MeltingpotQA: Multi-hop Question Answering

Nathan Dass
Department of Computer Science
Stanford University
`ndass@cs.stanford.edu`

Saelig Khattar
Department of Computer Science
Stanford University
`saelig@cs.stanford.edu`

Ankit Mathur
Department of Computer Science
Stanford University
`ankit96@stanford.edu`

Abstract

MeltingpotQA is a question answering model that works on the HotpotQA dataset. HotpotQA is a question answering dataset featuring natural, multi-hop questions, with strong supervision for supporting facts to enable more explainable question answering systems [11]. We advance the baseline model [1] to more appropriately tackle the task of multi-document question answering. In particular, we focus on the problem of relevant paragraph selection, where the model has to decide which documents to use for answering questions. We experiment with using document embeddings and propose architectural changes to the model to take advantage of the supervision signal provided in the HotpotQA dataset. Overall, our final model achieves an F1 of 62.14 as compared to the baseline F1 of 54.93. This score should go even higher once we use our approaches of paragraph selection with the cleaned data.

1 Introduction

One fundamental goal of natural language processing systems is to be able to function as humans do in understanding information in natural language and using it for a given task. One such fundamental task is that of reading comprehension. Reading comprehension involves reading a variety of documents and being able to answer a question based on these readings. These questions might require several pieces of information for context to be synthesized to generate a version of the question that can be directly answered given the facts. Alternatively, multiple facts might combine to fully answer the question.

Unfortunately, many current question answering tasks do not approximate this and instead focus on the simpler case where a single sentence answers the question. HotpotQA [11] aims to provide a dataset on which researchers can innovate on models which aim to solve this task. The baseline model provided with this is the model introduced in [1].

In this work, we introduce MeltingpotQA, an improvement to the baseline model, in which we introduce some new techniques aimed at improving the performance on the HotpotQA task. We investigate more effective preprocessing techniques for the data which result in noticeable performance improvements on the dataset. We also evaluate performance while using document embeddings, both out of the box with doc2vec [5] embeddings and by training custom document embedding, using supervision from the data provided in the HotpotQA dataset. We investigated a variety of architectures for this supervised document embedding fine tuning as well.

2 Approach

We start with the baseline model (and code) described in the HotpotQA paper [11], which is built on the architecture described in [1]. It has a few additions, including character-level models, self-attention, bi-attention, and a 3-way classifier for answering yes/no questions.

2.1 Document Embeddings

When looking through the modeling architecture, we noticed that the initial representations of the question and the context are built from word and character embeddings. While this is probably fine for the question, this seemed like an issue for the context. In the distractor setting, the context is 10 paragraphs and the model needs to extract the relevant information for answering the question from all of these 10 paragraphs, which seems difficult to gather from a long sequence of word embeddings and an even longer sequence of character embeddings. The problem gets much worse when we extend to the full wiki setting when the model is presented with the first paragraph of all Wikipedia articles. In the full wiki setting, not only would it be hard to extract the necessary signal, but we would very likely run into out of memory issues when trying to allocate enough space for all of the character embeddings. This also seems impractical — to answer a single question, the model would have to comb through the first paragraph of all Wikipedia articles.

For these reasons, we chose to focus on the problem of relevant paragraph selection. Due to time constraints, we focus on the distractor setting in this project, but our work can be extended to the full wiki setting (and we expect to see larger gains using our approaches in the full wiki setting). As a first step, we propose adding paragraph-level embeddings to the model with the hope that it would provide a stronger, higher level representation of the question and each of the paragraphs in the context. More concretely, we first pretrain a gensim doc2vec model on each question and paragraph in the context in the training data [5]. We wrote a Pytorch wrapper around this module and a script to precompute these embeddings to speed up performance. Then, when training the full model, we look up the paragraph embedding for the question and the paragraphs in the context and concatenate these embeddings to the word and character embeddings for the question and the context. The rest of the architecture remains the same, but we are just adding in an extra source of information in the beginning of the model. Since the rest of the model builds off of the initial representations passed in, we expect that the addition of paragraph-level embeddings will increase performance. We experiment with 50, 150, and 300 dimensional document embeddings.

2.2 Supervised Document Embeddings

While individual paragraph and question document embeddings can help capture higher level signal, they do not directly capture the relevance between the paragraphs and questions themselves. To capture this relevance, it would be helpful to have some sort of scoring function that could provide a measure of paragraph-question relevance or some way to tune the paragraph embedding to include information about question relevance. Further, we notice that the HotpotQA dataset has supervision signal that the baseline model does not take advantage of. Specifically, for each question, the dataset has the paragraph titles that correspond to the supporting facts of each question. Since we are given that for each question, only two paragraphs out of the ten are relevant, we can extract the exact paragraphs that are relevant to each question. Then, we can provide a binary label to each question-paragraph pair, i.e pairs that have relevant paragraphs are assigned to the positive class and pairs with irrelevant paragraphs are assigned to the negative class. Now, we frame this as standard supervised binary classification problem, where we take in a question and paragraph document embedding as input and predict whether they are relevant or not.

To do this, we develop a neural network that provides this scoring function and learns a modified paragraph embedding in the process. We propose a network with 4 hidden dense layers that takes in a concatenated paragraph and question embedding and predicts a binary label. The last hidden layer outputs a vector with the same dimension as the question and paragraph document embeddings — this is our new "supervised document embedding" that will feed into the full model. The output layer of our network takes this supervised document embedding and maps it to a two dimensional vector containing the scores for each class (we compute scores for both classes since it is easier to use incorporate class weights in PyTorch loss functions). We then apply a softmax layer to obtain the probabilities of each class. Our final architecture for this supervision task is shown in 1. We then

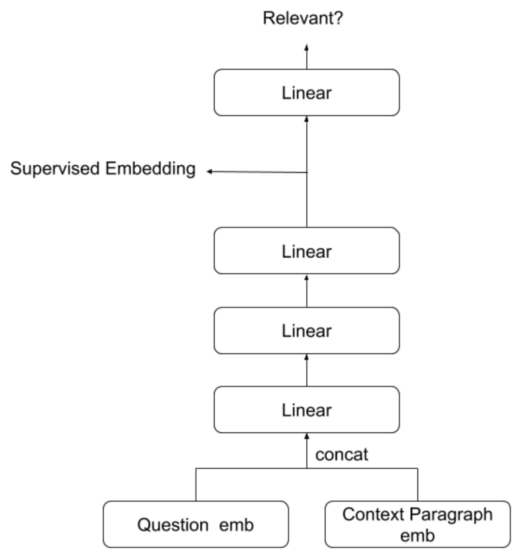

Figure 1: Architecture to learn supervised document embeddings for context paragraphs.

take the "supervised" embedding from this model and feed into the full model, whose architecture is depicted in Figure 2.

3 Related Work

3.1 Question Answering Datasets

There are a wide variety of datasets that target the task of answering questions. All of them define different tasks, so in this section we review the varying tasks and what makes the HotpotQA task special.

SQuAD [6] is the most popular and most cited of the question answering tasks that we were able to identify. SQuAD involves answering questions with a single paragraph of context. The answer to the question is located directly in a given sentence of the paragraph. Unfortunately, solutions designed for this dataset often fail to generalize to different settings where more complex questions and more varied contexts can be synthesized to answer questions.

TriviaQA [4] and SearchQA [3] are datasets that identify more documents, given a question answer pair. This makes the problem more challenging, but it does not address a the fundamental limitation of locality - the answer is usually attainable by looking over a relatively small span in the data. Therefore, models targeting these datasets do not require or learn complex reasoning over multiple paragraphs.

Therefore, the dataset that addresses the more complex tasks should involve multi-hop reasoning over large amounts of context. This concept is not new, as QAngaroo [9] and ComplexWebQuestions [7] have datasets with these. However, the target application of these datasets was knowledge graph systems. As a result, they are not really about the natural language setting. HotpotQA [11] is a natural language dataset that contains question-answer pairs, where the answer is derived from a relatively large context and the questions require being able to synthesize context from multiple paragraphs. As such, we felt this is a task more closely approaching how we would expect a reasonable human to answer questions given a certain amount of context. It more closely simulates the concept of reading comprehension.

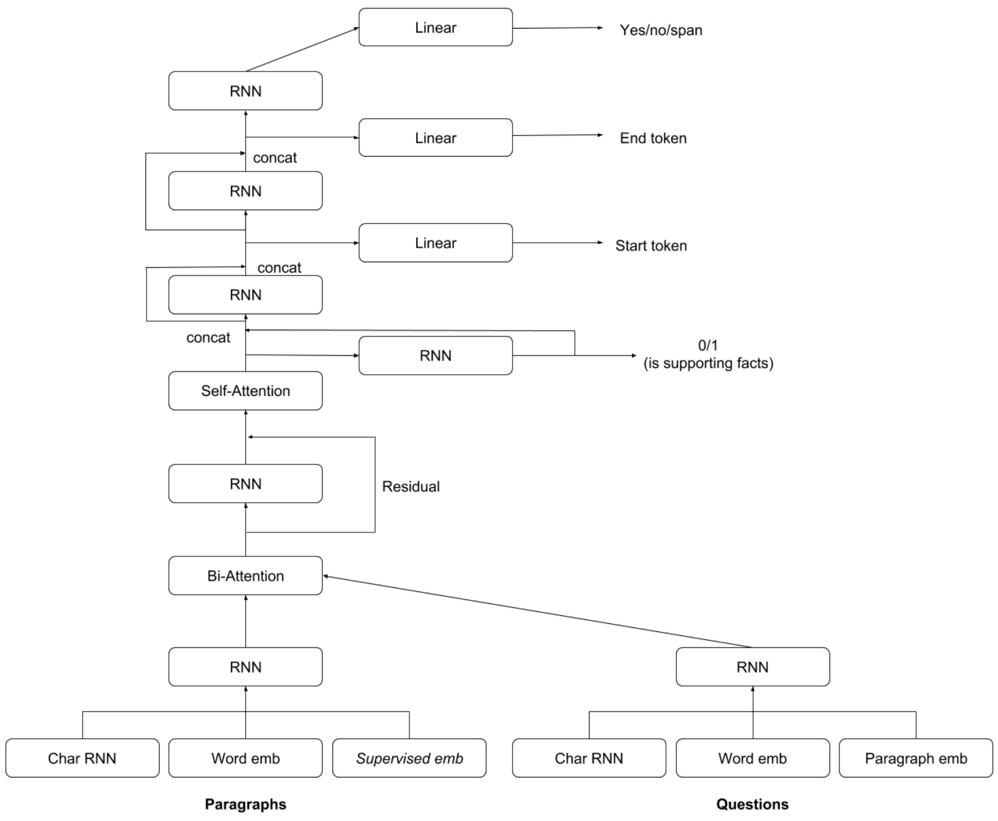

Figure 2: MeltingpotQA Model Architecture, built upon [11]. Takes in supervised document embeddings from 1.

3.2 Question Answering Models

The current state of the art model for the SQuAD1.1 dataset is the Transformer-based BERT [2] model released by Google during the course of this class. This model is actually a language model, but it can be fine-tuned to the task of question answering by training a start vector and an end vector, which are then used to compute the likelihood of which words correspond to the start and end tokens for the response. While this might work for relatively fixed size contexts where the answer is a single sentence, it is unlikely to be able to synthesize information across contexts or utilize information from other contexts to find relevant spans. As such, BERT does not work out of the box for the HototQA dataset.

Other state of the art models on other tasks like WikiQA [10] include models like those proposed by [8] Tay et al, who propose HyperQA, which uses a pairwise ranking objective which models the relationship between question and answer embeddings in hypterbolic space instead of Euclidean space. This paper focuses on the performance over huge datasats, but it does not have any specific mechanisms that might help generalize to a multi-hop question answering use case.

The state of the art model on the TriviaQA [4] dataset is more likely to be relevant to the HotpotQA dataset. That model [1] uses the S-norm to target the task of reading comprehension over large single document contexts. As such, the authors of the HotpotQA paper implement that model with some slight alterations in order to apply to the multi-document context.

4 Experiments

4.1 Data and evaluation methods

The dataset we are using is HotpotQA [11] for multi-hop questioning and answering. This dataset contains a diverse set of 113k Wikipedia-based question-answer pairs that require reasoning over multiple documents. We evaluate our performance using joint exact-match (EM) and F1 over supporting facts and questions. Joint exact-match is 1 only if both answers and supporting facts get an exact match, and 0 otherwise.

4.2 Experiment Details

4.2.1 Document Embedding Experiments

Our first approach involved incorporating document embeddings into the baseline architecture [11]. We separately trained 50, 150, and 300 dimensional document embeddings using a doc2vec model [5] over all the question and context paragraphs in the training set, and then passed these embeddings into the full model. In our first experiment, for each question, we average all of the paragraph embeddings together and repeat it to match the shape of the word and character embeddings. We use the published learning rate of 0.1 and decrease the learning rate by a factor of 2 whenever the dev F1 decreases between consecutive evaluations.

However, by averaging all of the paragraph embeddings together we likely lose the signal gained by the paragraph embeddings themselves. Thus, to correct this problem, we attach the paragraph embedding for the paragraph the word belongs to in our next experiment.

Both of these experiments took about 12 hours to train.

4.2.2 Supervised Document Embedding Experiments

For our supervised document embeddings model, we tested various different architectures for the 150 dimensional context paragraph and question embeddings. We vary the number of hidden dense layers as well as the number of units in each of these layers. We optimize our model over cross entropy loss. Our best model was a series of 4 hidden layers, with 500, 300, 200, and 150 units respectively. Note the last hidden layer must have the same number of units as the dimension of the context paragraph and question embeddings. We used a weighted cross entropy loss, assigning a weight of 10 to the positive class (relevant) and a weight of 1 to the negative class (irrelevant). We used a learning rate of 0.01.

Training the supervised model was fairly quick, usually taking around 20 to 30 minutes.

We then used the supervised document embeddings learned from this model, and fed them into the full architecture.

4.2.3 Other Experiments

We also conducted hyperparameter tuning on patience, and changed the way we preprocessed data.

To speed up the hyperparameter tuning process, we first downsampled the training and validation set to 10% of all the data. We then tested various different values for patience, i.e the number of epochs that don't result in F1 score improvement that we should wait for before halving the learning rate, on this downsampled data. Since we downsampled the validation set, we couldn't compare performance directly with full model performance, but we could still compare among the experiments themselves. We determined that the best performance resulted from a patience of 7, so we used this parameter on training the full model and compared performance with the baseline.

In addition, we noticed some oddities in the preprocessing script that came with the baseline model (see section 5). After changing these, we ran the baseline model on the updated preprocessed data (with patience 7).

4.3 Results

Table 1: Results on document embeddings and baseline improvements

	F1	**EM**
Baseline	54.93	40.90
Baseline, patience=7	56.38	42.26
Baseline, clean data, patience=7	**62.14**	**47.37**
Doc2Vec50	55.03	41.08
Doc2Vec150	55.44	41.15
Doc2Vec300	55.16	41.36

Table 2: Results on supervised document embeddings*

	F1	**EM**
Baseline, step 2400	40.04	28.71
Supervised document embeddings, step 2400	39.25	28.74

*Since with supervised document embeddings our model takes 15-20 hours to train, we were unable to get the final results in time for the paper. We will have these numbers for the poster, however.

5 Analysis

5.1 Preprocessing

We were initially very surprised by the results that we obtained from adding our document embeddings, both with the off the shelf unsupervised doc2vec model approach and with our supervised document embedding fine-tuning approach. Since the document embeddings should be providing a broader signal of the paragraph as a whole and the fine-tuned embeddings should be providing an additional signal of how relevant a paragraph is to a question, we expected the improvement in performance to be higher.

To analyze our results, we started by looking through the data. We immediately noticed something very odd. The preprocessed data had a lot of non-random out of vocab tokens throughout the context feature. Below is part of an example of a tokenized context paragraph:

```
<t> Greisen - -OOV- - Kuzmin limit -OOV- The Greisen - -OOV- - Kuzmin limit
( GZK limit ) is a theoretical upper limit on the energy of cosmic rays (
high energy charged particles from space ) coming from " distant " sources .
-OOV- The limit is , or about 8 -OOV- joules .  -OOV- The limit is set by
slowing - interactions of cosmic ray protons with the microwave background
radiation over long distances ( -OOV- million light - years ) .
```

We noticed that there is always an out of vocab token after the title ends and also at the start of every sentence in the actual context paragraph, which seemed like a big issue. Since the additional tokens were actual out of vocab tokens, the model would not know the difference between an actual out of vocab token and what seemed to be a more structured out of vocab token. Digging through the preprocessing code, we noticed that the title was surrounded with `<t>` and `</t>` tokens, but `</t>` was not a token in the vocabulary, so it was mapped to out of vocab. Additionally, most (but not all) of the sentences in the context paragraph had an extra token that was not a word at the start of the sentence and was also mapped to out of vocab. This turns into an issue because these extra out of vocab tokens are not meant to be out of vocab tokens and can provide actual signal to the model, but then there are actual out of vocab tokens that do not provide much signal to the model. To fix this, we changed the preprocessing code to use tokens that were actually in the vocab and add sentence boundaries (a common preprocessing step in NLP models). Below is the same paragraph as above with our new preprocessing:

```
<title> Greisen - -OOV- - Kuzmin limit <p> The Greisen - -OOV- - Kuzmin
limit ( GZK limit ) is a theoretical upper limit on the energy of cosmic
rays ( high energy charged particles from space ) coming from " distant
" sources .  <p> The limit is , or about 8 -OOV- joules .  <p> The limit
is set by slowing - interactions of cosmic ray protons with the microwave
background radiation over long distances ( -OOV- million light - years ) .
```

Now, out of vocab tokens actually imply what they are meant to imply and we can learn what the `<title>` and `<p>` tokens actually mean (start of title and start of sentence, respectively). Not only would this make the baseline better, but we also realized that this probably had a big impact on the performance of our document embedding experiments. We noted that the extra out of vocab tokens were only manifsted in the preprocessed data, but not in the raw data. Since we trained our initial doc2vec models on the raw data, the doc2vec models never saw the additional out of vocab tokens. Then, we try to obtain an embedding for what we think is the same paragraph but now has a bunch of extra out of vocab tokens. Since the doc2vec model has not been exposed to the out of vocab tokens, the model will give a different paragraph embedding from what we are expecting to get. The extra out of vocab tokens are essentially adding extra noise to the paragraph that we confirmed makes the embeddings actually obtained different from we expect them to be. Now, we have a paragraph embedding that is misrepresenting the actual paragraph and could even be adding additional noise to the model.

5.1.1 Impact on modeling performance

We discovered this bug very late into the project, but to confirm our suspicions with the time that was remaining, we reran the baseline model with a patience of 7 and only changed whether or not the model was trained and evaluated on the original preprocessed data or the cleaned preprocessed data and saw that the difference in metrics was very big. The best dev F1 of the model trained on the original preprocessed data is 56.38 and the EM at the same checkpoint is 42.08. On the other hand, the best dev F1 of the model trained on the cleaned preprocessed data is 62.14 and the EM at the same checkpoint is 47.37. Clearly, cleaning the data that is used has a big impact on the model with minimal tuning, so we are confident that redoing our document embedding experiments from scratch with the cleaned data will give better results.

5.2 Supervised Document Embeddings

We considered a variety of network architectures for the supervised document embeddings. The input to our model was concatenated versions of the question and document vectors for a given question, document pair. Conceptually, we want to learn a transform over the vector space that maps a pair of vectors to a new location in the space which is closer to the original question. We want this because we want document embeddings that are relevant to the question to be nearer, so the rest of the network can use this information to weight the likelihood that the answer lies in that document.

We made some progress on this, though there remain many experiments worth trying. We ruled out convolutional network architectures because they are best for local relationships between input dimensions, but that doesn't really make sense in context of this task - our input is a concatenated vector, and local information encodes nothing about the relationship between the two vectors. We experimented with fully connected layers because they more closely resemble the concept we wanted our model to learn. Fully connected layers learn nonlinear transform functions over the input - we wanted to learn such a transform that transforms the document vector into the question embedding space.

We experimented with a single layer model with 500 units and a deeper, wider model with 4 layers. Unfortunately, the best F1 score we were able to obtain on this model was around 0.33. We made sure to balance the training set (there were around 80 % negative examples and 20 % positive examples). To us, this represented a relative lack of improvement over other methods. As we've highlighted in previous parts of this paper, we believe a large portion of this to be a result of the OOV tokens messing up the dimensionality - our model learned about documents in a totally different vector space, and the OOV tokens inserted themselves at various dimensions.

We have several other ideas about how to better train this model - these include autoregressive approaches or not freezing the parameters learned when plugging the supervised document embedding transformation into the full model, since it is possible that our model was simply learning to optimize for the binary score. We also think a loss function that computes the distance between the question vector and the new document embedding vector might be better. A different loss function like max margin might also be effective, where we are measuring whether the relevance score of the relevant documents given the question is higher than the irrelevant paragraphs. Potentially, all of these options could intuitively have made a big difference, and if we had more time, we would definitely have tried these.

6 Conclusion

In this work, we introduced the idea of trying to use supervised document embeddings to understand how these embeddings can help inform the model in choosing which documents to focus on. We started with using doc2vec to pretrain document embeddings for the questions and paragraphs, and, subsequently, we designed a fine-tuning neural network that was supervised with the paragraphs that was trained to predict the relevance of a paragraph to a given question. We also investigated how data was preprocessed in the baseline, and made changes (along with hyperparameter tuning) that resulted in a F1 score that exceeded that of the baseline by more than 7 points. While we only achieved modest improvements with document embeddings, we believe that they could be more impactful when using the cleaned data.

7 Additional Information

Mentors: Peng Qi (external) and Anand Dhoot

References

[1] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. *CoRR*, abs/1710.10723, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[3] Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Güney, Volkan Cirik, and Kyunghyun Cho. Searchqa: A new q&a dataset augmented with context from a search engine. *CoRR*, abs/1704.05179, 2017.

[4] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611. Association for Computational Linguistics, 2017.

[5] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents, 2014.

[6] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392. Association for Computational Linguistics, 2016.

[7] Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. *CoRR*, abs/1803.06643, 2018.

[8] Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. Enabling efficient question answer retrieval via hyperbolic neural networks. *CoRR*, abs/1707.07847, 2017.

[9] Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Constructing datasets for multi-hop reading comprehension across documents. *CoRR*, abs/1710.06481, 2017.

[10] Yi Yang, , and Chris Meek. Wikiqa: A challenge dataset for open-domain question answering. ACL - Association for Computational Linguistics, September 2015.

[11] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.