REPRODUCTION CHALLENGE - DYNAMIC COATTENTION NETWORKS FOR QUESTION ANSWERING

Candidate 1004727
Department of Computer Science
University of Oxford

January 17, 2020

Repository: `https://github.com/AnonymousAMLStudent/AnonymousRepository`

1 Introduction

1.1 Question Answering and SQuAD

Question Answering (QA) is a crucial and well documented task in natural language processing. It is recognized as a hard task, as proper answering requires both understanding natural language and world knowledge ([1], [4]), and, in its general form, QA is AI-complete. Being a central NLP problem, QA has lead to many applications.

Retrieving pieces of information that answer our questions can be found today in large scale available smart assistants, such as Siri, Alexa, Google Assistant. This task can be factored in two parts ([2]): finding documents that might contain a relevant answer, and finding an answer in a paragraph or document. The latter one is often referred to as Reading Comprehension (RC), or Machine Comprehension of Text (MCT), as Burges calls it in his essay [3]. Burges also defines the task of MCT in terms of a machine comprehending a text if it can answer questions about a paragraph that a majority of native speakers can answer and if these speakers would agree that the produced string answers the question and does not contain irrelevant information.

As [4] notes, creating or acquiring large and natural datasets corresponding to a certain task has lead to cutting edge advancements in machine learning research, a notable example being ImageNet for the object recognition task. Introducing SQuAD in [4], which is a large, diverse, and high in quality human annotated dataset consisting of paragraphs, questions and answers, has paved the way to many breakthroughs in the Reading Comprehension field. QA datasets before SQuAD lacked either quantity, being manually created by people, or quality, being automatically or semi-autoamtically generated. SQuAD has two versions: v1.1, which has roughly 100.000 questions with answers in given contexts, and v2.0, which contains all the datapoints from v1.1 and adds 50% extra questions with no answers in the given contexts, thus requiring networks to also predict if a question in unasnwerable given a context.

1.2 Reproducing the paper: Dynamic Coattention Networks for Question Answering

We have chosen the paper [1]: Dynamic Coattention Networks for Question Answering as, when it was published, it provided the state of the art single model score of 75.9% F1 on SQuAD v1.1, compared to the previous best score of 71.0% F1. The DCN ensemble has also been the first network to break the 80% F1. We found particularly interesting the justification of both building a network that fuses the question meaning within the document through the proposed coattention mechanism, and of the proposed iterative way of hypothesizing answers that allows restoring from wrong predictions which can correspond to local maxima.

We implemented from the ground up the DCN architecture using Tensorflow on Python and got a baseline model which achieved 71.707% F1. We verified the experiments that the authors did and we present the results in this report.

We further improved our model and achieved a model with EM score of 62.195%*, and a model with F1 score of* 72.909%.

1.3 Our original contribution

1.3.1 Joint distribution of starting and ending indices - Outer product

By selecting, at prediction time, the starting and ending indices such that their joint probability is maximized and the ending index is greater or equal than the starting index we achieve an F1 score of 72.896%, which improves our baseline 71.707% F1 by roughly 1.2% at virtually no cost. We refer to this mechanism as adding outer product at prediction stage, as we basically generate the outer product of the probability distributions over starting and ending indices.

1.3.2 Squad v2.0

We extended the network to deal with unanswerable questions as well by taking the codependent representations of documents and questions through an LSTM which generates a sentinel corresponding to "no answer" which is treated by the rest of the network as a separate token, having obatained 49.44% F1 with no tuning and with only 2 training epochs, as we were limited by time.

1.4 Report structure

The structure of the report is as follows:

- **Section 2**: paper background, includes both technical aspects and explanations of our understanding of the model,
- **Section 3**: our implementation details and our original contribution to the paper,
- **Section 4**: our team work and my personal contribution,
- **Section 5**: experimental results - the reproduction results of the paper and the metrics of our contributions,
- **Section 6**: conclusions, assessment of our work, potential improvements given more time.

2 Dynamic Coattention Networks: Paper background

2.1 Motivation and overview of the model

The DCN architecture was the first one to break the barrier of 80% F1 on SQuAD v1.1, its ensmeble achieving 80.4%. It is an end-to-end network designed to answer questions, given paragraphs which certainly contain the answer we seek, which is specific to SQuAD v1.1. It has an encoder-decoder structure: the encoder builds a conditional representation of a paragraph given a question, while the decoder iteratively improves its answer over the document representation which encapsulated the question meaning.

2.1.1 Encoder

The encoder is coattentive, building a concantenated representation of the document and question, which contains:

- the document,
- the attention contexts of the question in light of each word of the document (which is the classic attention mechanism defined in [6]),
- the coattention, which is the attention contexts of the document in light of each word of the question, projected back to the space of document encodings.

We consider this coattentive representation as an efficient and intuitive way of merging the question meaning into the document. This representation is further compressed together through a Bi-LSTM, the output of which is defined by the authors as the **coattention encoding** and is passed further to the decoder.

As the authors say ([7]), the motivation behind focusing on representations in the document space lies in the natural behaviour that reading a document with a question in mind is much easier to answer than reading a question with the document in mind. Coattention builds a conditional representation of the document given the question, and a conditional representation of the question given the document (both being instances of classic attention), but represents both in document encodings space. As the representation of the document given the question is already in document space, it projects the representation of the question given the document back to document space.

We believe that building an accurate representation is the most important component of the network, which is confirmed by my correspondece to one of the authors of the DCN paper, Victor Zhong, who says:

"[..] with regards to our particular paper, we actually think that the encoder plays a much more prominent role with regards to performance. That is, the bottleneck is in building good representations [..]"

2.1.2 Decoder

The decoder (the Dynamic Pointing Decoder), alternates between predicting the starting and the ending indices of the answer span in the text iteratively, trying to come up with better predictions at each step. It basically mimics a state-machine, updating its state via an "LSTM-based sequential model". Using the current state and the previous starting and ending predictions, it tries to guess the next starting/ending index with a Highway Maxout Network ([8]) architecture.

The Highway Maxout Networks (HMN) generate a distribution of starting and endings scores over the document words. Essentially, highway networks in HMN stands for residual connections, which intuitively allow for a better gradient propagation. The maxout layers in the HMN are a simple way of pooling across different models, which may stand for multiple models corresponding to multiple question types, all summarized in one model.

The intuition behind the HMN role is confirmed by the correspondence with Victor Zhong:

"Our "highway" network is basically residual connections. One hypothesis for why these things work well in general is that they allow for better gradient propagation. [..] With regard to the Maxout layers, they are basically a way to create a soft ensemble mechanism (e.g. model averaging).".

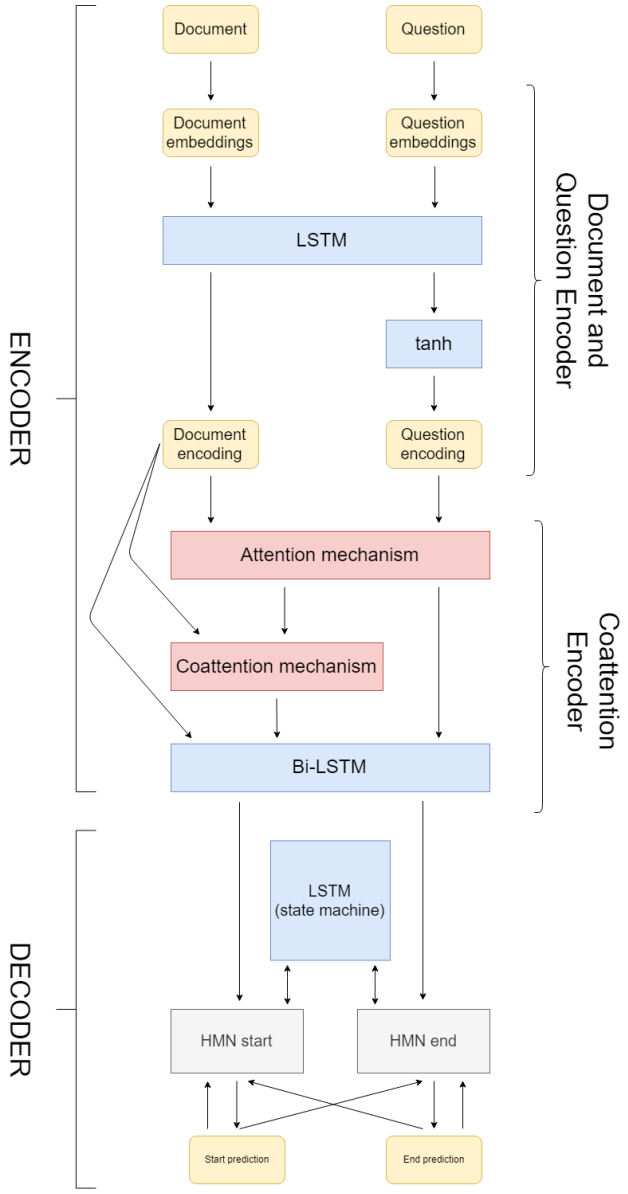


Figure 1: High level description of the DCN architecture.

In order to train the model, the cumulative softmax cross entropy of the start and end estimates across all HMN iterations is minimized. The HMN predictions are iterated either until a maximum number of iterations is reached, or until the predictions do not change anymore.

2.2 Authors' implementation and mathematical formulation of the network

Initially, the words in a given document and question are tokenized and embedded in a real vector space. The authors used Stanford CoreNLP for tokenizing and the pre-trained word vectors GloVe corpus ([5]) for embeddings: these are 300 dimensional word embeddings, generated on word-word co-ocurrences, containing 840B tokens.

2.2.1 Encoder

The first component of the encoder is the **Document and Question Encoder**: the question embeddings $(x_1^Q, \ldots, x_n^Q)$ and document embeddings $(x_1^D, \ldots, x_m^D)$ are encoded through the same LSTM, in order to share representation

power, thus obtaining encodings $(q_1, \ldots, q_n)$ and $(d_1, \ldots, d_m)$. Then, to each of these representations a **sentinel**[1] is added, thus defining the document encoding matrix $D = [d_1, \ldots, d_m, d_\phi] \in \mathbb{R}^{l \times (m+1)}$ and the intermediate question encoding $Q' = [q_1, \ldots, q_n, q_\phi] \in \mathbb{R}^{l \times (n+1)}$. Q' is further transformed through a fully connected $\tanh$ layer, in order to allow variation between the question and document spaced, therefore defining the encoding matrix $Q = \tanh(W^{(Q)}Q' + b^{(Q)}) \in \mathbb{R}^{l \times (n+1)}$. The document encoding D and question encoding Q constitute the output of the Document and Question Encoder.

The next component of the encoder is the **Coattention Encoder**. It first computes the affinity scores of each pair of question and document words: $L = D^\top Q$. Then, the **affinity matrix** L is normalized in order to produce attention weights of each question word in light of the document A^Q and of each document word in light of the question A^D. The authors mention normalizing row-wise, but we believe that, according to the original attention paper ([6]), the normalization should be done column-wise[2], such that all attention coefficients sum up to 1:

$$A^Q = \text{softmax}(L) \in \mathbb{R}^{(m+1) \times (n+1)}, \tag{1}$$

$$A^D = \text{softmax}(L^\top) \in \mathbb{R}^{(n+1) \times (m+1)}. \tag{2}$$

After this, the Coattention Encoder computes the summaries of the question in light of each document word $QA^D \in \mathbb{R}^{l \times (m+1)}$ and the summaries of the document in light of each question word: $C^Q = DA^Q \in \mathbb{R}^{l \times (n+1)}$, both representing the vanilla attention. C^Q is further projected into the space of document encodings, thus obtaining by concatenation the co-dependent representation of the question and document, which the authors call **the coattention context**[3] C^D:

$$C^D = \begin{bmatrix} Q \\ C^Q \end{bmatrix} A^D \in \mathbb{R}^{2l \times (m+1)}. \tag{3}$$

The output of the Coattention Encoder is the encoding obtained by fusing together D with C^D through a Bi-LSTM, i.e. encoding $\begin{bmatrix} D \\ C^D \end{bmatrix} \in \mathbb{R}^{3l \times (m+1)}$ via the Bi-LSTM and getting $U \in \mathbb{R}^{2l \times m}$, where the second dimension is m as we dropped the sentinels. In fact, U is the output of the entire decoder.

2.2.2 Decoder

Two HMNs are used iteratively for predicting the start and the end of an answer: HMN_{start} and HMN_{end}. Both take as input the coattention embeddings of the previous predictions for start and end, the coattention encodings $U = [u_1, \ldots, u_m]$, and the state of an LSTM state-machine, and produce the next predictions. In equations, following the notation in [1], if the starting scores over U indices at iteration i (which correspond to document indices) are $(\alpha_1, \ldots, \alpha_m)$ and the ending scores are $(\beta_1, \ldots, \beta_m)$, the HMN's preditct the start index s_i and the end index e_i to be:

$$s_i = \text{argmax}_t(\alpha_1, \ldots, \alpha_m), \tag{4}$$

$$e_i = \text{argmax}_t(\beta_1, \ldots, \beta_m). \tag{5}$$

The scores are computed as follows:

$$\alpha_t = \text{HMN}_{start}(u_t, h_i, u_{s_{i-1}}, u_{e_{i-1}}), \tag{6}$$

$$\beta_t = \text{HMN}_{end}(u_t, h_i, u_{s_i}, u_{e_{i-1}}), \tag{7}$$

where h_i is the hidden state of the LSTM state-machine at step i.

The HMN state-machine takes as inputs its previous state h_{i-1}, and the previous predictions encodings $u_{s_{i-1}}$ and $u_{e_{i-1}}$, then updating the current state:

$$h_i = \text{HMN}_{state}(h_{i-1}, u_{s_{i-1}}, u_{e_{i-1}}). \tag{8}$$

The HMN model mainly relies on the maxout layers, which pool the maximum along the first tensor component, and on the skip connections across layers. It uses a single non-linearity, applied to the state of the HMN state-machine, which is denoted r in the authors' figure (Figure 2).

[1] The sentinel enables the model not to attend to any word in particular. In general, attention coefficients sum up to 1, but it may be the case that, for example, some word in the question is completely unrelated to all the document words.

[2] We tested both options of normalizing, and they yield similar results (see experimental results section).

[3] The given formulation for C^D provides a way to parallelize matrix multiplications.

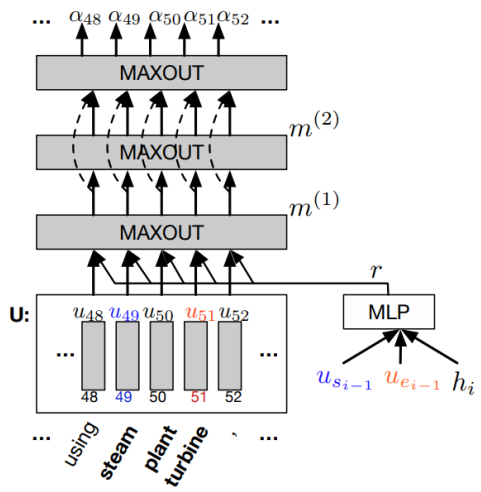


Figure 2: HMN architecture, as drawn by the authors of [1]. The "MLP" box stands in place of the LSTM state-machine, as a multilayer perceptron (MLP) can also be used as a state-machine, which the authors also try.

Training the network consists of minimising the cumulative softmax cross entropy loss of the start and end points over the predicted start and end scores.

2.2.3 Implementation details

The authors mention using a maximum sequence length of 600 during training and a hidden state size of 200 for all units. They use at most 4 HMN iterations, with a maximum pool size of 16 for each maxout layer. They only briefly talk about using dropout, but they clarified both dropout and regularization on Open Review ([9]): "We do not use L2 regularization. We apply 0.3 dropout on the question and document encodings.".

In the paper the authors say they randomly initialize all LSTM weights, and set all initial states to 0. Also, the sentinels are randomly initialized and optimized at train stage. They use the ADAM optimizer ([10]). Their entire implementation is using Chainer.

3 Our implementation and original contribution

3.1 Our implementation

We fully implemented the network in Python 3.5.2, using Tensorflow 1.12.0, while the authors used Chainer. We preprocessed the SQuAD data using the nltk tokenizer and the GloVe pretrained vectors on the large corpus (840B) to embed the tokens. Our implementation is highly customizable, easily allowing for model variations.

3.1.1 Potential differences to the authors' model

We used the same settings as the authors, with a few exceptions which we mention here. We used in our baseline model a max pool size of 4 instead of 16 in order to reduce the training time and to be able to perform more experiments. We trained our model on batches of size 64, the highest power of 2 which could fit on our GPU. In earlier models, which were not using so many resources, we noticed a trend of better model performance for larger batch sizes (up to 256), but unfortunately we had to set for a batch size of 64 given the complete implementation and the GPU limitation. On our model, decreasing the batch size lead to worse training times, as expected, but also to worse or similar results. We did not perform thorough experiments with different batch sizes, as the training was becoming considerably slower for smaller batch size, and we were getting memory exhaustion errors for larger batch size. The batch size the authors use is not mentioned anywhere.

We used ADAM for optimization, as the authors, but we did not really go through a thorough decay rate and learning rate search, as this would have been too time consuming. The optimizer parameters that the authors use are not mentioned neither in the paper, nor on Open Review.

We consider that the $\mathrm{softmax}$ should have been applied on a different axis for obtaining the attention weights A^D and A^Q, as the attention mechanism does. We stick to our interpretation, as both options performed similarly.

The last potential difference between our implementation and the authors' could be the dropout. They succinctly mention in the paper using dropout, and clarify on Open Review ([9]) using a dropout of 0.3 only in the encodings. However, there are many places in the encoder where dropout could be applied: the shared LSTM of the question and document, the $\tanh$ layer, the Bi-LSTM which fuses temporally the document, attention, and coattention. Also, there are various ways of applying dropout: applying it as a separate layer at the end of LSTM encodings, or applying it at cell level. We experimented various possibilities and tried to find the best interpretation of the authors' comments on Open Review. Consequently, we proceeded in our baseline model with dropout applied to the document representation, to the $\tanh$ layer, and to the Bi-LSTM.

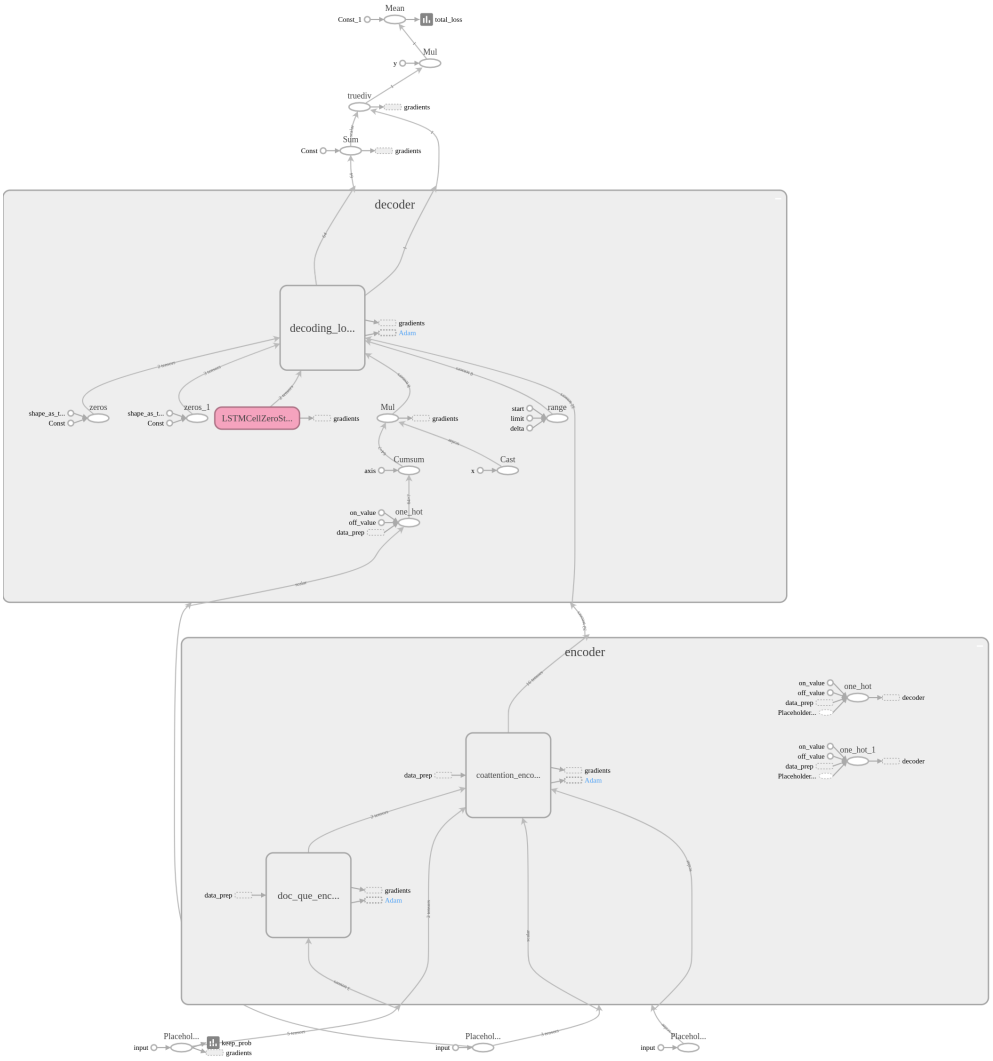


Figure 3: Our implementation visualized in Tensorboard.

3.1.2 Other implementation details

We follow authors' described implementation in all other aspects: we implemented all the described components, used a hidden size of 200, 4 HMN decoding iterations, 600 max sequence length, and no L2 regularization or dropout in the decoder (as mentioned on Open Review).

We included in our data preprocessing paddings, extending each document that has length below 600 to 600, using dummy embeddings (zero vectors). To avoid training over these dummy embeddings, we implemented masks everywhere in our code, which essentially had the role of disregarding the paddings in the graph. Otherwise, we would have got starting and ending scores over the paddings, and the model performance would have decreased, as it would have had to also learn to disregard paddings.

3.2 Our contribution to the paper

Here we present the principles of the novel elements we added to the network, and in the Experiments Section we will also present their experimental results.

3.2.1 Joint distribution over start and end

We implemented a simple mechanism which, at the extra cost of an outer product of 2 vectors per datapoint at validation time, consistently improved the results of any experiment we ran. Instead of outputting the indices with maximum α and β scores (the argmaxes of the final HMN scores), we output the indices whose joint probability is maximal and such that the end index is greater or equal than the start index. Formally, if $(\alpha_1, \ldots, \alpha_m)$ and $(\beta_1, \ldots, \alpha_m)$ are the final start and end scores, we let:

$$(\overline{\alpha_1}, \ldots, \overline{\alpha_m}) := \mathrm{softmax}((\alpha_1, \ldots, \alpha_m)), \tag{9}$$

$$(\overline{\beta_1}, \ldots, \overline{\beta_m}) := \mathrm{softmax}((\beta_1, \ldots, \alpha_m)). \tag{10}$$

The $\overline{\alpha_t}$'s and $\overline{\beta_t}$'s can be seen as probability distributions over document words for start and end indices. We make our model predict $1 \leq i \leq j \leq m$ which maximizes the joint probability $\overline{\alpha_i}\overline{\beta_j}$, i.e. the argmax of (i, j) above the main diagonal in the outer product matrix $(\overline{\alpha_1}, \ldots, \overline{\alpha_m}) \otimes (\overline{\beta_1}, \ldots, \overline{\beta_m})$.

3.2.2 SQuAD v2.0

We propose a simple adaption of the model to the SQuAD v2.0, which contains unanswerable questions. We consider that if a question has no answer in a document of length m, both the start and end indices are defined to be $m + 1$. We feed the appended document D with the co-dependent representation of the question and document C^D to an LSTM, whose final output we use as a sentinel that corresponds to the last position in the decoder.

The LSTM should learn how to build a sentinel correponding to "no answer" based on the entire coattention mechanism of the document and the question, and not just train a standard sentinel to work for all possible documents and questions.

Unfortunately, we only had time to run it for a few epochs, so the results are not conclusive.

3.2.3 Co-coattention

We investigated appending another deeper representation of the document conditioned on the question. A co-coattention interpretation is embedding the coattention in the question embedding space, and then back to document space:

$$\tilde{C_Q} := C_Q A_D A_Q A_D = D(A_Q A_D)^2. \tag{11}$$

The experiments showed that this version loses compactness of the representation [4].

4 Team work and personal contribution

4.1 Team work

We have been splitting the project work in parallel tasks and distributed them fairly among us. The initial planning consisted of 4 main parts: data preparation, the encoder, the decoder, and the train script. In time, the tasks got much more complex and we had to subdivide these main parts and work jointly on them.

Since the beginning of the project (Week 4, Hilary Term), we have had $2 - 3$ meetings per week which varied between 1.5 and 3 hours each. We discussed the theoretical aspects of the paper, potential additions to it, set milestones together, and helped each other. I believe that the organizing part went smoothly, allowing us to focus on the technical part of the project. Overall, each of us has had a considerable contribution.

4.2 Personal Contribution

In the initial stages of the development I have been mainly working on the Encoder and on preprocessing the data. I have implemented most of the Encoder successfully. I also did a large part of the data processing, working together with a teammate on it. Later on, I built entirely the validation part of the script, providing relevant metrics about the training process at the end of each epoch such that we know when we should stop the training. I have also split the training process, dividing epochs in further sub-epochs, and performing validation and metrics gathering between each of these, which allowed us to get for the first time above 60 EM.

I was also responsible for generating all predictions, logs, and metrics, both in human readable format, and in CodaLab format for proper SQuAD evaluation on the dev (validation) set. I have therefore implemented various scripts for these

[4] It got 58.258% F1, we do not report it in the experiments section

tasks, and also grouped and submitted all our experiments to CodaLab. Before using CodaLab for exact scores, I also built an approximation of SQuAD evaluation which we were using locally to approximate our scores.

I have also been communicating with V. Zhong, one of the paper authors, in order to clarify the motivation and intuition behind the paper.

I have been contributing equally to the other team members in terms of ideas, designing experiment cases, and debugging/reviewing other colleagues' code.

5 Experiments

We report our scores obtained on CodaLab with the official SQuAD evaluation script[5]. These are all evaluated over the dev set, therefore we only compare our results to the authors' reported dev scores.

As methodology, we have been plotting the train loss against the validation loss in order to decide when to stop training. We have been generating 3 models during every epoch, on which we also ran our local metrics and CodaLab evaluation.

The main scores that we report are the SQuAD success metrics, the **EM** and **F1**:

- the EM (Exact Match) score is the percentage of predictions which are identical to one of the ground truth answers,
- the F1 score is the harmonic mean of the precision and recall, where the precision is the percentage of words chosen from a correct answer, and the recall is the percentage of relevant selected words.

Both metrics neglect differences in trailing punctuation and articles ("a", "an", "the").

The F1 score is the more relevant metric, as predictions which slightly differ from the correct answers get a high F1 score, but an EM score of 0.

5.1 Baseline model

As discussed in Section 3, our baseline model has a hidden size of 200 for all cells, a max pool size of 4, at most 4 HMN iterations, uses dropout in the encoding Bi-LSTM, document encoding and question $\tanh$ layer of 0.3, and was trained with ADAM, using its default Tensorflow parameters ($learning\_rate = 0.001, beta1 = 0.9, beta2 = 0.999$). As argued in 3.1.1, we apply softmax normalization column-wise for obtaining the attention contexts.

5.2 Reproducing the paper experiments

Following the paper structure, we present and compare our results with the authors' reported main results, model ablation, performance across length and question type, and breakdown of F1 distribution. Unless mentioned otherwise, we perform the experiments on our baseline model.

5.2.1 Main scores

The authors report their single model scores on the dev set as: 65.4% EM and 75.6% F1 (with max pool size 16). Our baseline model achieved dev scores of: 60.757% EM and 71.707% (with pool size 4). The state of the art single model result before this paper was 71% F1, so we can confirm that this has certainly been the state of the art when it first appeared. We believe that we did not achieve the exact same scores because of the differences discussed in 3.1.1, mainly the way of applying dropout, the training parameters, and the resources and time limitations of having to use a batch size of 64.

Remark. The authors report in the "Breakdown of F1 distribution" perfect predictions (100% F1) for 62.2% of examples, which comes in contradiction with the reported 65.4% EM. We believe that it is either a typo, or an experimental inconsistency.

We also trained and tested our baseline model with a max pool size of 16, as the authors did, but have not employed it in further experiments because of limited training time. This is inoffensive as, from both this experiment and from the authors' observations, increasing the max pool size does not have a considerable impact.

[5]The results can be verified at `https://worksheets.codalab.org/worksheets/0x84771936337f4167aa0ab09762dd8280`.

	F1	EM
Authors' Reported Result with max pool of 4	75.2	65.2
Our Reproduced Result (On the baseline model, max pool 4)	71.707	60.757
Authors' Main Reported Result (max pool 16)	75.6	65.4
Our baseline model with a max pool size of 16	71.387	61.306

Table 1: The reported authors' scores on dev set against our baseline model, both presented with both 4 and 16 max pool sizes.

5.2.2 Ablation results

We run various ablation experiments. Strikingly, removing coattention and just fusing together the document with the attention through the Bi-LSTM achieved an F1 score lower than the Baseline Model with only 0.23%, compared to the authors' F1 drop of 1.9%. We consider this to be a limitation of our potential differences to the authors' implementation. Doing only 1 HMN prediction iteration results in an F1 score drop of approximately 0.6%, confirming the authors' claim qualitatively, but differing quantitatively from the drop they report of 1.6%.

	F1	EM
Baseline model (BM)	71.707	60.757
BM without Coattention	71.477	61.230
BM with only 1 HMN loop	71.138	60.691
BM with dropout only in document and question encodings	66.385	55.336
BM without HMN loop converge	70.394	60.009

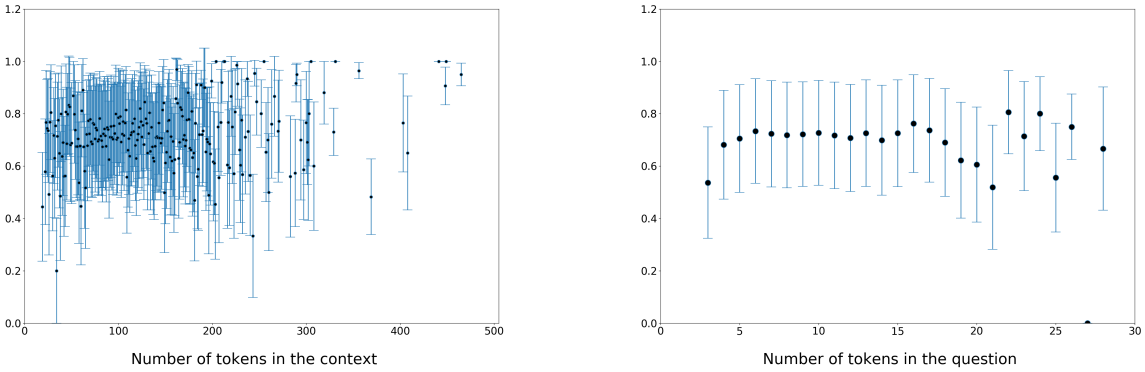
Table 2: Ablation results performed on our baseline model.

We found that giving up on Bi-LSTM dropout, and using only dropout layers on the document and question encodings results massive overfitting, leading to a decrease in F1 of 5.3%.

We also found out that removing converge in the HMN iteration loop, i.e. proceeding for 4 steps of alternating start/end predictions even if the previous predictions do not change anymore, decreased our performance by 1.3% F1.

5.2.3 Performance across length

The authors say that, contrary to the intuition, the model is "largely agnostic to long documents", without showing a sensible degradation for longer questions and documents. We confirm that this is the case, having obtained similar distributions to theirs for average F1 scores in terms of the token numbers in documents and questions. The results are presented in Figure 4.



(a) F1 score across document lengths.

(b) F1 score across question lengths.

Figure 4: We plot with black points the average F1 scores per document/question length, and with the blue vertical bar we represent the standard deviation of F1 at the given lengths.

The authors indicate a performance degradation correlated to longer ground truth answers, justified by the nature of F1. Our experiments also confirm this, as we have obtained very similar results (see Figure 5).

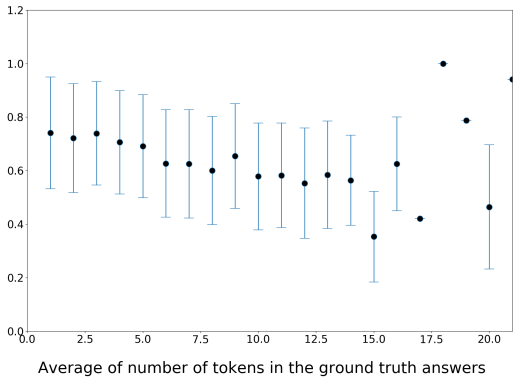


Figure 5: F1 score average and standard deviation across the average of ground truth answers lengths.

We also include the authors' versions of the 3 graphs in here, namely of the F1 scores across documents lengths, questions lengths, and ground truth answers lengths, all of which we reproduced really well.

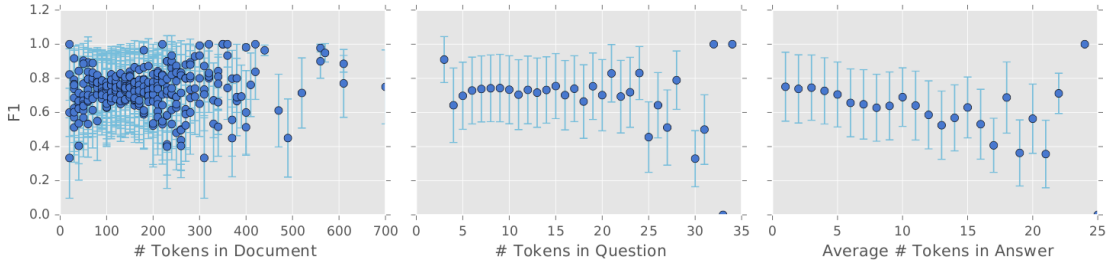


Figure 6: The authors' F1 scores across document, question, and ground truth answers lengths. This figure is from [1]. These are really similar to our results in Figures 4a, 4b, and 5.

5.2.4 Performance across question type

The paper reports the F1 scores across question types. As expected, it obtains the highest scores for "when" questions, and performs worst on the "why" questions, which are more complex. We report the same behaviour, having reproduced this experiment on our baseline model (Figure 7), which is similar to the authors' (Figure 8).

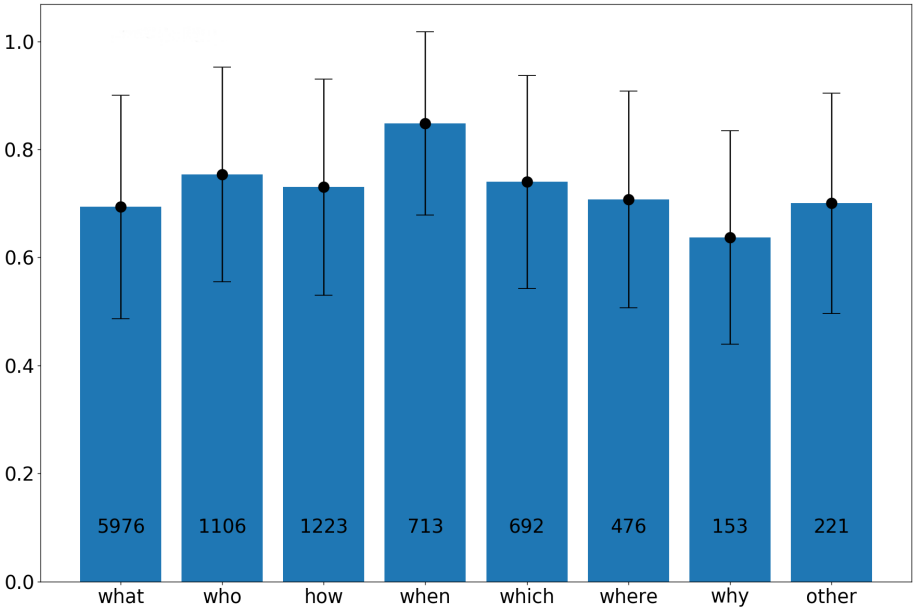


Figure 7: F1 performance of our baseline model across question types. The height of each bar represents the average F1 performance per question type, and the vertical black bar represents the standard deviation. The number written at the bottom of the bars represents the count of each question type.

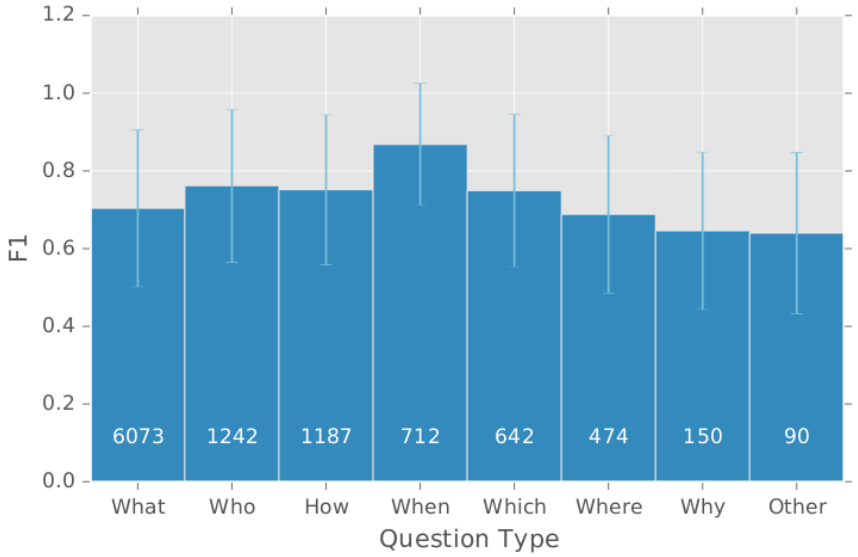


Figure 8: The authors' performance per question type. This figure is from [1]. It is very similar to our experiment: figure 7. There are small differences in the numbers of questions for each class, because of using a slightly different question classifier.

5.2.5 Breakdown of F1 distribution

The authors finally remark that the DCN performance is bimodal, perfectly predicting an answer on the dev set (100% F1) for 62.2% datapoints [6] and completely missing the answer ($0\% F1$) for 16.3%.

Our model perfectly predicts 60.757% of the datapoints from the dev set, and predicts completely wrong 19.451%. Therefore, we confirm that DCN performance is highly bimodal.

	Completely correct answers	**Completely wrong answers**
Authors' Model	62.6%	16.3 %
Our Baseline Model	60.757%	19.451 %

Table 3: The bimodal nature of the model is confirmed in both authors' and our experiments.

5.3 Testing our proposed experiments

5.3.1 Softmax axis

First of all, we compare the performance of our baseline model, which applies the softmaxes column-wise, to the performance of the same model applying the softmaxes row-wise, as the authors say. We previously made a point that our interpretation makes more sense under the classic Attention definition. The results are similar for the F1 score, but for the EM score the baseline model performs better by almost 0.6%.

	F1	**EM**
Baseline Model	71.707	60.757
Baseline Model with changed axis softmax	71.775	60.170

Table 4: Our baseline model applies softmax column-wise, while the authors apply it row-wise.

5.3.2 Joint distribution over start and end - Outer Product Prediction

The simple mechanism that we introduced in 3.2.1, which we refer to as Outer Product Prediction (OPP), consistently improved the scores for the model variations we tried (Table 5). *It allowed us to reach our best F1 performance of* 72.896% *and EM performance of* 62.195%.

	F1	**EM**
Baseline Model (BM)	71.707	60.757
BM with Outer Product Prediction (OPP)	72.896	61.419
BM with only 1 HMN loop	71.138	60.691
BM with only 1 HMN loop, with OPP	72.647	61.675
BM with only attention, no coattention	71.477	61.230
BM with only attention, no coattention, with OPP	72.648	61.911
BM with max pool size of 16	71.387	61.306
BM with max pool size of 16, with OPP	72.650	62.195

Table 5: The OPP we propose has improved consistently the performance of the models we have tried.

[6] This comes in contradiction with their reported EM scores, all approximately $65 - 66\%$

5.3.3 SQuAD v2.0

Unfortunately, we have not had time to experiment properly the adaption of our network for SQuAD, having only trained for around 2 epochs. The results on the dev set of SQuAD v2.0 [7] are however promising for only 2 training epochs and no tuning at all, achieving an F1 score of 49.44%, with an average F1 over unanswerable questions of 73.07%, and an average F1 over answerable questions of 25.41%. We believe that having trained the network more epochs, tuning it, and adding a larger penalty for predicting "no answer" would have lead to more consistent results.

6 Conclusions

6.1 Work and results assessment

We have successfully implemented the DCN architecture and obtained scores that would have beaten the previous state of the art results before the paper [1] was initially published.

Our baseline model F1 score was 71.707%, compared to the authors 75.6%, and to the previous state of the art of 71.0%. We justify the score difference in 3.1.1, which we believe is mainly a result of where to apply dropout (we approximated around 60 possible dropout applications that would make sense and be consistend with the authors' description on Open Review), of tuning the optimizier, and of a different batch size.

Using the Outer Product Prediction mechanism which we introduced at virtually no extra cost, we increased our baseline model performance to an F1 score of roughly 72.9%. By also using the max pool size of 16, we achieved an EM score of 62.195%.

We observed that the F1 performance of the DCN increased by adding coattention, having 4 HMN loop iterations instead of 1, and using the converge mechanism in the HMN loop. However, adding coattention increased the performance only marginally, which we believe is a result of the particularities of our implementation.

We confirm all the qualitative results of the authors regarding the performance of DCN with respect to the question, document, and ground truth answers lengths, and also with respect to the question type. The network seems to be largely agnostic to the size of the question and document, and to have a slowly decreasing trend with larger correct answer spans.

6.2 Further extensions and possible improvements

Given more time, a direction to proceed would have been going through a more thorough configuration search. Namely, we would have experimented most of the possible dropout configurations in order to clarify exactly what the authors meant by applying dropout in the encoding on Open Review([9]). Also, we would have tried various optimizer configurations and batch sizes, however limited by the GPU in terms of the maximum batch size. Furthermore, we would have definitely used a baseline model with max pool size of 16, as the authors do, instead of 4.

In the perspective of more time, the main addition we would have tackled rigorously would have been properly calibrating and training our SQuAD v2.0 network extension. We believe that creating a sentinel vector from the coattentive representation of the document and question which corresponds to "no answer" and passing it to the decoder would yield decent results.

Total Words: 4564

Headers: 42

Math Inline: 161

Math Display: 7

[7] The results can be found at `https://worksheets.codalab.org/rest/bundles/0x12b0f5e02f114b72b0259a5a4496138a/contents/blob/eval.json`

References

[1] Caiming Xiong and Victor Zhong and Richard Socher. Dynamic Coattention Networks for Question Answering. Salesforce Research, 2016.

[2] Yarin Gal and Thomas Luckasiewicz. Lecture Notes, Advanced Machine Learning Course, University of Oxford. https://www.cs.ox.ac.uk/files/10744/lecture11.pdf

[3] Christopher J.C. Burge Towards the Machine Comprehension of Text: An Essay. Microsoft Research, 2013.

[4] Pranav Rajpurkar and Jian Zhang and Konstantin Lopyrev and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. Computer Science Department, Stanford University, 2016.

[5] Jeffrey Pennington and Richard Socher and Christopher D. Manning . GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/, 2014.

[6] Dzmitry Bahdanau and Kyunghyun Cho and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. Published at ICLR 2015.

[7] Caiming Xiong and Victor Zhong. State of the art deep learning model for question answering. https://www.salesforce.com/products/einstein/ai-research/deep-learning-model-for-question-answering/#, 2016.

[8] Ian J. Goodfellow and David Warde-Farley and Mehdi Mirza and Aaron Courville and Yoshua Bengio. Highway Networks. arXiv:1302.4389

[9] Caiming Xiong and Victor Zhong and Richard Socher Open review page for "Dynamic Coattention Networks For Question Answering". https://openreview.net/forum?id=rJeKjwvclx

[10] Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. Published as a conference paper at ICLR 2015.