A Fully Attention-Based Information Retriever

Alvaro H. C. Correia, Jorge L. M. Silva, Thiago de C. Martins, Fabio G. Cozman
Escola Politecnica - Universidade de Sao Paulo
Sao Paulo, Brazil

***Abstract*—Recurrent neural networks are now the state-of-the-art in natural language processing because they can build rich contextual representations and process texts of arbitrary length. However, recent developments on attention mechanisms have equipped feedforward networks with similar capabilities, hence enabling faster computations due to the increase in the number of operations that can be parallelized. We explore this new type of architecture in the domain of question-answering and propose a novel approach that we call Fully Attention Based Information Retriever (FABIR). We show that FABIR achieves competitive results in the Stanford Question Answering Dataset (SQuAD) while having fewer parameters and being faster at both learning and inference than rival methods.**

I. Introduction

Question-answering (QA) systems that can answer queries expressed in natural language have been a perennial goal of the artificial intelligence community. An interesting strategy in the design of such systems is information extraction, where the answer is sought in a set of support documents. However, extracting information from large texts is still a challenging task, and most state-of-the-art models restrict themselves to single paragraphs. That is, in fact, the proposed focus of recent open-domain QA datasets, such as SQuAD [1].

In SQuAD, each problem instance consists of a passage $\mathcal{P}$ and a question $\mathcal{Q}$. A QA system must then provide an answer $\mathcal{A}$ by selecting a snippet from $\mathcal{P}$. That format reduces the complexity of the task and also facilitates training, as one can learn a probability distribution over the words that compose the passage. Since its publication in 2016, SQuAD has been targeted by many research groups, and the proposed models are gradually approaching (even overcoming) human-level performances. All but a few of these models rely on Recurrent Neural Networks (RNNs), which currently dominate the state-of-the-art in most Natural Language Processing (NLP) tasks. However, RNNs do have some drawbacks, of which the most relevant to real-world applications is the high number of sequential operations, which increases the processing time of both learning and inference. To address these limitations, Vaswani et al. have proposed the Transformer, a machine translation model that introduces a new deep learning architecture solely based on "attention" mechanisms [2]. We later clarify the meaning of attention in this context.

Inspired by the positive results of Vaswani et al. in machine translation, we have applied a similar architecture to the domain of question-answering, a model that we have named Fully Attention-Based Information Retriever (FABIR). Our goal then was to verify how much performance we can get exclusively from the attention mechanism, without combining it with several other techniques. We validated our model in the SQuAD dataset, which proved that FABIR not only achieves competitive results (F1:77.6%, EM:67.7%) but also has fewer parameters and is faster at both training and testing times than competing methods. Besides the development of a new architecture, we identify three major contributions of our work that have made these results possible:

- **Convolutional attention**: a novel attention mechanism that encodes many-to-many relationships between words, enabling richer contextual representations.
- **Reduction layer**: a new layer design that fits the pipeline proposed by Vaswani et al. [2] and compresses the input embedding size for subsequent layers (this is especially beneficial when employing pre-trained embeddings).
- **Column-wise cross-attention**: we modify the cross-attention operation by [2] and propose a new technique that is better suited to question-answering.

This article is organized as follows. We first introduce some of the related work in question-answering and then present FABIR's architecture and its basic design choices. Subsequently, we report and comment our results in the SQuAD dataset. Finally, we compare the performance of FABIR with RNN-based models and draw some conclusions, suggesting directions for future work.

II. Related Work

The vast majority of papers that address the SQuAD dataset have adopted RNN-based models [3]–[26]. They all follow a similar pipeline, with pre-trained word-embeddings that are processed by bidirectional RNNs. Question and passage are processed independently, and their interaction is modeled by attention mechanisms [27] to produce an answer. There are slight differences in how each model employs attention, but they all calculate it over the hidden states of an RNN. Vaswani et al. were the first to apply attention directly over the word-embeddings, and thus derived a new neural network architecture which, without any recurrence, achieved state-of-the-art results in machine translation [2]. In this section, we briefly discuss both types of attention models.

A. Traditional Attention Mechanisms

In recent years, attention mechanisms have been used with success in a variety of NLP tasks, such as machine translation

Alvaro H. C. Correia and Jorge L. M. Silva contributed equally to this work.

2799 978-1-5090-6014-6/18/$31.00 ©2018 IEEE

[2], [27] and natural language inference [28], [29]. Indeed, most models that target the SQuAD dataset use some form of attention to model the relationship between question and passage.

Attention can be defined as a mechanism that gives a score α_i to a vector p_i from a set $P = [p_1, ..., p_m]$ with respect to a vector q_j from $Q = [q_1, ..., q_n]$. This score is a function of both P and Q and is shown in its most general form in (1).

$$s_{i,j} = f(p_i, q_j), \tag{1a}$$

$$\alpha_{i,j} = \frac{\exp(s_{i,j})}{\sum_{k=1}^{n} \exp(s_{i,k})}, \tag{1b}$$

where s_i and α_i are scalars and f is a score function that measures the importance of p_i relative to q_j. Intuitively, a large weight α_i means that the vector p_i is somehow strongly related to Q. In the literature, two alternatives for f have been proposed, additive [27] and multiplicative [30] attentions:

$$f(p_i, q_j) = \begin{cases} W_3 g(W_1 p_i + W_2 q_j) & \text{(additive)} \\ p_i^T W_1 q_j & \text{(multiplicative),} \end{cases} \tag{2}$$

where W_1, W_2 and W_3 are learnable parameters and g is a elementwise nonlinear function. For small vectors, additive and multiplicative attention mechanisms have been shown to produce similar results [31].

In most models, the attention scores α are used to create a context vector c given by a weighted sum of P, which is processed by an RNN:

$$c^t = \sum_i \alpha_{i,t} p_i, \tag{3a}$$

$$v^t = \text{RNN}(v^{t-1}, c^t), \tag{3b}$$

where v^t is the hidden state of the RNN at time t. Notably, in the SQuAD dataset, P and Q are the vectorial representations of passage and question, respectively.

B. Google's Transformer

The Transformer is a machine translation model introduced in [2] that achieved state-of-the-art results by combining feed-forward neural networks with a multiplicative attention mechanism applied over position-encoded embedding vectors. It defines three different matrices U, K and V that are associated with queries, keys, and values, respectively. Every attention operation in the Transformer is performed by multiplying these matrices as shown in (4).

$$\text{att}(U, K, V) = \text{softmax}\left(U W_{UK} K^T\right) V \, W_V, \tag{4}$$

where $W_{UK}, W_V \in \mathbb{R}^{d_{model} \times d_{model}}$ are weight matrices and d_{model} is the embedding size of each word. Additionally, Vaswani et al. [2] suggest a multi-head attention, in which U, K and V are divided into n_{heads} heads and the attention in the i^{th} head is computed as

$$logits_i = U W_{U,i} (K W_{K,i})^T, \tag{5a}$$

$$\text{att}_\text{i}(U, K, V) = \text{softmax}\left(logits_i\right) \, V W_{V,i}, \tag{5b}$$

where $W_{U,i}, W_{K,i}, W_{V,i} \in \mathbb{R}^{d_{model} \times d_{head}}$ are again learnable weight matrices and d_{head} is the embedding dimension of each head. Finally, attention is computed by the concatenation of every head attention att_i, followed by an affine transformation:

$$\text{att}(U, K, V) = [att_1; ...; att_{n_{heads}}] \, W_O, \tag{6}$$

where $W_O \in \mathbb{R}^{n_{heads} * d_{head} \times d_{model}}$.

If one wants to model the interdependence of words within a single piece of text, U, K and V are all equal and consist of the text of interest embedded in some vectorial space. This type of attention is often called "self-attention" or "self-alignment" [2], [21]. Conversely, if one seeks the relationship between words from two different passages, then U represents one, while K and V represent the other. In that case, we talk about "cross-attention".

C. Other RNN-free Models

We also identified another QA model [32] that is inspired by the architecture introduced by Vaswani et al. [2]. Their model differs from ours in that it heavily relies on convolutions (46 layers against 2 in FABIR), which approximates it to other CNN NLP models [33], rather than purely attention based models. Although they report high F1 and EM scores (82.7% and 73.3%), our model is almost twice as fast in inference (259 samples/s against 440 in FABIR). Also, their model probably has a higher number of learned parameters due to the increased number of layers.

III. FABIR

In this section, we present FABIR's architecture and the main design decisions we have made to develop a lighter and faster question-answering model. In particular, we introduce the convolutional attention, the column-wise cross-attention, and the reduction layer, which build on the Transformer model [2] to enable its application to question-answering.

A. Embeddings

We model each piece of text at the level of a word, i.e., sentences are defined by a sequence of vectors ω, each one representing a word in a vectorial space $\mathbb{R}^{d_{input}}$. Thus, we build a new representation of question and passage to which we will refer as Ω_Q and Ω_P, respectively.

$$\mathcal{P}, \mathcal{Q} \xrightarrow{\text{embed}} \Omega_P \in \mathbb{R}^{P_{len} \times d_{input}}, \Omega_Q \in \mathbb{R}^{Q_{len} \times d_{input}}, \tag{7}$$

where Q_{len} and P_{len} denote the number of tokens in the question and the passage, respectively. These embeddings are composed by word-level and character-level representations. The former is denoted by $\omega_w \in \mathbb{R}^{100}$ and was imported from the pre-trained embeddings of GloVe "6B" [34]. The latter is denoted by $\omega_c \in \mathbb{R}^{100}$ and is computed for each word as a result of the composition of its characters. Given a word with length l, $C = [c_1, c_2, ..., c_l]$, in which $c_i \in \mathbb{R}^8$ are learned character embeddings, we compute ω_c by convolving C with kernel $H \in \mathbb{R}^{1 \times 5 \times 8 \times 100}$ and applying max-over time pooling

[35]. Finally, we squeeze ω_c values to $[-1, 1]$ using a hyperbolic tangent activation function and pass the concatenation of ω_w and $\tanh(\omega_c)$ through a two-layer Highway-Network [36] to obtain the final representation of a word.

$$\omega = \text{Highway}([\omega_w; \tanh(\omega_c)]). \tag{8}$$

B. Encoder

In contrast to an RNN, FABIR does not process words in sequence, and hence needs to model the position of each word in a sentence differently. We add positional information to each word embedding using a trigonometric encoder as proposed in [2]. Therefore, given a sequence of embedding vectors of even size d_{model}, the position of the i^{th} word is encoded in a vector e_i as follows:

$$e_i = \begin{bmatrix} \sin(i * f_1) \\ \cos(i * f_1) \\ ... \\ \sin(i * f_{d_{model}/2}) \\ \cos(i * f_{d_{model}/2}) \end{bmatrix}, \tag{9}$$

where f_k are scalars, which were chosen according to [2].

The encoding of an embedding matrix Ω is represented by E and the whole operation can be summarized as

$$\mathcal{P}, \mathcal{Q} \xrightarrow{\text{encode}} E_P \in \mathbb{R}^{P_{len} \times d_{model}}, E_Q \in \mathbb{R}^{Q_{len} \times d_{model}}, \tag{10}$$

where d_{model} is the size of each position encoding, which is not necessarily equal to d_{input}.

The encoding E can be summed to Ω to include the information of the position of each word in the text. Indeed, in [2], the final vectorial representation of a piece of text is defined by the sum of the embeddings Ω with the position encoding E, which would require $d_{model} = d_{input}$. However, we introduce a layer that processes embeddings and encodings separately before summing them up. Because we also use this layer to reduce the embedding size from d_{input} to d_{model}, we named it "reduction layer". The architecture of this type of layer is addressed further on.

C. Convolutional Attention

In FABIR the attention mechanism is inspired by the Transformer model introduced in [2]. However, we hypothesize the word-to-word relationship in (1a) fails to capture the complexity of expressions involving groups of words. To facilitate the modeling of the interdependence of surrounding words, we redefine $s_{i,j}$ as

$$s_{i,j} = f(p_{i-\frac{h-1}{2}}, ..., p_{i+\frac{h-1}{2}}, q_{j-\frac{w-1}{2}}, ..., q_{j+\frac{w-1}{2}}), \tag{11}$$

where h and w are the height and width of a convolution kernel. This new type of attention, which we named "convolutional-attention", is entirely defined by the following sequence of steps:

$$logits_i = UW_{U,i}(KW_{K,i})^T, \tag{12a}$$

$$logits_{i,padded} = \text{pad}(logits_i), \tag{12b}$$

$$logits_{conv} = \text{Conv}(logits_{padded}, H), \tag{12c}$$

$$att_{head,i} = \text{softmax}(logits_{conv,i})VW_{V,i}, \tag{12d}$$

$$\text{att}_{\text{conv}}(U, K, V) = [att_1; ...; att_{n_{heads}}]\, W_O, \tag{12e}$$

where Conv represents a single convolutional layer with a trainable kernel $H \in \mathbb{R}^{h \times w \times n_{heads} \times n_{heads}}$ that has height h, width w, and number of filters and channels both equal to n_{heads}. Note that in (12b) zero-padding is applied so that $logits_{conv}$ maintains the same dimension of $logits$.

D. Sublayers

After converting question and passage to their vectorial representation Q and P, we apply a series of operations that we call sublayers. In this section, we introduce each of these operations.

1) Self-attention: Self-attention (att_{self}) is the mechanism that models the interdependence between words in the same piece of text. It has been proven to help relating distant words, which is crucial to understand the long sentences that appear in context paragraphs in SQuAD. In FABIR, self-attention is a sublayer that applies such operation via convolutional attention and is defined as

$$\text{att}_{\text{self}}(P) = \text{att}_{\text{conv}}(P, P, P). \tag{13}$$

2) Column-wise Cross-attention: Cross-attention ($\text{att}_{\text{cross}}$) differs from other types of attention by relating two different pieces of text. Given P and Q, cross-attention of Q over P is defined as

$$\text{att}_{\text{cross}}(P, Q) = \text{att}_{\text{conv}}(P, Q, Q). \tag{14}$$

In contrast to Vaswani et al. [2], where the softmax in (12d) is applied in a row-wise manner, we suggest column-wise cross-attention. More precisely, we sum over i instead of j in (1b). Row-wise softmax is inadequate in QA because, in practice, it computes a weighted average of the question words for every passage word, and thus cannot model the likely scenario in which not every word in the passage is related to the question. In contrast, the column-wise softmax attributes greater weights to passage words that are more closely related to the respective question word, which seems appropriate for the SQuAD task.

Many question-answering models employ cross-attention in both directions: $att_{cross}(P, Q)$ and $att_{cross}(Q, P)$ [7], [21], [22]. However, in FABIR we have observed better results when only the former is used.

3) Feedforward: The feedforward sublayer is solely composed of a neural network with a single hidden layer, which is applied vector-wise. Following the architecture suggested by Vaswani et al. [2], the feedforward sublayer is implemented in (15) with a two-layer neural network:

$$x_{i,out} = \text{ReLU}\left(x_i\, W_1 + b_1\right) W_2 + b_2, \tag{15}$$

where $W_1 \in \mathbb{R}^{d_{model} \times d_{hidden}}$, $W_2 \in \mathbb{R}^{d_{hidden} \times d_{model}}$, $b_1 \in \mathbb{R}^{1 \times d_{hidden}}$ and $b_2 \in \mathbb{R}^{1 \times d_{model}}$ are all trainable parameters, d_{hidden} is the dimension of the hidden layer in (15) and $\text{ReLU}(x) = \max(0, x)$.

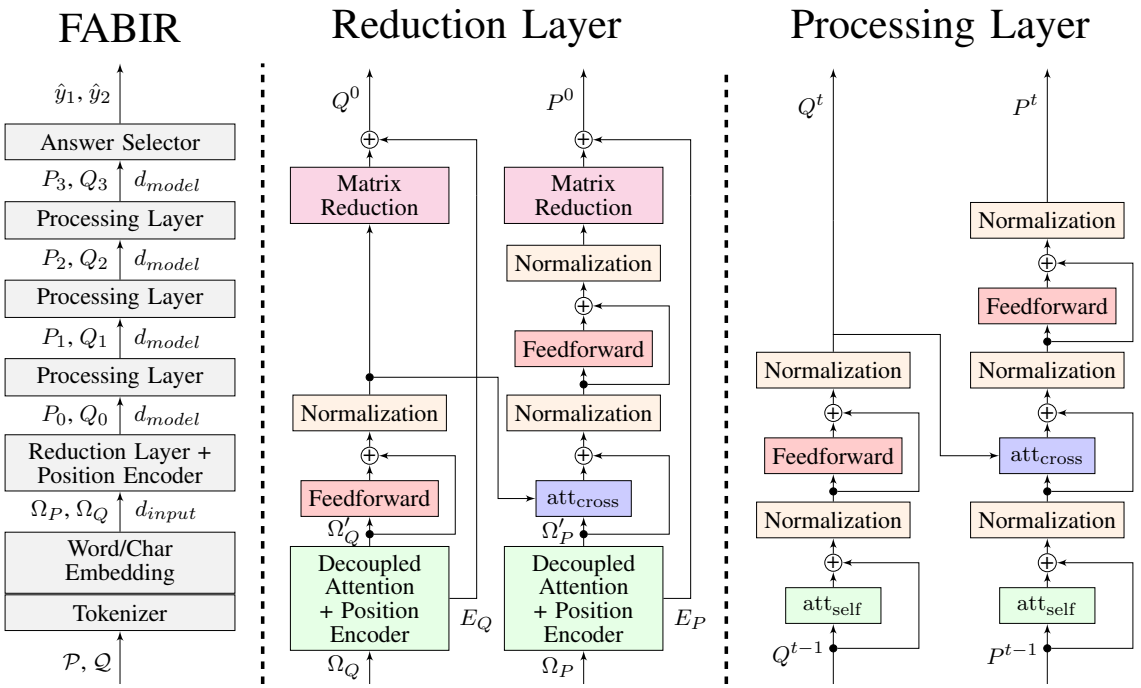

Fig. 1. On the left, a block diagram representation of FABIR, which receives as input the raw passage and question texts, $\mathcal{P}$ and $\mathcal{Q}$. The passage and question embedding matrices are Ω_P and Ω_Q, respectively, and they both have an embedding dimension of d_{input}, which is a result of the tokenization, followed by the word/char embedding process. After the layer reduction, subsequent representations of P and Q have embedding size d_{model} and already include the encoding of word positions. Finally, $\hat{y}_1$ and $\hat{y}_2$ are the indices, which define the answer to the passage-question pair $\mathcal{P}$ and $\mathcal{Q}$. On the center, a block representation of the reduction layer, which is the third layer in FABIR's pipeline. Finally, on the right side, it is the processing layer, which is similar to the reduction layer except by the absence of the "matrix reduction" sublayer and the substitution of the decoupled attention by a self-attention block.

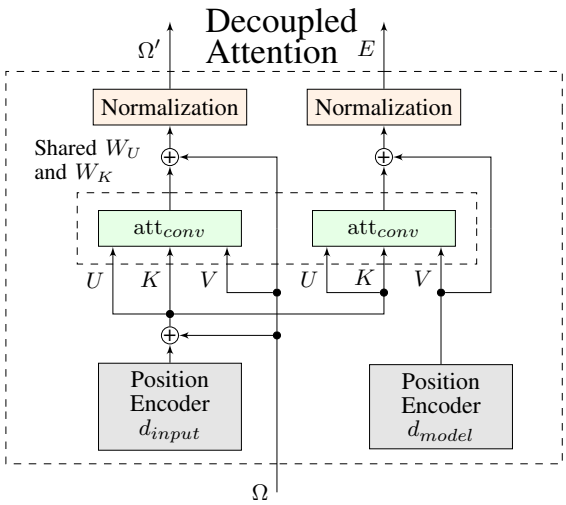

Fig. 2. Decoupled Attention. In contrast to previous attention mechanisms, this structure computes the embedding matrix $\Omega' \in \mathbb{R}^{\Omega_{len} \times d_{input}}$ and the encoder matrix $E \in \mathbb{R}^{\Omega_{len} \times d_{model}}$ separately. Shared weights W_U and W_K and same inputs U and K allow us to compute the convolutional attention pipeline only once until (12c), as in self-attention.

4) Normalization: This operation is also applied vector-wise and it normalizes the embedding of each word so that its variance and mean are reduced to 1 and 0, respectively. The primary goal of layer normalization is to accelerate training as shown in [37], [38].

E. Layers

A layer $\mathcal{L}$ is a combination of sublayers that produce a transformation in the representation of question and passage:

$$P^{l+1}, Q^{l+1} = \mathcal{L}(P^l, Q^l). \tag{16}$$

Typically, a layer is composed of self-attention with shared weights applied to P and Q individually, followed by cross-attention and feedforward sublayers. This standard layer is called "processing layer" and is illustrated in Figure 1. Note that, to facilitate training, every sublayer is followed by normalization.

FABIR is formed by stacking layers on top of each other as shown in Figure 1. Not counting the pre-processing and answer selector layers, our best performing model was composed of four layers, of which the last three are processing layers and the first is a reduction layer.

F. Reduction Layer

The SQuAD dataset is relatively small for the training of word embeddings, and pre-trained word vectors have been favored in the literature [21]. Nonetheless, we observed that the new architecture introduced by Vaswani et al. [2] is more susceptible to overfitting than RNNs when presented with large embedding sizes. Hence, we needed a method to compress the word representations, and thus facilitate and speed up training by reducing the number of parameters. A straightforward method to reduce the input embedding size is to multiply it by a matrix with the required dimensions:

$$\omega_{model} = W_{Reduction}\omega_{input}, \tag{17}$$

where $\omega_{model} \in \mathbb{R}^{1\times d_{model}}$, $\omega_{input} \in \mathbb{R}^{1\times d_{input}}$ are the embedding vectors, and $W_{Reduction} \in \mathbb{R}^{d_{model}\times d_{input}}$ is a weight matrix, to which we refer as "reduction matrix".

Although matrix reduction is quite simple, it discards information before any processing and hence might hamper performance by preventing the network from using some relevant data. To incorporate that information before discarding it, we could add a large processing layer followed by a matrix reduction, but our experiments have shown that this approach does not yield positive results in FABIR. Our interpretation of that behavior is that the position encoding is somehow dissolved in the matrix reduction process.

A possible solution is to process embeddings of size d_{input} and encodings of size d_{model} independently and thus limit the reduction operation to the embeddings. We then suggest decoupled attention, a mechanism that allows us to apply self-attention to embeddings and encodings separately to preserve their different sizes, as described in Figure 2. Both operations use the full embedding $\Omega \in \mathbb{R}^{\Omega_{len}\times d_{input}}$, but the decoupled attention outputs embeddings with size d_{input} and encodings with final size d_{model}.

After applying decoupled attention with shared weights in P and Q, we add a full processing layer for the embeddings Ω' with size d_{input}. That layer is equivalent to a regular processing layer $\mathcal{L}$, but it processes only Ω', leaving the encoding E untouched. Finally, we use a reduction matrix to scale Ω' down to d_{model} and add the encoder matrix E of same size, which is the output from the decoupled attention sublayer. Figure 1 describes this whole process, which we named "reduction layer".

G. Answer Selection

Given that in SQuAD the answer is always contained in the supporting paragraph $\mathcal{P}$, the output of the model is merely the indices $\hat{y}_1$ and $\hat{y}_2$ that represent the first and the last word of the answer, respectively. Therefore, we can model the answer as two probability distributions π_1 and π_2 over the passage $\mathcal{P}$, and train the model to minimize the negative log-likelihood. Given the true indices y_1 and y_2, the cost function is then defined as follows:

$$J = -(y_1 \log(\pi_1) + y_2 \log(\pi_2)). \tag{18}$$

To compute the cost function, we apply a two layered convolutional neural network with hidden layer size 32 and output size 2, as we need one dimension for each probability distribution. Both convolutions have kernel size 9 and the activation function (ReLU) is applied only after the first layer. Subsequently, each output dimension passes through a softmax operation to compute the probability distributions $\hat{\pi}^{y_1}$ and $\hat{\pi}^{y_2}$. Finally, selecting the indices $\hat{y}_1$ and $\hat{y}_2$ becomes an optimization problem:

$$\begin{aligned} \underset{i,j}{\text{maximize}} \quad & \hat{\pi}_i^{y_1} * \hat{\pi}_j^{y_2} \\ \text{subject to} \quad & i \leq j < i + 15, \end{aligned} \tag{19}$$

where 15 represents the maximum allowed answer length. This superior limit is imposed to avoid long answers, since short answers are more frequent.

IV. Experimental Results

We have trained our FABIR model during 54 epochs with a batch size of 75 in a GPU NVidia Titan X with 12 GB of RAM. We developed our model in Tensorflow [39] and made it available at https://worksheets.codalab.org/worksheets/0xee647ea284674396831ecb5aae9ca297/ for replicability.

We pre-processed the texts with the NLTK Tokenizer [40]. As suggested in [2], we have chosen the Adam optimizer [41] with the same hyperparameters, except for the learning rate, which was divided by two in our implementation. For regularization, we applied residual and attention dropout [2] of 0.9 in processing layers and of 0.8 in the reduction layer. In the character-level embedding process, a dropout of 0.75 was added before the convolution. Additionally, a dropout of 0.8 was added before each convolutional layer in the answer selector. We set processing layers dimension d_{model} to 100, the number of heads n_{heads} in each attention sublayer to 4, the feed-forward hidden size to 200 in processing layers and 400 in the reduction layer. Convolution kernels in attention sublayers had spatial dimensions 1×5.

A. Architecture Evaluation

To better evaluate FABIR's architecture, we ran controlled tests on each of its key elements. Table I show the results of these experiments regarding the F1 and EM scores, and the Training Time (TT) over 18-epoch runs. This analysis confirms the effectiveness of char-embeddings, as its addition increased the F1 and EM scores, by 2.7% and 3.1%, respectively. Most importantly, when the convolutional attention was replaced by the standard attention mechanism proposed in [2], the performance dropped by 2.4% in F1 and 2.5% in EM. That validates the contribution of this new attention method in building elaborate contextual representations. Moreover, the tests also indicate that the reduction layer is capable of producing useful word representations when compressing the embeddings. Indeed, when we replaced that layer by a standard feedforward layer with the same reduction ratio, there was a drop of 2.1% and 2.5% in the F1 and EM scores, respectively. Finally, we observed that the column-wise cross-attention

outperforms its row-wise counterpart by 2.0% and 1.9% in F1 and EM, respectively. It confirms the intuition that applying the softmax over the passage words is more adequate in QA.

The training times indicate that each one of the new mechanisms introduced in FABIR incurred an increase in the processing cost. Nonetheless, that was outweighed by the improvement in performance, especially because FABIR is still significantly faster than competing RNN models.

TABLE I
ARCHITECTURE VARIATIONS. F1 AND EM SCORES IN THE DEV SET.

Architecture	F1(%)	EM(%)	TT
FABIR	75.6	65.1	2h14m
FABIR without char embedding	72.9	62.0	1h48m
FABIR without convolutional attention	73.2	62.6	1h49m
FABIR without the reduction layer	73.5	62.6	1h59m
FABIR with row-wise cross attention	73.6	63.2	2h08m
FABIR with 2 attention heads	73.8	63.2	2h12m
FABIR with linear answer selector	74.0	62.8	2h10m
FABIR with 4 layers $\mathcal{L}_{Q\to X}$	75.5	64.7	2h47m
FABIR with 2 layers $\mathcal{L}_{Q\to X}$	75.0	64.1	1h55m

B. FABIR vs BiDAF

This section compares our model to traditional RNN-based question-answering models. To have a comprehensive comparison, we took a state-of-the-art model [21] developed in Tensorflow [39] that had its code openly available at https://github.com/allenai/bi-att-flow. That way, we could run our experiments with both models in the same piece of hardware to have a fair comparison between them. In Table II, the BiDAF scores without parentheses were achieved after training their model for 18,000 iterations of batch size 60 in our hardware. Conversely, values in parentheses are BiDAF's official scores in the SQuAD ranking [42]. Note that for both models, we batch the examples by paragraph length to improve computational efficiency.

TABLE II
COMPARISON BETWEEN FABIR AND BIDAF [21] MODELS.

	FABIR	BiDAF
# of Training Variables	**1,385,198**	2,695,851
Inference Sample/Second	**440**	78
Training Sample/Second	**202**	45
Training Time/Epoch	**7m13s**	32m30s
Training Epochs	54	**12**
Training Time	6h30m	6h30m
F1 in the dev set (%)	**77.6**	77.0 (77.3)
EM in the dev set (%)	67.6	67.3 (**68.0**)

Regarding EM and F1 scores, FABIR and BiDAF showed similar performances. Their similar scores render further comparisons even more telling, because their differences cannot be explained by their overall performances, but exclusively by their architectures. Although both models required similar training times to reach these scores, the time for training one epoch in FABIR was more than four times shorter, which could be useful for tackling larger data sets.

Concerning inference time, FABIR was more than five times faster in processing the 10,570 question-passage pairs in the development data set. FABIR's faster inference is a substantial advantage in large-scale applications, such as information extraction in large corpora. Indeed, when running applications such as search tools or user interfaces, the inference time is critical to tackle real-world problems. Concerning the number of training variables, FABIR has almost 50% fewer parameters than BiDAF, which incurs two major advantages. First, its training time is expected to be shorter, because the number of variables to be updated in every iteration is smaller. Secondly, it has lower memory requirements, which is attractive to applications that dispose of low computational power.

C. FABIR and BiDAF Statistics

In this section we analyze the performance of FABIR and BiDAF in the different types of question in SQuAD.

Figure 3 shows that shorter answers are easier for both models: while they reach more than 75% F1 for answers that are shorter than four words, for answers longer than ten words these scores drop to 60.4% and 67.3% for FABIR and BiDAF, respectively. Long answers are not only more challenging but are also underrepresented in the dataset, which introduces a bias towards short responses. More than 79% of the answers in SQuAD have five words or fewer.

In contrast to what has been observed for answers, longer questions seem not to increase the complexity of the task. In Figure 4, the F1 scores for both models varied by less than 2.5% in the considered question length intervals.

Question type is a strong predictor of performance. Figure 5 shows that both models had their best performance with "when" questions. Answers to these types of questions are

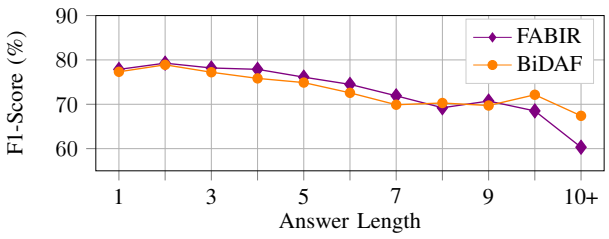

Fig. 3. F1-Score against answer length for FABIR and BiDAF.

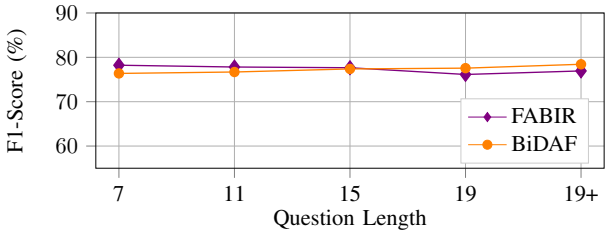

Fig. 4. F1-Score against question length for FABIR and BiDAF.

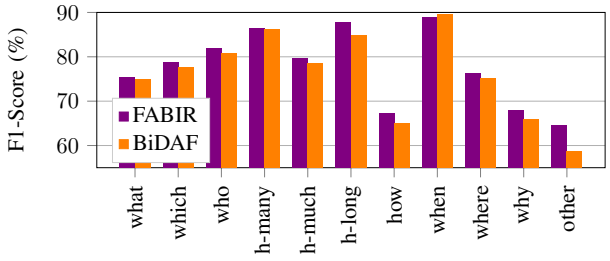

Fig. 5. F1-Score against question type for FABIR and BiDAF.

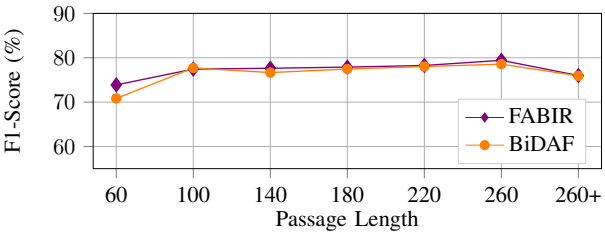

Fig. 6. F1-Score against passage length for FABIR and BiDAF.

easier to infer because they are usually composed of time-related words, such as months, years, seasons or weekdays, which are easier to distinguish from the rest of the text. Together with "when" questions, "how long" and "how many" also proved easier to respond, as they possess the same property of having a smaller universe of possible answers. In contrast to these, "how" and "why" questions resulted in considerably lower F1 and EM scores, as they can be answered by any sentence, and hence require a deeper understanding of the text. Note that "how" questions do not include "how many", "how much" or "how long" questions, whose answers are more predictable.

"Other" questions include alternatives, such as "Name a type of..." or "Does it..." or even questions with typos, such as "Hoe was ...", which should be "How was ...". These questions are more challenging because they might have multiple correct answers and require higher levels of abstraction. For instance, to respond to a question such as "Name an ingredient...", the model would need a deep understanding of the semantics of the word "ingredients" to identify "tomatoes" or "cheese" as possible answers. Questions which expect a "yes" or a "no" as an answer are also difficult because it is not always possible to find those words in a snippet from the passage.

Figure 6 shows the performance of FABIR and BiDAF against the passage length. It is curious that shorter passages showed the worst performance for both models. It is hard to interpret that result as, intuitively, one would expect brief passages to be easier to interpret. One possible explanation is that short passages give fewer options of simple questions, such as "when", "who", or "how many", and the annotators of the dataset had to resort to more elaborate alternatives.

TABLE III
EM AND F1 SCORES IN THE TEST SET FOR BEST PUBLISHED SINGLE MODELS IN THE SQUAD LEADERBOARD [42]

Model	EM (%)	F1 (%)
Reinforced Mnemonic Reader [3]	79.545	86.654
MEMEN [4]	78.234	85.344
FRC [32]	76.240	84.599
RaSoR + TR + LM [5]	77.583	84.163
Stochastic Answer Networks [6]	76.828	84.396
r-net [7]	76.461	84.265
FusionNet [8]	75.968	83.900
DCN+ [9]	75.087	83.081
Conductor-net [10]	74.405	82.742
BiDAF + Self Attention [11]	72.139	81.048
smartnet [12]	71.415	80.160
Ruminating Reader [13]	70.639	79.456
jNet [14]	70.607	79.821
ReasoNet [15]	70.555	79.364
Document Reader [16]	70.733	79.353
RaSoR [17]	70.849	78.741
FastQAExt [18]	70.849	78.857
Multi-Perspective Matching [19]	70.387	78.784
SEDT [20]	68.163	77.527
FABIR (Ours)	**67.744**	**77.605**
BiDAF [21]	67.974	77.323
Dynamic Coattention Networks [22]	66.233	77.896
Match-LSTM with Bi-Ans-Ptr [23]	64.744	73.743
Fine-Grained Gating [24]	62.446	73.327
OTF dict+spelling [25]	64.083	73.056
Dynamic Chunk Reader [26]	62.499	70.956

V. CONCLUSION AND FUTURE WORK

The experiments validate that attention mechanisms alone are enough to power an effective question-answering model. Above all, FABIR proved roughly five times faster at both training and inference than BiDAF, a competing RNN-based model with similar performance [21]. These results strengthen some of FABIR's compelling advantages, notably, an architecture that is both more parallelizable and lighter, with half of the number of parameters in comparison to BiDAF [21].

FABIR also brings three significant contributions to this new class of neural network architectures. The convolutional attention, the reduction layer, and the column-wise cross-attention individually increased the model's F1 and EM scores by more than 2%. Moreover, being thoroughly compatible with the Transformer [2], these new mechanisms are valuable assets to further developments in attention models. In fact, an intriguing line for future research is to evaluate their impact on other NLP tasks, such as machine translation or parsing.

Although FABIR is still far from surpassing the models at the top of the SQuAD leaderboard (Table III), we believe that its faster and lighter architecture already make it an attractive alternative to RNN-based models, especially for applications with limited processing power or that require low-latency. Also, being a distinct technique, FABIR might have low correlation with existing RNN-based models, increasing the potential of ensemble methods. How to combine FABIR with other systems is then an interesting topic for future research in diverse NLP applications.

REFERENCES

[1] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ Questions for Machine Comprehension of Text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, 2016, pp. 2383–2392.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *arXiv preprint arXiv:1706.03762*, 2017.

[3] M. Hu, Y. Peng, and X. Qiu, "Reinforced Mnemonic Reader for Machine Comprehension," *arXiv preprint arXiv:1705.02798*, 2017.

[4] B. Pan, H. Li, Z. Zhao, B. Cao, D. Cai, and X. He, "MEMEN: Multi-layer Embedding with Memory Networks for Machine Comprehension," *arXiv preprint arXiv:1705.02798*, 2017.

[5] S. Salant and J. Berant, "Contextualized Word Representations for Reading Comprehension," *arXiv preprint arXiv:1712.03609*, 2017.

[6] X. Liu, Y. Shen, K. Duh, and J. Gao, "Stochastic Answer Networks for Machine Reading Comprehension," *arXiv preprint arXiv:1712.03556*, 2017.

[7] M. R. A. Natural Language Computing Group, "R-Net: Machine Reading Comprehension with Self-Matching Networks," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Vancouver: Association for Computational Linguistics, 2017, pp. 189–198.

[8] H.-Y. Huang, C. Zhu, Y. Shen, and W. Chen, "FusionNet: Fusing via Fully-Aware Attention with Application to Machine Comprehension," *arXiv preprint arXiv:1711.07341*, 2017.

[9] C. Xiong, V. Zhong, and R. Socher, "DCN+: Mixed Objective and Deep Residual Coattention for Question Answering," in *2018 International Conference on Learning Representations*, Vancouver, 2018.

[10] R. Liu, W. Wei, W. Mao, and M. Chikina, "Phase Conductor on Multi-layered Attentions for Machine Comprehension," *arXiv preprint arXiv:1710.10504*, 2017.

[11] C. Clark and M. Gardner, "Simple and Effective Multi-Paragraph Reading Comprehension," *arXiv preprint arXiv:1710.10723*, 2017.

[12] Z. Chen, R. Yang, B. Cao, Z. Zhao, D. Cai, and X. He, "Smarnet: Teaching Machines to Read and Comprehend Like Human," *arXiv preprint arXiv:1710.02772*, 2017.

[13] Y. Gong and S. R. Bowman, "Ruminating Reader: Reasoning with Gated Multi-Hop Attention," *arXiv preprint arXiv:1704.07415*, 2017.

[14] J. Zhang, X.-D. Zhu, Q. Chen, L.-R. Dai, S. Wei, and H. Jiang, "Exploring Question Understanding and Adaptation in Neural-Network-Based Question Answering," *arXiv preprint arXiv:1703.04617*, 2017.

[15] Y. Shen, P.-S. Huang, J. Gao, and W. Chen, "ReasoNet: Learning to Stop Reading in Machine Comprehension," *arXiv preprint arXiv:1609.05284*, 2016.

[16] D. Chen, A. Fisch, J. Weston, and A. Bordes, "Reading Wikipedia to Answer Open-Domain Questions," *arXiv preprint arXiv:1704.00051*, 2017.

[17] K. Lee, T. Kwiatkowski, A. P. Parikh, and D. Das, "Learning Recurrent Span Representations for Extractive Question Answering," *arXiv preprint arXiv:1611.01436*, 2016.

[18] D. Weissenborn, G. Wiese, and L. Seiffe, "FastQA: A Simple and Efficient Neural Architecture for Question Answering," *arXiv preprint arXiv:1703.04816*, 2017.

[19] Z. Wang, H. Mi, W. Hamza, and R. Florian, "Multi-Perspective Context Matching for Machine Comprehension," *arXiv preprint arXiv:1612.04211*, 2016.

[20] R. Liu, J. Hu, W. Wei, Z. Yang, and E. Nyberg, "Structural Embedding of Syntactic Trees for Machine Comprehension," *arXiv preprint arXiv:1703.00572*, 2017.

[21] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional Attention Flow for Machine Comprehension," in *2017 International Conference on Learning Representations*, Toulon, France, 2017.

[22] C. Xiong, V. Zhong, and R. Socher, "Dynamic Coattention Networks For Question Answering," in *2017 International Conference on Learning Representations*, Toulon, France, 2017.

[23] S. Wang and J. Jiang, "Machine Comprehension Using Match-LSTM and Answer Pointer," *arXiv preprint arXiv:1608.07905*, 2016.

[24] Z. Yang, B. Dhingra, Y. Yuan, J. Hu, W. W. Cohen, and R. Salakhutdinov, "Words or Characters? Fine-grained Gating for Reading Comprehension," *arXiv preprint arXiv:1611.01724*, 2016.

[25] D. Bahdanau, T. Bosc, S. Jastrzebski, E. Grefenstette, P. Vincent, and Y. Bengio, "Learning to Compute Word Embeddings On the Fly," *arXiv preprint arXiv:1706.00286*, 2017.

[26] Y. Yu, W. Zhang, K. S. Hasan, M. Yu, B. Xiang, and B. Zhou, "End-to-End Reading Comprehension with Dynamic Answer Chunk Ranking," *arXiv preprint arXiv:1610.09996*, 2016.

[27] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation By Jointly Learning To Align and Translate," *arXiv preprint arXiv:1409.0473*, 2014.

[28] R. Socher, D. Chen, C. Manning, D. Chen, and A. Ng, "Reasoning With Neural Tensor Networks for Knowledge Base Completion," in *Advances in Neural Information Processing Systems 26*. Lake Tahoe, Nevada: Curran Associates, Inc., 2013, pp. 926–934.

[29] S. Wang and J. Jiang, "Learning Natural Language Inference with LSTM," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego: Association for Computational Linguistics, 2016, pp. 1442–1451.

[30] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, 2015, pp. 1412–1421.

[31] D. Britz, A. Goldie, M.-T. Luong, and Q. V. Le, "Massive Exploration of Neural Machine Translation Architectures," *arXiv preprint arXiv:1703.03906*, 2017.

[32] A. Wei Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, Q. V. Le, and L. Quoc, "Fast and Accurate Reading Comprehension by Combining Self-Attention and Convolution," in *2018 International Conference on Learning Representations*, Vancouver, 2018.

[33] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional Sequence to Sequence Learning," *arXiv preprint arXiv:1705.03122*, 2017.

[34] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Doha: Association for Computational Linguistics, 2014, pp. 1532–1543.

[35] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-Aware Neural Language Models." in *Thirtieth AAAI Conference on Artificial Intelligence*. Phoenix, Arizona: AAAI Press, 2016, pp. 2741–2749.

[36] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway Networks," *arXiv preprint arXiv:1505.00387*, 2015.

[37] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37. Lille, France: The International Machine Learning Society, 2015, pp. 448–456.

[38] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[39] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," *arXiv preprint arXiv:1603.04467*, 2016.

[40] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. Sebastopol, California: O'Reilly, 2009.

[41] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *2015 International Conference on Learning Representations*, San Diego, 2015.

[42] P. Rajpurkar. (2018, Apr.) The Stanford Question Answering Dataset. [Online]. Available: https://rajpurkar.github.io/SQuAD-explorer/