Reading Comprehension Using Modified Match-LSTM

Edwin Park
Department of Computer Science
Stanford University
edpark@stanford.edu

Shubha Raghvendra
Department of Computer Science
Stanford University
sraghven@stanford.edu

Jeremy Wood
Department of Computer Science
Stanford University
jwood3@stanford.edu

Abstract

In this paper we detail our attemtps to replicate an extended Match-LSTM models that uses a dynamic loss function and a boundary based model with extensions based on observed parts of ynamic Coattention Models and a decoder that mirrors an AttentiveReader model.

1 Introduction

With the advent of large-scale data management systems has come a unique opportunity to not just index information, but also to begin to comprehend it. Systems like IBM's Watson and Google's Knowledge Graph have made strides in this direction; Watson has defeated human players in the question-answering game of Jeopardy, and the Knowledge Graph successfully offers up a response to fact-based queries (Sborio).

In this project we attempt to, in the vein of groups such as Wang et al and others, leverage deep learning to train a model that can identify the answer to a question from a context paragraph (Wang). In particular, this model was trained and evaluated on the Stanford Question Answering Dataset (SQuAD), described below. This question-answering problem, constrained as it is, a useful contribution to machine comprehension given that using context clues, sifting through a corpus to find documents relevant to a search term is already relatively achievable (Aggrawal).

Specifically, we began by attempting to emulate the architecture of Match-LSTM with Answer Pointer model presented by Wang et al. in 2017 (Wang). In the process, we introduced a number of structural simplifications and hyper parameter optimizations, elaborated on below; though we were unable to attain the dev accuracy reported by Wang and others, we made steps toward demonstrating that architectural simplicity could potentially more efficiently learn similar outcomes. Here, we define efficiency both in terms of model training time and it terms of neurons utilized.

1.1 Previous Work

As mentioned above, our model is based on the Match-LSTM model with Answer Pointer configuration described by Wang et al. in 2017. This architecture constitutes a uniquely synergistic combination of the Match-LSTM model, developed by the same research group for problems of textual entailment, and the Pointer Net model suggested by Vinyals et al in 2015. Pointer networks are a neural architecture used to predict the probability of a sequence of output tokens. Other models

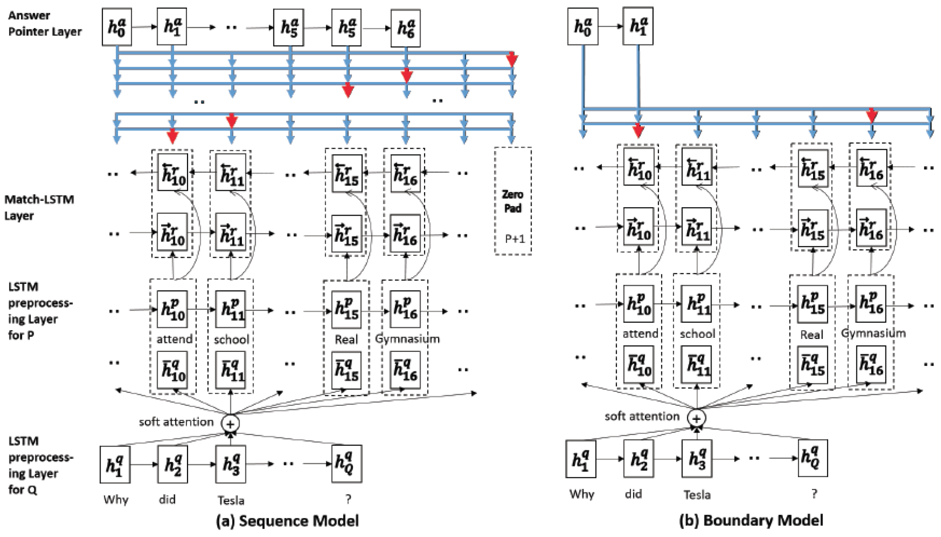

Figure 1: A schematic of the MLAP model in both its sequence model and boundary model forms, from Wang et al. in 2017. As noted below, it consists of three main layer, the preprocessing layer, the Match-LSTM Layer, and the Answer Pointer layer (Vinyals). The sequence model (a) represents an output of discrete tokens, while the boundary model (b), which we adopted, simply outputs start and end indices.

tackling the problem of question-answering currently in development by various groups include dynamic co-attention networks, bilateral multi-perspective matching, and fine-grained gating (Wang).

We were drawn to the Match-LSTM with Answer Pointer (MLAP) model specifically because it represented a very significant improvement in accuracy over the prior state-of-the-art architecture, which was a form of modified logistic regression. Additionally, it is a relatively straightforward architecture in comparison to contemporary models solving for similar problems, including multi-perspective matching and ensemble methods, so we believed we would be able to rapidly launch a baseline and quickly iterate upon it (though, as we later found, this was a somewhat optimistic assumption) (Wang).

1.2 Match LSTM with Answer Pointer

We will begin with a brief discussion of the model in question, MLAP. The model consists of three layers: the LSTM preprocessing layer, the Match-LSTM layer, and the Answer Pointer Layer (see Fig. 1 above). The LSTM preprocessing layer is intended to couch the representation of each word in the context paragraph and question in the context of the other words in that passage, and consists of a simple 1-D LSTM on a one-hot encoding of the tokens in question. This can be written as

$$H^p = \overrightarrow{LSTM}(P),\ H^q = \overrightarrow{LSTM}(Q)$$

where H^p and H^q (the hidden representations of of the passage and question) have dimensions $l \times P$ and $l \times Q$ respectively.

The bidirectional Match-LSTM layer then sequentially processes the question and paragraph information, subjecting the resulting weights and hidden states to tanh and softmax in order to obtain the attention weights α, using the following equations:

$$F_k = \tanh(VH^T + (W^a h^a_{k-1} + b^a) \otimes e_{(P+1)})$$

$$\beta_k = \text{softmax}(v^T F_k + c \otimes e_{(P+1)})$$

Using the above representation, we have $\beta_{k,j}$ as the probability of selecting the jth token from the passage as the kth token in the answer, given the previous $k-1$ tokens in the answer. We

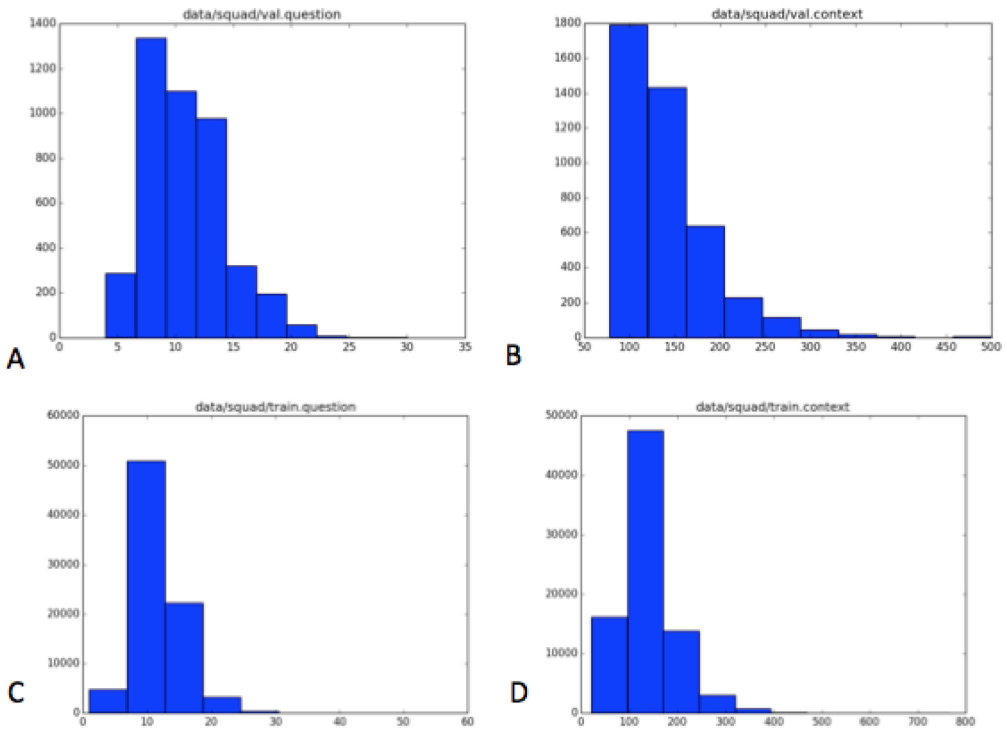

Figure 2: Above are histograms capturing the frequency of questions and context paragraphs of various lengths in the training and validation sets. On the basis of figures B and D it is evident that the vast majority of context paragraphs are under 200 tokens in length, and from A and C it is clear that in both the val and train sets, questions tend to not exceed 20 tokens.

train the model by choosing $\beta_{k,j}$ for each $k = 1, ..., N$ that maximize the probability of generating the answer sequence $p(a|H^r) = \prod_k p(a_k|a_1, ..., a_{k-1})$, or equivalently, minimizing the sum of negative log probabilities as the loss function: $-\sum_{n=1}^{N} \log p(a_n|P_n, Q_n)$.

However, we ended up applying this with the boundary model: wherein we predict only the start and end indices, the latter being conditioned on the former. In papers reveiwed, this had slightly better performance than a sequence model that predicts loosely-enforced sequential tokens.

2 Methods

2.1 Dataset

All experimentation was conducted on the Stanford Question Answering Dataset (SQuAD) v1.1 (Rajpurkar), which is a bank of over 87000 question and answer pairs extracted from 536 distinct Wikipedia articles spanning various topics, from Nikolai Tesla to the geography of California.

In particular, we utilized GloVe word embeddings, with dimensionality 300. Further, the data was split into a train set (87,599 pairs) and a dev set (10,570 pairs). Before constructing our model, we did a preliminary analysis of our dataset, summarized in Fig. 2.

The take-aways from Fig. 2, namely that 200 tokens are likely sufficient to capture most context paragraphs, and that 20 tokens are generally sufficient to capture most questions, became important when we later incurred several out-of-memory errors and had to shave down the size of the parameters being trained on. .

2.2 Approach

We began by attempting to implement the MLAP model described above, and planned to evaluate or models on the basis of training and validation loss, as well as F1 and ExactMatch (EM) scores. The EM score is simply the percentage of start and end-token predictions that exactly match the ground truth answers. The F1 score represents the harmonic mean between the precision and recall of a model's predictions, and is given by:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

We incurred a series of challenges (described below) in the implementation of our model, culminating a lackluster F1 and EM scores (F1 never rising above 8, and EM never above 2.0) over all ten epochs of training. This prompted us to conduct ablative analysis of our model; we essentially sequentially deleted models of our layer and drastically simplified remaining models to create a minimum viable model to assess results on, and this effort ended up driving up results significantly. This effort resulted in three novel architectures, discussed below. More thorough statistics from each layer are also reported in results.

First, we nixed the MLAP decoding layer and collapsed the Match-LSTM and encoding layers into a unidirectional dynamic encoder. Under this schema, the question and answer are encoded, and the paragraph is encoded on the basis of the question's hidden state. We then mixed the question and answer encodings with a simple dot product and ran a decoder to extract an output. We were surprised to note a significant jump in F1 scores to 35 on average after 10 epochs; EM scores also climbed modestly to 5.

Encouraged, we incrementally added in complexity to the model. Next, we shifted from a unidirectional dynamic encoder to a bidirectional dynamic encoder, which also improved F1 and EM scores from the unidirectional encoder, though we did not run this model beyond three epochs in an effort to iterate quickly.

Finally, we arrived at our most successful architecture to-date, which involves the introduction of a mix layer constituting an encoding of the paragraph conditioned on the question. Results for this model are reported more in-depth in the figures in "Results" below, but we were able to obtain an F1 of 42 and an EM of 24. After arriving at this optimal architecture, given the time constraints associated with completing this project, we shifted to hyperparameter tuning and regularization strategies, which are dicussed in "Results" below.

2.3 Extensions

One extension that we tried was modifying the loss function to penalize based on how close an answer was to the desired answer. Under the MatchLSTM paradigm (and many others), loss is distributed based on how likely the model rates the true answer to be. This works well if we only care about exact match scores. However, we wanted to try optimizing for F1 by scaling the loss based on the proximity of the top predictions to the real answer. Thus for the loss of both start and end we applied the following multiplier based on our predicted probability distribution, β and the true answer a:

$$l_{mult} = \sqrt{(\arg\max(\beta) - a)}$$

However we saw no significant improvement from adding this multiplier to loss, implying that our model likely lacked the architecture to drive an improvement in F1 score.

We also tried various methods of transforming the encoded MatchLSTM layer when feeding it through our eventual architecture with attentive reader decoders. We tried providing a nonlinear layer in between the decoding of the start position and the end position. However this negatively impacted performance. We also tried further mixing the MatchLSTM output with the originally encoded paragraph. However while this sped up the model's arrival at its peak performance, this peak performance was slightly less than our prior model.

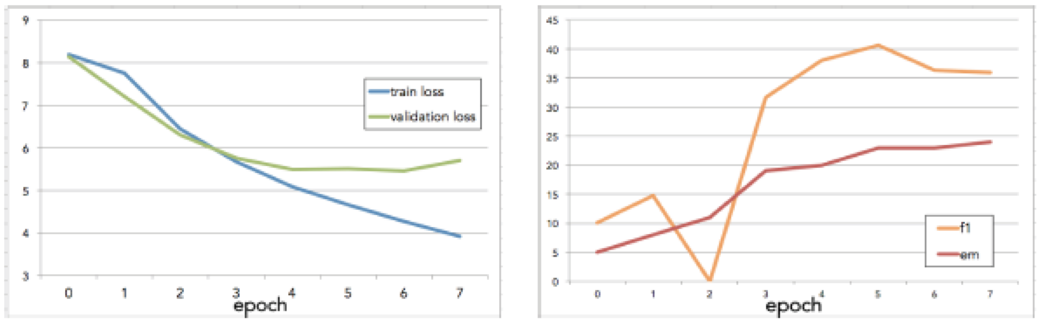

Figure 3: (L) Train loss consistently decreases epoch to epoch in the bidirectional dynamic encoder modification. While validation loss also trends downward, it plateaus out around the sixth epoch, prompting us to explore regularization options (see next panel). (R) EM consistently climbed from epoch to epoch, and with the notable exception of a dip around epoch 2, F1 did as well. We have not yet been able to complete another round of training on the BDE with mix model but are interested to know whether this early dropoff in F1 is an artifact.

2.4 Challenges

We encountered our fair share of challenges in attempting to implement this model, which contributed to our ultimately simplified submission.

First, we spent a great deal of time grappling with out-of-memory errors on our GPU, an issue that was solved both by reducing the sizes of the various hidden states being used, as well as avoiding manually unrolling LSTMs whenever possible.

After this issue was resolved, we struggled with understanding why our would-be MLAP implementation seemed to not be learning. In particular, error analysis revealed that often times padded (o) tokens would be predicted as the start and/or end index of an answer, indicating that we were likely not handling our mask appropriately; in addition, we sometimes observed start tokens predicted after end tokens. We attempted to incorporate a variety of heuristics to work around this issue, including adding an arbitrarily large number to loss to penalize the prediction of padded tokens. Ultimately, these fixes were only semi-successful and we found that the issues were only completed resolved after the re-instrumentation of the architecture, as described above.

3 Results

Results in terms of loss and F1 and EM scores for our best model, prior to hyper parameter tuning and regularization attempts, are presented in Fig. 3. Our best model achieved F1 score of 33.481 and EM score of 21.447 on the dev set. On the test set, this model achieved F1 score of 34.841 and EM score of 22.721, slightly higher than the dev set. As shown by the presented data, train loss consistently decreases epoch to epoch in the bidirectional dynamic encoder modification. While validation loss also trends downward, it plateaus out around the sixth epoch, prompting us to explore regularization options (see next section). EM consistently climbed from epoch to epoch, and with the notable exception of a dip around epoch 2, F1 did as well. We have not yet been able to complete another round of training on the BDE with mix model but are interested to know whether this early dropoff in F1 is an artifact; further rounds of experimentation with models much like this one lead us to believe that this was in fact a one-off-issue.

3.1 Hyperparameter Tuning

Wang et al, among others, relate the importance of tuning hyperparameters to the achievement of optimal results, particularly in terms of improving F1 and EM scores. To this end, we conducted a manual hyperparameter search, manipulating terms such as learning rate, decay rate (for the the learning rate), and the use of the Adam Optimizer, which makes gradient updates using the momentum method. We also implemented Xavier initialization to ensure that matrices were not intialized

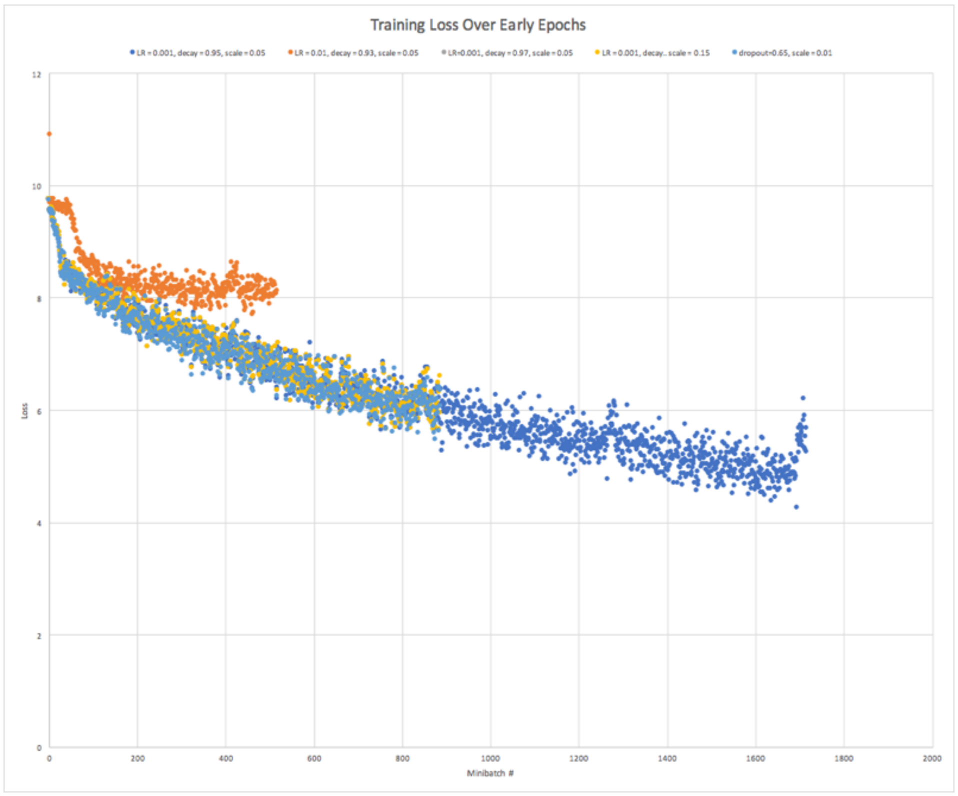

Figure 4: Presented above is a summary of training loss across multiple epochs for various experimental models with hyperparameter and regularization tuning. In dark blue is LR = 0.001, decay = 0.95, and scale = 0.05; in orange is LR = 0.01, decay = 0.93, and scale = 0.05; in gray is LR = 0.001, decya = 0.97, and scale = 0.05; in yellow is LR = 0.0001 and scale = 0.15; in light blue is 0.65 and scale = 0.01.

to all-0 values. We report a summary of the results of this hyperaparameter search (with only some experimental models shown) above in Fig. 4.

3.2 Regularization Strategies

As evidenced in Fig. 3, validation loss plateaus around epoch 7 of the optimal BDE model, meaning in its present state the model does not generalize as well as it could. Therefore, we began testing various regularization strategies including dropout (with various keep probabilities) and L2 normalization (with various λ values). We report the results of models with various permutations of these parameters above in Fig. 4, above.

3.3 Other Strategies

On both training and validation data, we saw some trade-offs of predicting spans. Particularly when optimizing for EM, the easier questions to predict were accompanied by shorter ($<$ 4 tokens) answers, whereas more difficult questions had long answers. When using a boundary model, we saw a few examples of questions with one-word answers where we did not get an EM because we predicted one of either `start` or `end` correctly, but not both. In these cases, our model predicted the answer to be within span, but had problem in selecting specific ranges. For long answers, it was difficult to predict the start and end correctly, since the boundary model has fewer assumptions about the tokens existing between the span. Additionally, we experimented by modifying the loss function to penalize the probability distributions of padded tokens by including the sum of their β distributions as a contribution to the loss, though this loss ended up favoring earlier tokens and discounting later tokens, which led to less inaccurate predictions in many cases.

4 Conclusions

As evidenced by the losses plotted in Fig. 4 across a variety of models, our experiments demonstrate that slow learning (that is, a learning rate of around 0.001) was optimal for tuning our specific model. Moreover, when manipulating other hyper parameters, including decay and scale, we found that differences were negligible, and that F1 scores and losses were comparable. This leads us to believe that beyond adhering to a relatively small learning rate, expanding our hyperparameter search likely will not be very fruitful; instead, we would considering implementing more complex architectures (discussed below), though time constraints did not allow for this iteration.

To conclude, our optimal model attained an F1 of 42 and an EM of 24, and consisted of a simplified bidirectional dynamic encoder with mix implemented. While we did not attain the accuracy of the MLAP model we based our initial architecture on, we were surprised by how far we got with a relatively simple model and hyperparameter tuning, and conclude that we built a efficient model in that relatively few layers (and hence neurons) contributed to respectable learning.

4.1 Further Work

As mentioned above, ideally we would like to implement more complex architectures to continue to build on our improvements in our EM and F1 scores. Before venturing to more complex architectures such as the afore mentioned multiperspective matching, we would like to resolve the issues with the Match-LSTM with Answer Pointer architecture we had initially attempted. It is clear to us that at some level we had a buggy implementation of the schema (else our initial Match-LSTM scores would have been much higher, and loss would have decreased), so resolving these issues is critical. Next steps would include incorporating other architectures and further tuning of regularization and hyper- parameters.

5 References

[1] Sborio, M. L. ”From knowledge graphs to cognitive computing”. IBM Research Ireland. https://www.ibm.com/blogs/research/2016/01/from-knowledge-graphs-to-cognitive-computing/. Accessed 20 March 2017.

[2] Wang, S. & Jiang, J. Machine comprehension using match-lstm and answer pointer. arXiv preprint arXiv:1608.07905, 2016.

[3] Aggrawal, C. & Yu, P. On Effective Conceptual Indexing and Similarity Search in Text Data. ICDM ’01 Proceedings of the 2001 IEEE International Conference on Data Mining 2001.

[4] Vinyals, O. et al. Pointer networks. Proceedings of the Conference on Advances in Neural Information Processing Systems, 2015.

[5] Rajpurkar, P. et al. Squad: 100, 000+ questions for machine comprehension of text. CoRR, abs/1606.05250, 2016.

[6] Chen, D. et al. A thorough examination of the CNN/Daily Mail reading comprehension task. In Proceedings of the Conference on Association for Computational Linguistics, 2016.