no-answer prediction, as before, alongside the new AvNA module.

A.3 Examples of relative positional encodings' gains in performance

We see the importance of encoding contextual information in examples like the following:

Question: What city later became Beijing?
Context: Kublai readied the move of the Mongol capital from Karakorum in Mongolia to Khanbaliq in 1264, constructing a new city near the former Jurchen capital Zhongdu, now modern Beijing, in 1266. In 1271, Kublai formally claimed the Mandate of Heaven and declared that 1272 was the first year of the Great Yuan in the style of a traditional Chinese dynasty...
Answer: Zhongdu

The vanilla QANet model incorrectly predicts the answer to be "Kublai", whereas the modified model with relative positional encoding is able to accurately predict the answer "Zhongdu". In analyzing this difference, we hypothesize that the vanilla QANet predicts "Kublai" as the answer because its lack of refined positional awareness causes it to merely identify Kublai as the main subject of the context paragraph. On the other hand, the addition of relative positional encodings allows for the model to learn more granular relationships between words in the context and correctly identify "now modern Beijing" as the key context in answering the question of which city later became Beijing.

As shown by our analysis, the vanilla QANet performs generally worse on "How" questions, because it learns global associations rather than using relative positional context to answer queries. For

instance, in the example:

Question: How effective was the military use of the "Afghan Arabs"?
Context: In 1979, the Soviet Union deployed its 40th Army into Afghanistan, attempting to suppress an Islamic rebellion against an allied Marxist regime in the Afghan Civil War. The conflict, pitting indigenous impoverished Muslims (mujahideen) against an anti-religious superpower, galvanized thousands of Muslims around the world to send aid and sometimes to go themselves to fight for their faith. Leading this pan-Islamic effort was Palestinian sheikh Abdullah Yusuf Azzam. While the military effectiveness of these "Afghan Arabs" was marginal, an estimated 16,000 to 35,000 Muslim volunteers came from around the world came to fight in Afghanistan.
Answer: marginal

The vanilla QANet model predicts "16,000 to 35,000", whereas the model with relative positional encodings is able to correctly extract the correct answer, "marginal". We hypothesize the Vanilla QANet model learned to associate "How + adjective" questions with measurable (i.e. numerical) responses, whereas the addition of relative positional encodings enables it to identify the phrase "effectiveness of these 'Afghan Arabs' was marginal" as relevant.