Building upon Multi-Perspective Matching for SQuAD

Louis Duperier duperier@stanford.edu codalab: duperier

Yoann Le Calonnec yoann@stanford.edu codalab: yoann

Abstract

In this paper, we describe a model to answer questions using informations contained in context paragraphs. An answer is a short contiguous segment of the context. The SQuAD dataset offers a hundred thousand question-answer pairs generated by humans from Wikipedia context. We present a model derived from multi-perspective matching from Z. Wang et al. (2016) that leverages perspective functions to match relevant context segments with the question.

1 Introduction

Question Answering (QA) is a crucial task in Natural Language Processing (NLP). It was shown that any NLP task can be transformed into a QA task. For instance, translating a sentence S from English to French is the same as asking: How do you translate the sentence S in French? Hence, QA can be seen as an universal, high-level abstraction of NLP tasks. Notably, QA ties to Machine Comprehension, because answering a question usually requires understanding the question itself, and other potential sources of information.
Reference datasets used to be manually crafted thus intrinsically very limited in size, which prevented researchers from building powerful, expressive models. The broad range of impactful problems that can be solved using QA motivated the need for a massive, high-quality dataset. For this reason, Rajpurkar et al. (2016) built the SQuAD dataset.
The SQuAD dataset is a very large (100k+ samples) dataset made of paragraphs of text, called contexts, and pairs of (questions, answer) about the context where the answer is a segment from the context. A very simple example could be: if the context is "the best way to learn NLP is to take CS224N", the question could be "what is the best way to learn NLP?", and the answer would be "take CS224N".

Manual analysis of SQuAD shows that it contains examples requiring various reasoning techniques to predict the correct answer. This diversity and richness poses an interesting challenge for the researcher, because models must be expressive enough to capture and reproduce those different forms of logical thinking. On the other hand, resulting models are expected to generalize very well to other tasks.

This paper first describes previous work in the area, which is easily accessible via the online leaderboard for SQuAD. Then, we dive into the technical approach we used to solve this problem. The next section details the experiment we ran, the results we obtained and the conclusions and questions they raised. Lastly, we'll conclude on the work done.

2 Related Work

Our approach is mainly inspired by the work of Z. Wang et al. (2016). He has demonstrated that a multi-perspective matching architecture similar to the one we implemented as described below is able to achieve results close to state-of-the-art on the SQuAD dataset.

The key takeaway from Wang's approach is the use of perspective functions as a way to correlate, cross-reference and fuse informations from the question and the context. They also proposed a classification of QA models targeting SQuAD into two classes: boundary identification, where we aim to predict directly the answer span, and chunking and ranking, where we first identify potential answers before ranking them.

Notably, Seo et al. (2017) have built upon the multi-perspective architecture, proposing a Bidirectional Attention Flow for Machine Comprehension (BiDAF). In addition to filtering context using the question, Seo symmetrically filters the question using the context, to extract relevant part of the questions. This allows him to entirely skip the perspective layer and achieve slightly better F1 score, while training performance increases. However, we found that this model was extremely hyperparameters-dependent since our implementation yielded a score of 31% F1.

Even though their model is very different from ours, S. Wang et al. (2017) yields several new ideas that we successfully adapted into our model: the tanh layer, doubly-stacked BiLSTM or a window-based answering fallback technique. In general, we took inspirations on several implementation details from various papers out of the scope of SQuAD papers.

3 Approach

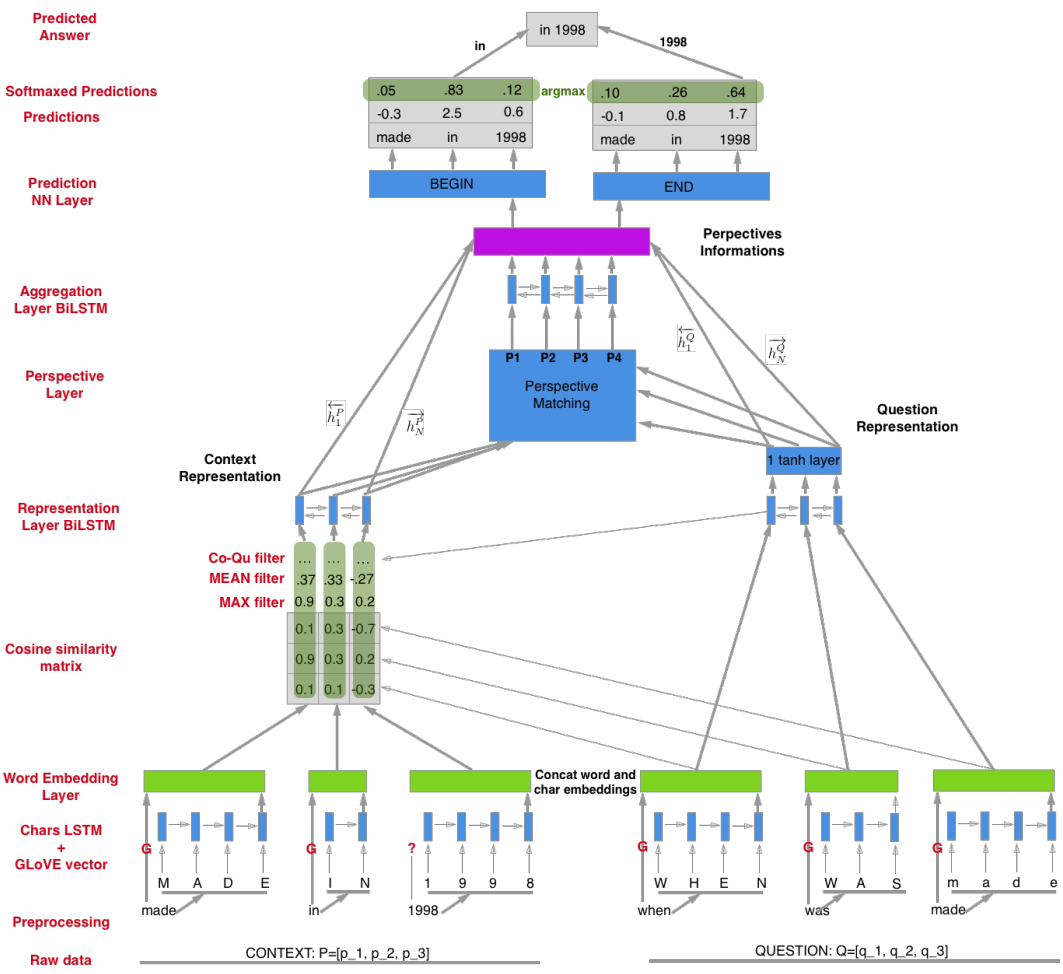

Figure 1: Multi-perspective matching network

We have implemented a multi-perspective matching algorithm. Let us assume that the paragraph, or context, has n words, and the question has m words. The model is made of 6 layers:

Embedding outputs context and question representations $\mathbf{P} = [p_1, \ldots, p_n] \in \mathbb{R}^{d \times n}$ and $\mathbf{Q} = [q_1, \ldots, q_m] \in \mathbb{R}^{d \times m}$ by embedding each word into a vector made of the concatenation of a pretrained word embedding and a char-based word embedding.

Attention filter selectively filters the context words that will be useful to answer this specific question, by weighing each word from the context: decreasing the norm of useless words to reduce their relative importance. Outputs $\mathbf{P'} = [p'_1, \ldots, p'_n] \in \mathbb{R}^{d \times n}$.

Representation fuses all words from the question (resp. the context) into one vector that captures the meaning of the question (resp. the context), using two BiLSTMs. The previous filtering step is crucial in making sure that useless words, noise, has as little influence as possible on the context representation. Turns $\mathbf{P'}$ and $\mathbf{Q}$ into $\mathbf{P}_h \in \mathbb{R}^{n \times 2h}$, $\mathbf{Q}_h \in \mathbb{R}^{m \times 2h}$.

Perspective "reads" the context from different perspectives while referencing to the question. The process is analogous to analyzing the (context, question) pair using different techniques and reasoning to extract relevant informations, where the techniques themselves are determined and learned by the algorithm. Outputs a perspective matrix $\mathbf{R} \in \mathbb{R}^{n \times 6p}$

Aggregation fuses extracted informations from the perspective layer with representations for context and question. Outputs $\mathbf{R}_a \in \mathbb{R}^{n \times h}$ that conveys the answer to the question on this specific context.

Prediction transforms the abstract, high-dimensional answer vector into two numbers: the predictions for the positions of the beginning and the end of the answer within the paragraph.

In this section, we will go into details through the successive layers of our model, from raw data to predicted answers.

3.1 Preprocessing

Upon evaluation of our models, we realized that three phenomena could impact performance:

- Words missing GLoVE encodings are encoded with random vectors. This can be solved using the larger corpora of GLoVE vectors (2.2M tokens), but since they are only provided in dimension 300, this negatively impacts training speed.
- Words that do have a GLoVE vector but are not in the vocabulary (because they are in test but not in train dataset) will also be encoded as the token *unk* (unknown). This causes on average a 6% drop in performance, but can be fixed by not restricting the embeddings to the train vocabulary when evaluating the model.
- Symbols-based words (such as 1970 or 10.5%).

In those three cases, the default encoding mechanism loses the meaning of those potentially meaningful segments. Hence, in addition to the default preprocessing (turning words into ids referencing the vocabulary list), we have added a character-wise encoding. Similarly to words, each letter of the context and question is encoded into an integer. Since there are only 170 distinct characters, this modification does not have a major impact on the number of parameters to the model (with encodings of size 40, this is 6800 parameters, or less than 1%). This allows for a better expressiveness in those cases.

3.2 Word embedding layer

Each word of the question and the context is represented by a d-dimensional vector. It is the concatenation of the word's GLoVE vector of dimension d_{glove} (typically 50 or 100) and a character-wise embedding of dimension d_{char} (typically 40). The GLoVE vector is obtained

by a simple lookup operation.
For the character-wise embeddings, we use the following process:

1. Let d_c be the number of characters in our dictionary. We choose an arbitrary embedding dimension for each character d_e. We initialize a random embedding matrix of dimension $d_c \times d_e$ using a Xavier initializer.

2. We lookup into this matrix to convert each character into its embedding.

3. We feed each character embedding in the order they appear in the word into a LSTM that fuses the embeddings. The final state represents the entire word meaning. It is called the character-wise word embedding, and is concatenated to the GLoVE vector for the word.

As a result, context (resp. question) is a list of n (resp. m) vectors of dimension d, one for each word. This vector summarizes as well as possible the word meaning.

3.3 Attention layer

Contexts are usually relatively long (several hundreds of words), because they discuss a broad topic, and most of the information contained in a passage is not relevant for answering the question. A specific question will only require to comprehend a subset (not necessarily contiguous) of the passage, which contains the answer itself and the clues needed to find the answer. The attention layer will make sure that our model identifies the crucial elements in the context, and pays very little attention to the noise generated by the rest of the passage.

As in Wang et al. (2016), we compute a coefficient of relevancy $r_{i,j}$ for each pair of words (p_i, q_j) from the passage and the question and pick the maximum over question words: for a word of the passage p_i, $r_i = \max_j r_{ij}$. The idea is that if a word in the passage has a high "similarity" with at least one word in the question, it must be important, and vice-versa. Then, we set $\mathbf{p}'_i = r_i \times \mathbf{p}_i$.

We use cosine similarity to compute $r_{ij} = \frac{\mathbf{p}_i^T \mathbf{q}_j}{||\mathbf{p}_i|| \cdot ||\mathbf{q}_j||}$, as it leverages the properties of word embeddings (words are clustered in this high-dimension space in a meaningful way). Radovanovi et al. (2010) have shown that since cosine distance is closely related to L2 norm, it can suffer from the curse of dimensionality (especially for GLoVE in dimension 300). Hence, we have experimented with dimensionality reduction via PCA before computing cosine similarity.

We have also found that applying a second filter, this time using a mean $r_i = \frac{1}{M} \sum_j r_{ij}$ proves to be useful. Indeed, the filter proposed by Wang might emphasize heavily a word of the context that triggered only one word in the question, when using the mean helps the model isolate words that are heavily related to the entire question. For similar reasons, good results can be obtained by using $r_i = cosine(p_i, [\overleftarrow{h}^q_1, \overrightarrow{h}^q_N])$, where h are final states from the BiLSTM that fuses the informations of the question (see representation layer). A high cosine similarity will mean that the context word echoes the entire question. We call this the Co-Qu filter.

Ablation study have highlighted the importance of this layer, so we spent a lot of time fine-tuning it. For this, we have developed a visualizing mechanism that shows the emphasis put on each word of the context. This allowed us to easily judge the quality of the filtering layer and build intuitions. That said, we are very aware that the behavior of the model should not be expected to mimic human reasoning: what sounds normal or good to us might not be ideal for the model.

3.4 Context/Question representation layer

We have representation of each word in the question and the now question-aware context. We use two BiLSTMs, one for the question and one for the context. Each of those BiLSTM produces two sets (backward and forward) of n (for the context) or m (for the question) vectors. We have $\mathbf{h}_i^p = [\overrightarrow{\mathbf{h}_i^p}, \overleftarrow{\mathbf{h}_{n-i}^p}] \in \mathbb{R}^{n \times 2h}$ where

$$\overrightarrow{\mathbf{h}_i^p} = \overrightarrow{\text{LSTM}}(\overrightarrow{\mathbf{h}_{i-1}^p}, \mathbf{p}_i)$$

$$\overleftarrow{\mathbf{h}_i^p} = \overleftarrow{\text{LSTM}}(\overleftarrow{\mathbf{h}_{i+1}^p}, \mathbf{p}_i)$$

We tried to add a second layer of BiLSTM stacked on top of the first layer proposed by Wang et al., to give more expressivity and power to the model. We saw small variations in accuracy that we deemed insignificant, suggesting that the LSTM structure was not sufficient. For this reason, we wanted to try using Convolutional Neural Networks (CNN) or a Dynamic Neural Network approach where representations are not built linearly but by a constant back and forth to identify part of interests in the text. Time constraints prevented us from training those models. That said, our analysis suggest that the main bottleneck in the performance of the model is not the representation layer.

Also, it is possible that questions and contexts are not encoded into the same "meaning space" by their respective BiLSTM. In other words, a same vector in the question space or the context space might have completely unrelated meaning and properties. To allow comparison, we added an intermediary reconciliation layer: let $\mathbf{W} \in \mathbb{R}^{2h \times 2h}$, $\mathbf{b} \in \mathbb{R}^{2h}$ be trainable variables, then $\mathbf{h}_i^q = \tanh(W\mathbf{h}_i^q + b)$ This simple modification yielded an increase of about 4% in F1 score.

3.5 Perspective layer

At this stage, we have a vector of size $2h$ for each word of the question and the context. The perspective layer allows us to compare the context against the question using different techniques, called perspectives. Basically, we try different matching/similarity methods, concatenate their results, and let the following layer learn how to use those informations to identify the answer within the passage. As in Wang, we define $\mathbf{r} = f_r(v_1, v_2, \mathbf{W}) = \text{cosine}(W \circ v_1, W \circ v_2)$, where $W \in \mathbb{R}^{p*2h}$ is trainable (p is the number of perspectives). We use six trainable matrices $W^1 ... W^6$, and define

$$\overrightarrow{\mathbf{r}_j^{\text{full}}} = f_r(\overrightarrow{h_j^p}, \overrightarrow{h_m^q}, W^1)$$

$$\overrightarrow{\mathbf{r}_j^{\text{max}}} = \max_{i=1..m} f_r(\overrightarrow{h_j^p}, \overrightarrow{h_i^q}, W^2)$$

$$\overrightarrow{\mathbf{r}_j^{\text{mean}}} = \frac{1}{m} \sum_{i=1}^{m} f_r(\overrightarrow{h_j^p}, \overrightarrow{h_i^q}, W^3)$$

Symmetrically, we define $\overleftarrow{\mathbf{r}_j^{\text{full}}}$, $\overleftarrow{\mathbf{r}_j^{\text{max}}}$ and $\overleftarrow{\mathbf{r}_j^{\text{mean}}}$. Concatenating those 6 vectors for each word of the context yields the matrix $R \in\in \mathbb{R}^{n*6p}$. The idea behind those three strategies is to allow the question to select relevant parts of the context, contiguous or not. For instance, if the part of the context that matches the question is on the left of the answer, $\overrightarrow{\mathbf{r}_j^{\text{full}}}$ will be useful. When the matching parts of the contexts are spread across the answer segment, mean and max matching are useful.

3.6 Aggregation layer

Each perspective function yields 6 perspective vectors per word of the context. Identically to the representation layer, we can fuse the information of all the perspective functions by using a BiLSTM. The output of this layer is simply the concatenation of the forward and backward hidden states of the LSTMs. From $\mathbf{R} \in \mathbb{R}^{n*6p}$ we obtain a matrix $\mathbf{R}_a \in \mathbb{R}^{n*h}$.

3.7 Prediction layer

We use two feed forward neural networks to extract the information from the perspective vector: one predicts the beginning of the answer span, the other predicts the end. We found that adding the representation of the context and the question to the data fed to this layer improved F1 score. Hence, the input matrix $\mathbf{S} \in \mathbb{R}^{n\times(6p+4h)}$ is the concatenation of all the BiLSTM-fused perspectives and the representation of context and questions: $\mathbf{S}_i = [\mathbf{R}_i, \mathbf{h}_N^p, \mathbf{h}_N^q]$. The output is $\mathbf{o}_b \in \mathbb{R}^n$ or $\mathbf{o}_e \in \mathbb{R}^n$, and the arg max yields b, e, integer positions of begin and end. We use a single hidden layer, and ReLU non-linearities.

4 Performance

In our case, performance issues were extremely time consuming. Without any optimization, the model described does not allow for a batch size greater than 5 to avoid out of memory errors on the 8Gb GPU of our Azure VM. Hence, we implemented several tricks to improve performance:

Clipping lengths Plotting the histograms for context and question lengths allowed us to see that less than 1% of the contexts are longer than 300 words, and questions 25 words. Keeping very long contexts and questions causes allocation of huge matrices. In other words, 99% of the data could be processed with a greater batch size, but the presence of those long samples prevented us from increasing the model's batch size. We simply removed those examples from our training set, and kept them as testing data. We expect the expressiveness of the model to be unnoticeably altered by the absence of those long sentences in the training data, bringing about 100% performance gain in training time.

Adaptative batch size We use adaptative padding: we pad questions and contexts to the length of the longest one in the batch, instead of padding everything to one single length, which adds a lot of computational overhead for processing useless states. Similarly, when a batch is made of only smalls questions/contexts, we extend its size to fill the memory as much as possible

Precomputing feed dictionaries We realized that by computing feed dictionaries on the fly at each batch, GPU usage was below 30% because of the delay between batches. Precomputing all feed dictionaries initially allow us to store them in a Tensorflow queue, which has a RAM impact but increases GPU usage to almost 100%.

5 Experiments

We have three datasets: train (80k samples), test (10k samples) to measure performance on an unseen dataset, validation (5k) for hyperparameter testing. Only once we reached a good model and we have shown it does not overfit, we retrain it using those three datasets and the dev datatset. Evaluation of the models are done using two criteria: average F1 and EM scores. F1 is the number matching words between prediction and correct answer, normalized by the length of the prediction to prevent model from simply returning the entire sentence. EM is a binary indicator that is 1 when the prediction exactly matches the correct answer. Sometimes, shifting the predicted segment by one word still makes a good quality answer but would yield an EM of 0, that is why the more progressive F1 score is used as the primary measurement.

5.1 Results

Even though we started with a simple reimplementation of Wang's model, our performance does not quite compare with his paper. We reached 54% on our test set, less on the leaderboard because of the unknown word issues described above. We have ran extensive debugging, displaying gradients and variables, and using Tensorboard to visualize learning. We identified several reasons for this lower performance than Wang's model:

1. An epoch takes about 14h on the VM: computing the perspectives is extremely time consuming, and the space requirements force us to use a very small batch size. Since we got the model working late and we had several experiments to run, we could not run the training for more than 2 epochs and stop it even though it was still learning.
2. We have noticed the impact of dropout on performance, but we have not been able to figure out the ideal places to use dropout. That said, comparing performance on train and test sets show that we do not overfit.
3. We manually analyzed results from the attention layer, and they seemed coherent with our expectations.
4. We spent a lot of time trying to modify and improve the model, as per the course guidelines that said that simply implementing an existing model is not enough. For this reason, we did not have enough time to fully diagnose Wang's basic model.

We have completed several model runs throughout the course of the projet. We started with a minimum viable model, with only one perspective function, no chars embeddings, glove vectors of size 50. Training was fast and quickly yielded 21% F1, confirming that the model worked. Then, we added the two remaining perspective functions, and reached a F1 score of 40% after two epochs. We added char embeddings and noted an increase to 45%. Then, we cross-experimented with the tanh layer, doubled BiLSTMs (unconvincing results), glove vectors size (higher dimensions systematically outperformed lower dimensions, with the inter-dimension gap reducing as dimension was increasing). We also changed prediction network depth (we used two hidden layers for our biggest models to summarize more smoothly the huge matrices into two integers. Again, the inability to run training for 10 or more epochs made hard to confirm model power, we were merely confirming that they could learn in a few epochs, but never actually reached a hard plateau. Our best model has a F1-score of 51% on our test dataset.

Due to time and computing power limitations, we did not have to build an ensemble model. The literature shows it would yield an almost guaranteed increase in performance of a 2-3%. Our plan was to add the outputs of the final softmax functions (the probability distributions for begin and end) of several models, and pick the argmax on the sum.

5.2 Analysis of errors

Manual analysis of errors made by the model reveal that the question type if very frequently well understood: *when* is answered by a date, *whose* is answer by a named entity, etc. Indeed, questions that are of the true/false type, a more blurry type, have the lowest success rate, because it is harder to interpret what is expected.

1. "the university's center in beijing is located next to what school's campus?", answers "hong kong" instead of "renmin university"
2. "when did the warsaw uprising begin?", answers "1944" instead of "August 1944"

Some answers appear correct to our human brains. For instance, "what did davies want to build?" has "nationwide network" and "proposed to build a nationwide network in the uk" as official answers, but our model predicts "nationwide network in the uk" which seems correct as well. One answer is "11" when the context only says "around 11", which is the predicted answer and should be an accepted answer.
We also noted that for hard contexts (i.e. several words are OOV), our model tends to predict very wide or completely unrelated answers. It seems to underutilize the information

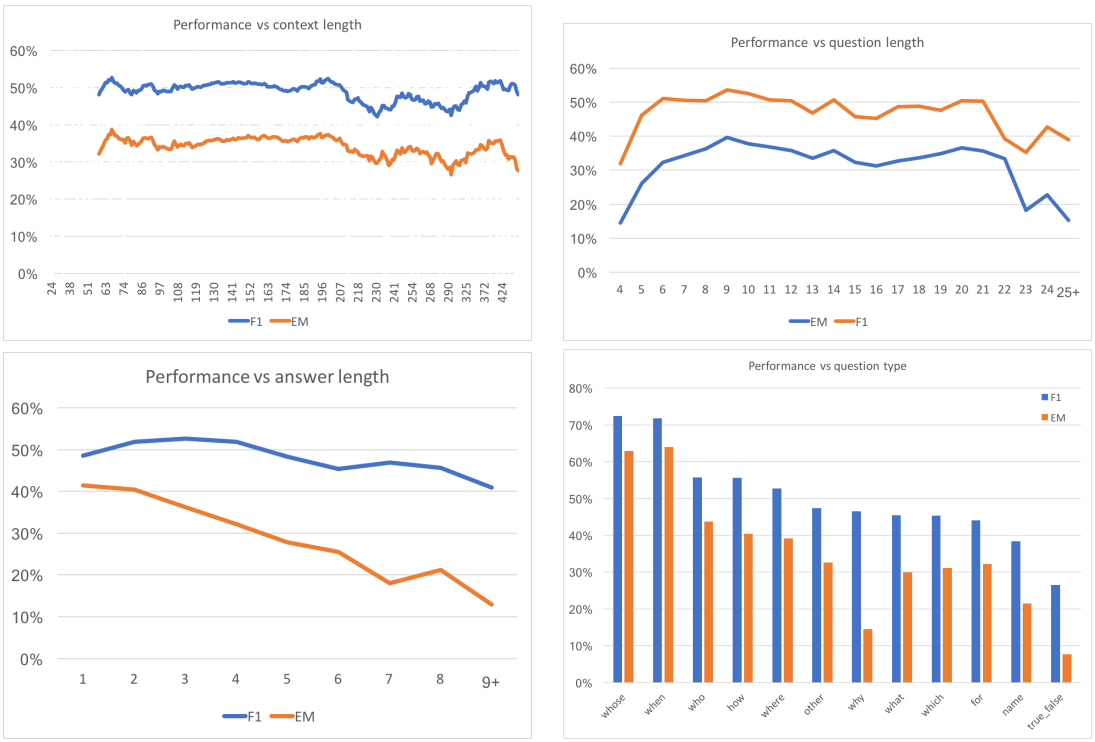

Figure 2: The performance is strongly damaged when increasing the answer length, whereas our model's prediction is not affected by neither the context or the question length. The bottom right histogram presents the F1 and EM scores given the type of question.

from the known words to extrapolate that the unknown word is indeed the answer, and the predictions are mere results of numerical instability as the probability distribution seems uniform on several potential answers.
In cases where the answer is spatially and syntactically close to other false answers, the model fails. For instance, if the context states that "A played against B, and B won against A", the close proximity and the identical grammatical statuses of A and B confuse the model, and probability distribution reveal that it can't differentiate. This hints at the lack of real, deep understanding of the model, which merely matches question and context and predicts the closest entity with the correct grammatical type (person for who, date for when, etc).

5.3 Hyperparameters

The model has various hyperparameters (learning rate, dropout, etc.) and adjustable techniques (activation functions, similarity metric, etc.). Among the reasonable option, the ability to pinpoint the ideal set of parameters is mainly dependent on available time and performance, because those can only be determined through experimenting. Hence, we have not been able to explore all the parameter space, and we had to rely on our intuition to make those choices as we lacked the time required to run more experiments.

We apply dropout over the char LSTM cells, the representation layer BiLSTM cells, the aggregation layer final BiLSTM cells, and all the hidden layer of the feed-forward network in the prediction layer.

6 Conclusion

We'd like to thank the CS224N team for providing us with this unique opportunity to work on a real-world research task. It was extremely positive in terms of improving our NLP and Tensorflow skills. Unfortunately, we estimate that we spent more than 50% of the time working on issues that were not directly related to the model: bugs, preprocessing, performance issues, etc. Even though those setbacks prevented us from improving our model as much as we wanted it, we learnt a lot from our mistakes and will be more efficient for our next NLP project.
By the end of the project, we have started exploring promising improvements and new approaches to the initial model: CNNs as a representation layer, improved prediction layer, etc. We also noticed that carefully adding variables at some points of the model could yield improved expressivity (for instance, doubling the BiLSTM or adding a tanh layer). Future improvements should include continuing exploration of those areas, as well as collecting low-hanging fruits via: systematic hyperparameters optimization, ensembling, improved prediction function (use a fallback window-based approach instead of predicting empty answers).

References

[1] Zhiguo Wang, Haitao Mi, Wael Hamza and Radu Florian. *Multi-Perspective Context Matching for Machine Comprehension*. arXiv:1612.04211, 2016.

[2] Minjoon Seo1, Aniruddha Kembhavi, Ali Farhadi, Hananneh Hajishirzi. *Bi-Directional Attention Flow for Machine Comprehension*. arXiv:1611.01603, 2016.

[3] Shuohang Wang, Jing Jiang. *Machine Comprehension Using Match-LSTM and Answer Pointer*. arXiv:1608.07905, 2016.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, Percy Liang. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. arXiv:1606.05250, 2016.

[5] Caiming Xiong, Victor Zhong, Richard Socher. *Dynamic Coattention Networks for Question Answering*. arXiv:1611.01604, 2017.

[6] M. Radovanovi, A. Nanopoulos, M. Ivanovi. *On the Existence of Obstinate Results in Vector Space Models*. SIGIR, 2010.