Natural Language Processing: Answering Hard Questions

Niels Pichon, Emma Hagerup, Carlos Tierno

November 1, 2018

Abstract

1 Introduction

Answering questions based on a given written document isan important development in modern computer language processing. It also echoes in the consumer world with, among other, as demand for such algorithms that has skyrocketed in last years, with the proliferation of personal AI assistants. Yet it remains a very challenging task as it requires a good understanding of natural language and being able to extract and store tremendous amounts of knowledge, as well as generate comprehensive answers from the two combined. Fortunately, some recent development in neural network based methods have shown some promising results, even performing better than humans in certain cases on the Stanford Question Answering data set [1].
These results can probably still be improved though, as the recent BERT algorithm showed [2]. In an attempt to tackle the issue of answering hard questions ourselves, we proceed in the following article to a review of some of the most advanced State-of-the-art techniques on the SQuAD database, and implement a network, largely inspired by BiDAF [12] and BERT [2].

2 Background and Related Work

2.1 GloVe Embedding

Among all vector space representation of language, the GloVe [10] algorithm stands out. The reason it does is that it combines local windows methods such as the skip-gram/CBoW method, and global matrix factorisation methods such as the LSA (Latent Semantic Analysis), thus allowing to capture properly word relations which the Vec2Word methods usually finds relatively well, and the word statistics, better captured normally by the latter. To achieve such a result, the GloVe algorithm typically considers word-word co-occurrence probability. To determine if a word is more related to a second word rather than a third one, all we have to do is compute the ratio of the co-occurrence probability of the first two words by the co-occurrence of the first and thrid words. If the ratio is large then the first word is most likely related to the second one rather than the third. Mathematically, the learning algorihtm actually looks for the vector representation w_i of the word i in the vocabulary, and the vector representation $\tilde{w_j}$ of the context word, V that minimizes the following cost function :

$$J = \sum_{i,j}^{V} f(X_{ij}) \left[w_i^T \tilde{w}_{j\,j} + b_i + \tilde{b}_j - \log(X_{ij}) \right]^2 \tag{1}$$

where X_{ij} is the co-occurrence count of words i and j and b_i the bias associated with the word vector i. $f(X_{i,j})$ is a weighting function defined as follows :

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} \text{ if } x<x_{\max} \\ 1 \text{ otherwise} \end{cases} \tag{2}$$

$x_{\max}$ and α are parameters to be chosen empirically, but the J. Pennington *et al.* [10] recommend to use $\alpha = 3/4$ and say $x_{\max}$ choice doesn't really matter. Note that in the end, because the matrix X of co-occurrence counts is symmetric, the vectors of words and context words should be equivalent and usable interchangeably.

Practically, for such methods which are context dependant the choice of the context size is crucial. The authors of the method recommend using a decay with distance (the further away in the context, the less a word contributes in meaning) such as 1/distance and a context of 10 words to the left and 10 words to the right.

2.2 Attention

A classic Feed-Forward network would process all the data, here the text of our corpus or question, at once, which makes it very poorly suited for the task at hand because it would completely ignore the sequentiality of the words, from which meaning emerges. Therefore a Recurrent Neural Network would definitely perform better, if we could determine how to feed it the appropriate sequences that is. This is where 'Attention' becomes handy. This is a process through which the machine would select the subset on which to focus it's attention (as the name suggest) at every step, or in other words, select given an input and the context, the attributes on which it relies to take a decision. For instance, given a block of text, a well trained attention network would select only the relevant sentences to our question and feed them to our answering network. Such networks can be trained via reinforcement network or even classic backpropagation if the applied policy is differentiable.

One of the most common attention units is the dot product attention. A set of vector queries, keys, and values are fed, by set stored in matrices, into the attention matrix. After applying a dot product on the queries and keys, the output layer if fed through a softmax which is finally multiplied by a dot product with the values. A. Vaswani *et al.* [8] argued that in certain case such structure can become very slow as the gradients become huge in the softmax calculation. To avoid such problem, they suggest to scale the output of the first dot product by dividing by the square root of the input vector dimension. This is summed up by the following formula :

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{\dim_K}} V \tag{3}$$

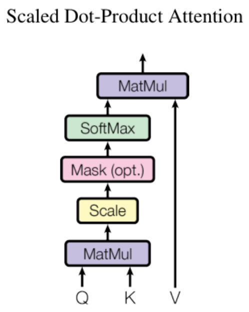

Figure 1: Scaled Dot Product Attention

The same authors introduced Multi-Head Attention layers which are simply n parallel scaled dot-product attention layers preceded by various projection matrix, and the output of which is then concatenated. The idea is that each parallel attention layer could focus on a different aspect of the input. an illustration is given on figure 2

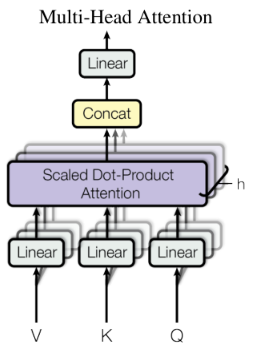

Figure 2: Multi-Head Attention

2.3 Dynamic Neural Network

As mentioned before, to properly understand a sentence or question, the words correlation has to be considered which suggest the use of a recurrent neural network. This also allows for to deal more easily with the variable length of the question and answer. However, understanding the relation between the words of a sentence is often not enough as many sentences may exhibit similar relation patterns. This is where Dynamic Neural Networks (DNN) play a role. Instead of directly confronting the knowledge corpus to a the question, we introduce a so-called episodic memory unit.

This unit will compare the question, a given sentence from the corpus and a memory to decide if the sentence is relevant to answering the question, and only then the question will be answered using this sentence. In a way this is a form of attention unit, as described above. This network structure is illustrated in figure 3, taken from [7].

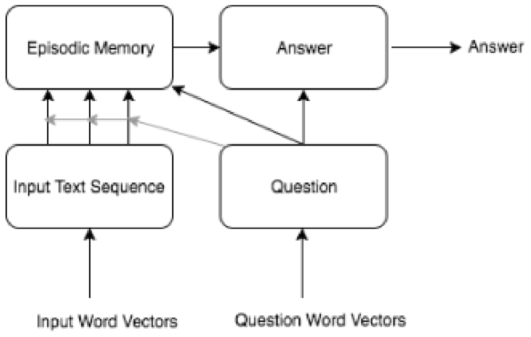

Figure 3: DNN structure. The input and question units are GRU layers.

2.4 Transformer unit

A more recent alternative to recurrent networks, which as gained a lot of popularity since it was first published by A. Vaswani *et al.* [8], is transformer networks. Recurrent network are classically the most widely used networks for any task that requires analyzing the link between sequential inputs. The problem is that such networks are usually hard to train. Instead, transformer networks solely use attention units, without any form of recurrence or convolutional networks. Such networks are both faster to train and allegedly more efficient on a number of native language processing tasks. This is for instance the kind of networks used in the latest leader on the SQuAD leaderboard [2].

The structure of a transformer network is detailed on figure 4. On the higher level it is a classic encoder-decoder structure. Each of them is made of N (=6 in the original publication) identical layers. On the encoder side, we have in serie a multi-head attention sublayer followed by a fully connected feed-forward sublayer. Note that in between each of the layers we allow for some residual connections such that the output of a sublayer actually is $\text{LayerNorm}(x) = \text{Norm}(x + \text{Sublayer}(x))$, where $\text{Sublayer}(x)$ is the activation function of the sublayer.

On the decoder part, we have the same structure, preceded by an extra multi-head attention layer applied directly on the output embedding shifted to the right so that an input i can only depend on the outputs up to the i-th one.

2.5 BiDAF

The Bi-Directional Attention Flow (BiDAF) network is a hierarchical multi-stage architecture for modeling the representations of the context paragraph at different levels of granularity [12]. BiDAF consist of 6 layers where 3 of them are embedded layers of characters, word, and context, respectively

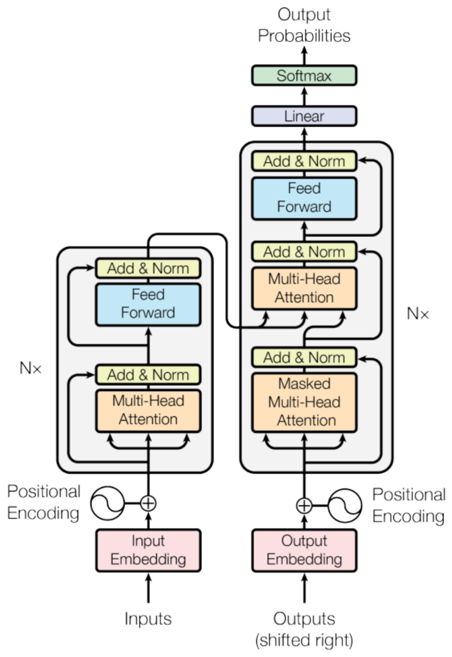

Figure 4: The transformer network structure

and by implementing bi-directional attention from query to context it is able to obtain a more query-aware context representation:

- **Character Embedding Layer**
  The character layer maps each word to a corresponding vector from the vector space. To achieve this we use a Convolutional Neural Network which outputs vectors (hence 1D). This output is then max pooled to obtained the definitive input of our BiDAF Network. The name comes from the fact that the CNN actually works on a character level.

- **Word Embedding Layer** The word embedding does the same mapping with words as in the character layers. We use pre-trained word layer vector, GloVe, to obtain fixed word embeddings of each word.

  We use a two-layer Highway Network on the concatenated character and word embedding vectors to produce an output of sequences of d-dimensional vectors, or what can also be regarded as, a matrix $\mathbf{X} \in \mathbb{R}^{d \times T}$ for the context and a matrix $\mathbf{Q} \in \mathbb{R}^{d \times J}$ for the query.

- **Contextual Embedding layer** To model the temporary interactions between words we place a bi-directional Long Short-Term Memory Network on top of the embeddings from the previous layers and concatenate the outputs from both. Thereby we get $\mathbf{H} \in \mathbb{R}^{2d \times T}$ from applying the network on the context vectors $\mathbb{X}$ and $\mathbf{U} \in \mathbb{R}^{2d \times J}$ from the query vectors $\mathbb{Q}$.

- **Attention Flow Layer** The attention flow layer takes in the inputs from the embedded layers, ie. the vectors $\mathbf{H}$ and $\mathbf{Q}$. By pairing the vectors in $\mathbf{H}$ and $\mathbf{Q}$ we can achieve a set

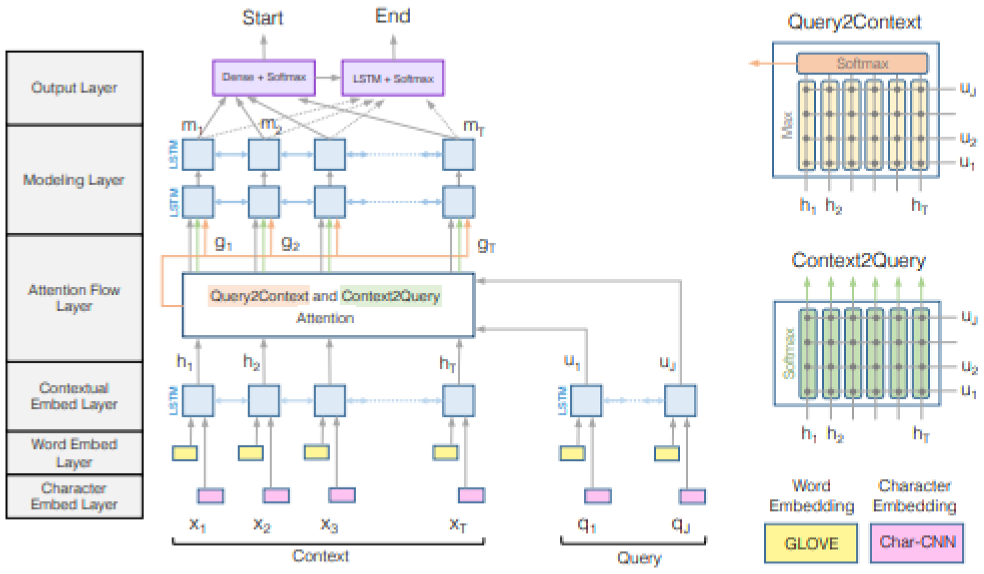

Figure 5: : Bi-Directional Attention Flow Model

of query-aware feature vectors for each word in the context, which is referred to as **G**, along with the contextual embeddings from Contextual Embedding Layer.
As mentioned before, the attention is bi-directional: we compute the attention from context to query and from query to context. By this we can compute the attentions and the attended vectors in both directions from the shared similarity matrix, $\mathbf{S} \in \mathbb{R}^{T \times J}$, between **H** and **U**.

- **Model Layer** The purpose of this layer is to get as output the interaction among the context words conditioned on the query. This layer is composed by two Long Short-Term Memory network, with an output of size d for each direction. Therefore we will get a $M \in \mathbb{R}^{2d \times T}$ matrix, which is used in the output layer for predicting the start index of the answer over the entire paragraph given.

- **Output Layer** The output of this layer will predict the start and the end indices of the phrase in the paragraph given, which is obtained with the probability distribution of the start index given by:

$$p_{\text{start}} = \text{softmax}(W_{p_{\text{start}}}^T [G; M]), \tag{4}$$

For the end index, it goes through another one Long Shot-Term Memory layer obtaining $M \in \mathbb{R}^{2d \times T}$ and using it in a softmax for getting the probability distribution of the end index.

$$p_{\text{end}} = \text{softmax}(W_{p_{\text{end}}}^T [G; M]) \tag{5}$$

2.6 BERT

The **B**idirectional **E**ncoder **R**epresentations from **T**ransformers is a brandnew network type that recently beat most scores on the major Native Language Processing tasks [2]. Achieving such a performance lies in two factors : a novel network structure, heavily dependant on transformer units, and a thorough and clever pre-training. We will first discuss the structure. It is illustrated in figure 6

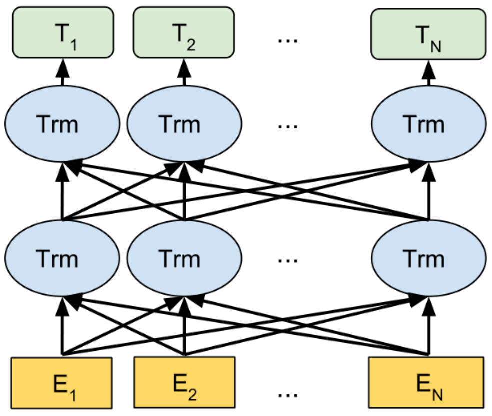

Figure 6: BERT Structure

The input is made of the corpus text plus a separator "word" followed by the question. Each word of the whole input is then embedded into a vector representation. The original publication uses Word Piece embedding for this task. We thus get a set of input vectors. These vectors are now fed as inputs into a network resembling a fully connected feed-forward network, except that each "neuron" is actually a whole transformer unit. Finally, we get a set of vectors (the same number as input vectors since each layer add the same number of transformer units as the number of words in the input) which represent our output. Such a network is rather general and it is hard to make sense of the output. And this is the actual strength of this network. Because the output is not specific, the network can then be fine-tuned to match any NLP problem, such as the SQuAD one, which is illustrated in figure 7. In this case, a softmax is applied to output tokens after the separator to obtain the start and end indices.

Such network however yield one major drawback. In a transformer unit, when the output is fed back in, it is offset by one index to the right so that there is some sort of causality of the new output: it doesn't depend of itself. This is a necessary condition for training to be possible. However, because in the BERT network, the transformer units are connected bidirectionally, each output is actually going to "see itself". To circumvent this issue, the original authors introduced a method that resembles dropout in a way : every step, some inputs are randomly selected and dropped or replaced by some other random word. Then, only the output tokens corresponding to the discarded tokens are considered. With about 15% of such drop out of the input word tokens, it was shown that the network can then train properly. Also, to mitigate the mismatch such a method would create between the pre-training and the actual training on the desired task, sometimes (10%

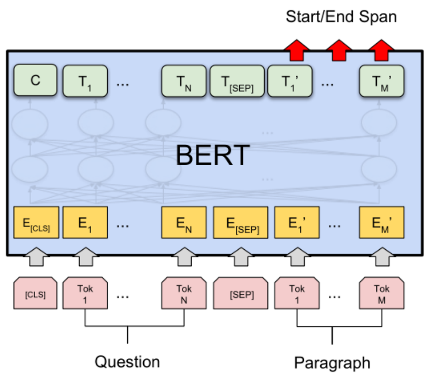

Figure 7: BERT Structure for answering the SQuAD problem

of the time), the "discarded" tokens are actually kept.

3 Methods

The idea of correctly answering a question on a corpus can be formalized as the need to maximize the probability of getting the right answer a from all the possible Answers (noted as $\{\text{Answers}\}$), given a document d and a question q:

$$O = \max_{a \in \{\text{Answers}\}} (P(a|q,d)) \tag{6}$$

To try and reach such objective we work in 2 steps. First, we create a vector representation of our Vocabulary using the GloVe algorithm. To reach maximal efficiency, we use a pre-trained model, as provided on the official GloVe website.

Using such embedding we can now create some input for our network. This network is actually a BERT network. The main difference, apart from the choice of word embedding is that don't do any pre-training, and train directly on the SQuAD database to see what results we can obtain. As in the original paper, we try to maximize the log likelihood of the correct start and end position.

In the eventuality where such network would fail to train properly because of the very nature of its structure, we would then implement a more simple and accessible BiDAF network.

4 Experimental Setup

4.1 GPU

Executing the training process will need a really big amount of time for training the model and a process capacity which we don't have in our personal computers. Therefore, we will use one of the following cloud computing services:

- AWS will be used for training the model, considering that we are able to run the code on a Tesla v100 GPU (16GB).

- HPC DTU resources has been also considered which offers us 4 GPU cores: 3 Tesla and 1 GTX TITAN.
- Google Cloud offers us NVIDIA Tesla K80, P100, P4 or V100 being able to select the number of GPUs we want, as the CPU and the posibility of using SSD.

4.2 Language Code, Framework and Control of Versions

The project will be settle up in a Python 3 environment using Pytorch as main framework. It is needed version controller being GitHub the choose. The repository can be found in:
https://github.com/careway/DL_QA

5 Results

6 Discussion

7 Conclusion

References

[1] P. Rajpurkar *et al.*, SQuAD: 100,000+ Questions for Machine Comprehension of Text, unreviewed, arXiv:1606.05250

[2] J. Devlin *et al.*, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, unreviewed, arXiv:1810.04805

[3] A. Radford *et al.*, Improving Language Understanding by Generative Pre-Training, Open AI, unreviewed

[4] T. Mikolov *et a.*, Efficient Estimation of Word Representations in Vector Space, Google AI, unreviewed, arXiv:1301.3781

[5] X. Rong, word2vec Parameter Learning Explained, unreviewed, arXivarXiv:1411.2738

[6] D. Guthrie *et al.*, A Closer Look at Skip-gram Modelling, Proc. of the Fifth International Conference on Language Resources and Evaluation, 2006

[7] A. Raghuvanshi and P. Chase, Dynamic Memory Networks for Question Answering, Stanford University report, 2016

[8] A. Vaswani *et al.*, Attention is all you need, unreviewed, arXiv:1706.03762

[9] srush aka the unknown nice guy, The Annotated Transformer, http://nlp.seas.harvard.edu/2018/04/03/attention.html

[10] J. Pennington *et al.*, GloVe: Global Vectors for Word Representation, 2014

[11] T. Fan *et al.*, Co-Attention with Answer-Pointer for SQuAD Reading Comprehension Task, Stanford report

[12] Min Joon Seo and Aniruddha Kembhavi and Ali Farhadi and Hannaneh Hajishirzi, Bidirectional Attention Flow for Machine Comprehension, arXiv:1611.01603