#SQuADGoals: A Comparative Analysis of Models for Closed Domain Question Answering

Andrew Duffy **Daniel Shiferaw** **Eric Musyoka**

Abstract

The task of machine comprehension has long been one of the greatest unsolved tasks in natural language processing. Thanks to a proliferation of datasets, numerous deep neural architectures have been created. We implemented two such architectures: the Match-LSTM with Answer Pointer[1] and a flavor of Dynamic Coattention Networks[2]. Our end-to-end models are able to exceed an existing logistic regression baseline with millions of derived features without having to perform any feature extraction, demonstrating the neural architectures are well poised to be the dominant model for conquering QA.

1 Introduction

The task of question answering is one of the greatest unsolved problems in the field of NLP. We are still a long way off from building systems that can truly understand and answer questions in an open domain, but Weston et al. propose a set of prerequisite tasks that researchers can focus on first[3]. The hope is that creating models for these constituent tasks will eventually allow us to compose models and create more advanced systems.

The main subtask the field is currently concerned with is machine comprehension. The formulation of the task is quite simple: given a reading passage, evaluate how much the model is able to learn. The way we measure machine comprehension of text will be familiar to anyone who has gone through elementary education: we ask the model questions about the text. If the model answers correctly, we determine that it has learned and reasoned effectively given the question about the context text. This specification allows us to cast the comprehension problem as a supervised classification problem, for which we already have many models. In general, reading comprehension not only requires the ability to reason effectively to answer questions about text but also pool from outside knowledge of the world, which is a challenging and involved enough task in its own right. To focus evaluation of models on the reasoning side and avoid a substantial focus on whatever information retrieval/extraction techniques necessary to solve general reading comprehension, closed question answering developed, which is effectively reading comprehension in which the information needed to answer the question is contained with the context test. Existing models that attain decent performance on machine comprehension have in the hundreds of millions of parameters[4], while still performing far below human level performance on the task. There is clear room for improvement on this front both in terms of performance and model efficiency. We reviewed the recent literature and found a pair of deep learning models that achieved decent performance on this task. While we do not achieve the same results as the original authors, we try numerous different variations on the models and are able to exceed the logistic regression baseline with less than 3% of the parameters of traditional models using end-to-end deep learning techniques.

1.1 Datasets

The formulation of the task is simple enough, but access to strong datasets has proven to be a bottleneck up until recently. Facebook Artificial Intelligence Research released the open source bAbI dataset[3], while Google DeepMind released the CNN/Daily Mail dataset[5]. Both datasets have been very helpful in taking steps forward, but there are a variety of drawbacks to each. The babI dataset provides text-based simulation data to target twenty question answering tasks. However, the simulated nature of the data limits the context text it can present and thus does not generalize well to context texts for reading comprehension. The CNN/Daily Mail dataset loosely leveraged computational abstractive summarization techniques to automatically convert online CNN and Daily Mail articles and their matching summaries to document-query-answer triples. While Google DeepMind provided one of the first large-scale supervised reading comprehension datasets, the automatic query generation process inherently limited the nature of the questions posed, as they could not be reflective of the wide variety of questions and the associated reasoning techniques needed for closed question answering in general. Thus existing machine comprehension models working on these datasets tend to actually learn very little about the passages[6]. Below, we discuss the recently introduced SQuAD dataset in depth.

1.2 The Question Answering Task and SQuAD

The SQuAD dataset[4] was released by Rajpurkar et al. in 2016 to fill the void for high quality QA datasets. It asks questions based off of paragraphs in Wikipedia text, where all answers are *spans* of the passage, i.e. some contiguous run of tokens from the original text. All models are trained on (q, p, a) triplets, where given the provided question and passage, the answer span can be extracted from the passage.

SQuAD is run like a contest reminiscent of the Netflix Prize: researchers are ranked on a public leaderboard, a training and development set are provided and then all submissions are scored against an internal hidden test set by the SQuAD authors. We decide to try and implement some of the modest models from the leaderboard, **Match-LSTM with Answer Pointer** as well as **Dynamic Coattention Networks**. These two are good choices for re-implementation because they both topped the leaderboard when first submitted, and the teams behind both have published detailed research on their model architectures.

Our main contribution is faithful implementations of these two models modulo a set of extensions. Both models achieve decent performance on the development set. Section 2 goes into detail on the models and their version, Section 3 gives details on our experiment setup, Section 4 explains the results of our experiments and Section 5 details future work. In Section 6 we include some concluding thoughts on the success of neural models for NLP tasks in general.

2 Models

Specify task for SQUAD: We take this section to describe at a high-level the models we implemented, as well as some of the extensions we made from their original formulations.

2.1 Match-LSTM with Answer Pointer

Wang et al. published the Match-LSTM with Answer pointer, and for a time enjoyed the top seat on the official SQuAD leaderboard[1]. It has since been replaced with other models, but we wanted to look into Wang's model as a starting point for more advanced models.

The model makes use of sequence attention with a pointer network[7]. Pointer Networks were put forward by Vinyals et al. as a new neural architecture for learning functions whose outputs were some possibly permuted subset of the inputs. In their work, they formulated a pointer network that could learn to solve the traveling salesman problem approximately. Here, Wang et al. propose using pointer networks to predict the start and end tokens [2].

[1] `http://stanford-qa.com`

[2] In the original paper they also propose a sequence model, but we did not use it in our experiments.

2.1.1 Match-LSTM formulation

The Match-LSTM with Answer Pointer makes use of three sequential layers: **LSTM preprocessing**, **Match-LSTM** and **Answer Pointer**

The preprocessing layer runs the question Q_i and passage P_i through an LSTM, yielding matrices $H_i^Q, H_i^P \in \mathbb{R}^{m \times l}$, where m is the maximum length of a passage and l is the hidden state size.

The Match-LSTM layer is a second encoding layer that runs over the hidden states of the passage h_i^P, calculating an attention distribution α_i against each question token. It outputs an encoded matrix $H_i^{(r)} \in \mathbb{R}^{P \times l}$.

The answer pointer layer takes in the $H_i^{(r)}$ and runs it through another LSTM layer that outputs a probability distribution β_i over the potential start tokens, and a second distribution over the potential end tokens. In this simple boundary model, we take the maximum over the first distribution and choose that as the start, and do the same for the end token distribution. Together that gives us our predicted answer span.

2.1.2 Variations on the Basic Model

Aside from the boundary model from the original paper, we also adopted a BiLSTM encoder for the preprocessing layer to account for all surrounding context for questions and passages.

We also applied masking to our logit distributions before applying softmax to disable the boundary model from predicting padding tokens as the start or end of a span. As the penultimate step before our answer pointer decoder, we had a logit distribution β_{il}. We create a special masking vector m_i that is 0 for all the token positions, and then a large negative number such as -1000 for the positions with padding. Adding these vectors together before performing the softmax gives us a new probability distribution that prevents predicting the padding tokens.

2.2 Dynamic Coattention Networks

We also implemented a variant of the Dynamic Coattention Network architecture, hereafter referred to as DCN[2]. DCN first develops an attentive co-dependent representation of the question and document for the encoding to capture the strongest relationships contained therein. We by and large preserve the DCN architecture for this encoding in our variant except we neglect to include sentinel vectors for attention. For the decoder, DCN employs a dynamic architecture, which iterates over possible answer spans until it decides to output an answer. Given time and resource constraints, we implemented a simpler but focused decoder. Our decoder formulated the answer prediction problem as a boundary problem by predicting the start and end tokens of an answer by two separate but unified sub-architectures. We first developed separate representations for the start and end tokens of an answer via an LSTM working on the attentive co-dependent representation of the question and document. We then concatenated these representations as our co-dependent answer representation and finally utilized two separate Bidirectional LSTMs to predict the start and end answer token probabilities. We will elaborate on the final parametrization of our architecture in the experiment section.

To elaborate on the encoder architecture, we ran the same LSTM (to unify the representation) to encode the GloVe representations of our question and document. Let's call those final representations Q and D, respectively. To enable the network to learn a relationship between the question and document space, we added a non-linear projection layer on top of the question with a $Q = tanh(W^{(Q)}Q')$. For coattention, we first compute an affinity matrix, $L = D^T Q$. This matrix computes some initial score for each document-question word pair. We then softmax this matrix row-wise to produce normalized attention for a document relative to each word in the question (A^Q) and then column-wise to produce normalized attention for each question relative to a word in a document (A^D). Next, we derive the summaries of the documents informed by its attention for a question by $C^Q = DA^Q$. We do the same for the question informed by its attention for each word in a document QA^D. Lastly, we compute the attention context of C^Q informed by each word in a document ($C^Q A^D$), concatenate that with QA^D to form a codependent representation of the ques-

Table 1: DCN Hyperparameters vs Loss

STATE SIZE	DECODER	DROPOUT	MIN VAL LOSS
128	Decoder with no pre-processing LSTMs	0.1	4.53
96	Decoder with no pre-processing LSTMs	0.1	4.35
96	Decoder with pre-processing LSTMs	0.1	4.24
96	Decoder with pre-processing LSTMs	0.2	4.43

tion and document, and then run a bidirectional LSTM on on that representation to define U, the initial encoding of our start and end answer tokens.

To begin decoding, we leveraged two separate LSTMs to prepare our initial encoding of the start and end answer spans, which we'll call U_s and U_e. We then concatenated those representations into another unified representation of the start and end answer spans, U'. We finally ran two separate Bidirectional LSTMs with a linear projection layer on top to output our final probabilities for start and end answer tokens.

In the experiment section below, note we also to start had a decoder that did not perform the inital encoding and concatenation of the start and end answer spans and instead started prediction directly from U. We found that this addition to our decoder architecture helped performance marginally but significantly enough to include in our final model.

3 Experiment Details

The vocabulary for our models was constructed from the training and validation passages and questions, and contained over 115k distinct token types. For our word embeddings, all experiments were run with 300-dimensional GloVe vectors from the Common Crawl Corpus[8]. All our software was implemented in Tensorflow 1.0 and written in Python 3 [9].

The Match-LSTM model had 3.7 million parameters, and was trained using the Adamax optimizer[3]. We used a batch size of 64 while training. Our hidden state size l is 150, some experiments were performed with $l = 300$ but initial results were not promising so all reported results are with the former choice.

The most common technique in regularizing recurrent neural networks is dropout[10]. Various values of the drop probability were tried, but the optimal choice was found to be $p = 0.4$, in keeping with the wisdom of Srivastav et al.'s original work that recommends using a value between 0.2 to 0.5. All Match-LSTM based models we report here were trained with this drop probability. For details about DCN dropout values, see Table 1.

A copy of the model parameters was saved at the end of every epoch of training, and we report the one with the highest overall F1 score on the development set.

For the DCN model, our final model had 1,377,925 parameters. Our hyper-parameter testing is summarized in the table above.

4 Results

We list our results for the various models on Table 2. Notably, our Match-LSTM model performed significantly higher when using the BiLSTM encoder for preprocessing, likely due to the increased contextual information available in the encodings.

For our DCN model, we surprisingly found that reducing the state size helped our model's performance quite a bit. We anticipate that this reduction in feature size forced the architecture to learn more robust representations for the reading comprehension task. We also found that dropout beyond

[3]Implementation of Adamax was taken from OpenAI: `https://github.com/openai/iaf`

Table 2: Best F1/EM Score Across Epochs For Match-LSTM Based Models

MODEL	F1	EM
Match-LSTM	46.14	34.20
Match-LSTM + BiLSTM	51.98	39.89

a small threshold tended to hurt our model's performace. We believe that this resulted because our model's architecture was already lean, and dropout would cause it to underfit. Lastly, we found that increasing the number of layers beyond one through the sub-networks in our architecture dramatically hurt our model's performance. We think that, given the complexity of the architecture, gradients could not backpropagate effectively through time to the beginning of the network. Keeping the networks shallow helped our lean model learn quickly and effectively and take advantage of the representation and attention mechanisms to tackle the task.

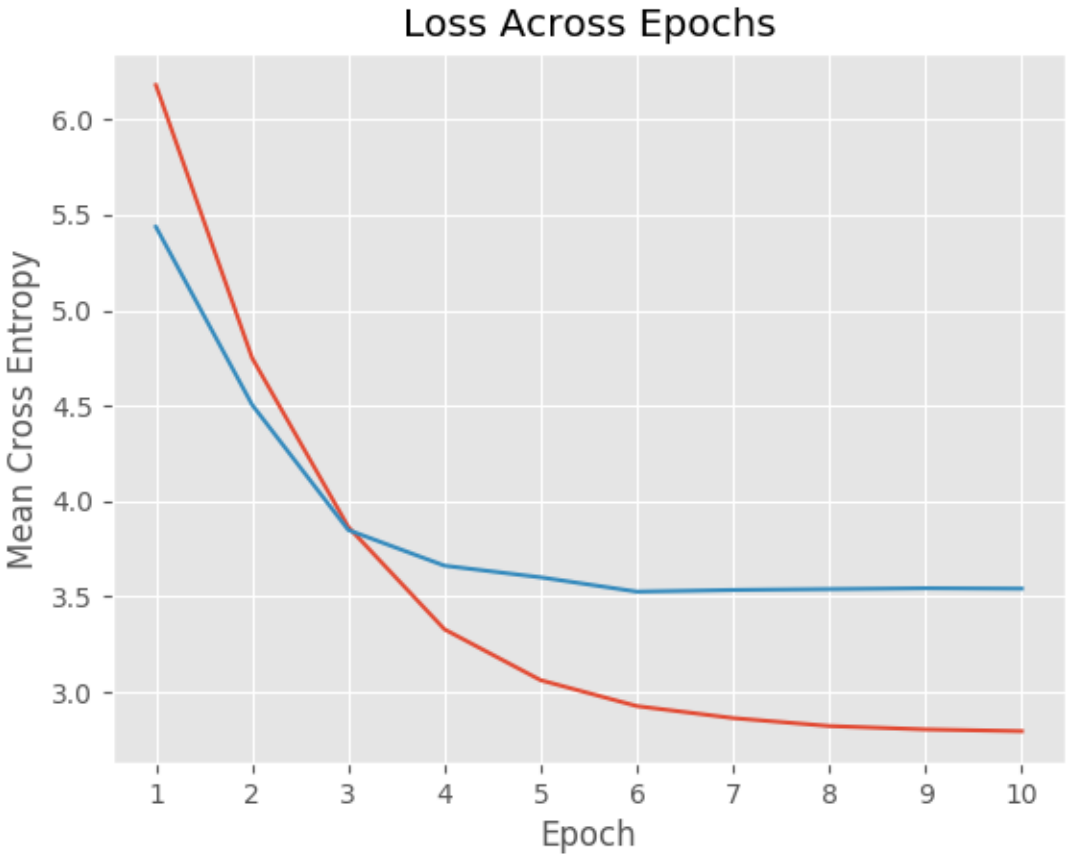

Figure 1: Training and validation error across epochs for the Match-LSTM with BiLSTM preprocessing. Train in red, validation in blue.

5 Future Work

As with any experiment, there are countless extensions that we were not able to try. One of the simplest is to add features beyond simple tokens. As we can see in Figure 5, the distribution over

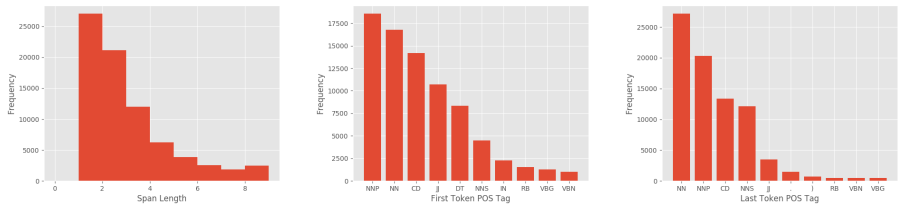

Figure 2: Plots of the distributions of span lengths, and most common starting and ending token POS tag.

the training dataset of span lengths and start and end token part of speech are largely clustered around a small handful of tags. Both of these pieces of information alone could be combined into a third model that classifies all possible spans up to a maximum length directly. We would then find either the most likely span, or perhaps take a learning to rank approach where we build a pairwise model that finds the best of each pair of possible spans until we are reduced to a "winner". This approach bears some similarity to the work of Lee et al. that builds out spans[11]. They claim that their performance comes form being able to classify spans rather than generating new spans, and based on their results is an interesting avenue for future research.

It might also be feasible to perform ensembling of our models. Each of our models outputs a logit distribution over tokens in the last stage. If we summed these scores before performing softmax normalization, we would theoretically end up with a variable that is the sum of each model's opinion of each character in the passage.

6 Concluding Remarks

Over the course of CS 224N, we've seen how to apply deep learning models to achieve state-of-the-art results for a variety of tasks, including translation, sentiment analysis and dependency parsing. Translation seemed to require the most ingenuity to get to its current state right now, and now that systems such as Google NMT[12] have achieved phenomenal results, it has taken decades to reach this point. The question answering task is likely to be similar, but it benefits from the fact that so many deep learning models for other learning tasks exist at this point in time. All the models we have studied seem to borrow bits and pieces of old models as the old saying goes, *there is nothing new under the Sun*.

Our models' performance was modest but decent, and the fact that both surpassed the score for a strongly engineered traditional machine learning model gives us hope that more great leaps for NLP will come from the application of neural architectures.

Acknowledgments and Contributions

We would like to thank Richard Socher and Christopher Manning for their detailed and well-organized instruction. Many of the more practically minded lectures gave us the ideas for techniques we applied in our study. In terms of the division of work, Eric and Andrew worked primarily on the coding the Match LSTM model and exploratory data analysis. Daniel primarily worked on coding the DCN model. Everyone together worked on brainstorming and experiment analysis.

References

[1] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016.

[2] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016.

[3] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698, 2015.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[5] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015.

[6] Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *Association for Computational Linguistics (ACL)*, 2016.

[7] 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.

[8] 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.

[9] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[11] Kenton Lee, Tom Kwiatkowski, Ankur P. Parikh, and Dipanjan Das. Learning recurrent span representations for extractive question answering. *CoRR*, abs/1611.01436, 2016.

[12] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.