Alma Mater Studiorum · University of Bologna

Question answering on the SQuAD dataset

NLP course final project

Leonardo Calbi (leonardo.calbi@studio.unibo.it)
Lorenzo Cellini (lorenzo.cellini3@studio.unibo.it)
Alessio Falai (alessio.falai@studio.unibo.it)

January 30, 2021

Contents

1 Summary

The tasks of machine comprehension (MC) and question answering (QA) have gained significant popularity over the past few years within the natural language processing and computer vision communities. Systems trained end-to-end now achieve great results on a variety of tasks in the text and image domains.

In this work we address a question answering problem and, in particular, the Stanford Question Answer Dataset (SQuAD) problem: given a large collection of Wikipedia articles with associated questions, the goal is to identify the span of characters that contains the answer to the question.

This project focuses on the SQuAD v1.1 dataset, that contains more than 100,000 question-answer pairs on more than 500 articles. Here each question has at least one associated answer, whereas in the SQuAD v2.0 dataset there are also questions with no answers at all.

In this work we implement and compare five different models:

- a naïve LSTM encoder-decoder that will act as our baseline;
- the Bi-Directional Attention Flow (BIDAF) network (CITAZIONE);
- a model that wraps a pretrained Bidirectional Encoder Representations from Transformers (BERT), as language model, coupled with a custom output layer on top of it;
- a model that wraps DistilBERT, a smaller general-purpose language representation model, 40% smaller then BERT but with 97% of its language understanding capabilities and 60% faster;
- a model that wraps ELECTRA, a more sample-efficient pre-training task called "replaced token detection".

For the last three models, we developed a modular reusable output layer, allowing us to implement different models by only change the language model levels (input, embedding and modelling).

Our experimental evaluations show that our models achieve .

2 Background

As already described above, the question answering task is based on the idea of identifying one possible answer to the given question as a subset of the given context.

Since the input data is of textual form and the latest models for the task are all based on neural architectures, there is the need to encode such text into a numeric representation. As of today, there are two main approaches to embed words in numerical format: sparse embeddings, like TF-IDF [12] and PPMI [8], and the modern dense embeddings, such as Word2Vec [6] and GloVe [11]. In the latter case, embeddings for each word in the input vocabulary are usually computed with shallow encoders.

These emebddings are then used as inputs for models specialized in processing sequential inputs. Nowadays, such models are mostly based on the Transformer architecture [13], which has the attention mechanism at its core. Instead, before this revolution, NLP competition's leaderboards were mostly populated by models based on recurrent and convolutional modules.

The most influential recurrent networks are LSTMs [5] and GRUs [3]. They are both based on processing sequential inputs and they keep an evolving series of hidden states, such that the output $h_t^{(0)}$ is a function of $h_{t-1}^{(0)}$ and the input x_t at position t. Recurrent layers can also be stacked to increase the capacity of the network: in that case, $h_t^{(0)}$ would act as an input to $h_t^{(1)}$, i.e. the first hidden state of the next depth-wise layer, and the same goes for the successive layers, as shown in figure 2.1.

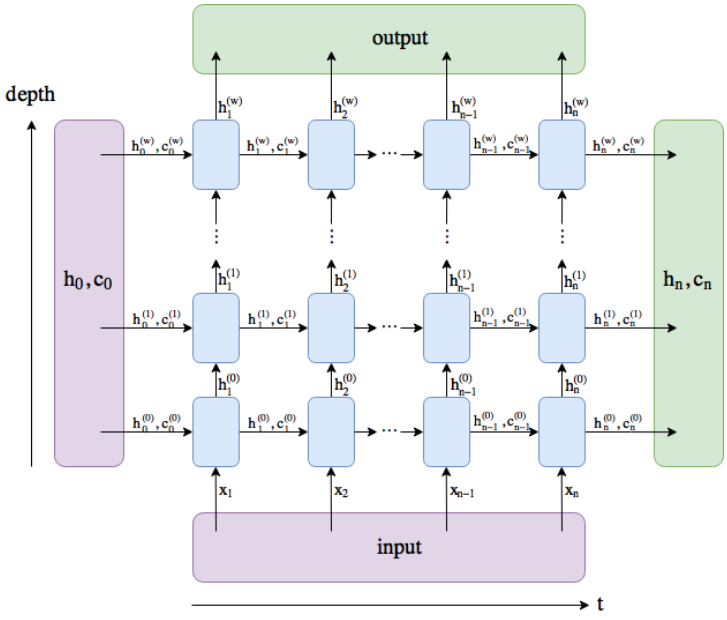

Figure 2.1: Recurrent module (image courtesy of [7])

In natural language, meaning does not usually simply flow from left to right, so that in RNNs other techniques may be necessary to reach a higher level of understanding. One example of such improvement is bi-directionality: the input is processed in two ways, from left to right (the "forward" pass) and from right to left (the "backward" pass); then, outputs of the two computations are merged together (usually by concatenating them over their last dimension). The intuition behind bi-directional RNNs is that they tend to understand the overall context better than their uni-directional counterparts, since they are able to aggregate information from the "past" and from the "future".

About Transformers, at a high-level, they comprise a stack of encoder-decoder modules, where each encoder features a multi-head attention block (to focus on different subsets of the input sequence when encoding one specific token) and a point-wise fully-connected layer, while each decoder features the same architecture as the encoder, but with an additional multi-head attention block in the middle (that helps the decoder focus on relevant parts of the input sequence).

The multi-head attention modules in the Transformer architecture are the ones responsible for gathering information from other tokens in the input sequence, thus resembling the task of hidden states in RNNs (i.e. incorporate the representation of previous inputs with the currently processed one).

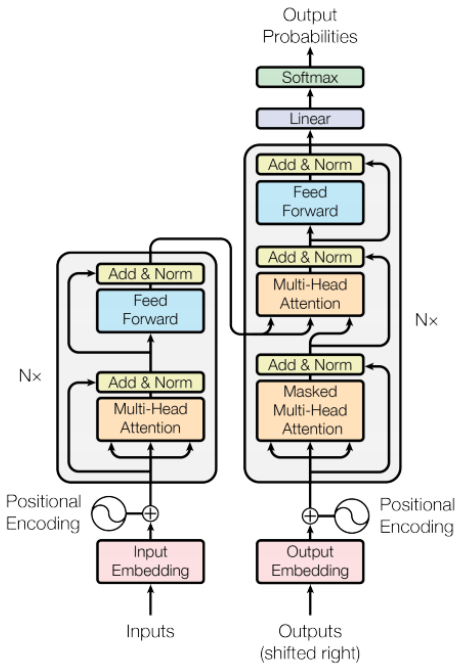

Figure 2.2: Transformers (image taken from [13])

While Transformers may always seem the way to go, since they throw away all the computational burden of training recurrent modules (which are sequential in nature and, thus, scarcely parallelizable), they also suffer from limitations due to the quadratic complexity of attention and the inherent maximum number of input tokens (which is implicitly linked to architecture choices themselves).

3 System description

In this work we implement three distinct models: a naïve LSTM encoder-decoder that will act as our baseline, the Bi-Directional Attention Flow (BIDAF) network (CITAZIONE) and a model that wraps a pretrained Bidirectional Encoder Representations from Transformers (BERT), as language model, coupled with a custom output layer on top of it.

Each model is trained on the SQuAD v1.1 training set and evaluated on the SQuAD v1.1 dev set, available on the github repository of the SQuAD project (link).

The models evaluation is carried out on the same preprocessed data: indeed, we defined a preprocessing pipeline, applied it to the training and dev set once and the resulting dataset is fed to each model. (CONTROLLARE QUI mi pare che per bert abbiamo tokenizzato diversamente)

In order to perform an effective models comparison, each model shares the same input and output layer: what changes is the language model (embedding + attention layers) and the modelling layer.

The following is a detailed description of the models.

3.1 Baseline

.

3.2 BiDAF

Sintesi tra paper bidaf e medium

3.3 BERT

.

4 Experimental setup and results

4.1 Data handling

Given the JSON training set, we decided to parse it into a suitable tabular data structure, which turned to be useful even for subsequent pre-processing tasks. In particular, since each context has multiple question/answer pairs and each question can have multiple answers, we allocated one row for each context/question/answer triple, thus replicating the same context and question multiple times, in order to consider it as a separate example.

Regarding the splitting strategy for the dataset, we went for a simple holdout, thus setting aside 20% of the whole dataset for validation purposes. The way the validation set is built guarantees that the entire set of rows referencing to the same context is kept into the same split, thus avoiding unjustifiable boosting of results.

As an additional test, we evaluated all of our models on the official SQuAD v1.1 dev set. The main difference between this dataset and the training one is that the same question can have multiple correct answers.

4.2 Tokenization

Tokenization has the following meaning: given a string of text, its task is to split it into substrings (called tokens), depending on the chosen algorithm. Different models call for different tokenization pipelines. In our case, we have two macro-categories:

1. Recurrent-based models tokenizer: splits words by whitespaces and punctuations, removes accents and applies a lowercasing function to all the tokens. Moreover, questions are padded (not truncated), while contexts are truncated and padded. Taking inspiration from [9], the maximum number of tokens for truncating contexts was fixed to 300.
2. Transformer-based models tokenizer: splits words using the WordPiece algorithm (introduced in [15]), removes accents and applies a lowercasing function to all the tokens, while also merging together questions and contexts as $[CLS]q_1q_2 \dots q_n[SEP]c_1c_2 \dots c_m[SEP]$ (it leverages the special tokens $[CLS]$ and $[SEP]$). Moreover, the combined question/context sentence is truncated to a maximum number of tokens (512, as in [2]) and padded to the right.

Tokenization happens on the fly at the batch level, thus enabling us to perform dynamic padding (based on the longest sequence in a batch) and avoiding the pre-tokenization overhead.

The tokenizer is also used as some kind of pre-processor, to remove from the training dataset those rows whose contexts would not contain the relative answer, after truncation.

4.3 Embeddings

In recurrent-based models, tokens are numerically encoded using the standard GloVe embeddings. In particular, we tested different types of GloVe models, with different embedding dimensions:

- Wikipedia 2014 and Gigaword 5 (6B tokens, 400K vocab, uncased, 300d vectors)
- Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 200d vectors)

We additionally embedded two more tokens (not present in the standard GloVe vocabulary):

1. $[PAD]$: the corresponding vector contains all zeros
2. $[UNK]$: the corresponding vector, which is used to handle OOV (Out Of Vocabulary) words, is the mean of all the GloVe vectors, as reported by Pennington (one of GloVe's authors)

   > *I've found that just taking an average of all or a subset of the word vectors produces a good unknown vector*

In Transformer-based models, we instead rely on the wrapped language models of the HuggingFace library [14], so that there is no need to manually handle token embeddings.

4.4 Metrics

Models are evaluated at the end of each epoch and metrics are computed for both the training and validation set. In our case, the validation set was mainly used to track models' performance on unseen data, in order to avoid overfitting problems.

Metrics used for evaluating the models are F1 and EM (Exact Match) and they were taken from the official SQuAD evaluation script. The F1 score ($f1 = \frac{2 \cdot p \cdot r}{p+r}$), defined as the harmonic mean between precision ($p = \frac{TP}{TP+FP}$) and recall ($r = \frac{TP}{TP+FN}$), is computed per question and then averaged across all questions. The EM score is instead computed as the number of questions that are answered in exact same words as the ground truth, divided by the total number of questions.

Since the same question can have multiple ground truths (specifically, in the dev set), there is the need to select just one for each question. To fulfill this requirement, only the nearest ground truth for each prediction is considered: given a prediction $p = [p_s, p_e]$, where p_s is the start index of the predicted answer and p_e is the corresponding end index, and a set of associated ground truths $G = \{[g_s^{(0)}, g_e^{(0)}], \ldots, [g_s^{(n)}, g_e^{(n)}]\}$, the selected ground truth is $\min_{g \in G} \|p - g\|_2$.

Moreover, we noticed that the training set does not have multiple answers for the same question, so this "nearest answer" approach comes into play only at inference time: this is actually the desired behavior, since at training time the described technique could lead to an unstable learning process, given by the non-stationarity of the target distribution.

4.5 Environment

The main third-party libraries on which the project is based on are PyTorch [10], an open source machine learning framework with automatic differentiation and eager execution, and HuggingFace [14], a library to build, train and deploy state of the art NLP models.

All of the training and validation processes were executed on Google Colaboratory [4], a platform that gives the possibility to exploit some computational resources for free. In particular, Colab allows you to select a GPU runtime, that boosts the training time of neural models and most of the times we were assigned an NVIDIA Tesla T4 GPU, with 16GB of RAM.

Training and evaluation metrics, along with model checkpoints and results, are directly logged into a W&B project [1], which is openly accessible here.

4.6 Results

	Training		**Validation**		**Test**	
	F1	**EM**	**F1**	**EM**	**F1**	**EM**
Baseline						
BiDAF						
BERT						
DistilBERT						
ELECTRA						

Table 4.1: Results

5 Analysis of results

6 Discussion

Bibliography

[1] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. cite arxiv:1810.04805Comment: 13 pages. 2018. URL: http://arxiv.org/abs/1810.04805.

[3] Yuan Gao and Dorota Glowacka. "Deep Gate Recurrent Neural Network". In: *Proceedings of The 8th Asian Conference on Machine Learning*. Ed. by Robert J. Durrant and Kee-Eung Kim. Vol. 63. Proceedings of Machine Learning Research. The University of Waikato, Hamilton, New Zealand: PMLR, 2016, pp. 350–365. URL: http://proceedings.mlr.press/v63/gao30.html.

[4] Google. *Colaboratory*. URL: https://colab.research.google.com/.

[5] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed Representations of Words and Phrases and their Compositionality". In: *NIPS*. Curran Associates, Inc., 2013, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[7] Nikolai Morin. *What's the difference between hidden and output in PyTorch LSTM?* URL: https://stackoverflow.com/questions/48302810/whats-the-difference-between-hidden-and-output-in-pytorch-lstm.

[8] Yoshiki Niwa and Yoshihiko Nitta. "Co-Occurrence Vectors from Corpora vs. Distance Vectors from Dictionaries". In: *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*. COLING '94. Kyoto, Japan: Association for Computational Linguistics, 1994, pp. 304309. DOI: 10.3115/991886.991938. URL: https://doi.org/10.3115/991886.991938.

[9] Do-Hyoung Park and Vihan Lakshman. *Question Answering on the SQuAD Dataset*. Tech. rep. Stanford University.

[10] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[11] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://www.aclweb.org/anthology/D14-1162.

[12] "TF–IDF". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 986–987. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_832. URL: https://doi.org/10.1007/978-0-387-30164-8_832.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[14] Thomas Wolf et al. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *CoRR* abs/1910.03771 (2019). arXiv: 1910.03771. URL: http://arxiv.org/abs/1910.03771.

[15] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). arXiv: 1609.08144. URL: http://arxiv.org/abs/1609.08144.