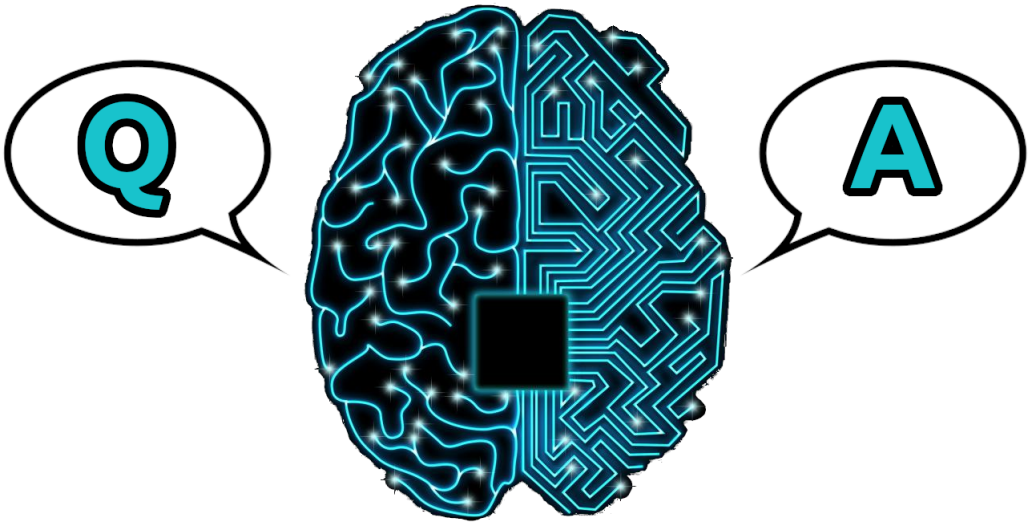
Question Answering on SQuAD dataset



Master in Artificial Intelligence - University of Bologna

A.Y. 2020-2021

Authors:

- Lorenzo Mario Amorosa
- Andrea Espis
- Mattia Orlandi
- Giacomo Pinardi

Executive summary

In this project we have addressed the problem of Question Answering (QA) using the SQuAD dataset. We started from the vanilla Bi-Directional Attention Flow (BiDAF) model, and then we developed and tested several variants. The best variant we obtained comprises the following features: a non-learnable character embedding layer based on the one-hot encoding of the most frequent characters, a convolutional highway network, and a simple constraint on the output span. While the baseline obtained an *f1* score of 0.43, these changes allowed our best variant to reach an *f1* score of 0.60.

1 Background

Question Answering (QA) task has gained significant popularity over the past years. The implementations of such systems have varied across the years, starting from knowledge-based technologies to deep learning approaches based on recurrent neural networks, attention mechanisms and transformers.

In particular, the attention mechanism is often implemented as follows:

1. the computed attention weights are used to extract the most relevant information from the context for answering the question by summarizing it into a fixed-size vector;
2. they are temporally dynamic, since the attention weights of a certain time step are a function of the attended vector at the previous time step;
3. they are usually uni-directional, namely the query attends on the context.

We have implemented the Bi-Directional Attention Flow (BiDAF) network (Seo et al. 2018): it consists in a hierarchical multi-stage architecture for modeling the representations of the contexts and the queries at different levels of granularity. These levels include character-level embedding following Kim 2014, word-level embedding with GloVe (Pennington, Socher, and Manning 2014) and contextual embedding using recurrent neural networks.

Notably, it tries to improve some issues of regular attention:

1. the attention layer does not summarize the context into a fixed-size vector, but instead it computes attention at each time step; then, the attended vector, along with the representations from previous layers, is allowed to flow to the subsequent layer (called *modelling layer*);
2. the attention mechanism is memory-less, namely at each time step the attention is a function of only the context and the query from the current time step (this simplification should lead to the division of labor between the attention layer, focusing on learning the attention between context and query, and the modelling layer, focusing on learning the interaction within the query-aware context representation).
3. the attention flow is in both directions, from context to query and vice versa, providing complementary information to each other.

In this work we have trained our model using the Stanford Question Answering Dataset (SQuAD) from Rajpurkar et al. 2016.

The models are implemented in PyTorch and the code is available on GitHub.

2 System description

The hierarchical multi-stage architecture of the BiDAF model consists of six layers that are described in the following sections.

2.1 Character Embedding Layer

This layer performs the context and query embeddings at the character level. We tested two implementations of Character Embedding Layer, discussed below.

Baseline

In the baseline (from *BiDAF*) each character of the sentences is encoded using a trainable 8-dimensional Embedding layer from PyTorch which consists in a lookup table. The embedding of each word is then obtained by summing the embeddings of the characters composing the word itself. After this operation

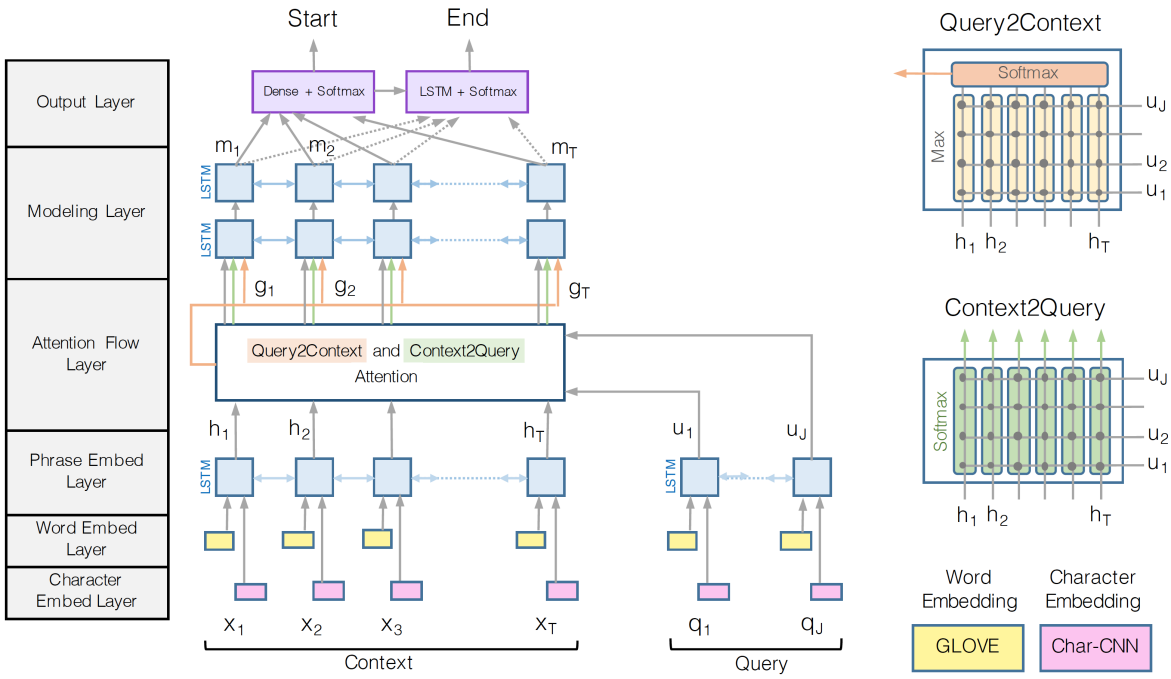


Figure 1: Hierarchical multi-stage architecture of the BiDAF model. Image from Seo et al. 2018.

the tensor dimensions are: batch size, sequence length[1], char embedding dim (8). The tensor is then processed by a convolutional layer with 1×5 kernel and c_{out} output channels (100). The result of the convolution is further processed by summing along the char embedding dimension in order to obtain a 3D tensor (batch size, sequence length, c_{out}).

Our improvements

In this implementation we firstly assign a d-dimensional one-hot vector to each character in the corpus using a non-trainable one-hot encoder. We have analysed the dataset and discovered that there are over 1100 distinct characters, but it would be unfeasible to use 1100-dimensional one-hot vectors. We observed that the 99 most frequent characters represent the 99.92% of the total characters. As a consequence we decided to use a 101-dimensional one-hot encoder:

- one vector is assigned to each of the 99 most frequent characters;
- the all zeros vector represents the padding;
- the vector $[0, ..., 0, 1]$ is assigned to the infrequent and not represented characters (unknowns).

The one-hot encoder produces 4D tensors, whose dimensions are: batch size, sequence length, word length[2] and 101-dim character embedding. These tensors are processed by a convolutional neural network inspired by Kim 2014, called Char Embedder.

The first step is to reshape the tensor in order to have *batch size* $\times$ *sequence length* number of matrices, each with dimension *word length* $\times$ 101 ($n \times d$ in Figure 2). In the first step each word is processed through a 2D convolution with kernel width equal to the character embedding dimension, so the filter is slid only vertically. As a consequence this operation resembles a 1D convolution since there is no horizontal sliding.

In the following step through max-over-time pooling the maximum value is extracted from each of the m channels. These m values are fed to a two-layers fully connected neural network. The output tensor is reshaped to restore the original batch size and sequence length dimensions, obtaining a 3D tensor of shape: (batch size, sequence length, output embedding dim).

The advantage of the encoding provided by the Char Embedder is that every word is represented by a fixed size vector independently from the word length.

[1] We refer to *sequence length* as the maximum number of words in the sentences of the same batch.
[2] We refer to *word length* as the maximum number of characters in words of the same batch.

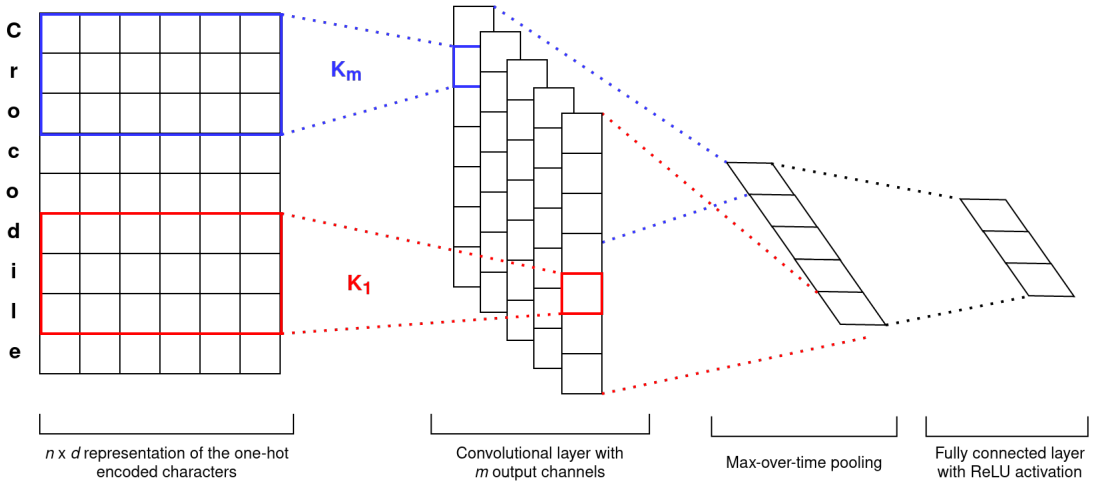


Figure 2: The structure of the Char Embedder.

2.2 Word Embedding Layer

This layer performs the context and query embeddings at the word level and it is implemented as a PyTorch Embedding layer. The weights are initialized using GloVe (Pennington, Socher, and Manning 2014).

This word embedding will be concatenated with the Char Embedder output: as a consequence the embeddings dimensions are chosen so that they are similar, to avoid that one embedding is less represented than the other. For instance, if the GloVe embedding has dimension 50 then also the char embedding has to match this dimension.

The output 3D tensor has shape: (batch size, sequence length, output embedding dim).

2.3 Contextual Embedding Layer

Before the Contextual Embedding Layer the char and word embeddings are concatenated and fed to a Highway Network, based on the work of Srivastava, Greff, and Schmidhuber 2015.

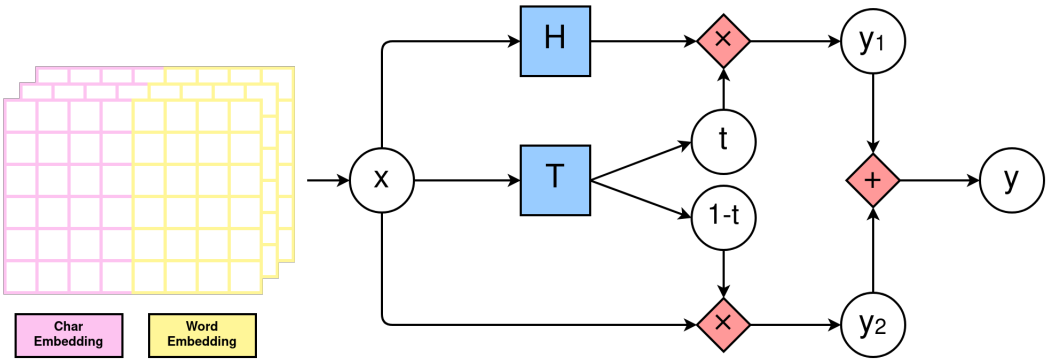


Figure 3: The structure of the Highway Network block. T is the transform gate, H is the transform function.

The concatenation is performed on the sequence length dimension, such that each row contains both the char and the word embeddings of the corresponding word. The resulting tensor is then processed by the Highway Network, which consists in two blocks that are both a generalization of the residual block. In each block there are two branches, whose activations are re-weighted according to the output $t \in [0, 1]$ of the transform gate T:

- in the first branch the input x is transformed using a neural network H and re-weighted according

to t;
- in the second branch the input x is carried out as it is and re-weighted according to $1 - t$.

The activations from the two branches are then added together. The operations performed by a single block are summarized by the following formula:

$$y = H(x) \cdot t + x \cdot (1 - t) \tag{1}$$

Finally, the Highway Network output is processed by a bidirectional Recurrent Neural Network (GRU).

We tested two implementations of such Highway Network, discussed below.

Baseline

The baseline relies on a dense Highway Network implemented using fully-connected layers (cfr. *BiDAF*) both for the transform gate T and the transform function H. We can observe that in this implementation the weighting value t is distinct for each element of the activation tensor.

Our improvements

In our improved implementation the transform gate output is computed using a 2D convolution and a mean operation so that t is shared among all the elements of the same batch. The transform function H processes the input x using a 2D convolution with 5×5 kernel.

2.4 Attention Flow Layer

This layer is responsible of merging information between context and query through a peculiar attention mechanism called **bidirectional attention flow**. It consists in computing attention both from context to query and vice versa by using a shared similarity matrix $\mathbf{S} \in \mathbb{R}^{c \times q}$,[3] which we obtain from the contextual embeddings $\mathbf{H} \in \mathbb{R}^{2d \times c}, \mathbf{U} \in \mathbb{R}^{2d \times q}$, respectively of the context and the query; each element of $\mathbf{S}$ is computed as $\mathbf{S}_{ij} = \alpha(\mathbf{H}_{:i}, \mathbf{U}_{:j})$, where $\alpha(\cdot)$ is defined as a linear layer taking in input the tensor $\text{concat}(\mathbf{H}_{:i}, \mathbf{U}_{:j}, \mathbf{H}_{:i} \odot \mathbf{U}_{:j})$.

2.4.1 Context-to-Query Attention

The Context-to-Query (C2Q) attention determines which query words are more relevant w.r.t. a context word. Such information is contained in the matrix $\tilde{\mathbf{U}} \in \mathbb{R}^{2d \times c}$, which is computed in the following way: first, for each context word i a vector of attention weights $\mathbf{a}_i \in \mathbb{R}^q$ is computed by applying a softmax on the i-th row of $\mathbf{S}$; then, each column i of $\tilde{\mathbf{U}}$ is computed as the vector-matrix product between $\mathbf{a}_i$ and $\mathbf{U}$:

$$\tilde{\mathbf{U}}_{:i} = \sum_j \mathbf{a}_{ij} \cdot \mathbf{U}_{:j}$$

2.4.2 Query-to-Context Attention

The Query-to-Context (Q2C) attention is about determining the context word on which the query must focus: for each context word we compute the maximum similarity w.r.t. all the query words; these values, after applying a softmax, are used as attention weights.

Given the vector of attention weights $\mathbf{b} \in \mathbb{R}^c$, obtained as explained above, we perform a vector-matrix product to compute the intermediate vector $\tilde{\mathbf{h}} \in \mathbb{R}^{2d}$:

$$\tilde{\mathbf{h}} = \sum_i \mathbf{b}_i \cdot \mathbf{H}_{:i}$$

This vector is then replicated c times across the columns, in order to obtain the matrix $\tilde{\mathbf{H}} \in \mathbb{R}^{2d \times c}$.

Finally, we obtain the matrix $\mathbf{G} \in \mathbb{R}^{8d \times c}$, containing the query-aware representation of each context word. Each column of $\mathbf{G}$ is obtained as follows:

$$\mathbf{G}_{:i} = \text{concat}(\mathbf{H}_{:i}, \tilde{\mathbf{U}}_{:i}, \mathbf{H}_{:i} \odot \tilde{\mathbf{U}}_{:i}, \mathbf{H}_{:i} \odot \tilde{\mathbf{H}}_{:i})$$

[3] c is the length of the context embedding, q is the length of the query embedding

2.5 Modeling Layer

The matrix $\mathbf{G}$ is fed as input to a 2-layers bidirectional LSTM which produces as output the matrix $\mathbf{M} \in \mathbb{R}^{2d \times c}$. Such matrix captures the interaction between context words *taking into account the query.* By contrast, the output of the contextual embedding layer captures the interaction between context words *independently from the query.*

2.6 Output Layers

In this section we describe the structure of the output layer which consists in two parallel layers: one to determine, for each token, the probability of being the start of the answer span, and the other, analogously, to determine for each token the probability of being the end of the answer span.

2.6.1 Output layer for the answer span's start

To obtain the probability distribution for the start of the answer span, the following operations are performed:

1. matrix $\mathbf{G}$ and matrix $\mathbf{M}$ are concatenated into a matrix with shape $\mathbb{R}^{10d \times c}$;
2. the concatenation is fed into a linear layer with $10d$ input features and 1 output feature, giving as output a $\mathbb{R}^c$ vector;
3. a softmax is applied on such vector, resulting in vector $\mathbf{p}^{\text{start}}$.

2.6.2 Output layer for the answer span's end

To obtain the probability distribution for the end of the answer span, the following operations are performed:

1. the matrix $\mathbf{M}$ is fed into an additional bidirectional LSTM, which produces as output the matrix $\mathbf{M}^2 \in \mathbb{R}^{2d \times c}$;
2. as before, matrix $\mathbf{G}$ and matrix $\mathbf{M}^2$ are concatenated into a matrix with shape $\mathbb{R}^{10d \times c}$;
3. the concatenation is fed into another linear layer with $10d$ input features and 1 output feature, giving as output a $\mathbb{R}^c$ vector;
4. a softmax is applied on such vector, resulting in vector $\mathbf{p}^{\text{end}}$.

Additionally, we also tested a variant in which $\mathbf{p}^{\text{start}}$ is concatenated with $\mathbf{M}$ before being fed into the LSTM in order to take into account the probability of the start of the answer span when determining the probability of the end of the answer span.

2.6.3 Loss function and optimizer

The following loss function, which sums the negative log probabilities of the true start and end indices (averaged over all examples), was defined to train BiDAF:

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

where:

- θ is the set of BiDAF's trainable parameters;
- N is the number of all the samples;
- y_i^{start} and y_i^{end} are, respectively, the true start and end indices of the answer span for the i-th sample;
- $\mathbf{p}_k^{\text{start}}$ and $\mathbf{p}_l^{\text{end}}$ are, respectively, the probability that the k-th token is the start and that the l-th token is the end of the answer span.

We trained the model using Adam optimizer, with a learning rate of $5 \cdot 10^{-3}$ and mini-batches with size 8.

3 Experimental setup and results

All tests were performed on a NVIDIA GeForce RTX 2080 Super-MaxQ GPU, with 8 GB of VRAM. As a baseline, we trained the BiDAF model described in the paper:

- learnable character embedding layer;
- LSTM layers;
- fully-connected highway network;
- dropout rate of 0.2;
- no concatenation of $\mathbf{p}^{\text{start}}$ when computing $\mathbf{p}^{\text{end}}$.

The training results are shown in the figures 4 and 5.

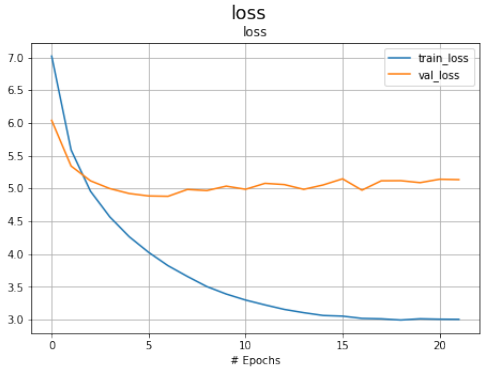


Figure 4: Loss trend of the baseline.

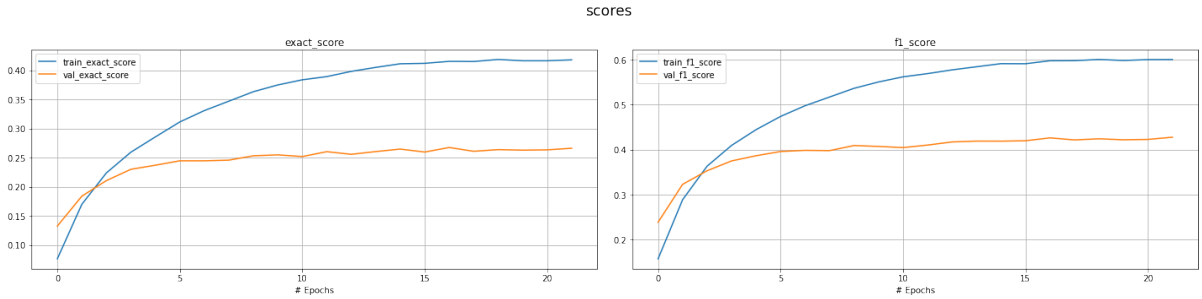


Figure 5: Scores trend of the baseline.

As it can be seen, while the training loss decreases smoothly from ~ 7.0 to ~ 3.0, the validation loss initially decreases from ~ 6.0 to ~ 5.0 but then it starts oscillating; the validation exact score and *f1* score experience a similar trend, settling around ~ 0.25 and ~ 0.4, respectively.

Additionally, we developed and tested the following variants:

- **Variant 1**: convolutional highway network and concatenation of $\mathbf{p}^{\text{start}}$ when computing $\mathbf{p}^{\text{end}}$.
- **Variant 2**: as the variant 1, but with a non-trainable character embedding layer based on the one-hot encoding of the most frequent characters (cfr. subsection 2.1).
- **Variant 3**: as the variant 2, but with no dropout;
- **Variant 4**: as the variant 3, but with the additional constraint $\mathbf{p}^{\text{end}} > \mathbf{p}^{\text{start}}$ (which acts only on the inference).

Concerning the Variant 1, as it can be seen in the figures 6 and 7 the situation is slightly improved, since the validation loss decreases to around ~ 0.47, whereas the validation exact score and *f1* score increases to ~ 0.27 and ~ 0.43, respectively.

In the Variant 2 the non-trainable character embedding layer leads to far better results: as shown in the figures 8 and 9, the validation loss settles around ~ 3.7 and the validation exact score and *f1* score reaches a value of ~ 0.40 and ~ 0.58, respectively.

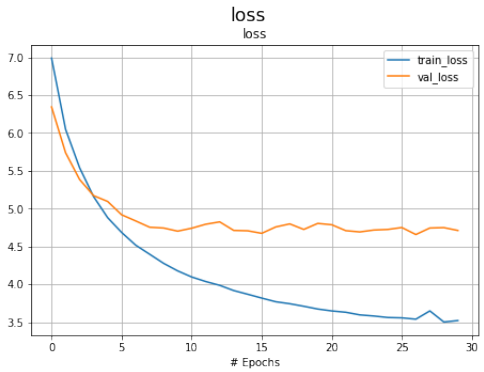


Figure 6: Loss trend of Variant 1.

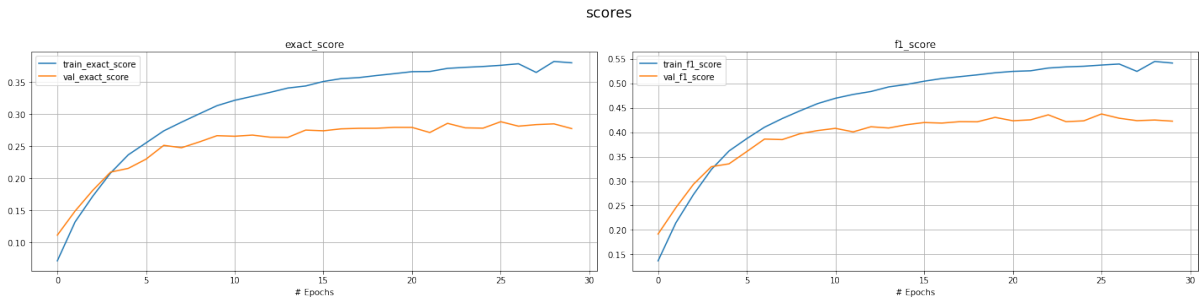


Figure 7: Scores trend of Variant 1.

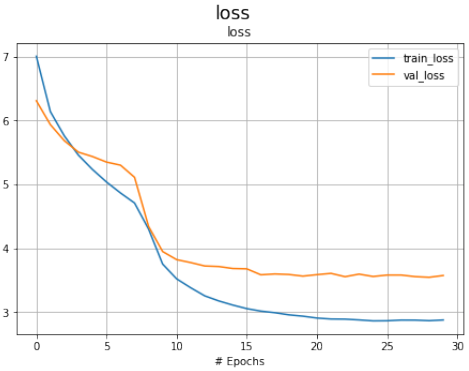


Figure 8: Loss trend of Variant 2.

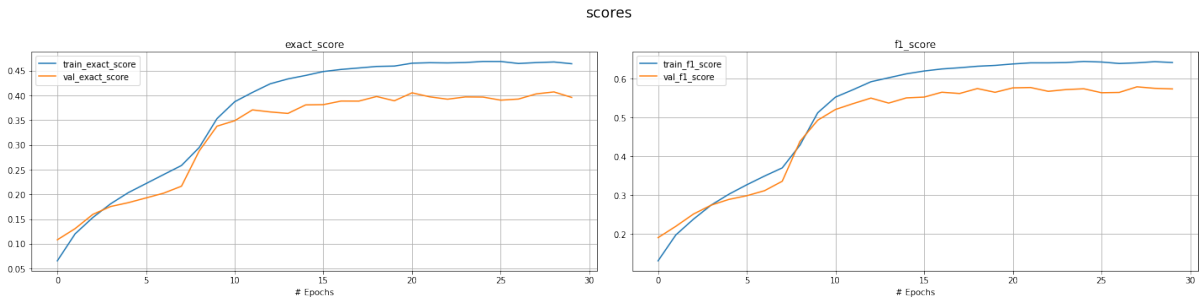


Figure 9: Scores trend of Variant 2.

As far as the Variant 3 is concerned, we can observe from figures 10 and 11 that the absence of dropout results in a more stable training (in fact, as opposed to the trends of the Variant 2, in this case there are no "bumps"), and in slightly better performances (~ 3.6 for the loss, ~ 0.41 and ~ 0.59 for the exact

score and *f1* score).

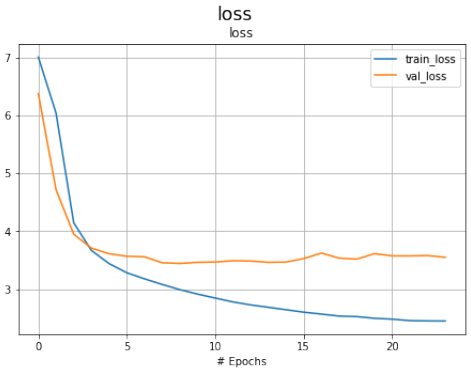


Figure 10: Loss trend of Variant 3.

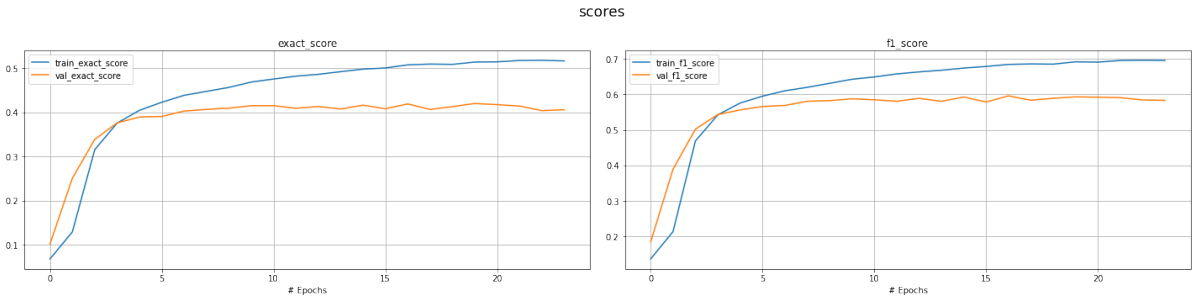


Figure 11: Scores trend of Variant 3.

Finally, the Variant 4, thanks to the constraint, performs even better: while the loss is identical to the Variant 3, the exact score and the *f1* score improve, reaching a value of ~ 0.42 and ~ 0.6 (in fact, as stated before, the constraint has an effect only on the inference).

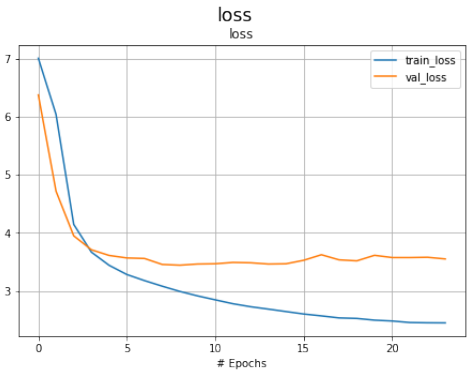


Figure 12: Loss trend of Variant 4.

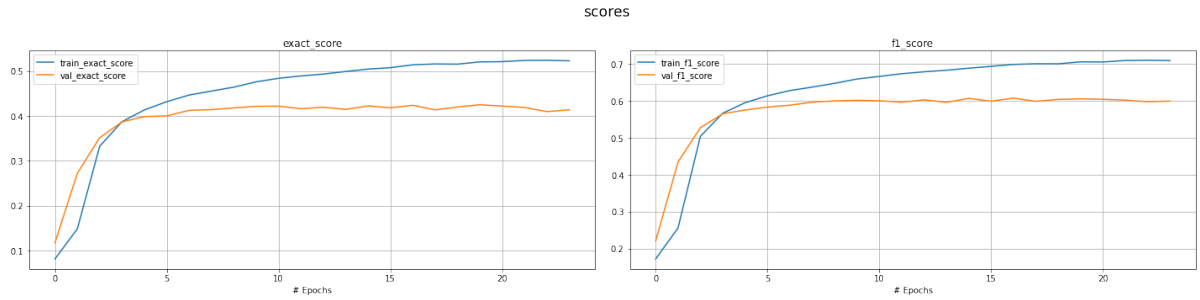


Figure 13: Scores trend of Variant 4.

The results of each model on the test set are represented in the following table: as expected, the Variant 4 outperforms the baseline and the other variants.

Performance on test set					
Model	Baseline	Variant 1	Variant 2	Variant 3	**Variant 4**
Loss	5.07	4.73	3.64	3.67	**3.65**
Exact score	0.266	0.288	0.402	0.412	**0.421**
f1 score	0.432	0.443	0.587	0.599	**0.603**

Table 1: Results obtained on test set by the different BiDAF's variants.

4 Analysis of results

In this section we show and discuss some of the results obtained by using the Variant 4 model discussed in the previous section. Some errors show that the model is able to understand the kind of answer it must be associated to the question, for instance if the answer should be a number or a name, but it gives the wrong answer anyway:

- **Example 1**:
  - **context**: Beyoncè Giselle Knowles-Carted (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer of R&B girl-group Destiny's Child. Managed by her father, Mathew Knowles, the group became one of the world's best-selling girl groups of all time. Their hiatus saw the release of Beyoncè's debut album, Dangerously in Love (2003), which established her as a solo artist worldwide, earned five Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in Love" and "Baby Boy".
  - **Query**: When did Beyoncè start becoming popular?
  - **Correct answer**: in the late 1990s.
  - **Our answer**: 2003).
- **Example 2**:
  - **Context**: Emigrants from Siberia that walked across the Bering land bridge into North America may have had dogs in their company, and one writer suggests that the use of sled dogs may have been critical to the success of the waves that entered North America roughly 12,000 years ago, although the earliest archaeological evidence of dog-like canids in North America dates from about 9,400 years ago. Dogs were an important part of life for the Athabascan population in North America, and were their only domesticated animal. Dogs also carried much of the load in the migration of the Apache and Navajo tribes 1,400 years ago. Use of dogs as pack animals in these cultures often persisted after the introduction of the horse to North America.
  - **Query**: Evidence places dogs in North America when?
  - **Correct answer**: 12,000 years ago.
  - **Our answer**: 9,400 years ago.

To solve this kind of error it could be tried a different use of the Attention.

Some errors are due to the inclusion of words which shouldn't belong to the answer. In some cases only a little information is added to the correct answer, in others it is too much.

- **Example 1**:
  - **Query**: In what city did Beyoncè grow up?
  - **Correct answer**: Houston.
  - **Our answer**: Houston, Texas.
- **Example 2**:
  - **Query**: Time magazine named her one of the most 100 what people of the century?
  - **Correct answer**: influential.
  - **Our answer**: Top Female Artist of the 2000s and their Artist of the Millennium in 2011. Time listed her among the 100 most influential people.

In other cases the error is to not include some words which should belong to the answer:

- **Example 1**:
  - **Query**: Which three countries did Beyonce's song "Work It Out" achieve top ten status?
  - **Correct answer**: UK, Norway,and Belgium
  - **Our answer**: Belgium.
- **Example 2**:
  - **Query**: Where was the Olympic torch lit?
  - **Correct answer**: Olympia, Greece.
  - **Our answer**: Olympia.

To solve these kind of errors it could be used a model which aim is to predict the length of the answer.

In some cases the problem is related to the dataset, meaning that the answer could be correct also if different from the proposed one:

- **Example 1**:
  - **Context**: The latest study using magnetic resonance imaging (MRI) to humans and dogs together proved that dogs have same response to voices and use the same parts of the brain as humans to do so. This gives dogs [...].
  - **Query**: What technology was used to show that dogs respond to voices in the same brain parts as people?
  - **Correct answer**: MRI.
  - **Our answer**: magnetic resonance imaging.
- **Example 2**:
  - **Context**: In Islam dogs are viewed as unclean because they are viewed as scavenger. In 2015 city councillor [...].
  - **Query**: How are dogs viewed in Islam?
  - **Correct answer**: as unclean.
  - **Our answer**: as scavengers.

5 Discussion

In the end, the evidence shows us that exploiting high-level constraints, such as the information that the start span index must be smaller than the end span index used in Variant 4, can lead to significant improvements. Moreover, as shown starting from Variant 1, the output layer is able to learn to constraint the value of $\mathbf{p}^{\text{end}}$ based on the value of $\mathbf{p}^{\text{start}}$. We can expect that adding more constraints can result in further improvements.

One of the major weakness shared by our models is that they are not suited for parallelized training. This is mainly caused by the fact that the models make extensive use of recurrent neural networks (LSTM and GRU). In addition, the baseline model is slowed down by the dense implementation of the Highway Network used in the Contextual Embedding Layer. As a consequence, we introduced the convolutional Highway Network with the purpose of speeding up the training.

It would be possible to improve the models' results by better exploring the hyperparameters' space: e.g. the number of layers and the hidden size of RNNs, the GloVe embedding dimension, the Character Embedding Layer hyperparameters (filters and channels dimensions), the convolutional Highway Network structure (dense layers number, filters dimensions).

The above limitations can be partially solved by using Transformer architectures (Vaswani et al. 2017) which do not employ recurrent modules and therefore are trained more efficiently. In particular, we had a close look at XLNet (Yang et al. 2019), a transformer that overcomes the limitations of the BERT model (Devlin et al. 2018). One of the advantages of XLNet over BERT is that the latter accepts as input a maximum of 512 words while the former does not have this limitation thanks to a recurrent mechanism to handle long text sequences, and this matters since there are input tokens (context + query) with more than 512 words in the SQuAD dataset. As a future work we could compare BiDAF and XLNet performances on the task of question answering.

References

[1] Minjoon Seo et al. *Bi-Directional Attention Flow for Machine Comprehension*. 2018. arXiv: 1611.01603.

[2] Yoon Kim. *Convolutional Neural Networks for Sentence Classification*. 2014. arXiv: 1408.5882.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. *GloVe: Global Vectors for Word Representation*. 2014. URL: http://www.aclweb.org/anthology/D14-1162.

[4] Pranav Rajpurkar et al. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. 2016. arXiv: 1606.05250.

[5] Junki Ohmura. *BiDAF*. Accessed 04 Mar 2021. URL: https://github.com/jojonki/BiDAF.

[6] Rupesh Kumar Srivastava, Klaus Greff, and Jurgen Schmidhuber. *Highway Networks*. 2015. arXiv: 1505.00387.

[7] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762.

[8] Zhilin Yang et al. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2019. arXiv: 1906.08237.

[9] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. arXiv: 1810.04805.