Natural Language Processing

SQuAD question answering with Transformers

Alessandro Dicosola
alessandro.dicosola@studio.unibo.it

Pierpasquale Colagrande
pierpasqu.colagrande@studio.unibo.it

Sami Osman
sami.osman@studio.unibo.it

1 Summary

Natural Language Understanding is an ongoing research field in NLP that is trying to achieve superhuman performance on tasks such as question-answering, text classification, machine translation and automated reasoning. The real application are quite important: fake-news checking, assistant-application for people with disabilities, assistant for law documentation such as term of service and so on.

In NLP, starting from non-neural models, performance were increased using neural-model such as linear, convolution and recurrent networks (RNNs, LSTMs, CNNs etc.) reaching its peak with attention mechanisms.

In our work, we used a Transformer-based [1] architecture to solve the question-answering task, using the SQuAD 1.1 dataset [2]. Question answering is about extracting an answer to a given question from a given passage (also called context). The SQuAD dataset provides 100.000+ question-answer pairs regarding 500+ Wikipedia articles that can be used to train a model solving this task.

More specifically, we fine-tuned a pre-trained BERT-based [3] model to extract an embedding from the context-question pairs in order to find the probabilities of each token to be the start or the end of the answer span.

Moreover, we did some experiments to try to improve the performances of the baseline by extending its architecture or by changing the pre-trained model.

The models were then compared using F1 and EM scores computed on the same test set.

2 Background

2.1 Transformers

The **"Attention is all you need"** [1] paper has introduced a game-changing architecture: the **Transformer**, which exploits a self-attention mechanism to solve NLP tasks without the need of using recurrent models.

Until the advent of Transformers, RNN-based architectures allowed to achieve pretty good results on a variety of NLP tasks. Moreover, RNN encoder-decoder architectures with attention mechanisms are still performing very well if accurately designed and trained. The problem with RNNs, however, is in their sequential nature, which does not allow to exploit the parallelism offered by GPUs, **inhibiting parallelization.** Due to this problem, big RNNs are slow to train. Moreover, long range information are lost because of the **bottleneck problem**.

The parallelization problem was partly solved with CNNs and in particular with convolutional sequence models. Furthermore, to avoid the bottleneck problem, attention mechanisms and residual connections were implemented.

On the other hand, Transformers do not have recurrent layers and for this reason they can perfectly exploit GPU parallelization, allowing faster training time, and because they don't use a sequential approach like the RNNs, they can perfectly capture long range dependencies without the bottleneck problem.

Thanks to parallelization, the Transformer can be trained in a fraction of the time with respect to recurrent networks (which require the state of the model at previous timestep), reaching state-of-the-art performance in a more efficient way.

The Transformer is an encoder-decoder where **Multi-Head Self Attention** layers allow the network to focus on specific positions in the input. The encoder and decoder are repeated and each decoder receive as input the output of the last encoder. What makes this conceptually so much more appealing than some LSTM cell is that we can physically see a separation in tasks: for example, in an english to french translation task, the encoder learns what is english, what grammar is and more importantly what the context is, while the decoder learns how english words are related to french words.

Noticeable layers in the Transformer architecture are:

- **Multi-Head Self Attention** layer, which contains multiple *Self Attention* mechanisms, where a weighted similarity measure (in general dot-product) is computed on the input and then a softmax computes probabilities, summing up to 1, that should capture attention information on different parts of the input
- **Positional Encoding** layer, which encodes each i-th embedding of each p-th token position as:

$$\begin{aligned} P_{p,2i} &= \sin \frac{p}{1000^{\frac{2i}{d}}} \\ P_{p,2i+1} &= \cos \frac{p}{1000^{\frac{2i}{d}}} \end{aligned} \tag{1}$$

  Moreover the use of oscillating functions allows to learn relative position (e.g. two words at certain distance will have some of the positional embedding values equal)

The Transformer architecture is used in many NLP tasks and researchers achieved state-of-the-art results in those task in reasonable amount of training time.

2.2 BERT

Researchers also came up with novel architectures like **BERT** [3], **GPT** [4], **GPT-2** [5] etc. which allowed to achieve the actual state-of-the-art results for various NLP tasks.

BERT [3], in particular, introduced by Google in 2018, uses the Transformer to learn a language model based on **context vectors**, allowing to solve any kind of NLP tasks.

For this reason, the training of a BERT-based system is divided in two parts:

- **pre-training**: it consists in investing a lot of computing power to pre-train an architecture which understands the language using *Masked Language Modeling* (MLM) and *Next Sentence Prediction* (NSP).

  In this phase, a language model is learnt by masking a portion of the input text (usually around 15%), so the model learns to predict the masked words, training a model to fill the blank spaces.

  In case of the next sentence prediction, BERT takes in two sentences and determines if this second sentence actually follows the first in kind of what is like binary classification problem.

  This helps BERT understand the context across different sentences themselves. Using both these methods together, BERT gets a good understanding of the language it is being trained on.

- **fine-tuning**: it consists in tuning a previously learnt architecture (or language model) on a specific target task.

  We can now "adapt" the pre-trained BERT language model on very specific NLP tasks. For example, in the Question Answering task, what we need to do is to replace the fully connected output layers of the original network with a fresh set of fully-connected layers that can basically output the answer to the question we want in form of an answer span, indicating the position of the answer in the passage (the context).

  Then, we can perform supervised training using a Question Answering dataset. It won't take long since it is only the output parameters that are learned from scratch, while the rest of the model parameters are just slightly fine-tuned and as a result training time is faster.

BERT also have a special input representation: the input to BERT are two sentences A and B which are given as a unique sentence separated by a special [SEP] token, along with another special [CLS] token that marks the start of the input. It also uses a special input embedding indicating if a token in the input belongs to sentence A or to sentence B (sentence embedding), along with traditional positional embedding.

The embedding for the tokens is constructed from three vectors:

- **token embeddings**: this is a pre-trained embedding, the main paper uses WordPiece embedding that has 30 thousand tokens
- **segment embeddings**: this is basically a number indicating to which sentence each token belongs
- **positional embeddings**: this are embeddings of the positions of the tokens in the sentences

If we want to use BERT to solve the Question Answering task, the two input sentences are, respectively, the question followed by the context (the actual passage) from which the answer needs to be extracted. These input are used to predict an answer span, so the output layers are predicting two start/end labels that are indicating the start and end index of the answer inside the passage.

3 System description

In our project, the goal was to train a neural-based NLP system that handles the question answering task on the SQuAD v1.1 dataset. In particular the system must be able to predict the answer given the question and the context in which it is applied. The answer can be produced as text (generative task) or extracted using a span (classification task).

At the beginning we decided to solve the problem as a generative task by training a simpler Transformer on the contexts and questions in order to generate the answers but, due to bad performances, we decided to adopt another approach.

We tried different pre-trained models and we compared them using a baseline architecture which was a simple classifier. Then, we chose the best one and we extended it. The metrics used to compare the models were EM and F1.

The dataset used was SQuAD [2] 1.1 where for each argument there is a paragraph with specific questions; each question has an answer represented by its text and its character span in the context.

The framework used was **PyTorch**. In order to ease the development and the use of our scripts, we created a CLI project splitted in modules in order to decouple as much as possible all the tasks and avoid redundancy.

Moreover **Huggingface's Transformer library** [6] was exploited in order to use pre-trained models, such as BERT, ALBERT and DistilBERT. Although Question-Answering models are already present in the library, we didn't used them in order to solve the task by ourselves.

The library also contains for each pre-trained model its tokenizer (and pre-tokenized vocabulary) which was used by us in order to produce inputs for our models.

The system contains various components:

- **dataset splitting**: the JSON dataset file is handled by a specific class for executing common operations such as splitting in train, validation and test set, saving these splits or converting the dataset into a DataFrame easing the operation applied to data.

  Moreover, a JSON configuration file can be used for defining the splits.

- **training**: it's possibile to create a JSON configuration file which can be used to specify the architecture to train (if baseline or not), the pre-trained

model to use and also some parameters like the batch size, the learning rate and the number of epochs.

This scrips performs a number of training epochs specified by the user (using the given training set) and, at the end of each training epoch, performs a validation epoch using the specified validation set.

At the end of the training, the scrips also evaluates the model once using the specified test set.

For training purposes, the start and end character indexes are converted into start and end token indexes.

- **computing answers**: since we also want to simply compute answers for the sake of evaluating the model, we also wrote a script that uses a trained model to compute the answers to each sample of a JSON dataset specified by the user. The output of the script is a JSON file where containing a predicted answer for each input sample.
- **evaluation**: the creators of the SQuAD dataset also provided a script which evaluates a model using EM and F1 metrics. The script use two files in order to compute F1 and EM:
  - ground truth test set, whose structure is the same of the original dataset
  - prediction test set, which contains a predicted answer for each question in the ground truth test set

We defined different CLI python scripts that allow to easily perform each key-task of the training procedure independently. We are also providing pre-defined splits along with a pre-defined model which is already trained. This model is used by the *compute_ answers.py* script.

3.1 Architecture

We tried different architectures but all of them were mainly based on pre-trained Transformers.

The first experiment consisted in using a Transformer architecture with few encoders and decoders. We tried to solve the question-answering task with a **generative** approach but as we said the performances were poor.

All the following experiments were based on solving the question-answering task as a **classification** problem, by predicting the probabilities that a token is either a start or end token; in this way we extracted an answer span.

The second experiment consisted in fine-tuning a classification network based on a pre-trained language model (more specifically, BERT).

We then modified this baseline model, by changing the used pre-trained model and by modifying the architecture on top of the trained model (adding a LSTM

encoder-decoder, a highway network and a residual connection).

In total, we tested 6 different architectures:

- **Transformer**: we trained a Transformer to produce the answer to the question. The numbers of encoder and decoder were few due to out of memory error during the training.
- **baseline model with BERT**: we created a baseline model consisting of a BERT pre-trained layer with a simple classification layer on top of it which was taught to predict the answer span in the context in form of start and end indexes.
- **baseline model with DistilBERT**: the architecture seen before was used with DistilBERT as pre-trained model. DistilBERT is a smaller version of BERT, thanks to less number of used Transformers (6 compared to 12 of the base BERT), so it is lighter and faster to train. Moreover it is trained using **distillation**, therefore the output probability distribution of the student is trained to be as similar as the output distribution of the teacher.
- **baseline model with ALBERT**: the same architecture seen before was used with ALBERT as pre-trained model. ALBERT is a lighter version of BERT thanks to two techniques:
  - decoupling the embedding dimension from the dimension of the hidden layers in order to avoid an embedding matrix that occupy too much memory; the embedding matrix is also divided in two matrices
  - sharing of the cross-layer parameters in order to avoid parameters growing with depth and so reducing them, decreasing at the same time memory and computational cost

  ALBERT turned out to be the best performing pre-trained model among the ones we tried, therefore we extended the baseline model using it in order to improve the performances.
- **modified baseline model v1 with ALBERT**: we added, before the linear classifier, a LSTM encoder-decoder architecture and a highway network: the highway network allows to train very deep neural networks in an optimized way; highway networks use learned gating mechanisms to regulate information flow, inspired by Long Short-Term Memory, allowing the information to flow across different layer reducing the information loss problem of very deep networks.
- **modified baseline model v2 with ALBERT**: the first modified version has a simple problem: the encoder-decoder LSTM will of course suffer the bottleneck problem so part of the information computed through the pre-trained model will be loss because of this; to solve this problem, the first modified version was re-modified by simply adding a residual connection

from the output of the pre-trained model to the output of the LSTM encoder-decoder

These models were trained using the same hyperparameters and using also the same train, validation and test splits to mantain coherence between the different experimentations that will be described in the next section.

4 Experimental setup and results

Before experimenting, we had to set up the experimentation environment by performing two operations:

- we **splitted** the original dataset into training (70%), validation (15%) and testing (15%) sets, so we used the same splits for each experiment.
- we created a **configuration** file to be used among all the experiments; the main parameters were the same (i.e. number of epochs, learning rate etc) but ,in between the experiments, we only changed the pre-trained model to use or the architecture to train; also, when testing ALBERT and BERT, we had to lower the batch size because otherwise we would have had an out of memory crash because of the size of the two pre-trained model.

After defining the splits to use and the parameters of the model, we tested each model by running a **single training epoch**, using a learning rate of 1e-4 and a dropout rate of 0.3. We used Cross Entropy loss and AdamW optimizer. The only parameter changing, apart from the pre-trained model and the network architecture to use, was the batch size for the reason described before.

We did the following experiments:

- **Transformer**: we trained the Transformer to produce the answer from the question and the context; however, the experiment was aborted and the architecture was abandoned because we noticed that the loss was not lowering at all and the model was not learning.
- **baseline model with BERT**: we trained the baseline model with BERT pre-trained to predict the answer span using a batch-size of 16; this model obtained a EM score of 61 % and a F1 score of 74.58 % on the test set
- **baseline model with DistilBERT**: we trained the baseline model with DistilBERT pre-trained to predict the answer span using a batch-size of 32; this model obtained a EM score of 43.83 % and a F1 score of 54.47 % on the test set
- **baseline model with ALBERT**: we trained the baseline model with ALBERT pre-trained to predict the answer span using a batch-size of 16; this model obtained a EM score of 63.38 % and a F1 score of 77.36 % on the test set

- **modified model v1 with ALBERT**: we trained the first modified version of the baseline model with ALBERT pre-trained to predict the answer span using a batch-size of 16; this model obtained an EM score of 62.58 % and a F1 score of 76.16 % on the test set
- **modified model v2 with ALBERT**: we trained the second modified version of the baseline model with ALBERT pre-trained to predict the answer span using a batch-size of 16; this model obtained an EM score of 63.75 % and a F1 score of 77.79 % on the test set

The following table resumes the experiments scores. As we said before we aborted the experiment on the vanilla transformer, so it is not included in the table.

Model	EM	F1
BERT baseline	61 %	74.58 %
DISTILBERT baseline	43.83 %	54.47 %
ALBERT baseline	63.38 %	77.36 %
ALBERT modified v1	62.58 %	76.16 %
ALBERT modified v2	63.75 %	77.79 %

Table 1: Performances of the various models on the same test set

5 Analysis of results

The first experiment tried was aborted due to a model that wasn't learning and that was producing extremely bad results.

We then moved to the usage of pre-trained models. The best performing model was **ALBERT modified v2**, with a **EM score of 63.75 %** and a **F1 score of 77.79 %**.

To break this result down, we have to start from the first experiment. BERT was trained for one epoch producing good results but the training time was too high therefore DistilBERT was used in order to reduce it. The consequences of this were that the performances were decreasing too, although they were still acceptable. Hence we changed the pre-trained model to ALBERT in order to find a trade-off between training-time and performances, a model faster than BERT and better-performing than DistilBERT.

Then, we tried to increase the performances of ALBERT: we decided to modify the baseline model by adding a BiLSTM encoder-decoder on top of the pre-trained layer, before the linear classifier, to capture and learn more information about the relationships between the questions and the contexts. We also added, after the BiLSTM encoder-decoder, a highway network which allowed us to learn even more information without the problem of optimizing a very deep neural network, thanks to the nature of the highway network. Of course, this modification slowed the training a little bit and, because of the nature of the

LSTM encoder-decoder architecture, the scores lowered a little because of the bottleneck problem.

So, we decided to test another model by simply adding a skip/residual connection from the output of the pre-trained layer to the output of the BiLSTM encoder-decoder. The training time did not increase because we only added a residual connection, but we saw a little improvement on the scores, indicating that we were able to exploit better the information extracted by the BiLSTM without losing part of the information produced by ALBERT.

This last model was, as said before, the best performing one, and allowed us to reach good scores without taking too much time for completing one training epoch.

All of these experiments were done using the same training, validation and testing sets. Just to test even more the model, we also created manually a very small test set by picking one article randomly from Wikipedia and by feeding two questions about this article to the network. We saw that the network was performing very well and answered correctly to all the questions that we wrote.

In conclusion, we also noticed that the models that were performing best were also the models converging faster. In fact, by using the training and validation scores, we noticed that these models were converging and reaching higher scores faster with respect to the other models.

6 Discussion

A major improvement would be to add a residual connection from the output of the pre-trained layer to the output of the highway network to help preserve the information after the highway network.

Another test that could be done is to train these models for a higher number of epochs to see where these models reach their peak performances. Also some hyperparameter tuning will maybe increase the scores further.

Moreover we can exploit memory by enriching the features used to make predictions by storing a memory on common question type, hence for each *wh-question* we could compute an exponential moving average of the embedding found during the training; similarly, we can link the type of the context (e.g. sport, history, finance, politics, ...) to an embedding and use it for training and testing.

We can so conclude that transformer based model are allowing to reach actual state-of-the-art performances and that pre-trained models give a noticeable boost to the performances of novel NLP systems. This is due to the fact that, as said in the beginning of this report, these models are learning entire language they are trained on and thus this learnt information is very useful for solving various NLP tasks using that language.

However, even if pre-trained model are very useful, studying, implementing and

testing different and more complex architectures using these pre-trained models also helps in reaching better results.

7 Bibliography

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[2] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," *CoRR*, vol. abs/1606.05250, 2016.

[3] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[4] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training,"

[5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[6] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, (Online), pp. 38–45, Association for Computational Linguistics, Oct. 2020.