Deep Coattention Networks for Reading Comprehension

Amy Bearman
Department of Computer Science
Stanford University
abearman@cs.stanford.edu

Abstract

Machine reading comprehension of text is an important task in Natural Language Processing. A recently released dataset, the Stanford Question Answering Dataset (SQuAD) formulates the problem as question answering, and it provides a large corpus of challenging, realistic questions. To address this task, we implement an end-to-end neural encoder/decoder model. The encoder consists of the coattention model proposed by [10], which intelligently interleaves both the question and context paragraph encodings into one co-dependent knowledge representation, and the decoder is a simple linear classifier. Our experiments show that this model achieves an F1 score of 59.37% and an exact match score of 42.4% on the validation dataset.

1 Introduction and Related Work

Reading comprehension is an essential task in Natural Language Processing. It poses multiple challenges, including the need for multi-sentence reasoning, the ability to maintain long-range context, and the ability to output answers of variable length. Researchers have developed many tasks to assess a machine's ability to process and understand text. Typically, the machine is presented with a piece of text, such as a news story or essay. The machine is then given questions to answer based on the context text. This task is essentially what Google Search does with its OneBox and Knowledge Panel features. When a user asks a question about a specific entity, Google reads the corresponding text (usually a Wikipedia article) and attempts to locate the answer within the document (see Figure 1).

To address this problem, researchers have developed large-scale, annotated datasets. Recently, [6] released the Stanford Question Answering Dataset (SQuAD), which is an order of magnitude larger than previous datasets and contains more challenging questions. Moreover, the questions and answers in SQuAD were produced by crowdsourced humans, which makes them more realistic. We use this dataset to evaluate our models for the reading comprehension task.

Machine comprehension of text has gained interest in recent years as a popular task, and many of the recent approaches have involved neural architectures. These authors have frequently used recurrent neural networks (RNNs), including long short-term memory units (LSTMs), to process the provided question and paragraph and predict the answer [3, 9, 10]. Many papers also use the attention mechanism [2, 3, 7] on top of the recurrent units in order to capture dependencies between the question and the context paragraph.

In this paper, we implement the coattention model for question answering introduced by [10]. The model consists of a coattention encoder, which captures the interactions between the question and the document, and a simple linear decoder that predicts the start and end index of the answer span. Our model obtains an F1 score of 59.37% and an EM score of 42.4% on the validation set.

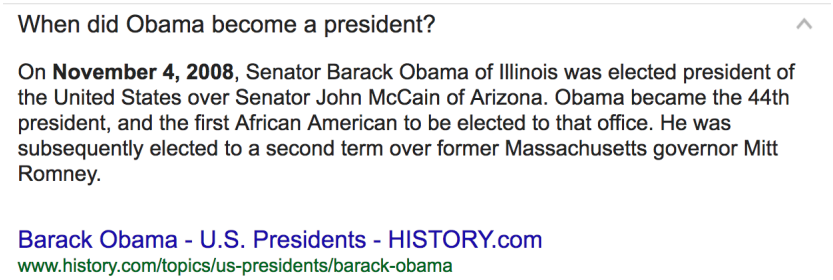


Figure 1: Google Search OneBox demonstrating the reading comprehension task.

2 Method

We use an encoder/decoder approach, as in Neural Machine Translation. Our encoder is largely based on the one developed by [10].

2.1 Document and Question Encoder

Let $(x_1^Q, x_2^Q, \ldots, x_n^Q)$ be the sequence of word vectors (each of dimension $\mathbb{R}^d$) corresponding to a question of length n, and $(x_1^D, x_2^D, \ldots, x_n^D)$ be the word vectors corresponding to an answer document of length m.

We use an (one-directional) LSTM to encode the question as: $q_t = \text{LSTM}_{enc}(q_{t-1}, x_t^Q)$. We define an intermediate question representation as the outputs of this LSTM stacked horizontally: $Q' = [q_1, q_2 \ldots q_n] \in \mathbb{R}^{\ell \times n}$, where ℓ is the size of the LSTM. We also add a non-linear projection layer to the question encoding so that the question space and the document space are not linearly related:

$$Q = \tanh\left(W^{(Q)} Q' + b^{(Q)}\right) \in \mathbb{R}^{\ell \times n} \tag{1}$$

We encode the document with the same encoding LSTM as the question: $d_t = \text{LSTM}_{enc}(d_{t-1}, x_t^D)$. The document encoding matrix is the outputs of this LSTM: $D = [d_1, d_2 \ldots d_m] \in \mathbb{R}^{\ell \times m}$.

2.2 Coattention Encoder

We use a coattention mechanism that simultaneously generates attention contexts for both the question and document, and then fuses them together. See Figure **??** for a diagram of this architecture.

First, we compute the **affinity matrix**, L, which contains affinity scores for all pairs of question word encodings q_t and document word encodings d_t:

$$L = D^T Q \in \mathbb{R}^{m \times n} \tag{2}$$

We use L to generate two matrices of **attention weights**, A^Q and A^D, which are just the attention scores normalized over different dimensions. A^Q is L normalized over rows, and it contains the attention weights across the *document* for every word in the question. A^D is L normalized over columns, and it contains the attention weights across the *question* for every word in the document:

$$\begin{aligned} A^Q &= \text{softmax}(L) \in \mathbb{R}^{m \times n} \\ A^D &= \text{softmax}(L^T) \in \mathbb{R}^{n \times m} \end{aligned} \tag{3}$$

Next, we compute the **attention contexts**, C^Q and C^D. The attention contexts are a weighted (using the attention weights) average of the source hidden states (question word encodings or document word encodings). C^Q is the attention context of the document over every word in the question, and is calculated simply as following:

$$C^Q = D A^Q \in \mathbb{R}^{\ell \times n} \tag{4}$$

C^D is the attention context of the question over every word in the document. Its computation is slightly more complicated. Rather than just computing QA^D, we also compute $C^Q A^D$ and horizontally concatenate these results:

$$C^D = [Q; C^Q]A^D \in \mathbb{R}^{2\ell \times m} \tag{5}$$

Intuitively, $C^Q A^D$ is the mapping of the question encoding into the document encoding space. C^D is a co-dependent representation of both the question and document, or the "co-attention context."

We then horizontally concatenate D and C^D to get our final knowledge representation:

$$K = [D; C^D] \in \mathbb{R}^{3\ell \times m} \tag{6}$$

Finally, we fuse the temporal information of the document to the coattention context by using a bi-directional LSTM:

$$u_t = \text{Bi-LSTM}(u_{t-1}, u_{t+1}, K) \in \mathbb{R}^{2\ell} \tag{7}$$

The encoder returns the outputs of this LSTM: $U = [u_1, u_2, \ldots, u_m] \in \mathbb{R}^{2\ell \times m}$.

2.3 Decoder

We omit the more complicated decoder of [10], and instead simply train a m-way linear classifier to output a_s, the start index of the answer in the context document:

$$a_{s,scores} = UW \in \mathbb{R}^m \tag{8}$$

We train a second m-way linear classifier to output a_e, the end index of the answer in the context document. However, we first pass U through one more (one-directional) LSTM. This is because the question/document encoding, U is not expressive enough to determine both a_s and a_e; we need to first transform U into the "end answer" space.

$$\begin{aligned} u' &= \text{LSTM}_{dec}() \in \mathbb{R} \\ U' &= [u_1, u_2, \ldots u_m] \in \mathbb{R} \\ a_{e,scores} &= U'W' \in \mathbb{R}^m \end{aligned} \tag{9}$$

2.4 Loss Function

During training, we optimize our model using a basic softmax cross-entropy loss. $a^{(i)}_{e,gt}$ and $a^{(i)}_{s,gt}$ are the ground truth labels for the one training example's start index and end index of the answer in the paragraph, respectively.

$$\begin{aligned} a^{(i)}_{s,pred} &= \arg\max a^{(i)}_{s,scores} \\ a^{(i)}_{e,pred} &= \arg\max a^{(i)}_{e,scores} \\ \mathcal{L} &= -\frac{1}{N}\sum_i a^{(i)}_{s,gt} \log \text{softmax}\big(a^{(i)}_{s,scores}\big) + a^{(i)}_{e,gt} \log \text{softmax}\big(a^{(i)}_{e,scores}\big) \end{aligned} \tag{10}$$

where N is the number of training examples. The softmax function simply normalizes the predicted scores into probabilities.

3 Experiments

3.1 Dataset

To evaluate our algorithm, we use the Stanford Question Answering Dataset (SQuAD). SQuAD has a vocabulary size of 115,613 words and is composed of 81,381 training examples and 4,284 validation examples. Each example is a triplet containing $\langle$*question, context, answer*$\rangle$. The context paragraphs were extracted from a set of 536 diverse articles from Wikipedia, and the questions were generated by humans.

The goal of the SQuAD task is to predict an answer span tuple $\{a_s, a_e\}$, where a_s is the index into the context paragraph of the first token in the answer, and a_e is the index of the last token in the answer. The following is an example of one such training example:

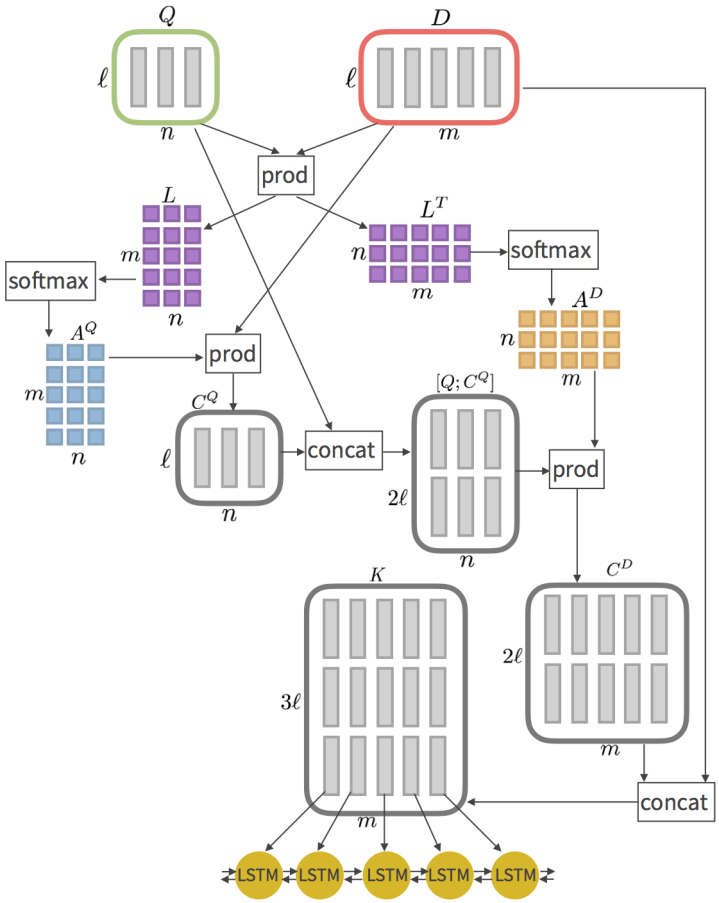


Figure 2: Architecture diagram of the coattention encoder.

Question: In what year was Nikola Tesla born?
Context paragraph: Nikola Tesla (10 July 1856- 7 January 1943) was a Serbian American inventor, electrical engineer, mechanical engineer, physicist, and futurist best known for his contributions to the design of the modern alternating current (AC) electricity supply system.
Answer: 1856

3.2 Data Preprocessing

We first tokenize all questions, documents, and answers into a vocabulary of 115,613 words. We then obtain GloVe word embeddings [5] trimmed to dimensionality $d = 100$ for each word in the vocabulary. These GloVe vectors were pretrained on Wikipedia 2014 and Gigaword 5. These word vectors are constant and are not updated during training.

In order for the model architecture to work, all questions and context documents must be the same length. We zero-pad all of the questions to length $n = 60$, which is the maximum length of any question in SQuAD. We truncate (or zero-pad) each document to length $m = 300$. The maximum length of a SQuAD context document is 766, but the vast majority of answer spans fall in the first 300 tokens of the document. Truncating the context document allows us to save computation and shrink the number of trainable parameters, while not sacrificing much accuracy.

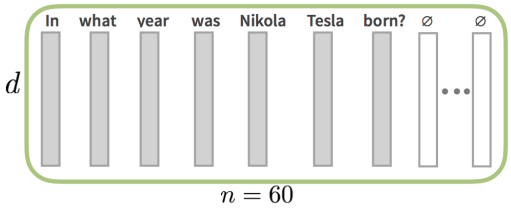


Figure 3: Visualization of a question after it its GloVe word embeddings are obtained, concatenated horizontally, and padded with zeros.

3.3 Implementation Details

We optimize our model with ADAM stochastic gradient descent [4], with a mini-batch size of 20. We train our model for 10 epochs, with an initial learning rate of 1e-3 which is annealed over time. To avoid exploding gradients, we clip our gradient norms at 10. We use dropout at a rate of 0.1 to regularize our network during training [8]. All LSTMs have a hidden state size of 200, randomly initialized parameters, and an initial state of zero. All models are implemented and trained with TensorFlow [1].

3.4 Evaluation Metrics

We use two metrics to evaluate our model's performance: ExactMatch (EM) and F1 score. Exact match measures the percentage of predictions that match one of the ground truth answers exactly, i.e., the predicted start token and predicted end token are exactly the ground truth values.

F1 score is a metric that loosely measures the average overlap between the prediction and ground truth answer. We treat the prediction and ground truth as bags of tokens, and compute their F1:

$$\text{F1} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{11}$$

where:

$$\text{precision} = \frac{\#\text{matching tokens}}{\#\text{predicted tokens}} \qquad \text{recall} = \frac{\#\text{matching tokens}}{\#\text{ground truth tokens}} \tag{12}$$

4 Results

4.1 Training the Model

The results of our models are shown in Table 1. Without dropout, the model clearly overfits to the training set. Adding a dropout rate of 0.1 increases the validation F1 and EM scores by nearly 10 points in both cases.

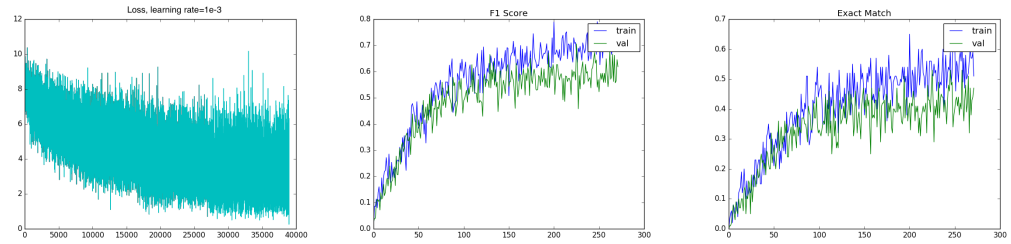
Model	Train F1	Train EM	Dev F1	Dev EM
Coattention w/ dropout	71.93	54.50	50.41	33.89
Coattention w/out dropout			**59.37**	**42.41**

Table 1: Results on the the train and validation set of the SQuAD dataset.

Performance Analysis

To better understand the model, we visualize the document attention weights (Figure 5) and the probability vectors outputted by the decoder (Figure 6).

In general, the attention weights activate on the parts of the document that are most relevant to different parts of the question. For example, in the third example, the attention weights are strongest at the words "church in" of the question, *The mosaic in the church of Thessaloniki is known as what?* and correspondingly at the answer in the document, "6th century Christ in majesty." Sometimes the



(a) Loss function (b) F1 score (c) EM score

Figure 4: Plotting the loss and the training/validation F1 score and EM score as the model learns. The loss function is plotted over 4 epochs with no regularization.The loss is noisy because the batch size is small, but the overall graph looks like the "hockey stick" function we hope to observe. The plots of both F1 score and EM are also exhibiting desired behavior, with the validation accuracy just lower than training accuracy. This means that the model is not overfitting to the training set. The plots for F1 and EM are noisy because a random batch of 100 samples of the training and validation data (not the same 100 samples from both sets) is selected for evaluation after each 100 steps.

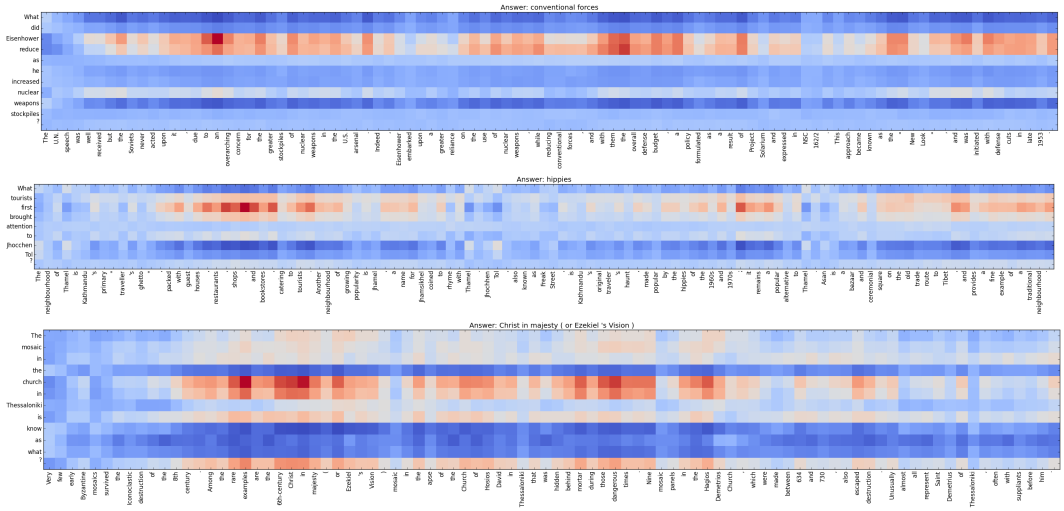


Figure 5: Visualization of the document attention weights A^D for three different examples.

attention weights only activate for a specific part of the question, as in the first example. Here, the model hones in on "Eisenhower" and "reduce" as the most important parts of the question, *What did Eisenhower reduce as he increased nuclear weapon stockpiles?*

We also notice from the probability heatmaps (Figure 6) that the probabilities are fairly diffuse; the model selects the correct answer span for these examples, but it does not place a high confidence on these indices. The probability distribution even appearing almost Gaussian in the first example. Allowing the model to train longer might increase its confidence in its predictions. We also hypothesize that incorporating a more complicated decoder might remedy this, such as a decoder that iteratively hones its predictions until arriving at the one with the highest confidence.

5 Conclusion

In this paper, we developed a model for the reading comprehension problem defined in the Stanford Question Answering (SQuAD) dataset. Our model consists of an encoder (which makes use of the coattention mechanism defined in [10]) that transforms the question and paragraph into a cohesive knowledge representation, and a decoder that uses this knowledge representation to output the an-

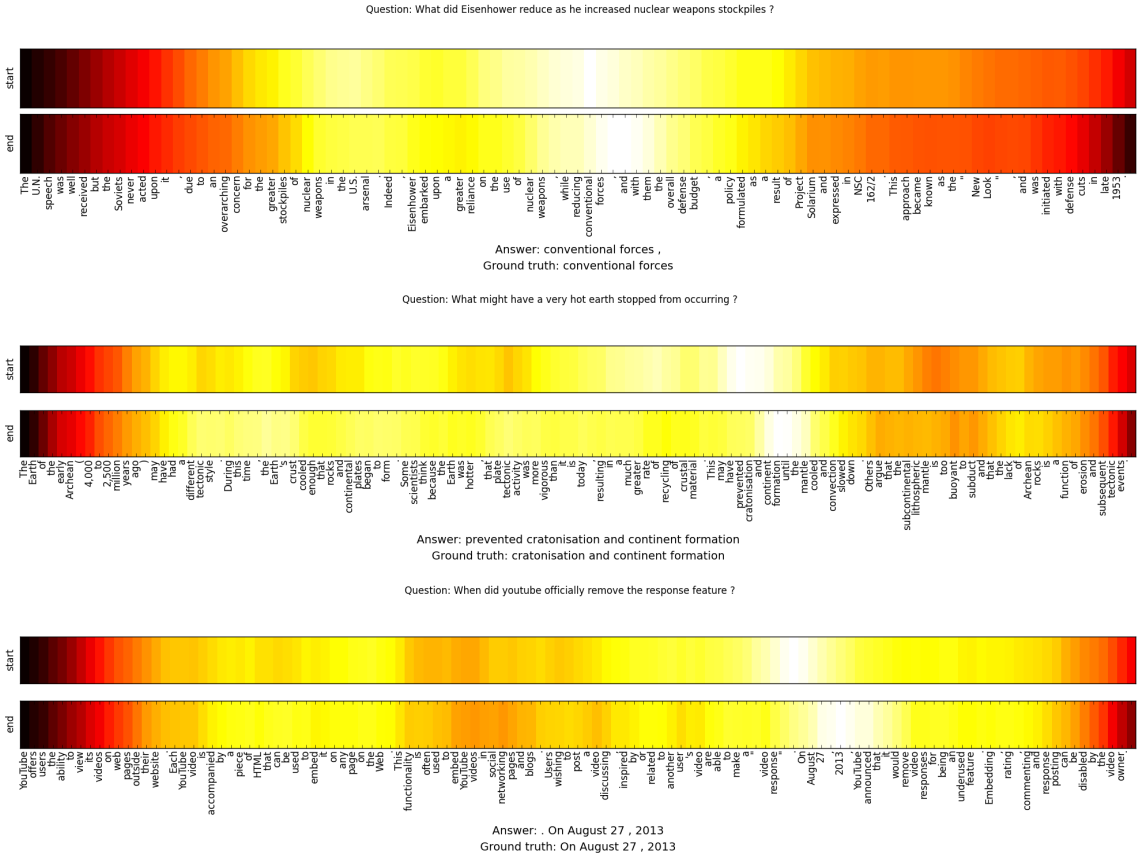


Figure 6: Visualization of the probability vectors for a_s and a_s returned by the model for three different examples.

swer span in the context paragraph. Experiments on the SQuAD dataset showed that our model obtained an exact match score of 42.4% and an F1 score of 59.37% on the dev dataset.

In the future, we plan to investigate more complicated decoders, such as the one proposed by [10], so that the start and end answers are not mapped into the same encoding space. We also plan to look into adding a simple constraint to ensure that $a_s \leq a_e$, which is a common problem that is unaddressed beyond merely flipping the values of a_s and a_e.

Acknowledgments

Thank you to the CS224n teaching staff for running a great course, and to Microsoft Azure for the GPU use.

References

[1] 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.

[2] Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. *CoRR*, abs/1606.02858, 2016.

[3] 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.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

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

[7] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[8] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

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

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