QuAVONet: Answering Questions on the SQuAD Dataset with QANet and Answer Verifier

Jesus Cervantes
Department of Computer Science
Stanford University
cerjesus@stanford.edu
https://github.com/CerJesus/cs224nfinalproject

Abstract

SQuAD has been a driving force behind the challenge of Machine Comprehension in recent years. In this paper, I set to first recreate a popular model, QANet, from SQuAD 1.1, before combining it with Answer Verifier from U-Net, what I refer to as QuAVONet. The best results achieved, however, was a QANet-lite with 4 attention heads and 96 hidden dimension size for an EM of 58.157 and F1 of 61.39 on the hidden test set. QuAVONet, however, only achieves EM of 57.532 and nearly identical F1 of 61.389 on the test dataset.

1 Introduction

SQuADv1.1 is a challenge and dataset of 100,000 passages and corresponding questions introduced in 2016 for the purpose of encouraging the advancement of Machine Comprehension (MC) through answering reading comprehension questions of passages [9]. In 2018, SQuADv2.0 introduced an additional 50,000 questions for which the answer could not be found in the given passage [10]. The ability to determine the answerability of questions resulted in the development of adjustments to existing models and the creation of entirely new models to account for this new challenge.

One model that was fairly successful on SQuADv1.1 but as of writing this report is still not on the leaderboard for SQuADv2.0 is QANet [1]. In an attempt to bring QANet to the modern world of SQuADv2.0, I set out to combine QANet with the Answer Verifier (AV) component of U-Net [12]. AV is specifically designed to predict whether the answer to a question is in a passage, the exact component that the original QANet model was lacking. I call this model the Question Answer Verifying Optimization Network (QuAVONet).

2 Related Work

The bulk of my work is predominantly focused on QANet, a feedforward model consisting exclusively of convolutions and self-attention [1]. The defining feature of QANet is the EncoderBlock, an analogue of Transformers applied to SQuAD [2]. QANet was introduced to tackle SQuADv1.1 where it achieved 73.6/82.7 EM/F1 scores. QANet was able to surpass State of the Art (SotA) with even higher scores through the use of data augmentation, however this is a practice I don't explore in this paper. QANet's data augmentation entailed translating the questions and contexts into French and back to English to obtain paraphrases of the original text and effectively expand the dataset. While this technique worked well, it was beyond the resources and time constraint of this project.

The other major work that serves as the basis for this project is U-Net [12]. U-net is an end-to-end model that incorporates many concepts from Seo et al.'s BiDAF. The primary component from U-net

that I make use of in this work is the answer verifier. The answer verifier is a component that predicts the probability of a question being unanswerable and comes in the form of a linear layer. U-net is not the first to use an answer verifier, however, prior approaches trained the answer verifier separately from the rest of the model. U-net utilizes information from the rest of the model, such as the answer pointer and attention, to help train the answer verifier. The complete U-Net model achieved 71.7 F1 on SQuAD 2.0.

Finally, I utilized BiDAF as described by Seo et al. and implemented by the teaching staff to serve as both a baseline and a third model to experiment with combining with QANet and Answer Verifier [11].

3 Approach

3.1 Problem Formulation

Here, I define the following terms of the problem that are referred to throughout the paper. The model takes as input a context C of length n defined as $C = \{c_1, c_2, ..., c_n\}$ and a question Q of length m defined as $Q = \{q_1, q_2, ..., q_n\}$, my models output indices i and j such that $0 \leq i \leq j < n$ the answer to Q is $S = \{c_i, c_{i+1}, ..., c_j\}$, a span in C.

3.2 Baseline

For my initial baseline, I used the provided modified BiDAF model with no character embeddings as described in the default project handout.

3.3 QANet

This subsection will focus on the implementation of QANet, consisting of an embedding layer, embedding encoding layer, a context query attention layer, a model encoding layer, and an output layer.

3.3.1 Embedding

The embedding layer involves getting 64-dimensional character embeddings for every letter in each word and running a 1D convolution on the resulting matrix before performing relu on it and taking the max element of each row to get a 64-dimension word representation. We then concatenate this with a 300-dimension word2vec vector for the word and run the concatenation of these vectors through a 1D convolution to get a 96-dimension hidden state, though I also experimented with hidden dimensions of 100 and 128. During training, the model also employs dropout with probability 0.1 for word embeddings and 0.05 for the character embeddings. Note that the original QANet model used 200-dimension trainable representations for the characters whereas my implementation simply uses the 64-dimension embeddings that were given in the starter code. These embeddings are still made trainable, as they are in the original QANet.

3.3.2 Embedding Encoder

The next major component of QANet is QANet's signature Encoder Block. The Encoder Block is a major part of QANet and is derived from Transformer.

Each Encoder block takes an input and scales up the values before adding a positional encoding to the vector. The original vector is scaled up to make the positional encodings relatively smaller. After the positional encoding, we then run through a varying number of depthwise separable convolutions each preceded by a layernorm [4]. Each convolution has kernel size 1 and output size equal to the input size. Next, the model runs through another layernorm before running the multi-head self-attention, with 4 heads, and finally ending with a feed-forward layer preceded by another layernorm. Note that the original QANet utilized 8 attention heads, but I was able to only use 4 with no performance loss. Each layer in the block is also wrapped in a residual block such that the output of any given layer is

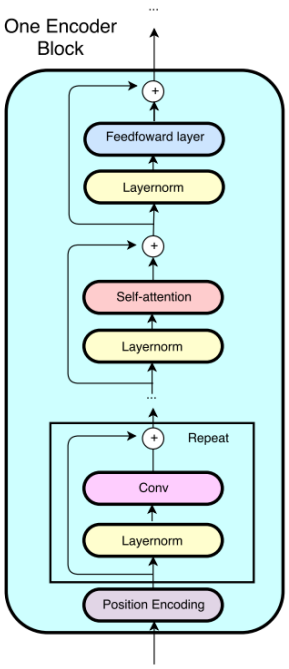

Figure 1: A visual representation of an Encoder Block, consisting of convolutional layers and multi-head self-attention.

$f(layernorm(x)) + x$ [1]. The overall output size for an Encoder Block is the same as the input size. In addition, along with regular dropout, during training the Encoder Block also employs sublayer dropout in which each sublayer has dropout probability $d_l = \frac{l}{L}(1 - p_L)$ where l is the current sublayer, L is the total number of sublayers, and $p_L = 0.9$ [9]. Since the Encoder Block is based on Transformer, I was able to source from a few different Transformer implementations [8]. First, I outsourced implementations for the Positional Encoder and Depthwise Separable convolution, though for the convolution I looked at various implementations and readings to better understand the concept and ensure the validity of the implementation I was using [8] [5]. I also took the implementation for multi-head self-attention from the same source before implementing my own feed-forward network to finish off the pieces needed for the Encoder Block. When it comes to making the Encoder Block itself, the rest of the implementation was done by me.

3.3.3 Context-Query Attention Layer

This attention layer is the exact same as that proposed by Seo et al. and implemented in the baseline [11]. A similarity matrix between C and Q is computed through the trilinear similarity function to output $S \in R^{nxm}$ [11]. Softmax is then applied to S to normalize each row and yield $\overline{S}$. The context to query attention is defined as $A = \overline{S} \cdot Q^T \in R^{nxd}$. The query to context attention is then computed as $B = \overline{S} \cdot \overline{\overline{S}}^T \cdot C^T$ where $\overline{\overline{S}}$ is the column normalized matrix of S. The output of this layer is [C,A,C*A,C*B] where * is the element-wise multiplication of the two matrices. The code for this layer was entirely copied from the baseline implementation.

3.3.4 Model Encoder

The Model Encoder layer of QANet consists of 7 stacked Encoder Blocks, each with 2 convolution layers. The input to these Encoder Blocks is the hidden size. Since the output of the CQAttention Layer is 4*hidden size, we use a 1D convolution to scale the output from CQAttention back down to hidden size to be input D to the Model Encoder blocks. This layer outputs 3 matrices M_1, M_2, M_3. M_1 is the result of D going through the stacked Encoder Blocks once. To get M_2, I run M_1 through the Encoder Blocks again. M_3 is the result of running M_2 through the Encoder Blocks. These matrices serve as the input for the output layer.

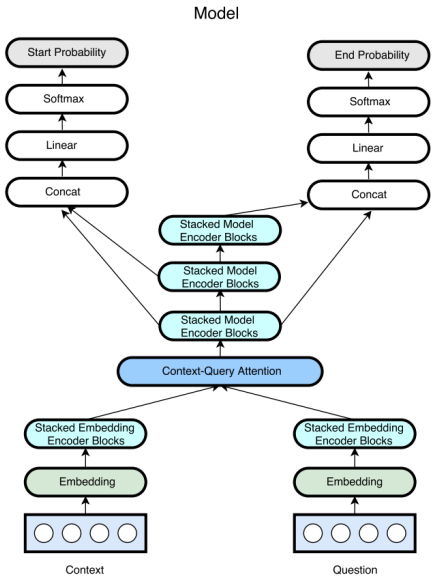

Figure 2: A visual representation of the QANet architecture.

3.3.5 Output Layer

The output layer is adapted from Seo et al. and the baseline and predicts the probability of each position from being either the beginning or end of the correct answer span [1]. Like in the baseline, each context is prepended with a token such that the (start,end) pair (0,0) represents a prediction of No Answer. The probabilities for the start and end positions are calculated as:

$$p^1 = softmax(W_1[M_0; M_1]), p^2 = softmax(W_2[M_0; M_2])$$

where W_1 and W_2 are trainable weight variables. The score for a specific span is calculated by taking the product of the probabilities of the start and end positions of that span. For the model's objective function, QANet takes the negative sum of the log probabilities of the predicted probabilities of the correct endpoints, averaged over the training examples. More formally, this is written as:

$$L(\theta) = -\frac{1}{N}\sum_{i}^{N}\{log(p^1_{y_i^1}) + log(p^2_{y_i^2})$$

where y_i^1 and y_i^2 are the true start and endpoints for example i. For predictions, the predicted span output (s,e) is chosen to maximize the product of the probabilities of s and e and such that $s \leq e$. The final architecture of QANet can be seen in Figure 2, taken from the original QANet paper [1].

3.4 QANet + Answer Verifier - QuAVONet

The final component of my model is Answer Verifier and it is what transforms QANet to QuAVONet. Answer Verifier is a linear layer that is run on $F = (c_q; o_{m+1}; c_s; c_e)$ to output p^c, the probability a question is answerable [12]. Before I define each of these terms, I will first note that these matrices are based on other aspects of U-Net that don't always have direct analogues in QANet. As such, these terms have some slightly different representations in my model than in the original U-Net paper.

First, c_q is meant to be a representation of the question. To create this, in the CQAttention layer I performed a matrix multiplication similar to that used to create A and got the max of each row along the query length dimension. This is more formally written as $c_q = \overline{\overline{S}}^T \cdot C$ to create an attentive representation of the question.

Next, o_{m+1} is what U-Net describes as a universal node and is effectively the attention representation of the prepended No-Answer token in the starter code. My representation is slightly different from U-Net's due to different attentions but is functionally similar.

Finally, c_s and c_e are representations of the passage based on the log probabilities of the start and end answer pointers. We define them as follows:

$$c_s = p^1 \cdot B, c_e = p^2 \cdot B$$

The concatenation of these four components yields $F \in R^{bx7*h}$ where b is the batch size and h is the hidden size. The loss for this layer is the binary cross-entropy loss:

$$L_{AV} = -(\sigma log(p^c) + (1-\sigma)(1 - log(p^c)))$$

where $\sigma in \{0, 1\}$ indicates whether a question is answerable or not. This loss is added to the NLL loss of QANet. At inference time, I use this layer to override the prediction from QANet if the AV probability of a question being answerable is below a threshold. In the U-Net paper, the threshold they found best was 0.3 [12].

4 Hybrid Models

Using the 3 models described above, I created several models that are hybrid combinations of the baseline, QANet, and AV. The most prominent hybrid is QuAVONet, the combination of QANet and AV. However, I experimented with various other combinations of these models that yielded interesting results. First, A model I call QAEmb takes the baseline but replaces the embedding layer with the QANet embedding layer. AVQAEmb is QAEmb with AV. Next, QAEmbEnc is a similar premise but also replaces the embedding encoder layer of the baseline with the Encoder Block used by the embedding encoder layer of QANet. Finally, QARNNet is QANet but with the Embedding Encoder layer replaced with the RNNEncoder from the baseline and a 1D convolution to bring the output size back down to hidden size. While QARNNet may seem like a strange idea, the reasoning behind this will be made clear in the experiments section of this paper. During experiments, I will refer to models based primarily on QANet (QANet, QuAVONet, QARNNet) as the QANet family and the models based primarily on BiDAF (baseline, QAEmb, QAEmbEnc, AVQAEmb) as the BiDAF family of models.

5 Experiments

5.1 Data

The data for this project comes from the official SQuAD 2.0 dataset, consisting of passages from Wikipedia and corresponding questions and answer spans [10] [3]. The train data consists of 129,941 examples taken from the SQuAD 2.0 training set. This is not the entirety of the official training set as the official dev set is used to create this project's dev and test set. The dev set consists of 6078, roughly half of the official dev set, randomly selected samples of the official dev set. Finally, the test set consists of 5915 examples from the official dev set with a few hand-labeled examples to detect cheating by teams who may try to train their model on the official dev set. For the evaluation metrics, I use the EM and F1, as is common practice for SQuAD and outlined in the starter code.

5.2 Implementation Details

For the word embeddings, I used the starter code's 300-dimensional GloVE vectors trained on the CommonCrawl dataset [6]. These embeddings remained unchanged and were not trained for any of my models. As for the character embeddings, I used the starter codes 64-dimension character embeddings and unfroze these embeddings to let them be trained for each model.

During training of all models, as per the starter code I maintain moving averages of all parameters with a decay rate of 0.999 and use these moving averages as parameters during test time. For the QANet family, I use the Adam optimizer as described in the QANet paper with $_1 = 0.8, _2 = 0.999, \epsilon = 10^{-7}$ and a learning rate of 0.0001 with logarithmic growth for the first 1000 steps [1] [7]. For the BiDAF family, I used the Adadelta optimizer with constant learning rate of 0.5 as defined in the starter code [11] [13].

I trained 2 versions of QANet, one with 8-head attention and 128 hidden size, as defined in the QANet paper, and another with 4 attention heads and 96 hidden size for speed as described in the CS224N staff implementation on Piazza [1]. QuAVONet and QARNNet both use the 4-head attention and 96 hidden size. The BiDAF family all use the default parameters like 100 hidden size from the starter code.

Finally, I trained QAEmb, QAEmbEnc, and 4-head QANet on an NV6 instance. I trained QuAVONet, AVQAEmb, the baseline, AVQAEmb, 8-head QANet, and QARNNet on an NV12 instance. I trained each model to different number of iterations depending on the potential seen for each on Tensorboard. For example, it became quite obvious early on that QAEmbEnc was not going to surpass baseline so I only trained it for roughly 2M iterations whereas QuAVONet was still growing after 30 epochs expired so I trained it for an additional 10 epochs, amounting to roughly 5.3M iterations. As a result, QuAVONet appears as 2 separate colors on my graphs: grey until 30 epochs then becoming a rust color.

5.3 Results and quantitative analysis

In this section, I compare the results from the various models trained and analyze some errors and other key differences between the models.

5.4 Model Performance and Quantitative Analysis

The best performing model was the 4-head QANet implementation, achieving 60.813/64.458 EM/F1 on the dev set and 58.157/61.390 EM/F1 on the hidden test set. QuAVONet achieved nearly comparable results with decreases of 0.067/0.101 on the dev set and 0.626/0.001 on the hidden test set compared to the 4-head QANet. The 8-head QANet, interestingly enough, was slightly inferior to both of these models, with a loss of 0.218/0.066 relative to 4-head QANet on the dev set and a loss of 1.200/1.578 on the hidden test set. It would seem that the larger model size resulted in increased overfitting during training, stunting performance on the test set.

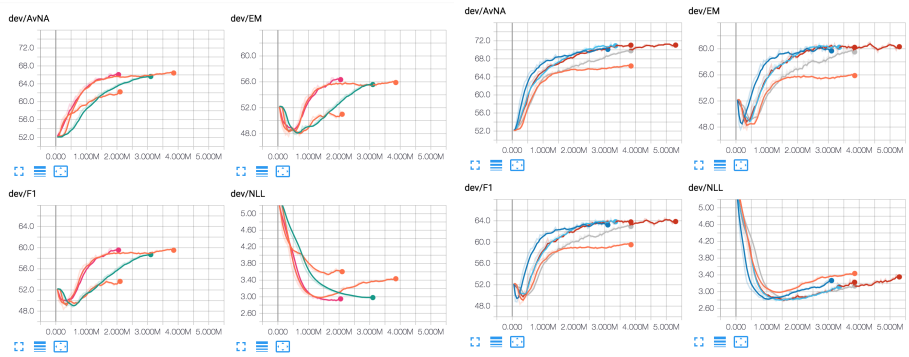

Figure 3: Plots of the AvNA, F1, EM, and NLL scores for the models, weaker ones on the left and stronger on the right. Left: baseline, orange; QARNNet, pink; AVQAEmb, mint; QAEmbEnc, orange (bottom) Right: baseline, orange; QuAVONet, gray then rust; QAEmb, dark blue; QANet 8-head light blue; QANet 4-head dark red

Another interesting note is that QAEmb performed nearly as well as the 4-head QANet (loss of 0.369/0.699 on dev set) but was 3.5x faster than QANet, taking 7 hours to complete 3M iterations compared to 4-head QANet's 25 hours. The results of various models I implemented on the dev and test set can be seen in Table 1. This comparable performance in reduced time is why I created AVQAEmb, hoping that AV would be able to boost QAEmb beyond QANet while retaining the speed advantage. While AVQAEmb remained very fast, finishing 3M iters in 6.5 hours on NV12, its performance was severely stunted, barely catching up to the baseline after 3M iterations.

Another key area of quantitative interest is the effect of the Encoder Embedding layer of QANet. For QAEmbEnc the Encoder Block severely stunted performance over the RNNEncoder in the baseline (QAEmb), yielding a loss of about 7.4 on AvNA, 8.7 on EM, and 9.7 on F1 after 2M iterations, which was worse than baseline, though it performed marginally faster than QAEmb. This performance loss is why I created QARNNet, replacing the embedding encoder in QANet with the RNNEncoder, to see if it would boost performance of QANet. However, after 2M iterations, QARNNet (in pink in Figure 3) was significantly underperforming both QAEmb and QANet, so I cut the training short given the scarcity of time and resources, though it was able to surpass the baseline in that time.

5.5 Qualitative Analysis

In this section I will analyze different error types I noticed were common throughout the main high-performing models: QANet 8- and 4-head, QuAVONet, and QAEmb.

5.5.1 Making Short Answers Shorter

- **Question:** When had the Brotherhood renounced violence as a means of achieving its goals?
- **Context:** While Qutb's ideas became increasingly radical during his imprisonment prior to his execution in 1966, the leadership of the Brotherhood, led by Hasan al-Hudaybi, remained moderate and interested in political negotiation and activism. Fringe or splinter movements inspired by the final writings of Qutb in the mid-1960s (particularly the manifesto Milestones, a.k.a. Ma'alim fi-l-Tariq) did, however, develop and they pursued a more radical direction. By the 1970s, the Brotherhood had renounced violence as a means of achieving its goals.
- **Answer:** By the 1970s
- **Prediction:** 1970s

Figure 4: Example Question, Passage, and Answer from QANet 4-head in which the predicted answer was just slightly off from the true answer, but similar in meaning.

A common issue among all the QANet family has been cutting already short answers slightly shorter, like missing an article. Figure 4 show as such an error from 4-head QANet, but 8 head QANet has similar incorrect answers like predicting "Scotland Act" instead of "Scotland Act 1998". QuAVONet also has a similar issue, one example being predicting "Catholic church" instead of "Catholic church in the region". These slightly shortened answers don't seem to impact the meaning of the answer too much, likely retaining F1 score, but the slight missteps are likely a significant impact on the EM scores of the models given the relative frequency of their occurrence.

5.5.2 Relationship Dependency without Logical Backing

- **Question:** What causes elevated vitamin D levels in the elderly?
- **Context:** It is conjectured that a progressive decline in hormone levels with age is partially responsible for weakened immune responses in aging individuals. Conversely, some hormones are regulated by the immune system, notably thyroid hormone activity. The age-related decline in immune function is also related to decreasing vitamin D levels in the elderly. As people age, two things happen that negatively affect their vitamin D levels. First, they stay indoors more due to decreased activity levels. This means that they get less sun and therefore produce less cholecalciferol via UVB radiation. Second, as a person ages the skin becomes less adept at producing vitamin D.
- **Answer:** N/A
- **Prediction:** The age-related decline in immune function

Figure 5: Example Question, Passage, and Answer from QuAVONet correctly identifying a relation in the passage, but of the wrong nature from that defined in the question.

Another error type for QuAVONet is answering based on the existence of a correectly modeled relationship without accurately taking into account the nature of the relationship. In Figure 5,

QuAVONet predicts that "The age-related decline in immune function" causes elevated Vitamin D levels when, in fact, the elderly don't experience elevated vitamin D levels. However, the passage creates a logical link between the predicted span and "decreased Vitamin D levels". QuAVONet seems to accurately extract the link between its predicted span and "Vitamin D levels" but does not model the distinction between the "elevated" in the question and the "decreased" in the context.

6 Conclusion

For this project, I implemented a neural network to answer questions from the SQuAD dataset. My network was a combination of QANet with the Answer-Verifier from U-Net, what I dub as QuAVONet (Question Answer Verifier Optimizing Network) [1][12]. While AV was an idea to bring QANet from it's original application to SQuAD1.1 to SQuAD 2.0, my QANet achieved an EM of 58.157 and F1 of 61.39 on the hidden test set whereas QuAVONet achieved 57.532 EM and 61.389 F1, slightly worse than the regular QANet.

What's even more interesting is that, in spite of the speed of QANet relative to RNNs that is touted in the paper, the QAEmb model achieves only slightly inferior results relative to these two models on the dev set (less than 0.4 loss of EM and 0.8 loss for F1) but in a fraction of the time. QAEmb reached 3M iterations in 7 hours on NV6 whereas QANet took 25 hours on NV6 to reach 3M iterations and QuAVONet required nearly 16 hours on an NV12 to reach the same number.

While QANet and QuAVONet were able to achieve marginally better results, they are significantly less efficient. As to why this occurs, that is work that could make for an interesting project in the future. Another project I would like to explore is incorporating recent improvements to Transformers, into QANet and QuAVONet. My original project was going to be QANet-XL: applying the segment level recurrence and relative positional encodings from Transformer-XL to QANet, so I'd certainly be interested to see how a QuAVONet-XL would perform [14].

As of writing this paper, the best performer on the SQuAD 2.0 leaderboard is 0.158 EM and 0.305 F1 short of human performance [10]. This only goes to show that there is still plenty of work to be done in the field of MC and progress is being made every day.

7 Acknowledgements

I'd like to thank the CS224N teaching staff for the constructive feedback and guidance for this project. I'd also like to thank Microsoft for sponsoring the class' use of GPU VMs.

8 Notes

8.1 Qualitative Notes

QANet at 2.25m iters with 4 heads and batch size 12 on NV6, lr=0.0001 Seems to often miss articles in the beginning of answers (i.e predicts "Parliamentary Bureau" instead of "the Parliamentary Bureau" for a who question or "Article 5" instead of "in Article 5" for a "where" question. After about 2.6M iters, performs about as well as QAEmbed, but takes 21h on NV6 as opposed to 6h for QAEmbed. Way slower!

8.2 Tests

QANet at 2.25m iters with 4 heads hidden size 96 and batch size 12 on NV6, lr=0.0001 Takes around 62 minutes per epoch (30-35it/s), scored EM: 60.813 (+0.370 from QAEmb) F1: 64.458 (+0.700 from QAEmb)

QANet with 4 heads,hidden size 96 and batch size 12 on NV12, lr=0.0001 (just to check speed): 60 it/s, twice as fast! Didn't run to results for time and instead just looked at speed

QANet iters with 8 heads, 128 hidden, and batch size 12(*2) on NV12, lr=0.0001: 45 it/s! 53min/epoch. Almost identical performance to 8 head, 96 hidden size performance, but thanks to NV12 only took 19h 35min instead of 24h 50 min to get to 3M iters.

QANet with 4 heads, hidden size 96, batch size 12, RNNEncoder for EmbEnc, NV12, lr = 0.0001: 48 min/epoch, 45-50 it/s, this is slower than same dims but with EncoderBlock for EmbEnc

QuAVONet with 4 heads, hidden size 96, batch size 12, NV12, lr = 0.0001, Adam Optimizer after 40 epochs: Dev: EM: 60.746 (-0.067) F1: 64.358 (-0.101) Test: EM: 57.532 (-0.626) F1: 61.389 (-0.001)

QAEmb + AV, hidden size 100, batch size 64(?), NV12, lr = 0.0001: not great

ANSWER VERIFIER: https://arxiv.org/pdf/1810.06638.pdf

9 References

References

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

[2] Niki Parmar-Jakob Uszkoreit Llion Jones Aidan N Gomez Łukasz Kaiser Ashish Vaswani, Noam Shazeer and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, page 5998–6008, 2017.

[3] Danqi Chen and Adam Fisch et al. Reading wikipedia to answer open-domain questions. *Association for Computational Linguistics*, 2017.

[4] Franc¸ois Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR, abs/1610.02357*, 2016.

[5] Pytorch Forums. How to modify a conv2d to depthwise separable convolution?, 2019.

[6] Richard Socher Jeffrey Pennington and Christopher D. Manning. Glove: Global vectors for word representation. 2014.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR, abs/1412.6980*, 2014.

[8] S. Lynn-Evans. How to code the transformer in pytorch - towards data science, 2018.

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

[10] Robin Jia Pranav Rajpurkar and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

[11] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hananneh Hajishirzi. Bi-directional attention flow for machine comprehension. 2018.

[12] Fu Sun, Linyang Li, Xipeng Qiu, and Yang Liu. U-net: Machine reading comprehension with unanswerable questions. 2018.

[13] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[14] Yiming Yang-William W Cohen Jaime Carbonell Quoc V Le Zihang Dai, Zhilin Yang and Ruslan Salakhutdinov. Transformer-xl: Language modeling with longer-term dependency. 2019.