Machine Comprehension Using Multi-Perspective Context Matching and Co-Attention

Andrei Bajenov
abajenov@stanford.edu

Tarun Gupta
tarung@alumni.stanford.edu

Abstract

Several deep-learning models have been proposed for the Stanford Question Answering Dataset (SQuAD). We explore the effectiveness of different layers of these proposed models, and attempt to reproduce their results. Our most successful results stem from a combination of Multi-Perspective Matching models, as well as Coattention Networks. We build an ensemble of these models with different layer configurations and hyper-parameters to achieve a final **F1 / EM** scores of **69.075 / 57.957** on the test dataset.

1 Introduction

Question answering is a popular topic in NLP research. To aid with research in this area, Stanford developed the SQuAD dataset. It consists of over 100k triplets: (context paragraph, question, answer span within the context paragraph). The context paragraph is an arbitrary piece of text with an associated question. The answer to the question is selected as a span within the context paragraph.

The task given to NLP researchers is to predict the start and end indexes of the span. For each question, three potential answer spans are provided. Two measurements are used to evaulate the success of a model:

- F1 score which is based off of how many words intersect between the predicted span and the given spans
- Exact Match (EM) score is the percentage of answers that are predicted exactly.

1.1 Previous Work

Many deep-learning models have been proposed that attempt to make accurate predictions on the SQuAD dataset. We implement, evaluate, and explore the performance of a few of them:

- Multi-Perspective Context Matching [1]
- Dynamic Co-attention Networks [2]
- Match-LSTM with Answer-Pointer [4]

What we found is that all these models have four basic layers and seem to follow the simple formula of embed-encode-attend-predict:

- **Paragraph / Question Representation Layer** - This layer is usually just a set of single or bi-directional LSTM networks that take word embeddings of the paragraph and question and build hidden layer representations for each word. The output matrix contains rich contextual information about each word (has context from both left and right of word). The output is :

- a paragraph representation ($2h \times P$) that encodes contextual information about each word in the paragraph independent of the question
- a paragraph representation ($2h \times Q$) that encodes contextual information about each word in the question independent of the paragraph

- **Mixing Layer** - A layer that attempts to mix the hidden layers of the question and context representation. We try mixing via co-attention and matching to capture interaction between context and question words. Output is a paragraph representation $2l \times P$
- **Query-Aware Paragraph Representation Layer** - We consume the output of the mixing layer and feed to a BiLSTM to generate P hidden states. This layer generates a paragraph representation that encodes contextual information about each word conditioned on the question.
- **Prediction Layer** - a layer that takes the mixed hidden states, passes them through some other network to finally get probabilities for start and end indexes.

The three papers we explored have slightly different implementations and strategies for implementing these layers. We implement a few of them and evaluate their effectiveness.

2 Model Implementation Details

We tried implementing several models. Below we describe the different implementations for layers across the Multi-Perspective Context Matching Model [1] (Figure 1) and Co-attention [2] (Figure 2).

2.1 Multi-Perspective Context Matching

The first model we tried implementing is described in [1], and has all the layers we described in section 1.1. The overview of the implementation is shown in Figure 1. Below is a brief overview of some of the key layers:

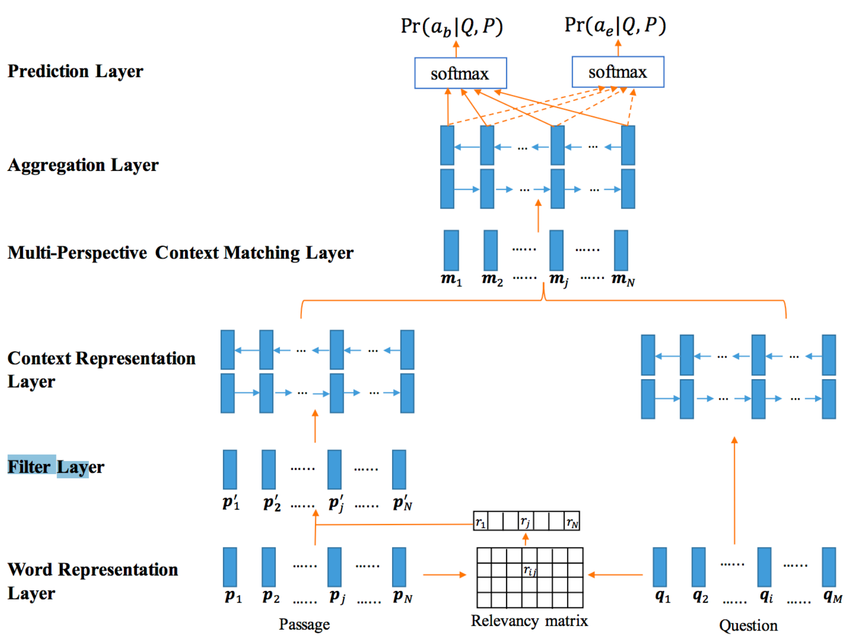


Figure 1: Architecture of Multi-Perspective Context Matching Model [1]

2.1.1 Paragraph / Question Embedding Layer

As described in section 1.1

2.1.2 Filter Layer

One key idea is to pass the paragraph vectors, pp through a filter that emphasizes the relevant words in the passage and removes the redundant parts. The steps in this process are described below:

- Calculate passage-word to question-word cosine similarity, $r_{ij} = cossim(pp_i, qq_j)$
- For each passage-word, define relevancy as $r_i = max_j(r_{ij})$
- Use relevancy to scale each passage-vector, $pp_i = r_i * pp_i$

2.1.3 Paragraph / Question Representation Layer

As described in section 1.1, pass paragraph-vectors and question-vectors through a BiLSTM to obtain forward and backward context representations for question and paragraph vectors.

2.1.4 Matching Layer

- We compute forward and backward matching matrices as $cosine(W_1 \circ pp_{fw}, W_2 \circ qq_{fw})$ and $cosine(W_2 \circ pp_{bw}, W_2 \circ qq_{bw})$
- We reduce these via mean pooling to compute P vectors containing question-aware representation for each paragraph word.
- We feed a fusion of these matching vectors to a BiLSTM to obtain P vectors, one for each passage-word

2.1.5 Softmax Prediction Layer

Consumes the $P \times l$ dimensional output, x of the BiLSTM to generate probability distributions over the P passage words. We assume that the probability distributions are generated independently. $ps = xW_{start} + b_{start}$ $pe = xW_{end} + b_{end}$

2.2 Dynamic Co-attention Networks

The next model we implemented also has similar layers we described in section 1.1, with slightly different implementations. The full description of the model can be found in [2]. Below is a brief overview of the key layers.

2.2.1 Embedding Layer

We map each word in paragraph and question to d-dimensional vector space via glove embeddings to generate $P \times d$ and $Q \times d$ matrices.

2.2.2 Paragraph / Question Representation Layer

As described previously in 1.1

2.2.3 Co-Attention Layer

In this layer, the goal is to mix the the vectors representations pp and qq to compute a heat-map like matrix for each example. This attention matrix helps us to localize the interesting parts of the paragraph and helps us identify the ”patch” corresponding to the answer (i.e. parts of passage that respond/interact heavily with the question).

Algorithm details are given below:

- Given the contextual representation for each word in paragraph and question, $pp : 2h \times P$, $qq : 2h \times Q$

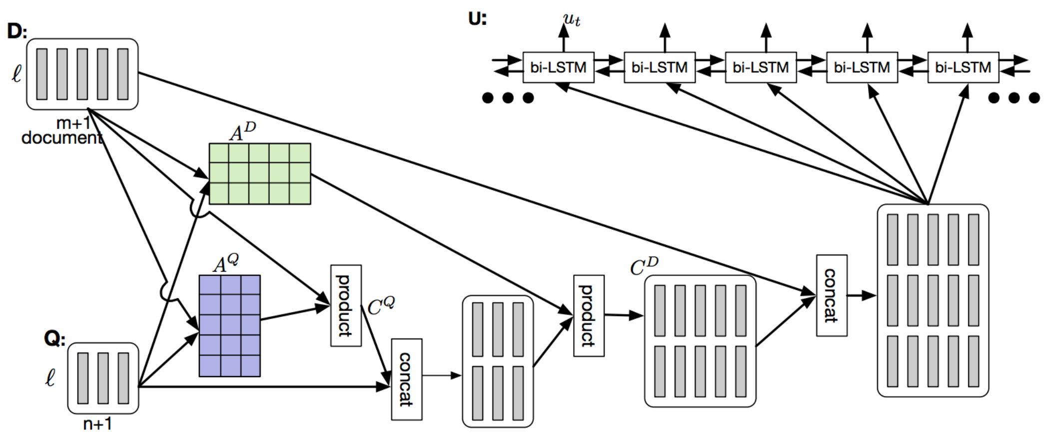


Figure 2: Architecture of Co-attention encoder [2]

- Produce affinity scores and generate context-to-query attention and query-to-context attention. $S = pp^T qq$ and $A^q = softmax(S)$ and $A^p = softmax(S^T)$ Produce an attention context vector $C^q = ppA^q$
- Map question encoding into the space of paragraph encodings. The concatenation represents the document encoding conditioned on question representation $C^p = [qq; C^q]A^p$: $2hk \times P$
- Finally, feed fusion pp_{att} to a BiLSTM to obtain P vectors corresponding to each word in the paragraph, where:

$$pp_{att} = [pp, C^p] \tag{1}$$

These vectors contain all information about the paragraph word, its context and question-to-paragraph interaction.

2.2.4 Softmax Layer

Consumes the $P \times l$ dimensional output, x of the BiLSTM to generate probability distributions over the P passage words. We assume that the probability distributions are generated independently. $ps = xW_{start} + b_{start}$ $pe = xW_{end} + b_{end}$

2.3 Match-LSTM with Answer-Pointer

The final model we implemented is described in [4]. It is very similar to the Multi-Perspective Context Matching model, with a slightly different implementation. Unfortunately, our implementation of the model turned out to be quite slow (5+ hours per epoch to train), so we opted to stick with the Multi-Perspective Context Matching model, which had a similar idea with a faster implementation.

2.4 Training

During training, we used a slightly modified version of a cross-entropy loss function. We penalized cases where the predicted answer span length was significantly different from the actual length.

We let y_e^i and y_s^i be the true end and start indices for the i-th example and ps^i and pe^i be the predicted start-index and end-index probability distributions over the P words in the passage. Then,

$$L(\theta) = -\frac{1}{N}[\sum_{i=1}^{N} \log ps^i[y_s^i] + \sum_{i=1}^{N} \log pe^i[y_e^i]] + \frac{1}{N} \sum \alpha(l_p^i - l_y^i)^2] \tag{2}$$

where the predicted span-length, $l_p^i = \arg\max_k(pe^i[k]) - \arg\max_k(ps^i[k]) + 1$ for $k \in [0, P]$ and the actual span-length $l_y^i = y_e^i - y_s^i + 1$

Therefore, we effectively enriched the vanilla cross-entropy loss with an L_2 penalty for incorrect predictions of answer-span length. In addition, this ”span-loss” term penalizes negative lengths and forces the end-index to be greater than the start-index.

In addition, in some experiments, we find that we can control some trade-off between F1 and EM using this parameter. Increasing this penalty, can lead to an increase in EM score while decreasing F1 score.

We used the Adam Optimizer to train our model with varying learning rates ranging from 0.01 to 0.0001.

2.5 Test

2.5.1 Prediction Strategy

During training, we tried a very naive prediction approach of predicting start-index and end-index independently i.e. $start = \arg\max_k ps[k]$ and $end = \arg\max_k pe[k]$. This is computationally cheap $O(P)$ and sits well with the well-known cross-entropy loss function.

However, for making predictions a more mathematically sound approach is to predict the span-tuple (start, end) jointly while enforcing $end > start$ i.e. $(start, end) = \arg\max_{(si,ei),ei>=si} ps[si]pe[ei]$. This is more expensive $O(P^2)$ but should lead to higher performance on dev-set.

2.5.2 Ensemble Strategy

The performance of each of our individual models wasn't as good as we'd hoped, so we decided to try an ensemble of the models, and got significantly better results.

Given M model-predictions for an example, we select a span s defined by (i, j) where $i < j$ is our predicted (start-index, end-index) tuple for that example, such that:

$$(i, j) = \arg\max_s \prod_{m=1}^{M} P^{(m)}[s|p, q] = \arg\max_{(i,j)} \prod_{m=1}^{M} ps^{(m)}[i]pe^{(m)}[j]$$

Hence, we effectively combine the learnings from multiple models by performing element-wise multiplications of the predicted probability vectors and choosing the span with maximum overall predicted probability. This approach beats the naive approach of independently making predictions for start and end indices for each model. Rather we make a span prediction and choose the span with the maximum probability across all predictions.

2.6 Gradient Clipping

Gradient Clipping is performed when the norm exceeds a threshold value. This has been provided to be an effective strategy for dealing with exploding gradients in recurrent networks.

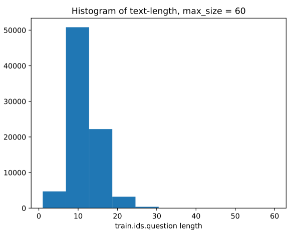
2.7 Model Regularization: Dropout

We experimented with dropout across various layers. We strictly enforced dropout on all LSTM layers.

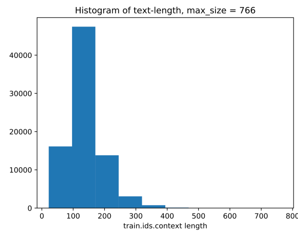
3 Experiments

3.1 Initial Settings

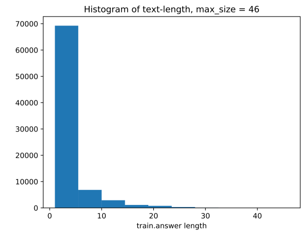
Since the amount of work done is proportional to the paragraph length, P, we identify an optimal value of P and truncate or pad all paragraph-sequences to that length. We find that $P = 400$ gives



(a) Histogram of question sequence-length, Q



(b) Histogram of context sequence-length, P



(c) Histogram of answer sequence-length

Figure 3

us good coverage (99.98% of examples have paragraph-length $< P$) across the training examples. For question text, we use a max-sequence length of $Q = 60$ that gives us 100% coverage across the training set. We allowed the word-embeddings to be learn-able and controlled over-fitting via dropout. We clipped gradients if they exploded above the threshold value of 10.

We start with size 100 for all embedding and hidden-state layers. We allowed trainable embeddings. All models are implemented in TensorFlow.

We found that starting with a a large learning rate to be an effective strategy for initial exploration of parameter-space followed by lowering the learning rate as the validation-loss starts to flatten out to explore effectively in the neighbourhood of true minima.

3.2 Some Ideas Explored

We tried several modifications to existing architectures. For example, we tried introducing a filter layer post-coattention. The goal was to emphasize the relevant words more rather than passing the full paragraph vector to down-stream layers.

- We experimented with a blend of filter layer within the co-attention layer i.e. replace pp in equation 1 with filtered passage $pp^{'}$ which emphasizes the important parts of passage (based on relevancy) and removes redundant parts. A visualization of this implementation is shown in figure 4

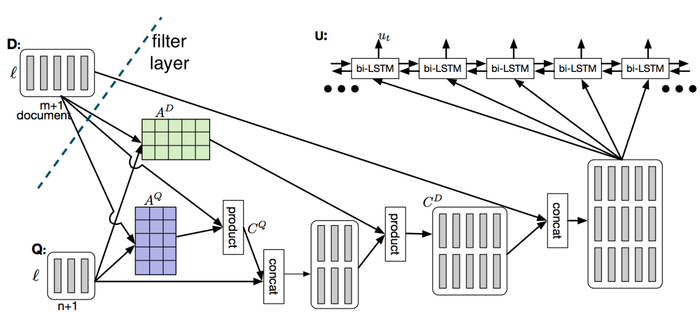


Figure 4: Architecture of Mixture Model encoder

- Architecture Ensemble: We experimented with an ensemble of two architectures. We used the logits from the constituent models and added them before applying a softmax layer to generate the final prediction. This network would then learn the parameters of all constituent models jointly.

Joint parameter-learning appeared computationally inefficient, after running a few iterations. Instead we found that learning the parameters for constituent models independently and then combining the predictions gives best results.

3.3 Hyperparameter Tuning

We experimented with various hyperparameters. Tuning is very hard for this problem given the number of hyperparameters. Table 1 lists several choices that were explored in our experiments. We loaded pre-trained glove embeddings and fine-tuned by allowing them to be "trainable" during the learning process. We experimented with the choice of adding dropout after the embedding layer. Intuitively, it seems reasonable to add dropout given the embeddings are "learn-able" and re-trained embeddings have a tendency to over-fit leading to loss in generalization.

Batch sizes we tried were 32, 64. We experienced memory issues for larger batch sizes.

Table 2 details some of the experiments that have been performed.

Table 1: List of Hyperparameters

Category	Hyperparameter
Embedding Layer	Trainable or fixed embeddings
	Dropout post-embedding layer
	Gigaword (6B) or Common Crawl (840B) corpus
Architecture Choices	Number of layers
	Type of Layers
Representation Sizes	embedding size (100-300)
	lstm units (100-200)
	perspective units
Regularization Choices	dropout ratio (0.2-0.4)
	span l1 regularization strength (0.001-0.0001)
Optimization Choices	optimizer
	max gradient norm (1-10)
	learning rate (0.01-0.0001)
	epochs run
	batch size (32/64)

Table 2: Hyperparameter Tuning Experiments

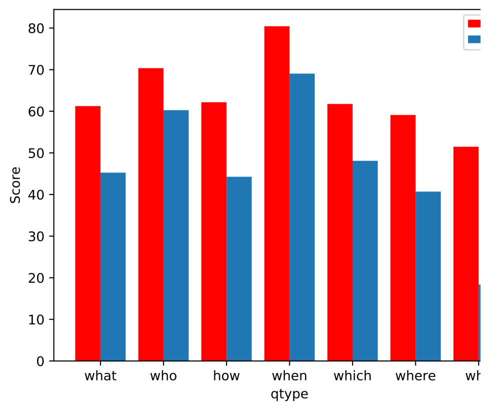
Model Name	Knobs Turned	Val F1	Val EM
MPCM	Initial Settings	56.21	43
MPCM-fixed	Fixed embeddings	49.3	36.4
MPCM-fixed-0.001	Fixed embeddings, span l2 increased to 0.001	50.82	37.38
MPCM-fixed-0.002	Fixed embeddings, span l2 increased to 0.002	50.37	37.64
COATT	Initial Settings	61.77	47.56
COATT-fixed	Fixed Embeddings	56.95	43.75
COATT-fixed-200	Fixed Embeddings, Increased layer sizes to 200	55	40.65
COATT-fixed-200-mix	Added filter layer	56.38	41.74

4 Results And Discussion

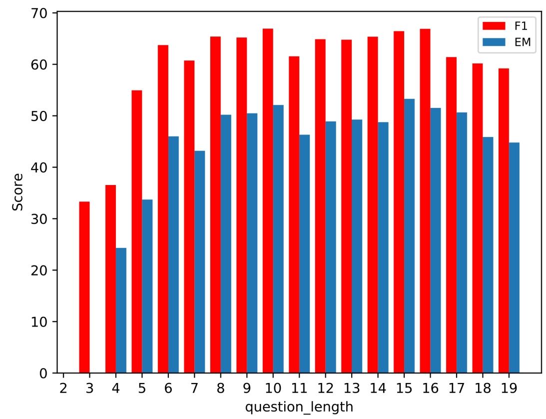
Looking deeper into the predictions, we saw some interesting things. Key observations from figures 5a, 5b, 5d and 5c:

- The model does well for simple questions like 'when'/'who' and worse as complexity increases 'why'/'how' type questions

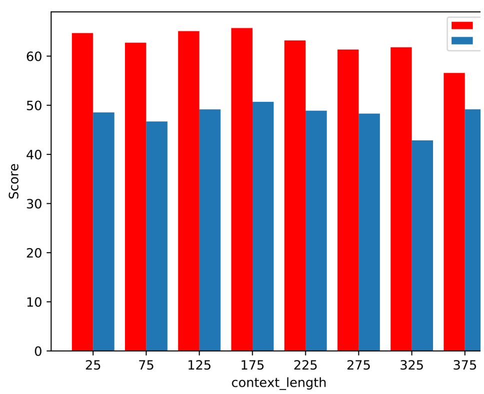
- The model seems to perform worse as the answer-length increases. Longer answers typically correspond to more complex questions.
- The model seems to perform surprisingly well across different question and context-paragraph lengths. We would expect performance to deteriorate for long questions and long documents.



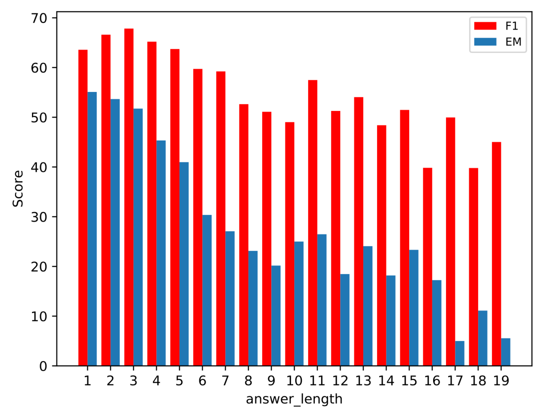
(a) Performance across question-types



(b) Performance across question-lengths



(c) Performance across context-lengths



(d) Performance across answer-lengths

Figure 5: Performance across different dimensions

4.1 Ablative Analysis

We tried to peel off layers from our architecures to do sensitivity analysis and identify the key components of the network. Table 3 shows the importance of the filter layer in the MPCM model.

Table 3: Layer Ablation

Model Name	Val F1	Val EM
MPCM	57.31	46.00
w/o filter layer	38.20	26.22

Table 4: Performance of models attempted so far

Model Name	Description	Val F1	Val EM	Dev F1	Dev EM
SQuAD LR Baseline	Logistic Regression w/ hand-crafted features			51.00	40.04
MPCM	Multi-perspective context matching	56.21	43.00	57.31	46.00
COATT	Co-attention network	61.16	46.97	62.45	51.96
MPCM + COATT Ensemble	Our first ensemble			64.21	53.15
Human	Crowd-sourced answers			91.221	82.304

4.2 Ensembling Results: Progress Path So Far

We identified several stand-alone weak learners through our tuning experiments and added them to the ensemble if they cleared a baseline performance threshold. The baseline threshold was set to F1 $> 55\%$.

We notice excellent performance gains by chaining the model predictions into an ensemble-prediction. The ensemble performance exceeds the performance of best stand-alone model by 5% F1. Table 5 details the various steps taken so far.

Our final ensemble achieves **F1 / EM** scores of **68.3 / 56.9** on dev-set and **69.075 / 57.957** on test-set.

Table 5: Progress of ensemble-models attempted so far

Ensemble Model Name	Description	Dev F1	Dev EM
MPCM + COATT	First ensemble, initial settings, see 3.1	64.2	53.2
MPCM + COATT	Tuned COATT	64.34	53.34
MPCM + COATT	Make start prediction, force $end > start$	65.72	54.26
MPCM + COATT + COATT-fixed	Added COATT(fixed embeddings, size = 100)	66.5	55.3
MPCM + COATT + COATT-fixed	Joint (start,end) prediction, see 2.5.1	66.71	55.4
MPCM + COATT + COATT-fixed + COATT-fixed-200	Added COATT(fixed embeddings, size = 200)	67.51	55.9
MPCM + COATT + COATT-fixed + COATT-fixed-200	Added filter layer to co-attention encoder	68.3	56.9

5 Conclusions

- We have explored and implemented ideas from several stat-of-art papers on question-answering.
- Our ensembling strategy yields a high performing model compared to any stand-alone model
- Our model doesn't get affected by question-length or paragraph-length
- Our model seems to be doing well for low and medium-complexity questions

6 Future Work

- Instead of predicting the start and end index independently we want to try out predicting (start-index, span-length) or (end-index — start-index). We decided not to try this because this may be computationally expensive.
- More effective ensembling strategy that effectively utilizes the local bumps in probability distributions across different models to make the overall span prediction. We could train a meta-model on the local maxima-minima.
- The current implementation of predicting the optimal span is $O(P^2)$. We want to optimize this step.

Acknowledgments

We would like to thank the instructors Prof. Chris Manning and Prof.Richard Socher for teaching an excellent course. We would also like to thank the TAs for helping out with questions on Piazza.

References

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

[2] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question-answering. *arXiv preprint arXiv:1611.01604*, 2016.

[3] Jeffrey Pennington, Richard Socher and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. *Proceedings of ACL*, pp. 1532-1543

[4] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905* , 2016.