Question Answering on SQuAD 1.1 dataset

Andrea Policarpi - 0000950326
andrea.policarpi@studio.unibo.it

Nicola Amoriello - 0000952269
nicola.amoriello@studio.unibo.it

Nicola Poggialini - 0000949125
nicola.poggialini@studio.unibo.it

Corentin Magyar - 1900089029
corentin.magyar@studio.unibo.it

April 2021

Contents

1. Summary

Question Answering (QA) is a well-known task in the Natural Language Processing (NLP) field. It got a lot of interest among many companies and research groups in the last few years because of its latent potential in understanding unstructured text.

For our task, a QA model refers to a model which takes at least two inputs, a question and a context paragraph containing the answer, and output two values: the start and end indices of the answer in the context.

One of the key factors behind the success of these models is the neural attention mechanism, which enables the system to focus on a targeted area within a context paragraph, that is most relevant to answer the question. By now, lots of others attention mechanisms have been designed, in particular the self-attention one in 2017. This mechanism is the building-block of a new great variety of models called transformers. These are nowadays the state-of-the-art models in NLP.

In our work, we used the SQuAD (Stanford Question Answering Dataset) dataset, which consists of questions related to a paragraph (called context). The answer of a given question is contained in the corresponding context.

In the SQuAD webpage, there is a leaderboard which classifies the most successful models for this task according to two metrics, the F1 score and the Exact Match (EM): looking at these models, we observed that some of the best ones are non-transformer-based. So we decided to focus on both transformer-based and non-transformer-based architectures, in order to compare the results produced by these two different approaches.

In particular, we have implemented the BIDAF model, which doesn't make use of transformers, and some alternatives based on the pre-trained BERT.

We used the F1 score and the EM as metrics to compare our trained models: we observed that there aren't relevant differences between the models that use BERT, but the results achieved with these are considerably better than the ones obtained with the BIDAF model.

2. Background

2.1 Character embedding

In NLP, the most common way to obtain features from tokens is to use an embedding layer. This layer transforms tokens indices into vectors of numbers. Therefore, these tokens are almost always words and not characters. Thus, it is a bit tricky to handle out of vocabulary (OOV) words. In most cases, we simply assign a random vector to these tokens, but sometimes it is not enough. Indeed, this could end up confusing the model.

It is why we need character embedding. This layer helps to better deal with OOV words. Indeed, there are only few characters in a language. By concatenating or processing these characters embedding, we end up with a word embedding that is more reliable than a random vector.

We now need to work at the character level (so we need to create a new vocabulary). Then, we use a simple embedding layer (like in the word embedding part) to obtain fixed-sized characters embedding. We then use a one-dimensional convolutional neural network followed by a max pooling: the kernels will slide over the word (i.e the characters embeddings), we end up with a new vector on which we apply a max pooling (to obtain fixed-sized features). The dimension of the word embedding is equal to the number of kernels used.

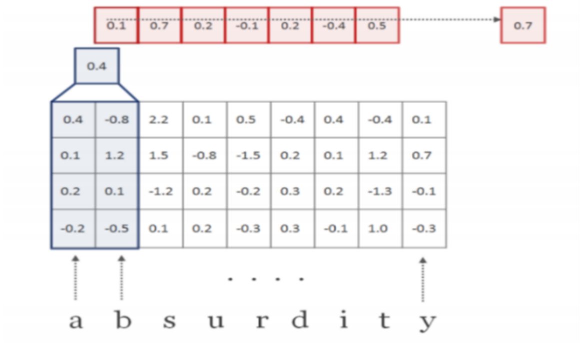

Figure 2.1: character embedding

2.2 Bi-directional attention flow mechanism

Attention mechanism in the QA domain typically uses attention to focus on a small portion of the context and summarize it with a fixed-sized vector. It is what we call a uni-directional attention (because the query is not processed). The BIDAF network introduces a multi-stage hierarchical process that represents the context at different levels of granularity and use bidirectional attention flow mechanism to obtain a query-aware context representation without early summarization.

Hence, the attention is computed in two-directions: from query to context as well as from context to query. Both attentions are derived from a shared similarity matrix S, between the contextual embeddings of the context and the query.

$\alpha(\mathbf{a}, \mathbf{b}) = w^T \times [\,\mathbf{a} \,;\, \mathbf{b} \,;\, \mathbf{a} \circ \mathbf{b}\,]$, where...

- $\mathbf{a}$ and $\mathbf{b}$ are the two input vectors
- $w^T \in \mathbb{R}^{6d}$ is a trainable weight vector
- $\circ$ is element wise multiplication
- $[\,;\,]$ is vector concatenation across row

Figure 2.2: similarity matrix

In a nutshell, the similarity matrix computes the similarity between each feature of the query and each feature of the context and vice versa. (there is a need to tile both inputs matrices). The output matrix has as many rows as they are features in the context, and as many columns as they are features in the query.

Then, we can compute the context to query attention. That signifies to find which query words are most relevant to each context word. To do so, we just need to compute the row-wise softmax of the similarity matrix, which give for each context feature, the attention probability distribution. To finally obtain the new weighted query features, we need to do a dot product between this new attention probability distribution matrix and the query contextual embedding matrix.

To compute the query to context attention, we need this time to compute the max of the similarity matrix across each column. We then compute the softmax of the resulting vector to end up with the attention probability distribution. Finally, we obtain the new weighted context features vector by doing a dot product between this new attention probability distribution vector and the context contextual embedding matrix, we then need to tile the resulting vector as many times as they are features in the context to obtain the final query to context attention matrix.

The final step of this attention mechanism is the megamerge. The contextual embeddings and the attentions matrices are processed together to yield the matrix G. Each column of this matrix can be considered as the query aware representation of each context word.

$\beta(\mathbf{a}, \mathbf{b}, \mathbf{c}) = [\, \mathbf{a} \,;\, \mathbf{b} \,;\, \mathbf{a} \circ \mathbf{b} \,;\, \mathbf{a} \circ \mathbf{c} \,]$, where...

- $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}$ are the three input vectors
- $\circ$ is element wise multiplication
- $[\,;\,]$ is vector concatenation across row

Figure 2.3: similarity matrix

In the formula above:

- a is the context contextual embedding matrix
- b is the context to query attention matrix
- c is the query to context attention matrix

2.3 Highway Network

A highway network is defined by the following equations:

$$x_{proj} = ReLU(W_{proj}x + b_{proj}) \tag{2.1}$$

$$x_{gate} = \sigma(W_{gate}x + b_{gate}) \tag{2.2}$$

$$x_{highway} = x_{gate} * x_{proj} + (1 - x_{gate}) * x \tag{2.3}$$

(* is the element - wise multiplication).

Highway networks can help improve performance because their gating mechanisms can filter out potentially irrelevant information.

2.4 Bidirectional Encoder Representations from Transformer (BERT)

BERT is a model which makes use of transformers: in a nutshell, a transformer is a model which uses attention mechanisms to forward a more complete picture of the whole sequence from an encoder to a decoder at once rather than sequentially. BERT's architecture, then, can be described as a multi-layer bidirectional transformer encoder. The pre-training of this model is based on two different tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). Without going into the details, we can just point out that this pre-training allows to exploit the BERT model for obtaining good results in many different tasks, such as question answering.

2.4.1 BERT for Question Answering

For the question answering task, BERT takes the question and the context as a single packed sequence. The input embeddings are the sum of the token embeddings and the segment embeddings:

- Token embeddings: A [CLS] token is added to the input word tokens at the beginning of the question and a [SEP] token is inserted at the end of both the question and the context.
- Segment embeddings: A marker indicating Sentence A or Sentence B is added to each token. This allows the model to distinguish between sentences (for example, all tokens marked as A belong to the question, and all those marked as B belong to the paragraph).

To fine-tune BERT for a Question-Answering system, it introduces a start vector and an end vector. In the base model, the probability of each word being the start-word is calculated by taking a dot product between the final embedding of the word and the start vector, followed by a softmax over all the words. The word with the highest probability value is considered. An analog process is followed to find the end-word.

3. System description

3.1 BIDAF

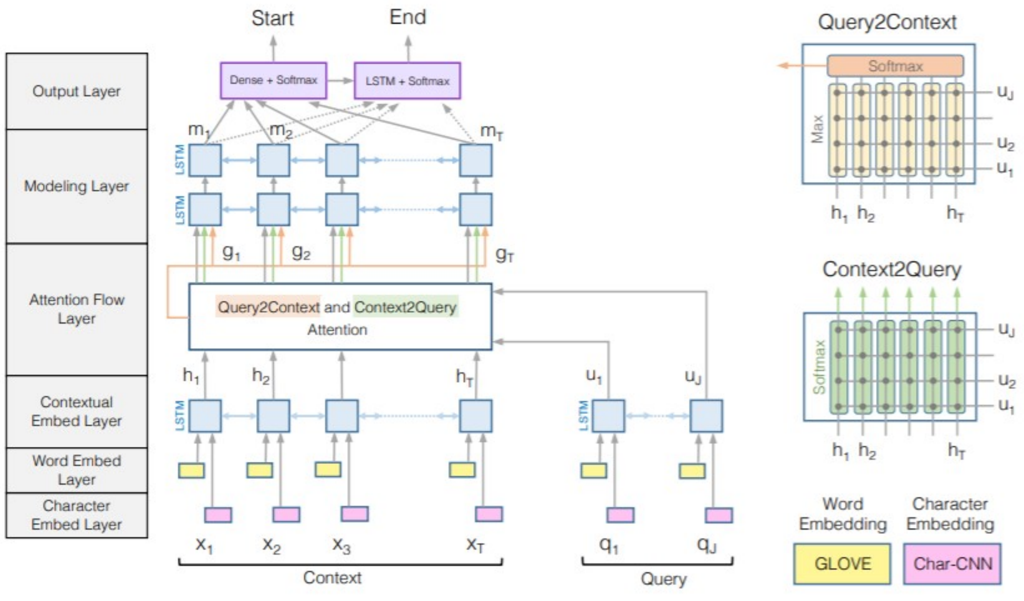

Figure 1: BiDirectional Attention Flow Model *(best viewed in color)*

Figure 3.1: BIDAF model architecture.

As depicted in figure 3.1, our model is composed of three main sections:

1. **Embedding layers**

   The model has 3 embedding layers (word-level, character-level and contextual). The purpose of these layers is to change the representation of words in the query and the context from strings into vectors of numbers.
   Before the contextual layer, embeddings at the word and character level are concatenated and passed through a highway network.

2. **Attention and Modelling layers**

   These layers use several matrix operations to fuse the information contained in the query and the context. The output of these steps is another representation of the context that contains information from the query.

3. **Output layers**

   We then obtain the probability distribution of the start index by means of a final [dense+softmax] layer, while the end index is obtained by a [bidirectional LSTM+softmax] layer that also takes as input the start index information.

3.2 BERT

We implemented the BERT architecture both in the standard version and with the introduction of an "encoder-decoder" tail: the latter is one of the models proposed in [3] for the SQuAD 2.0.

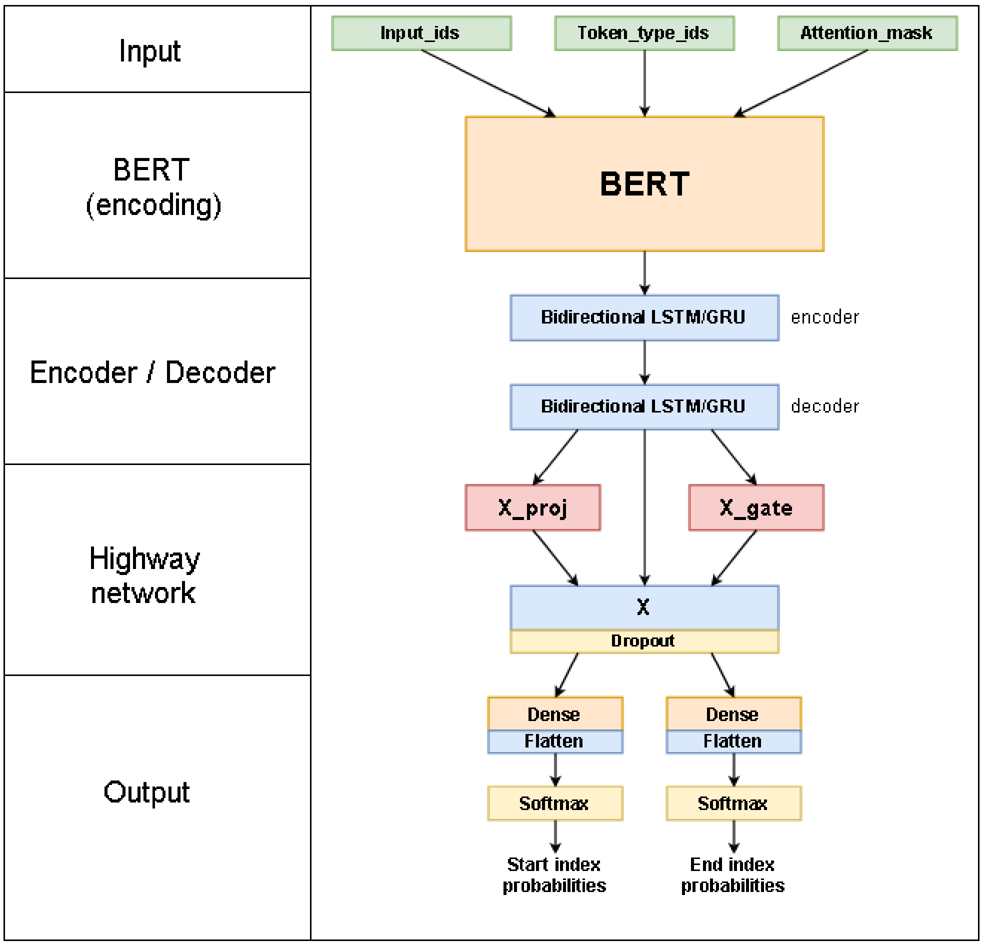

Figure 3.2: BERT with enc/dec model architecture.

The extended model is composed of four main sections:

1. **BERT**

   This section uses a BERT structure to encode the three inputs into a vector representation that will be processed by the following layers.

2. **Encoder/Decoder**

   In order to create an encoder-decoder structure, two recurrent layers are put sequentially: for these, we used a biLSTM, like it's been done in the original paper, and we also experimented with a GRU.

3. **Highway network**

   An highway network is inserted, with the aim of filtering irrelevant information before the final dense layers elaboration. With respect to the original paper, we added an optional dropout layer between the highway network and the following layers.

4. **Output layers**

   Two last dense layers, with a final softmax activation function, are in charge of predicting the start and end index probabilities; differently from the BIDAF model, here the two positions are computed separately.

This comprehends the "extended" model structure; for the vanilla BERT architecture the "Encoder/Decoder" and the "Highway Network" sections are not present, and the BERT block is directly connected to the output layers.

4. Experimental setup and results

The following is a summary of the hyper-parameters used for training the models.

4.1 BERT

- **Split training set**: 0.8
- **Encoding dimension**: 128
- **Decoding dimension**: 64
- **Loss**: Sparse Categorical cross-entropy
- **Optimizer:**: Adam
- **Batch size**: 256
- **Learning Rate**: 5e-5
- **Number of epochs**: 3
- **Dropout**: 0.5

4.2 BIDAF

- **Split training set**: 0.8
- **Loss**: Categorical cross-entropy
- **Optimizer:**: Nadam
- **Batch size**: 10
- **Learning Rate**: 5e-4
- **Number of epochs**: 5
- **Dropout**: 0.2

4.3 Results

This section provides all the results of the models described above. Following the directives given to us by the SQuAD challenge page, we evaluated the effectiveness of our models using F1 and exact match (EM) scores as metrics.

Model	F1	EM
BIDAF model	65.8105	51.0768
BERT Base model (without dropout)	73.0947	58.2248
BERT Base model (with dropout)	72.7792	57.5766
Encoder decoder model with GRU (without dropout)	73.0172	58.1029
Encoder decoder model with GRU (with dropout)	72.8409	57.7484
Encoder decoder model with biLSTM (without dropout)	73.6249	58.0974
Encoder decoder model with biLSTM (with dropout)	73.6399	58.1417

Table 4.1: Results of our implementations on the validation set

As a comparison, in the following table are summarized the human performance and top model performances for the task.

Model	F1	EM
Human performance	91.221	82.304
Top 1 model (LUKE)	95.379	90.202
...	...	...
Top 52 model (BIDAF)	77.323	67.974
...	...	...
Top 60 model (RQA+IDR)	71.389	61.145
...	...	...
Top 63 model (UQA)	64.036	53.698

Table 4.2: Human performance and top model performances for SQuAD 1.1 dataset.

5. Analysis of results

5.1 BERT

To train the BERT-based models, we used the Exact Match on the validation set to decide the number of epochs in order to avoid under-fitting or over-fitting. For all the alternatives, we observed that just 3 epochs were necessary (using a distribution strategy with a TPU, each epoch takes about 3 minutes): in fact, the value of the metric is stuck since the 4th epoch, and even decreases at some point in the following ones.

Comparing the results given by the trained models, we observe that the best score of the F1 is given by the model that uses a biLSTM encoder-decoder and dropout. On the other hand, the best score of the EM is given by the base model that doesn't use dropout. We could state that, overall, the first model is the best one, since its EM score is similar to the one of the base model, but the F1 is a bit better.

However, we can also observe that the difference between the scores is always almost negligible, so there isn't a relevant distinction between the performances of the models: we can conclude that the goodness of the results is given in the major part by the fine-tuning of the pre-trained BERT model.

5.2 BIDAF

The main metric used to train the model is the validation loss. Indeed, we saved the value of this loss at each epoch and we saved the weights each time this value reach a minimum. Thank to that, we observed that 5 epochs is enough to train the model (each epoch takes about 45 minutes).
We used the same architecture of the original model but we ended up with a lower F1-score: we will try to investigate some issues in the discussion part.

5.3 Comparison between approaches

To evaluate the goodness of our results, we used the SQuAD dataset leaderboard in [11]. We can state that the performances of both approaches are satisfying: in fact, our F1 scores obtained with BERT-based models and BIDAF are, respectively, within the top 60 and top 70 in that leaderboard. So, even without the usage of pre-trained architectures, the BIDAF still manages to produce results comparable with the ones obtained with the BERT-based systems: however, the latter ones perform considerably better in our task.

6. Discussion

6.1 Limitations

Despite the relatively low number of epochs needed to train our models, the main problem that arose during both training and test steps resides in the time and memory complexity: while the models are optimized in order to work on a TPU (training time of $\approx$ 10 minutes, prediction time over the entire validation dataset of $\approx$ 1-2 minutes), when used on a GPU the training and the prediction are slow ($\approx$ a few hours for the training, $\approx$ 15-20 minutes for the prediction). This could be a problem since a TPU could not always be available, depending on the application. In addition, since our models are very big (order of magnitude of 10^8 weights), it is difficult to save and share the weights file of the trained models, due to their size of over 400MB. The same problem applies to the embedding matrix file (1 GB).

We could cope with these critical issues by exploring lighter models and/or strategies: for example, for the extended BERT (which is a Transformer-derived architecture) we could substitute the attention mechanism with a log-sparse attention mechanism, which approximates the original behaviour while substantially reducing the number of weights.

For what instead concerns the results, our models achieved performances comparable to the top 50 models of the SotA, but we feel that we can achieve more. Our models seems to be a bit weaker mainly on the EM metric: it seems that they efficiently manage to identify the answer section in the context, but struggle to exactly delimit its boundaries. This weakness could be addressed by using models with more sophisticated input (for example a part-of-speech token mask, obtained from a POS-tagging preprocessing step) in order to locate the answer bounds more accurately.

6.2 Future implementations

6.2.1 BIDAF

Our model yields good F1 and EM scores, but it does not reach the scores of the original model. We could try to investigate some issues that could improve the performances.

- **Use pre-trained charachter embeddings**
  Currently, the embedding layer (at the character level) is trainable. Instead, we could make this layer non-trainable by relying on GloVe to create embeddings. For instance, to create the embedding for the character 'a', we could make an average of all words embeddings containing this letter.
- **Use BERT embeddings**
  We can use BERT embedding to enhance BIDAF for the task of question answering.
- **Use GRU instead of LSTM**
  Training with GRU is usually faster than LSTM. The performance could also be improved.
- **Use gradient clipping**
  It is a common technique in recurrent neural networks because, as gradients are being propagated back in time, they can "explode" or become exponentially large when they are continuously multiplied by numbers greater than one. Clipping these gradients between two manually-set values accounts for this problem.
- **Use others merging techniques different from concatenation**
  Embeddings from the word embedding layer and the character embedding layer are concatenated along the time axis. We could try others techniques, like averaging or summing these vectors.

6.2.2 BERT

Our models achieved good results on the F1 metric, and less but still satisfying results on the EM metric. Here we'll propose some ideas that could improve the results.

- **Lighten the attention mechanism**
  One of our model's problems resides in its time and memory complexity. Since the long training time and the memory burden are mainly caused by the size of the attention matrix (complexity $O(L^2 * d)$, where L is the input length and d is the network size), we could substitute the BERT self-attention mechanism with lighter attention strategies, such as the log-sparse attention mechanism (complexity $O(L * log(L) * d)$). This could approximate our model performances while reducing its complexity burdens.
- **Tune the encoding/decoding dimensions**
  In our experiments, we tried to optimize the performance of the encoder/decoder model focusing on the type of recurrent layers and on the usage of dropout, keeping fixed the dimensions for the encoder and decoder: we could train the models tuning these hyperparameters, in order to see if this can lead to some improvements.
- **Different encoder**
  We could implement a CNN encoder, like it's been done in [3]: in the paper, they just used it in combination with the biLSTM decoder, so it could be interesting to see the results also with the GRU decoder.

Bibliography

[1] Keras: the Python deep learning API. https://keras.io/

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding https://arxiv.org/pdf/1810.04805.pdf

[3] Yuwen Zhang, Zhaozhuo Xu. BERT for Question Answering on SQuAD 2.0 https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/default/15848021.pdf

[4] Danny Takeuchi, Kevin Tran.
Improving SQUAD 2.0 Performance using BERT + X https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/default/15737384.pdf

[5] NLP: Contextualized word embeddings from BERT https://towardsdatascience.com/nlp-extract-contextualized-word-embeddings-from-bert-keras-tf-67ef29f60a7b

[6] Question Answering System with Fine-Tuned BERT https://medium.com/saarthi-ai/build-a-smart-question-answering-system-with-fine-tuned-bert-b586e4cfa5f5#:~:text=To%20fine%2Dtune%20BERT%20for,softmax%20over%20all%20the%20words.

[7] Bi-directional attention flow for machine comprehension https://arxiv.org/pdf/1611.01603.pdf

[8] Medium article about the BIDAF architecture https://towardsdatascience.com/the-definitive-guide-to-bi-directional-attention-flow-d0e96e9e666b

[9] Medium article about character embedding https://towardsdatascience.com/besides-word-embedding-why-you-need-to-know-character-embedding-6096a34a3b10

[10] Github repository with GloVe character embedding https://github.com/minimaxir/char-embeddings

[11] The Stanford Question Answering Dataset https://rajpurkar.github.io/SQuAD-explorer/