Question Answering with Deep Learning

Tristan McRae
Department of Aeronautics and Astronautics
Stanford University
Stanford CA, 94305
Codalab Username: trimcrae

Abstract

Recent progress in deep learning has allowed for significant strides in natural language processing. Here, a model for using deep learning to answer questions was proposed and implemented. The model can accept a text based question and a text based context paragraph in which the correct answer can be found. The model will generate a prediction for where in the context paragraph the answer to the question lies. The model involves representing questions and contexts as lists of word vectors, applying an attention mechanism to the context words and passing the results through a bidirectional LSTM. This method produced a model that can answer questions in a way that gives an F1 score of 33% and an exact match score of 19%.

1 Background

The ability for a machine to answer any question asked of it has numerous applications from artificial personal assistants, to medical diagnoses, fact checking and beyond. For this reason, it is worthy of much research. Past approaches have involved representing words as numerical vectors and allowing a model to learn temporal patterns in sentences and the approach laid out in this paper will follow a similar path. The model presented in this paper learns based on seeing questions answered correctly. Because of the way it learns, the model is not effective at answering questions that are very dissimilar to what it has seen before.

Models in this paper were trained and tested on the Stanford Question Answering Database (SQuAD) which contains thousands of questions, contexts and answers. Three sections of this dataset were taken for this project and split into datasets for training, validating and testing. All computations in this paper were executed on a Microsoft Azure standard NV6 GPU with 6 cores and 56 GB memory.

2 Metrics

Two primary metrics were used in evaluating models, F1 score and Exact Match (EM) score. Both metrics compare the similarity of the predicted answer to a question to the actual answer. F1 score rewards predicting words that occur in the actual answer while punishing false positives and false negatives. It is computed as follows:

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

Where

$$precision = \frac{\#\ of\ words\ common\ to\ prediction\ and\ actual\ answer}{\#\ of\ words\ in\ prediction}$$

$$recall = \frac{\#\ of\ words\ common\ to\ prediction\ and\ actual\ answer}{\#\ of\ words\ in\ actual\ answer}$$

EM score is a simple measure of what percentage of the time the predicted answer is exactly the same as the actual answer.

Both F1 and EM can be computed in two different ways. One way is to use the words predicted by the model and compare them to the words in the actual answer. The other is to compare the predicted and actual starting and ending locations of the answer in the context. The former is the standard used for evaluating EM and F1 scores provided by the team that created SQuAD, the latter was constructed for the purposes of this research to gain more insight into the results. Though they produce similar scores, the two models have subtle and instructive differences.

The core concept behind evaluating where in the context the model looks for the answer rather than what answer it produces comes from the way the model is trained. The model is not learning to produce an answer completely on its own, but rather, it is learning where to look in a context when a question is asked. For this reason, having this alternative formulation for F1 and EM can be useful in that it allows us to select models that are looking in the right places better than the SQuAD standard calculations.

The difference between calculating F1 and EM scores based on answers vs context locations is best illustrated with examples. The following is an example context-question-answer triplet from the SQuAD dataset used to train the model.

Context:
The map of earthquake intensity published by CEA after surveying 500,000 km2 of the affected area shows a maximum liedu of XI on the China Seismic Intensity Scale (CSIS), described as "very destructive" on the European Macroseismic Scale (EMS) from which CSIS drew reference. (USGS, using the Modified Mercalli intensity scale (CC), also placed maximum intensity at XI," very disastrous".) Two south-west-north-east stripes of liedu XI are centered around Yingxiu, Wenchuan (the town closest to the epicenter of the main quake) and Beichuan (the town repeatedly struck by strong aftershocks including one registering MS 6.1 on Aug 1, 2008), both in Sichuan Province, occupying a total of 2,419 km2.

Question: *What was the intensity scaled at?*
Correct Answer: *XI*
Model's Answer: *XI*

On the surface, it appears that the model correctly identified the answer to the question and, indeed, this is what SQuAD models for EM and F1 conclude. However, where the model found this answer shows that it is not that simple.

The correct context for finding the answer to this question is as follows:

*"The map of earthquake intensity published by CEA after surveying 500,000 km2 of the affected area shows a maximum liedu of **XI** on the China Seismic Intensity Scale (CSIS)"*

This is intuitive to a human reader and it is indeed the location SQuAD gave as correct. The model, on the other hand, looked for an answer in a different part of the context:

*"Two south-west-north-east stripes of liedu **XI** are centered around Yingxiu"*

This does not represent where a human would find the correct answer and does not represent the kind of behavior from the model that should be rewarded. In other words, we do not want to reward the model for just getting lucky. In addition, SQuAD F1 and EM calculations omit the common articles "the", "an" and "a" from its calculations. This prevents cases where one of these articles is the predicted answer and right next to the correct answer from being considered as any better than guessing nothing at all. For these and similar reasons, location based EM and F1 scores were considered in addition to the SQuAD versions when determining which model architectures to pursue.

The EM and F1 scores shown throughout the rest of the paper refer to scores on the validation dataset unless otherwise noted. All scores, with the exception of final test scores, are calculated with 100 fixed samples from the validation dataset.

3 Model Architectures

3.1 Architectures Overview

Several different architectures and features were tested in this research for their effect on the question answering model's effectiveness. These are described below.

3.1.1 Baseline

This is the model that all other architectures in this paper are based on. Where descriptions of other models in this paper lack detail, the models simply copy what was implemented in this model. This is the minimum functional model. In this model, both question and context are received in text format. They are then converted to word vectors of length 100 using pretrained GloVe embeddings. This model contains two different bi-directional long short term memory cells (bi-LSTMs), one for the question and the other for the answer. Both LSTMs have the same hidden size, here set to 75. The question and context are put through their respective bi-LSTMs to generate hidden vectors and outputs. These bi-LSTMs allow the model to look for patterns in the questions and contexts both forward and backward in time. The two hidden vectors of size [hidden size] for the final state of the question bi-LSTM (one in each direction) are concatenated to form vector *q* of size [2*hidden size]. The outputs of the context bi-LSTM for each word are concatenated as well to form matrix *X* of size [2*hidden size, context length], where context length is the number of words in the context paragraph. The inner product of *q* and *X* is taken to create a new vector of length [context length]. A softmax is applied to this vector and the resulting [context length] vector is called *p*. Vector *p* represents the probability that the answer to the question lies at each index in the vector. In this case, predicted answers are only ever one word. Cross entropy between vector *p* and the one-hot vector *a_s*, which represents the actual index of the start of the answer is one component of loss, L1. The other component of loss, L2, is determined by cross entropy between vector *p* and the one-hot vector *a_e*, associated with the index of the end of the answer. These loss components are combined to make total loss.

$$loss = L1 + L2 = CE(a_s, p) + CE(a_e, p)$$

Loss is minimized with the Adam optimizer with a default learning rate of 0.001. When it is producing answers, the index of *p* with the highest probability is taken as the index of the answer in the context paragraph.

3.1.2 Attention

The attention model adds a key feature to the baseline model. Before the context word vectors are passed into their bi-LSTM, they are weighted according to their relevance. First, both question vectors and context vectors are normalized. Then, for each context word, the inner product between the normalized context word vector *c* and each normalized question word vector, q_i, is found. The highest product for that context word becomes a scaling factor for the context word vector. This scaling factor is used to create an attention-weighted context word vector, c_attn.

$$c_{attn} = c * \max(c * q_i)$$

This modified c_attn is then passed into the context bi-LSTM and the rest of the model runs as usual. The introduction of this feature results in significant improvements in performance, as shown in Table 1, and was implemented as part of the final model. This is an important feature because it allows the model to focus on areas of the context that are more relevant to the question and not become distracted by superfluous information. This attention model was inspired by Wang et al [1].

3.1.3 Question Attention

This model is the same as the Attention model with the modification that now, the question word vectors are also modified before going into their bi-LSTM. Each word vector in the question is multiplied by a scaling factor proportionate to how similar the word is to the most similar context word. This method did not exhibit an improvement in performance and was not included in the final model. A likely reason for this is that the words similar between the context and the question were already sufficiently emphasized with just the context attention method.

3.1.4 Multiple Attention Passes

This methods attempts to capture the fact that it can be helpful to read over a context multiple times before deciding what the important parts really are. Here, the architecture is similar to the Attention model except the outputs of the context bi-LSTM are re-fed into the context bi-LSTM before moving on. This mechanism was found to hinder model performance. This is likely because, the way it was implemented, this model just dilutes the outputs of the context bi-LSTM rather than adding any new insights.

3.1.5 LSTM Decoder

This model starts out the same as the Attention model but the matrix *X* is multiplied elementwise by *q* and the product of these is fed into a single direction decoding LSTM. The hidden states of this decoding LSTM are separately multiplied by two different trainable weight matrices. The outputs of these products are then put through a softmax and called p_s and p_e respectively. Loss for this model is calculated as follows:

$$loss = L1 + L2\ CE(a_s, p_s) + CE(a_e, p_e)$$

This model then found its prediction for where in the context the answer ends by finding the index in p_e with the highest probability, provided it was a higher index than the answer's start index. This model did not improve performance, possibly because it was overcomplicating the process of finding the start location and was not able to find where the answer began anymore.

3.1.6 Separate Start and End Predictions

This model is similar to the LSTM Decoder model only without the LSTM. Predictions for p_s and p_e are made using trainable weights but this time the weights are applied to *X* rather than a new LSTM's outputs. This model did not show improvement and was not implemented. It is

possible this model just needed more time to train but a more effective way of handling sentences of variable length was found in the next model making further study of this model less pressing.

3.1.7 Length Prediction

This model is the Attention model with the addition of the ability to predict the length separately from the starting position. Here, the elementwise product of q and X is taken, multiplied by a trainable weight and added to a trainable bias. The result is passed through a rectified linear unit to force it to be non-negative and called the length of the answer (where length 0 is one word, length 1 is two words, etc.). This length is added to the predicted index for the answer's start to give a predicted index for the answer's end. This index is then turned into a one-hot vector p_e and used to compute the new L2 for loss.

$$L2 = CE(a_e, p_e)$$

This feature showed improved performance on the validation dataset and was incorporated into the final model. The improvement is likely because it removes the restriction that the predicted answers can only be one word long. The loss calculation in this case suffers the disadvantage that it is all or nothing in regards to whether a prediction is correct but empirically this mode still improves performance.

3.1.8 Dropout

This model is the same as the Attention model except it applies dropout to the matrix X. This model shows no improvement over the Attention model at one epoch but, as implementing dropout causes training to slow, requires additional evidence against it to exclude it from the final model.

3.2 Proof of concept testing

Table 1: Performance across architectures

Model	Time (min)	SQuAD F1	SQuAD EM	Location F1	Location EM
Baseline	**11**	21.7%	13.0%	22.1%	11.0%
Attention	**11**	35.0%	20.0%	34.6%	18.0%
Question Attention	12	32.9%	15.0%	29.4%	7.0%
LSTM Decoder	15	4.9%	1.0%	7.3%	1.0%
Separate Start and End Predictions	13	15.2%	10.0%	18.2%	10.0%
Multiple Attention Passes	22	35.5%	20.0%	35.1%	16.0%
Length Prediction	12	**37.7%**	**23.0%**	**37.9%**	**19.0%**
Dropout	12	33.1%	19.0%	32.1%	16.0%

To ensure they were viable, all model architectures were tested after training for 1 epoch. The results of this testing are shown in Table 1. Attention, and Length Prediction both showed an immediate improvement.

3.3 Further model testing

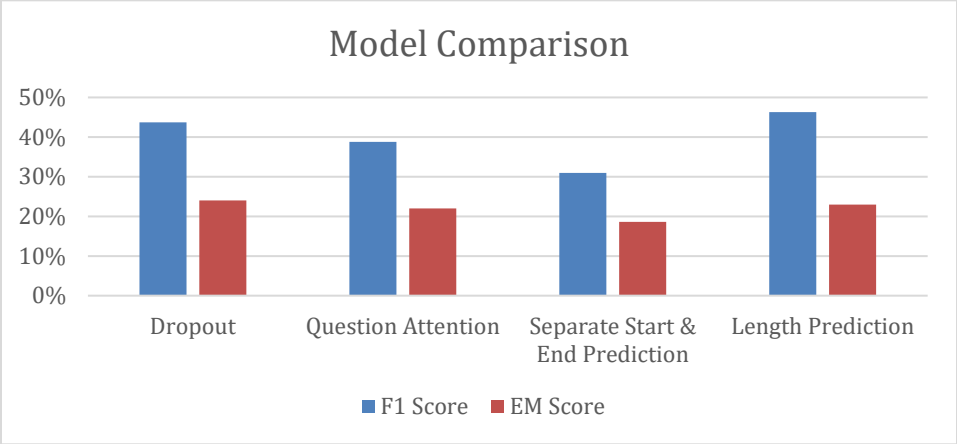

Figure 1: Model performance

Figure 1 shows the results of training the most promising model architectures for five epochs. Somewhat counterintuitive was that dropout does not give an improvement. Analysis of the training scores showed that training scores and validation scores were very similar, suggesting that no overfitting was occurring and that dropout was unnecessary. This analysis showed that the Length Prediction method yielded the best results, which is congruent with what was found when training for only one epoch. Because of this, the Length Prediction method was chosen to perform further tuning on.

4 Hyperparameter Tuning

Once the final architecture of the model had been solidified, hyperparameters were tuned to optimize the performance of the model. The two key hyperparameters that were tuned were learning rate and hidden size.

4.1 Learning Rate

Learning rate is a parameter used by the model's optimizer, in this case Adam, to influence how much the model parameters should change based on new information. Higher learning rates allow for bigger changes and can allow the model to converge faster but are vulnerable to overshooting and being too sensitive to a new input which can prevent the model from converging entirely. To find the right balance, various learning rates were tested for performance. During these tests, hidden size was kept constant at 75 and number of epochs was kept constant at 1.

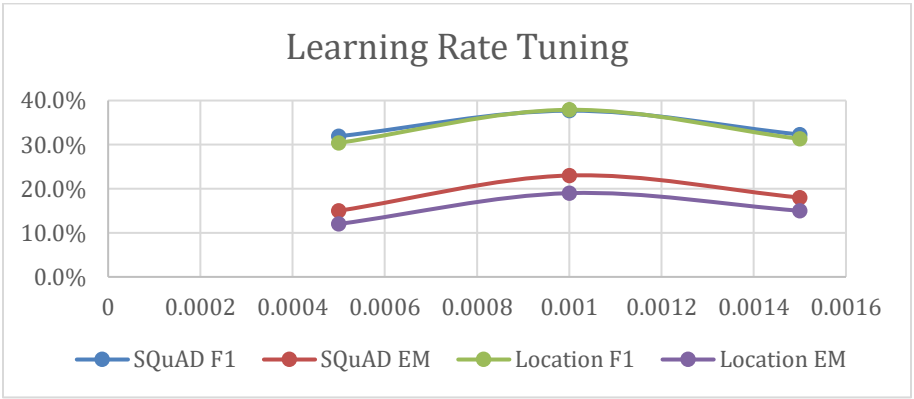

Figure 2: Learning rate tuning

All metrics used in this analysis agree that a learning rate of 0.001 is optimal for this model. This learning rate was used for all experiments and results going forward. Since the learning rate was optimized over a hidden size and number of epochs that was not used in the end, these results are imperfect but still provide useful insight to guide the shaping of the model. With a larger hidden size, the model can take more time to learn and a higher learning rate may be desired. Conversely, with more epochs to train over, the model has more time to learn and learning rate may not need to be as high to pick up on patterns. The very basic assumption made here is that these effects cancel each other out and the optimal learning rate found here is valid for the final model. Further investigation would be necessary to determine the validity of this assumption but for now the assumption should be sufficient to build a functional model.

4.2 Hidden Size

Hidden size is the dimension of the hidden vector used in the LSTM process. It is common between both LSTMs used. Having the hidden size be constant between LSTMs takes away some of the customizability of the model but simplifies the evaluation and simplifies the interactions between the LSTMs outputs. Hidden size was varied while learning rate was kept constant at 0.001 and number of epochs was kept constant at 5.

Figure 3: Hidden unit tuning

Variation over hidden size shows that for SQuAD EM, the optimal hidden size is 100 while for SQuAD F1, the optimal hidden size is 50. Here, the location based F1 and EM scores both favor a hidden size of 100 and, with that added information, a hidden size of 100 was selected for the final model. Since the bulk of the computational time comes from training hidden variables, increasing the hidden size has a significant effect on computation time as shown below.

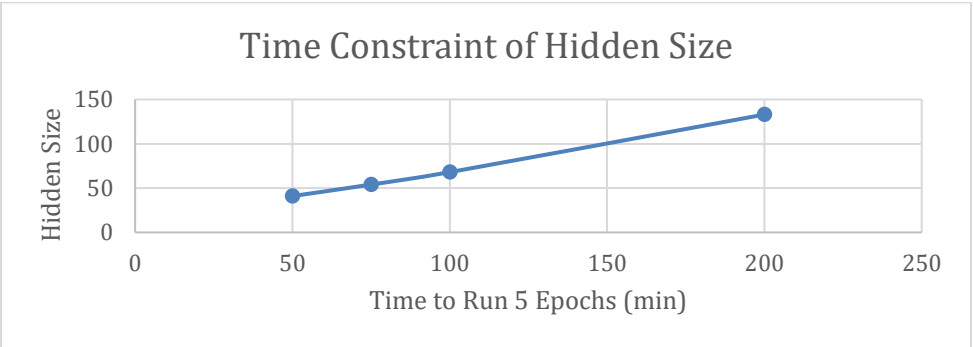

Figure 4 Effect of hidden size on computation time

While the relationship between hidden size and computational time is linear, it was determined that the computations were still fast enough at the optimal size 100 hidden unit that the increased computational time was not a major setback.

5 Results

5.1 Quantitative

A tuned Length Prediction model training on 8 epochs was used as the final model for this paper. When applied to a previously unseen testing dataset, this model produces a SQuAD F1 score of 33% and a SQuAD F1 score of 19% (location based scores were unavailable on the unseen dataset). This is significantly lower than the scores that were being seen during validation testing. This discrepancy is likely due to the small size of the validation dataset that was used to select model architectures and tune hyperparameters leading to overfitting in the design of the model and overconfidence in the model's performance.

5.2 Qualitative

Answers produced by this model tended to be very short. Most of the actual answers to the questions asked are short as well but that does not necessarily mean the model is learning to have shorter answers because short answers are common in the dataset. Having shorter predicted answers significantly reduces the number of false positives which leads to better F1 scores. Theoretically, this should be somewhat balanced by an increase in false negatives but there is no assurance of that.

The model seems to be most effective at answering questions where the answer is a single number such as a year. This is likely because saying what year something happened in and asking what year something happened in is a fairly common kind of question to ask and the model saw many example of it during training time.

The model struggles a lot when several parts of the context are highly related to the question. The model is able to tell that the different parts are good places to look because they are highly relevant to the question, but has a hard time distinguishing between them due to the limited nature of the attention model beyond simple word similarity.

6 Conclusions

6.1 Significance of results

The scores achieved by this model are far from state of the art, which is around 80% on F1 and 75% on EM. They do, however, represent a large improvement over random guessing and demonstrate that there is some learning occurring. With further work, these results could be improved upon significantly, particularly with continued use of the location based EM and F1 scores.

6.2 Future work

As discussed in 5.2, the limited nature of the attention model used here limits the effectiveness of the question answering model. A more advanced attention structure such as co-attention or making multiple passes of attention over the context are just two of several more advanced attention mechanisms that could extend this work.

Many of the tests described in this paper were done on models that only had one epoch to train. Although time constraints made this necessary for this paper, training models for longer could provide more helpful insight as to which models are worth pursuing more.

Acknowledgments

Thank you to Microsoft Azure for the use of their GPUs for this project.

References

[1] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. *arXiv preprint arXiv:1702.03814*, 2017.