CAESAR: Context-Awareness Enabled and Summary-Attentive Reader

Kshitiz Tripathi
Yahoo Inc.
kshitiz@stanford.edu

Long-Huei Chen
Stanford University
longhuei@stanford.edu

Mario Rodriguez
Salesforce.com
mario.rodriguez@salesforce.com

Abstract

Comprehending meaning from natural language is the primary objective of Natural Language Processing(NLP), and text comprehension definitely is the cornerstone for achieving this objective upon which all other problems like chat bots, language translation and others can be modeled in some way or the other. In this report, we first present a brief analysis of some of the state of the art(SOTA) solutions for the problem of Reading Comprehension using the recently published Stanford Question Answering Dataset (SQuAD) [1]. Next we detail our implementation of two of these models: Match LSTM [2] and Dynamic Coattention Networks [3]. We also describe our attempts of using the recently developed attention mechanism constructs, sequence to sequence learning(seq2seq) [4]. Our best trained model reached close to matching the results obtained by these advanced solutions. Finally, we describe our novel attempt of employing a summary attentive reader which tries to make the training process efficient yet retaining the same scores.

1 Introduction

Endowing machines with the ability to understand and comprehend meaning is one of the ultimate goals of language processing, one that holds promise to revolutionize the way people interact with and retrieve information from machines [5]. The goal was outlined by Richardson et al. with the MCTest dataset they proposed [6], in which questions are provided for which the answer can only be found in the associated passage text. To perform well on such task, machine comprehension models are expected to possess some sort of semantic interpretation of text along with probabilistic inference to determine the most probable answer.

Building on the basis of similar evaluation metrics of machine comprehension, Rajpurkar et al. recently released the Stanford Question Answering dataset (SQuAD) [1], which has quickly become the de facto standard of comparison among machine comprehension models [7]. The reason of the dataset's wide applicability is several-fold: first, the dataset is at least 2-orders of magnitude larger from other similar releases. Secondly, the questions are human-curated and are realistic approximation of real-life needs. Thirdly, the corresponding answers to each question are of a variety of nature and can test the generalizability of machine comprehension. Finally, the answer can be an arbitrary span instead of one of the pre-determined choices. SQuAD is comprised of around 100K question-answer pairs, along with a context paragraph. The context paragraphs were extracted from a set of articles from Wikipedia. Humans generated questions using that paragraph as a context, and selected a span from the same paragraph as the target answer.

Here we propose a novel approach to better question-answering by preprocessing document text before answer selection from the passage. This is achieved by a summarization engine that is applied to truncate document length to enable narrower focus on the correct answer span. The summarization takes into account the nature of the question, and therefore can help eliminate unrelated answer candidates in the passage that are likely to be erroneously selected. The summarization engine also assists in machine efficiency, since the encoder is more likely to focus directly around the answer span. The summarized document is then passed through a coattention-based encoder, and the encoded message is passed along to an answer pointer decoder to predict the answer.

2 Related Work

2.1 Sequence-to-sequence Models and Attention Mechanism

Sequence-to-sequence models [4] map input sequences directly to output sequences, as illustrated in figure 1. Such architectures generally consist of two recurrent neural networks (RNNs): an encoder RNN that processes the input and a decoder RNN that generates the output. In the basic model depicted in figure 1, every input has to be encoded into a fixed-size state vector, as that is the only thing passed to the decoder.

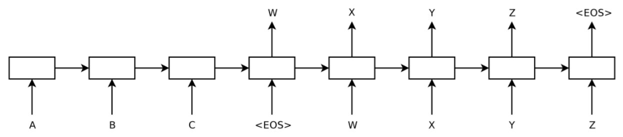


Figure 1: A sequence-to-sequence model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model consists of two recurrent neural networks (RNNs): an encoder that processes the input and a decoder that generates the output. Each box in the picture represents a cell of an RNN, most commonly a GRU cell or an LSTM cell. The model stops making predictions after outputting the end-of-sentence token.

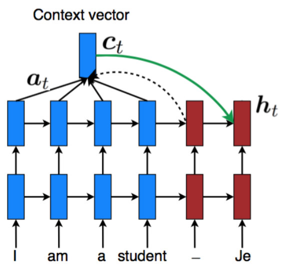


Figure 2: A 2-layer recurrent architecture depicting an attention-based mechanism for translating a source sequence "I am a student" into a target sequence prompted by the GO symbol (dash). The context vector c_t, a weighted average of all the vectors in the input sequence, becomes another input into the cell that makes the prediction at time step t.

In order to provide the decoder with more direct access the entire input sequence, not just the fixed-size state vector output by the encoder, an attention mechanism was introduced in [8], thus better allowing these kinds of models to cope with longer input sequences. [9] analyzes variations on attention architectures as applied to neural machine translation sequence-to-sequence models. In its simplest case, in addition to each cell in the decoder RNN receiving as input both, the output of the previous time step's cell (or the GO symbol if starting the output sequence) as well as the hidden state of the previous' time step cell (or the fixed-size state vector output by the encoder if starting the output sequence), it also receives as input what is known as the context vector. This context vector, c_t, is essentially a weighted combination of the vectors forming the input sequence, where the weights are learned parameters (e.g. a_t). One of the many variations of this attention-based mechanism is depicted in figure 2, in a 2-layer sequence-to-sequence model.

In our work, broadly, we aim to map from question-paragraph sequences directly to answer sequences; thus, sequence-to-sequence models enhanced with attention mechanisms seem like a good fit for baseline models.

2.2 Machine Comprehension and Question-Answering

Here we outline an overview of the various high-performing models evaluated on the SQuAD dataset. Instead of summarizing and repeating similar functionality shared among the implementations, in the following we list features are are unique to the models, from which we drew inspiration to further improve our CAESAR question-answering model.

- **Multi-Perspective Context Matching** : During preprocessing, character-level embedding are created from a LSTM RNN to assist with words with which pre-existing word embeddings are not available [10]. Then the document is filtered based on word-level cosine similarity between question and document. The key to the model is the MPCM layer, which compares the document with question in multiple perspectives that are applicable to different kinds of answer spans as seen in passage.

- **Match-LSTM**: In this model, bi-directional match-LSTM RNN created from concatenation of document and attention-weighted question is applied to predict the answer span [2]. The answer-pointer layer predict answer from the 2 directions of the LSTM bi-directional RNN hidden state. Sentinel vector is appended at the end of document to facilitate determination of answer span ends.

- **Dyanmic Coattention Network** : In the dynamic coattention network, the coattention mechanism uniquely attends to both the question and the document simultaneously [3]. In addition to the conventional attention of question in light of document words and the attention of the document in light of the question, the authors also calculated the summaries of attention in light of each word of the document. They also applied an iterative dynamic pointer decoder to predict on the answer span. Because the model alternates between predicting the start and the end point, recovery from local maxima is more likely.

- **Bi-directional Attention Flow** : The model also employed character-level embeddings with Convolutional Neural Networks [11], which are then combined with word embeddings via a Highway Network. Interestingly, in this model the attention flow is not summarized into single feature vectors, but is allowed to flow along with contextual embeddings through to subsequent modeling.

- **RaSoR** : To address lexical overlap between question and document, for each word in document they created both passage-aligned and passage-independent question representation [12]. For passage-aligned representation, fixed-length representation of the question is created based on soft-alignments with the single passage word computed via neural attention. For passage-independent representation, the question word is compared to the universally learned embeddings instead of any particular word.

- **ReasoNet** : The ReasoNet specifically mimics the inference process of a human reader by reading a document repeatedly, with attention on different parts each time until a satisfied answer is found [13]. The authors present an elegant approach for ReasoNet to dynamically determine whether to continue the comprehension process after digesting intermediate results, or to terminate reading when it concludes that existing information is adequate to produce an answer. Since termination state is a discrete variable and non-trainable by canonical back-propogation, they use Reinforcement Learning to solve that.

- **Dynamic Chunk Reader** : After obtaining attention vectors between each word in the question and the document, the dynamic chuck reader creates chunk representation that encodes the contextual information of candidate chunks selected from the document [14]. Then cosine similarities is evaluated between the chuck repreeation and the question, and the highest-scoring chuck is taken as the answer.

- **Fine-grained Gating**: While other methods typically use word-level representation, or character-level representation, or both [15], in fine-grained gating a gate is applied to dynamically choose between the word or character-level representations for each document word. Word features such as named entity tags, part-of-speech tags, binned document frequency vectors, and the word-level representations all form the feature vector of the gate.

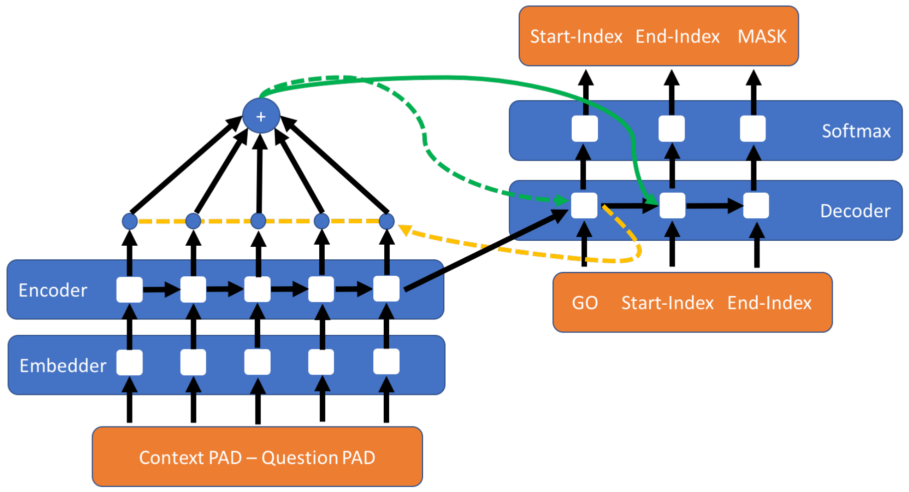


Figure 3: Sequence-to-sequence with Attention Baseline.

3 Sequence-to-sequence with Attention Baseline

Our baseline sequence-to-sequence model with attention mechanism, was implemented in Tensorflow, using the built-in seq2seq library, and is illustrated in figure 3. The input data was encoded as follows: (1) first the question and context sequences of token IDs were padded and/or trimmed to their respective max-length of $Clen = 300$ and $Qlen = 25$, and (2) then they were concatenated, context first, and question second, separated by an arbitrary separator symbol (thus the input sequence to the encoder always had a fixed length of 300+1+25=326). Though we tried bucketing, the majority of the runs involved a single bucket. Bucketing is a method to efficiently handle sentences of different lengths that is compromise between constructing a graph for every pair of lengths and padding to a single length. The seq2seq library internally accepted the input sequence as well as an embedding matrix, with which it mapped the input sequence to the corresponding sequence of dense vector embeddings (various embedding vector lengths were tried: 50, 100, 200, 300). Other parameters on the encoder side included: (1) whether or not to continue to adapt the embeddings or to treat them as fixed, (2) the dimension of the RNN cell used (100, 200, 300), (3) the type of RNN cell used (GRU, LSTM), and (4) whether the encoded RNN was stacked (1, 2, and 3 layers). Similar settings where also used with the decoder.

On the decoder side, the aim was to predict the start and end indices of the answer in the context span. A small variation was to predict the start of the span, and the length of the span, since the length of the span had a smaller variance than the end index, thus seemingly easier to predict. The decoder input was also prefixed with an arbitrary GO symbol, to trigger the decoder when predicting. During training, if the even if the decoder output the wrong prediction at time t, the cell at time $t+1$ was still provided with the correct index. At test time, however, the output of the previous time-step was fed into the decoder at the next time-step. Finally, the embeddings used were the GloVe [16] word vectors.

In the end, all of the variations of this model, ended up overfitting the training data, and not performing well on the test data. The plot in figure 4 is representative of the performance of the training loss in just about every variation of parameters of the sequence-to-sequence model, where there is an initial steep decrease in the loss, and subsequent gradual decrease that does not appear to slow down. The decrease in loss is matched with a corresponding decrease in F1 and EM metrics in the training data, as seen in figure 5, however, the metrics on the test data remain below 3% for F1 and EM, whereas perplexity did not go below 4.7. Though further experimentation is needed to properly regularize this model, these results prompted pursuing other models which had been proven to perform well on this task.

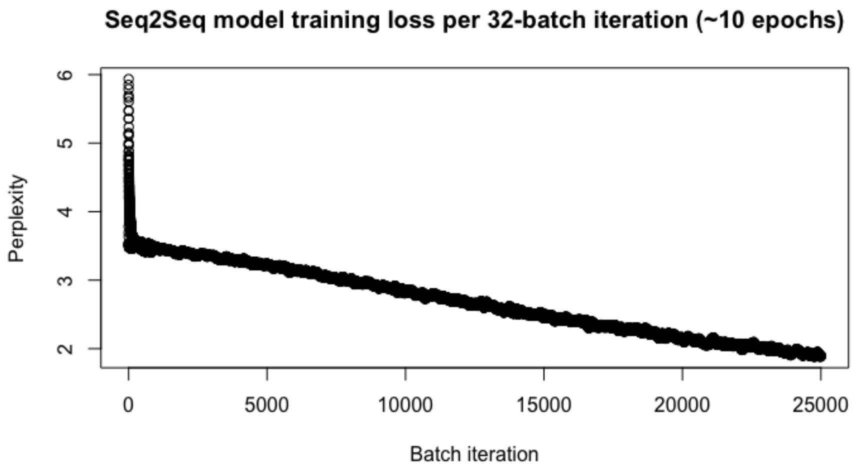


Figure 4: Perplexity went down consistently over the course of model training, but the same was not observed on the test set, where it remained above 4.7.

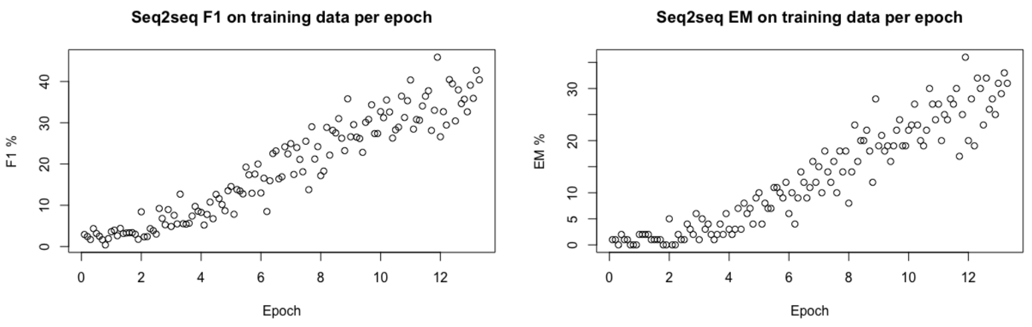


Figure 5: Similar to what was observed with perplexity, F1 and EM also decreased on the training set, but the same was not observed with the test set, where both remained below 3%.

4 Match-LSTM with Answer Pointer Decoder

In Match LSTM model, the authors present a layered network architecture consisting of three layers:

- An LSTM preprocessing layer that preprocesses the passage and the question using LSTMs. This layer uses standard one dimension LSTM to process the embedded passage and the question and collect the entire sequence of hidden states to generate Document and Question representations, H^p and H^q, respectively.

$$H^p = LS\vec{T}M(P), H^q = LS\vec{T}M(Q)$$

  Note that we shared a single LSTM unit for generating both the representation(same as in 5.1) unlike using two different ones as mentioned in Match-LSTM paper.

- A match-LSTM layer that tries to match the passage against the question. This is the core of the network where attetion vector α_i is computed by globally attending to the hidden representation of each word of the passage for the entire question.

$$\vec{G_i} = tanh(W^q H^q + (W^p h_i^p + b^p) \otimes e^Q)$$

$$\vec{\alpha}_i = softmax(w^T \vec{G}_i + b \otimes e^Q)$$

where W^p, $W^q, W^r \in R^{lXl}$ $b^p, w \in R^l$ and $b \in R$. h_i is the hidden state for the one-directional match-LSTM which recives the the concatenation of passage representation and attention weighted qustion representation. A similar LSTM is used for representation in reverse direction. Omitting the detail as they exactly match the Match-LSTM paper. Finally, the hidden states from both the LSTM are concatenated which is passed to the next layer for inference.

- (3) An Answer Pointer (Ans-Ptr) layer - the top layer is motivated by the Pointer Net introduced by Vinyals et al. (2015)[17]. The authors mentions two variants for this layer: predicting the entire answer Sequence vs predicting only the answer span. The initial testing showed us that boundry model was better than the sequence model, which was also inline with the results mentioned in the paper, so we conntinued with the boundary model which was also simpler to implement.

4.1 Experiment Detail and Evaluation

- **Embedding Dataset**: Our initial runs were with vocabulary data generated using glove 6B 100D dataset but towards the ends we switched to glove 840B 300D dataset which resulted in superior performance. Unknown words' embeddings were randomnly initialized.
- **Tokenizer**: Similar to embedding dataset, we switched from NLTK to superior Stanford Core-NLP tokenizer.
- **Document Length**: We filtered out data with passage length > 300, quesiotn lenght > 25 and answer length > 15 to keep the training set small for faster iterations.
- **learning rate** = 0.001 (tested with diffrent LR and settled to this)
- global norm **gradient clipping** threshold = 5
- **Adam optmizer** with $\beta_1 = 0.9$, $\beta_2 = 0.999$
- state size = 200

After many iterations, we acheived a significant score of **(EM=39%, F1=53%)** (complete score in 5.5) and settled for that as a baseline score. Although there was huge scope for improvement, but we wanted to explore other advanced models.

5 Dynamic Coattention

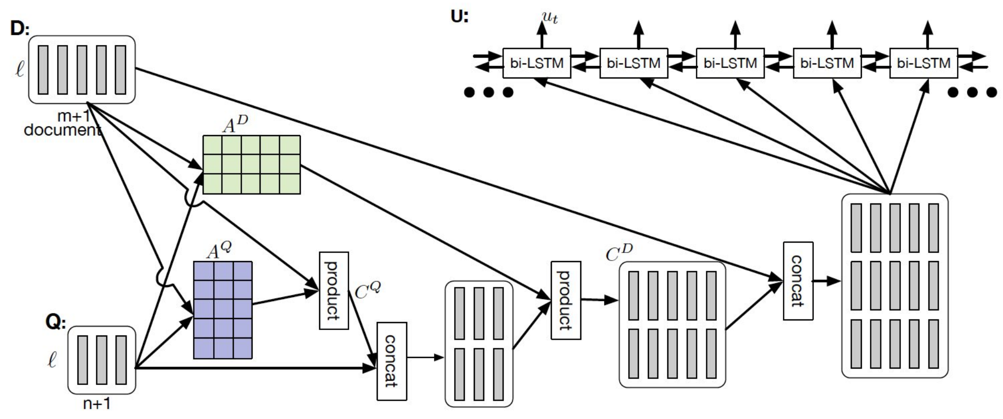


Figure 6: The Coattetion Encoder as descibed in original paper and implemented here.

After setting up a baseline with the Match LSTM model, we started to implement Dynamic Coattention Model [3] which had an advanced attention mechanism and a novel iterative approach for finding the answer span. Similar to Match-LSTM, this model also consisted of three layers describe below

5.1 Dcoument and Question Encode Layer

This layer is similar to the preprocessing layer of Match-LSTM model but with two modifications: the question and context tokens are fed to the same LSTM unit to obtain their encoding. And, there is an additional linear layer on top op the question encoding which results in general scoring function[9] instead of simple dot product between question and document encoding for attention weights calculation. A sentinel vector d_ϕ is added to not attend to any word in the document or question. Note: we did this only for the dynamic pointer decoder and skipped this for feed-forward and answer-pointer.

5.2 Coattention Encoder Layer

The coattention Encode layers attends to the question and document simultaneously by first calculating a affinity matrix, $L \in R^{mn}$ which is the dot product of the question encoding and the document encoding. This is followed by taking row-wise and column-wise softmax normalisation to get the question and document attention matrices. These are further processed which is neatly described in fig:6 before being fed to a BiLSTM unit to generate coattention encoding.

5.3 Decoder Layer

We started our training with a simple feed forward network and after initial success we used the Answer pointer layer as described in Match-LSTM model above which performed better. However, the original paper presents a dynamic pointer decoder which tries to iteratively find better solution and take the network out of local minima. The implementation is based on Highway Maxout Network, which we were able to implement and partially test at the time of writing this report. We expect a significant lift in the scores with the iterative decoder approach. And the next natural extension of this would be to implement a network deciding when to terminate out of iteration instead of iterating for a fixed number of times, much like a human reader. This network is described in ReasoNet model[13] and has demonstrated better outcomes.

5.4 Experiments

- **Embedding Dataset**:glove 840B 300D
- **Tokenizer**: Stanford Core-NLP.
- **Document Length**: $passage\_length\_max = 500, question\_length\_max = 35, answer\_length\_max = 20$.
- **learning rate** = 0.001
- global norm **gradient clipping** threshold = 5
- **Adam optimizer** with $\beta_1 = 0.9$, $\beta_2 = 0.999$
- **Dropout** = [0.2, 0.3, 0.5] - We picked 0.2 as the dropout based on Fig 10 and 11.
- **Constant Embeddings**. We tested with both and trainable and constant embeddings and found the latter to give better validation score (1-2% lift)
- state size = 200

5.4.1 Unknown Word Lookup

A major bottleneck of the model in case of test/dev prediction was the presence of many new words in the test/dev dataset which were missing in the vocabulary constituted purely of training dataset. Since our network performed better with constant embeddings, we had the flexibility of enhancing the vocabulary before test/dev prediction. And upon including the Unknown Word Lookup module, we got a lift of 6-9% in each F1 and EM score which was very crucial for the success of our model. We used an efficient hash join technique for the lookup and on test servers, it took just 60-100sec for looking up the missing word in the 2.19M Glove 840B 100D dataset which gave us more iterations to test.

5.5 Results

System	Val EM/F1	Dev EM/F1	Test EM/F1
Single Model			
Seq2seq Attetion Baseline	3.0/5.0	–	–
Match LSTM Baseline	38.8/ 53.8	42.1/55.4	–
Coattention + feedforward decoder	48.6/64.0	46.4/60.8	46.1/60.1
Coattention + ans-ptr	49.7/65.2	–	–
Coattention + ans-ptr + summarization	31.2/39.9	–	–
Coattention + ans-ptr + dropout +word lookup	56.0/69.8	61.6/72.3	**61.9/72.8**

Table 1: Exact match and F1 Scores for the Seq2seq baseline, Match-LSTM baseline and different variants of Dynamic Coattention model

Table 1 summarizes the F1 and EM scores for the two baseline models and different variants of the coattention model. The best score is for a single predictor and we are running multiple training iterations to create and ensemble of predictor to further boost the scores.

6 Summary-Attentive Reader

6.0.1 Summarization Engine

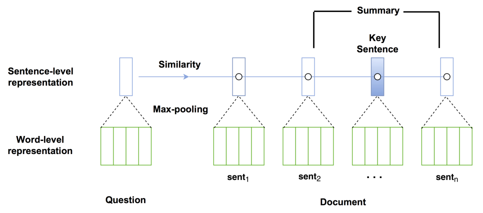


Figure 7: The summarization engine first selects the key sentence in the document, which has the highest similarity to the question. Then the document is truncated around the key sentence to obtain the summary.

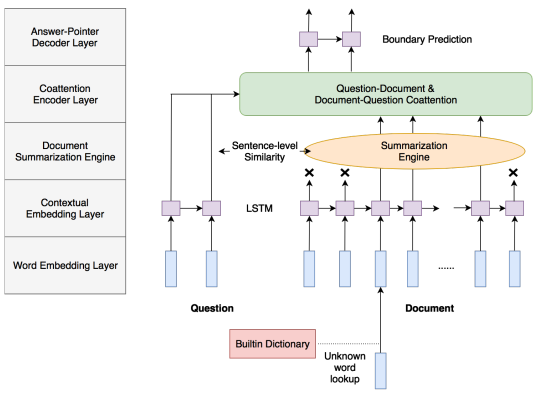

Answer-Pointer Decoder Layer
Coattention Encoder Layer
Document Summarization Engine
Contextual Embedding Layer
Word Embedding Layer
Boundary Prediction
Question-Document & Document-Question Coattention
Sentence-level Similarity
Summarization Engine
LSTM
......
Question
Document
Builtin Dictionary
Unknown word lookup

Figure 8: Overall Network Architecture

To facilitate better answer prediction, we created a summarization engine that can preprocess the document to generate a shortened summary that is more likely to contain the answer. This is motivated by the human readers' approach to question answering: when we are looking for an answer to some question from a paragraph, we would rarely read the article word-by-word in its entirety before selecting an answer. Instead we would often rapidly scan the document passage, identify sentences containing similar wording to the question, then select the answer based on our understanding of the text.

In an effort to emulate the aforementioned human question-answering process, we designed the summarization engine to truncate the document into a shorter paragraph before further processing. This is carried out through the following steps (Fig. 7):

1. A sentence-tokenizer is applied to the document to identify span indices of the sentences.
2. Max-pooling or mean-pooling is applied across all word embeddings of each sentence in the document, in order to obtain sentence-level representations of all sentences.
3. Similar pooling techniques is applied to the question.
4. The sentence-level representation of each sentence in document is compared with the question, in order to identify the sentence that has the highest cosine similarity as the key sentence.

5. The document is truncated around the key sentence to the pre-specified summary length.

After being shortened, the summarized document is allowed to flow through to the subsequent layers. Our summarized method ensures that similar attention mechanisms and prediction methods can be applied regardless of whether the document is full-length or has being shortened, since the word ordering and sentence meaning are preserved in this summarization engine. Fig: 8 shows our new summarization layer in the overall network architecture.

6.1 Experiment and Evaluation

To evaluate the summarization engine, we truncate each document in the train set to different word length. The success of the engine is measured by the ground truth answer retain rate, which is the percentage of ground truth that are wholly contained in the summary we selected. This can serve as a metric of how well the engine is doing as we can expect the same summarization, when applied to document with unknown answer from the test set, will also retain the actual answer for subsequent layer prediction.

Even though the maximum document size of our train set is 600 words, the summarization engine is able to retain more than 98.31% of the ground truth answer when we truncate the document to 200 words or more (Fig. 9). Even when we truncate the document down to 150 words, 92.17% of answers are contained in the summary. We also see that max-pooling performs marginally well than mean-pooling in the averaging across word embedding when obtaining sentence-level representation.

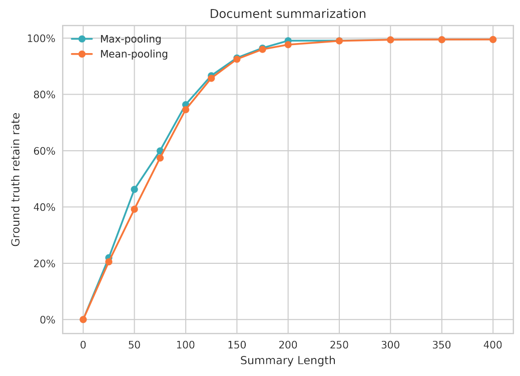


Figure 9: Ground truth answer retain rate with respect to the truncated document summary length. The original document has a maximum length of 600, and the summarization engine is able to retain more than 92% of ground truth answers in the summary when we truncate the document to length 150.

Figure 10: F1 Score vs Training epochs

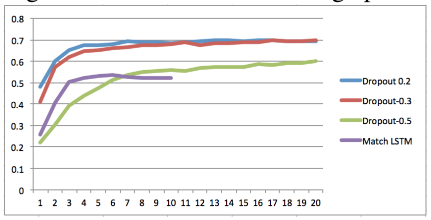
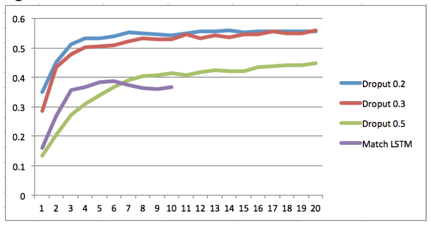


Figure 11: EM Score vs Training epochs



7 Discussion & Future Work

Even though the summarization engine did not effectively raise prediction accuracy in the current iteration in our model, we believe that the underlying idea is of great potential. As the current machine-based question-answering models do not perform well enough to challenge human readers, our attempt to draw inspiration from the human reading process is well-founded. Interestingly, other models performing the Q&A task has developed similar summarization features to ours [5], though their selection scheme is more sophisticated and thus presumably of higher quality. We plan to continue optimizing the summarization engine by (1) no longer limiting the summary to consists of consecutive sentences (2) develop better boundaries of selection in addition the natural sentence spans, and finally (3) incorporate semantic understanding of the text based on matching information flow to the question to better faciliate summarizing.

8 Conclusion

In this paper we analysed multiple state-of-the-art question-answering models capable of machine comprehension tasks at accuracy levels near that of humans. We also presented or implementation of Sequence-to-sequence learning, Match-LSTM and Dynamic Coattention model. Our best model with single predictor achieved a highly competitive score and we mentioned several enhancements, and we have high hopes of further improvement beyond the current competent implementation in the near future. Finally we presented our novel solution of summary attentive reader.

Contributions

Kshitiz was in charge of overall network architecture, monitoring model performance and built the baseline match-LSTM model. Long-Huei designed and created the summarization engine, experimented with its applicability and co-written the coattention encoder and answer pointer decoder with Kshitiz. Mario analyzed the sequence-to-sequence model, applied the attention mechanism to our specific task and thus allowing greater insight into model tuning and extension. The authors contributed equally to this work.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[2] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.

[3] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

[4] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, 2014.

[5] Wenpeng Yin, Sebastian Ebert, and Hinrich Schütze. Attention-based convolutional neural network for machine comprehension. *arXiv preprint arXiv:1602.04341*, 2016.

[6] Matthew Richardson, Christopher JC Burges, and Erin Renshaw. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*, volume 3, page 4, 2013.

[7] Arvind Neelakantan, Quoc V Le, Martin Abadi, Andrew McCallum, and Dario Amodei. Learning a natural language interface with neural programmer. *arXiv preprint arXiv:1611.08945*, 2016.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[9] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[10] Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211*, 2016.

[11] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[12] Kenton Lee, Tom Kwiatkowski, Ankur Parikh, and Dipanjan Das. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436*, 2016.

[13] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. *arXiv preprint arXiv:1609.05284*, 2016.

[14] Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end answer chunk extraction and ranking for reading comprehension. *arXiv preprint arXiv:1610.09996*, 2016.

[15] Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W Cohen, and Ruslan Salakhutdinov. Words or characters? fine-grained gating for reading comprehension. *arXiv preprint arXiv:1611.01724*, 2016.

[16] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[17] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015.