Alpha and Omega? Gauging the influence of Answer-Pointer Frameworks in Question Answering Models

Stanford CS224N Default Project
Track: SQuAD

Nicholas Paul Brazeau Sanchez
Department of Computer Science
Stanford University
npbsanc@stanford.edu

Eden Grown-Haeberli
Department of Computer Science
Stanford University
edengh@stanford.edu

Abstract

Answer-Pointer RNNs have factored into successful machine comprehension models, especially those designed for the SQUaD challenge. However, whether or not these RNNs may be applied, generally, to boost model performance is unclear. In our project, we perform an ablation study to isolate the precise performance benefits of these Answer-Pointer RNN components in different models. We observe the improvements of an Answer-Pointer RNN on a baseline BiDAF model, a BiDAF model with character embeddings, and a BiDAF model with a self-attention encoding layer, after the R-net model (1). We introduce a novel output layer that combines succesful elements of the BiDAF and R-net models (2). The Answer-Pointer output contributed to improved performance in some models, but worse performance in others. We conclude that an Answer-Pointer RNN must be applied with careful regularization, and propose that they may be less effective in models that leverage gratuitous self attention.

1 Key Information to include

- Mentor: Fenglu Hong
- External Collaborators: None
- Sharing project: No

2 Introduction

Many successful SQUaD models produce outputs to question-context text pairs directly from contiguous tokens of the context. These models often select the first and last tokens of the context independently from attention scores calculated across context tokens, as in the case of the BiDAF model (2), but frequently, this end token is conditioned on the start token. Researchers Jiang and Wang pioneered the Answer-Pointer network to predict the end token based on the start token (3). The Answer-Pointer accepts attention across the context tokens (treated as "pointers" to possible answers, hence the name) as input to an Answer-Pointer RNN. This RNN's output is then used as input over a context attention calculation used to predict the end token. The idea behind the Answer-Pointer, that the beginning of an answer bears importantly on the end of an answer, is not only intuitive, but has found many emulators. The R-Net (1) and the DCN (4) models both implemented answer pointer derivatives after Jiang and Wang, improving upon the basic answer pointer. Yet while answer-pointers have been generally applied to SQUaD output, this may not mean that they *should* be generally applied. The fact that the SQUaD dataset is constrained to only 150 thousand examples (5) means that SQUaD nets may risk overfitting by implementing too deep or sophisticated of a model. Because

of this, some models may see improved performance from favoring simpler output layers in lieu of an answer-pointer based output, perhaps even models like the R-Net and the DCN!

As such, understanding the types of models that could most benefit from an Answer-Pointer becomes essential. Knowing how the Answer-Pointer architecture interacts with other SQUaD improvements will lead to time saved for future modelers, disambiguating whether or not their model's architecture will benefit from Answer-Pointer output, and contribute to a greater theoretical understanding of how, precisely, Answer-Pointer networks improve machine comprehension. To improve and iterate on Answer-Pointer output, we need to understand its strengths and weaknesses.

In this paper, we isolate the precise benefits of Answer-Pointer outputs within the contexts of different SQUaD models. We perform ablation testing on several models, adapting all of their output layers to make use of an Answer-Pointer RNN, and then qualitatively and quantitatively observe the changes that these RNNs induce in the models. We ablate against four models in this paper: A baseline BiDAF, a BiDAF with character embeddings, a BiDAF with character embeddings and self attention, and a BiDAF with character embeddings and coattention. We also attempt, as other papers have, to improve upon Jiang and Wang's answer pointer output, implementing a "chocolate" output layer that directly factors raw context to query and query to context attention in the output layer.

3 Related Work

There are already many SQUaD papers that implement an Answer-Pointer RNN, or a derivative, in their models. The Match-LSTM(3), DCN (4), and R-Net(1) all leverage output layers using an Answer-Pointer RNN. All of these Answer-Pointer RNNs are based on the Pointer Network developed by Google Brain (6).

As discussed, the Match-LSTM output layer uses the attention prediction for the start token as input to an RNN, with output used to predict end token distributions. DCN uses a dynamic pointing decoder, which not only conditions the end token's position on the start token's position, but predicts several start-end token pairs iteratively to guard against selecting spans in local maxima. The R-Net paper additionally conditions the position of the first start token on attention from the question encodings, as a way to feed additional information about the question into the answer pointer. We implement the R-net's approach in our rendition of answer pointer output.

All of these papers, however, lack ablation testing on their models against non Answer-Pointer based output networks. We thus lack data on whether or not these models' architectures were directly improved by the inclusion of Answer-Pointer output, which is a knowledge gap that our paper attempts to fill.

4 Approach

Our approach was to modify layers of a baseline BiDAF model piecemeal, producing improved models, and then to apply Answer-Pointer output to all models. This way, we obtained several models to perform quantitative and qualitative ablation testing against.

4.1 Character Embeddings

Character embeddings are a popular means to improve SQUaD models. Although character embeddings are not present in the given, baseline BiDAF model, the published BiDAF model makes use of them. The BiDAF paper finds them to improve the model's handling of out of vocabulary tokens which lack representation within word2vec embeddings, such as words from foreign languages, by training on world morphology (2). Thus, even when our word embeddings don't accurately capture the relationship between two words due to a lack of frequency in English, the model is able to use their morphology to make educated guesses about their relationships. We adopted the BiDAF paper's approach to learning character embeddings. We take a 2D tensor of raw character embeddings for the question c_r^Q and passage c_r^P, and obtained fixed character vectors of hidden size H for each word in the passage and context, c^Q and c^P. To obtain c^Q and c^P, We token characters in c_r^Q and c_r^P, respectively, through a 1D convolutional layer with a kernel width of 5, and then maxpool them. We did not use a nonlinearity in our convolutional layer, since it was only one layer, and in fact found that convolutional non-linearities hampered the model's overall performance. We obtain our final token

encodings for the question u^Q and passage u^P, from their word embeddings w^Q, w^P and character embeddings c^Q, c^P:

$$v^Q = \text{highway}([w^Q, c^Q])$$
$$u_t^Q = \text{BiLSTM}(u_{t-1}^Q, v_t^Q)$$
$$v^C = \text{highway}([w^C, c^C])$$
$$u_t^C = \text{BiLSTM}(u_{t-1}^C, v_t^C)$$

Where highway refers to a highway network of depth 2, as described in the SQUaD handout (7). The convolutional component of this layer was implemented with original code.

4.2 Vanilla and Chocolate Answer-Pointers

The most crucial module to implement was our Answer-Pointer output. We imitated the general approach found in the R-Net paper(1), using question-token attention as the zeroth state of our answer pointer RNN.

We begin with question encodings u^Q, along with a one-dimensional primer parameter V_P acting as a query to extract relevant information from the question. Our zeroth hidden state h_0 is calculated with additive attention as follows:

$$h_0 = v^T(\tanh(W_Q u^Q + W_P v_P)$$

From here, we obtain attention weights a_1 and a_2, as start and end token attention calculations, respectively, with the following formula:

$$a_n = v^T(\tanh(W_{mod} H^C + W_{point} h_{n-1})$$

Where H^C represents the context embeddings to be fed into the output layer; in the default BiDAF, H^C are encoded and attention aware. W_{mod} and W_{point} put output into some attention dimensionality A, and v^T takes these A lengthed vectors and projects them to 1 dimension. Obtaining h_1 from a_n is also simple:

$$n_0 = \sum_{i=1}^{N} a_{1i} H_i^C$$
$$h_1 = \text{RNN}(h_0, [n_0])$$

for all N tokens within the context. We used a basic RNN: since we were running it for only one timestep, the memory storage capabilities of an LSTM were unnecessary. We can think of an answer pointer as a form of self attention, since the end token factors attention over the start token distributions. We call this output layer our vanilla Answer Pointer, since it corresponds to the R-Net. Yet where the vanilla model uses the same projection matrices to compute the start and end tokens, the BiDAF output layer gives them unique projection layers, puts the modeling tokens through an RNN before calculating the end tokens, and moreover, factors the useful information of raw context to query and query to context attention to its final output. To emulate the BiDAF output layer, we introduce the Chocolate Answer-Pointer RNN, which combines the nuance of the BiDAF output layer and the conditioning of the R-Net output layer. It is entirely identical to the vanilla output layer, except for how we compute a_1 and a_2:

$$a_1 = v^T(\tanh(W_{att1} A + W_{mod1} H^C + W_{point} h_0)$$
$$a_2 = v^T(\tanh(W_{att2} A + W_{mod2} H^{C\prime} + W_{point} h_1)$$

Where A is the output of our attention layer in the model, and $H^{C\prime}$ is the modeling layer passed through an output BiLSTM, as in the BiDAF model. Notice that while the modeling and attention data have different projection layers, to capture the nuances in how to predict the start token as opposed to the end token, the answer pointer uses the same projector W_{point}; this is to guarantee that the Answer-Pointer RNN learns how to transmute attention from the start pointer distribution into data that can be used to predict the end pointer distribution.

The code we developed for this layer was implemented from scratch, however, we did briefly refer to an open source implementation of the R-Net to make certain that we were correctly inputting h_0 into the answer pointer RNN (8).

4.3 Self Attention

The R-Net paper introduces a self attention layer before its output (1). The idea is to afford the context encodings additional knowledge about context tokens that may be crucial to answering an input question through attention over the context. To implement this, we take modeling layer context M^C, and obtain self-attended model encodings N as follows:

$$N = (\text{Softmax}(M^C {M^C}^\top / \sqrt{E}) M^C)$$

where E is our hidden size. note that N is obtained through dot-product, instead of additive, attention, in contrast to the R-Net paper, and is done in our code with the multihead attention module from pytorch, initialized with a single head. We passed in our M^C as the query, key, and vector parameters to this module to perform self attention, at the recommendation towards somebody else of Reddit user Slashcom((9)).

We then concatenate M^C with N to obtain tensor $[M^C, N]$, and run this vector through an attention gate designed to scale-down irrelevant portions of the passage during encoding, another piece of the R-Net's self attention layer:

$$g = \text{Sigmoid}(W_{gate}[M^C, N])$$

$$[M^C, N]' = g \cdot [M^C, N]$$

$$H_t = \text{RNN}(H_{t-1}, [M^C, N]'_t)$$

. We then pass self-attention encoded context tokens H to our output layer. Besides the use of the multihead attention module, we developed this code from scratch.

4.4 Coattention

The DCN introduces coattention to the SQUaD model. Similarly to the BiDAF model, it attempts to capture interaction between the Question and the Context through Context To Question (C2Q) and Question to Context (Q2C) attention (2). Unlike BiDAF, however, Coattention derives its context to question attention from further attention over the context (4). First we compute an affinity matrix, as in the C2Q and Q2C attention layer in the BiDAF, such that for Context hidden states C and Question hidden states Q:

$$L = C^\top Q$$

From this affinity matrix, we can take the softmax row-wise to extract question attention weights A^Q, and column-wise to extract attention question weights A^C.

$$A^C = \text{softmax}(L, axis = 1)$$
$$A^Q = \text{softmax}(L, axis = 0)$$

Using these attention weights, we next attend to the context paragraph and the question such that

$$C^D = [Q; CA^Q]A^C$$

Then C^D is concatenated to the context paragraph embeddings and passed into the output layer. Notice that coattention adds *another* layer of attention to our attention layer: the Context attends to the Question, and then this Question-attended context is attended by the Context attention. We do not, however, add learnable sentinel parameters to our Q and C before processing them, as the Coattention paper does, meaning that our layer loses some ability to attend to the null values that these sentinel vectors represent; the base model already contains OOV tokens in the context, representing a Null answer, so having our model attend to these produces a similar dynamic.
We implemented coattention from scratch.

4.5 Models

For this paper, we produced Eight Primary Models: A baseline BiDAF, A, derived from the CS224n IID SQUaD starter project (7), a BiDAF model with character embeddings, B, a BiDAF model with character embeddings and self attention C, and a Coattention model with character embeddings, D. These models used a standard, projection based output layer as exists in the default BiDAF model. We created four additional models, A', B', C' and D' which used a chocolate Answer Pointer Output. For a given model X', X acted as its baseline, since our goal was to isolate the effects of answer pointer output. We also produced Models A'' and C'', which had a Vanilla Answer-Pointer output, to compare the performance of the Vanilla Answer-Pointer with the chocolate Answer-Pointer. We used A' as the baseline for A'', and C' as the baseline for C''. Moroever, we compared improvements in B', C' and D' to the improvements in A'. Finally, we produced model E, which was a baseline BiDAF with added self-attention, to demonstrate that the addition of self attention improved upon the baseline BiDAF model; A was this model's baseline. Unless specified otherwise, each of these models used the default final project code given by the CS224N teaching staff.

5 Experiments

5.1 Data

We used the SQUaD 2.0 dataset to train our model((5)). It is composed of question and context pairs, and a machine comprehension algorithm is used to refer to the context to produce an answer to the input question. Unlike SQUaD 1.0, many of the inputted questions lack answers that can be parsed from context, meaning that a model must learn to abstain from answering, yielding output N/A, as much as how to answer. Instead of using the SQUaD provided test set, however, our Dev and Test sets were largely comprised of the default SQUaD dev set, randomly and evenly split.

5.2 Evaluation method

Quantitatively, we were primarily interested in the maximum EM scores a given model achieved on the Dev set throughout its training. EM, or exact match, represents whether or not a model provided answer exactly matches one of the SQUaD provided ground truth answers ((5)). We also observed the F1 score of models, which represents the rough correspondance of a model produced answer with a ground truth answer. We observed the F1 scores of our models when their EM scores were maximized. We also monitored the models' NLL on the dev set, as a model with lower dev NLL typically had better EM and F1 scores.

Qualitatively, we were primarily interested in finding questions where one model output a correct answer, but another model output an incorrect answer. Observing these discrepancies gives insight as to what qualitative power and weakness one model's improvement gives over another.

5.3 Experimental details

All models had constant learning rate of 0.5, using the Adadelta optimizer, with exponential decay rates of 0.999, as in the base BiDAF paper. We applied dropout of 0.2 after every RNN and before passing the embeddings into our model. In other words, in our final iteration, we departed very little from the default hyperparameters in the base BiDAF model.

This isn't to say that we didn't experiment with hyperparameters, however. We tried cutting the learning rate in half every million epochs, since we noticed that dev metrics improvements slowed down at increments of million epochs. on more sophisticated models (B, C, and D), we also noticed a dramatic decrease in dev accuracy past 1 million iterations in comparison to the baseline, suggesting that our models were overfitting. Thus, we reduced their hidden sizes to 80. Neither of these modifications yielded improvement over default hyperparameters. We theorize that since our models still tended to improve towards 2 or 3 million iterations in training, cutting down their learning rate retarded the learning during this period, causing our model to become stuck in local minima.

All models were run for at least 25 epochs, and ideally 30. In models C, C' and E, we reduced the depth of the modeling layer by 1 to improve performance; using a 2-layer model led to decreased performance on the dev set, likely due to overfitting.

5.4 Results

Figures:

Model	EM	F1
Model A	57.13	60.67
Model A'	58.34	61.82
ΔA	**+1.21**	**+1.15**
Model B	61.44	64.72
Model B'	61.07	64.31
ΔB	**-0.34**	**-0.41**
Model C	59.77	63.59
Model C'	62.06	65.33
ΔC	**+2.29**	**+1.74**
Model D	59.94	63.4
Model D'	59.23	63.06
ΔD	**-0.71**	**-.34**

Model	EM	F1
Model A'	58.34	60.67
Model A"	55.82	59.07
Δ	**-2.52**	**-1.67**
Model C'	62.06	65.33
Model C"	61.62	64.84
Δ	**-0.44**	**-0.49**

Model	EM	F1
Model A	57.13	60.67
Model E	59.18	62.24
Δ	**+2.05**	**+1.57**

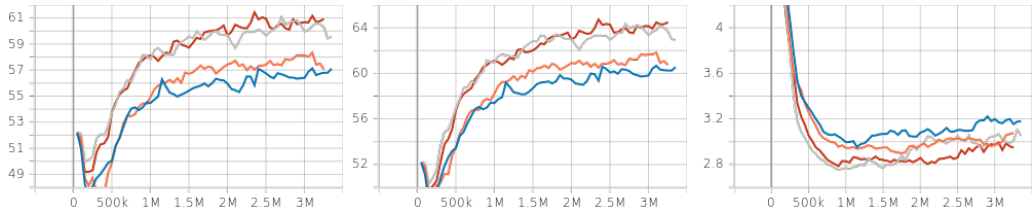

Figure 1: dev set EM, F1, and NLL, in order
A: Blue, A': Orange, B: Red, B': Grey

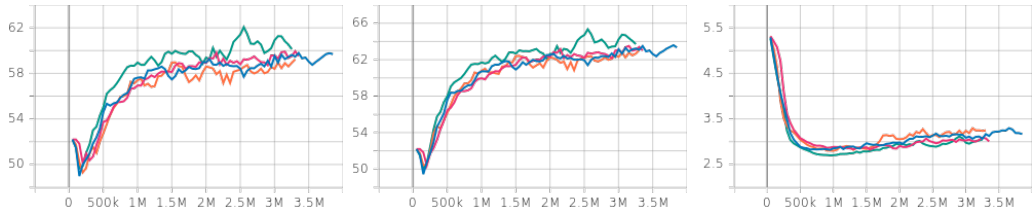

Figure 2: dev set EM, F1, and NLL, in order
Blue: C, Green: C', Pink: D, Orange: D'

SQUaD Test leaderboard results: **EM: 61.927, F1: 65.560**

Since it got the best performance on our Dev set, we submitted model C' to the IID SQUaD test leaderboard.

After performing ablation testing on all of the models, we have identified that the model which performed the best was the C' model with chocolate Answer-Pointer output, character embeddings and self attention. While models B, C and D did improve appreciably over A, we did not find any guarantee that an Answer-Pointer improved these models generally. To conclude this, we compared the baseline A' achieves over A, of about +1 in EM and F1 scores, to the improvement other prime models provide over their baselines. Model C' outdid this improvement by a factor of around +1. This is consistent with our suspicions, going into the project, that Answer-Pointers are not an improvement that can universally improve models. We are thus satisfied with our approach: we found standard models that improved over A, and were able to observe the effects that an Answer-Pointer had models A through D.

6 Analysis

After a quantitative examination of the results, we claim that character embeddings and the chocolate Answer-Pointer improvements made the biggest difference to the model. Although the chocolate answer pointer made a big improvement over the BiDAF baseline, the vanilla Answer-Pointer output actually made the model perform worse; hence, we tried to implement models with a chocolate Answer-Pointer, as opposed to a vanilla Answer-Pointer, as much as possible during testing. By comparing the prime model's performance to the original model's performance, we can use the results of our ablation testing to examine how the Answer-Pointer layer interacts with the other model improvements.

In order to assess how, qualitatively, the answer pointer improves a model, we looked at how a prime model responded differently to a base model in examples across the dev set. The prime models were better at encapsulating articles, like "the" or "and" than baseline models. This is a natural extension of a larger pattern, namely that the Answer-Pointer models tend to select more verbose answers than their BiDAF output counterparts, and were also more biased against giving N/A as an answer than their base models. Take the following dev example, and responses from model C and C':

- **Question:** What type of group is The Islamic State?
- **Context:** "The islamic State"...is a Wahhabi/Salafi jihadist extremist militant group...
- **Answer:** Wahhabi/Salafi jihadist extremist militant
- **C Prediction:** extremist militant
- **C' Prediction:** Wahhabi/Salafi jihadist extremist militant

We see here that the prime model was not only more verbose in its answer, but because of this, it produced a correct answer, whereas the default model produced a shorter, but incorrect version of the answer. However, this verbosity could be a curse, as well, and the outputs in Answer-Pointer models tended to be very sensitive to irregular punctuation schemas:

- **Question:** What is the Chinese name for the Yuan Dynasty?
- **Context:** The Yuan dynasty (... pinyin: Yuán Cháo), officially the Great Yuan...
- **Answer:** Yuán Cháo
- **C Prediction:** Great Yuan
- **C' Prediction:** Yuán Cháo), officially the Great Yuan

We see here that C' was able to extract the correct answer from the question, where C was not. It included, however, a lot of extraneous text, including punctuation, which we conclude to be due to the presence of a colon, parenthesis, and comma around Yuán Cháo.

We propose an elegant explanation for this behavior. Our model accepts attention based categorical distributions over start and end tokens as its output, taking the argmax of these to be the start and end tokens. Because the end token is conditioned on the start token in Answer-Pointer output, our model can learn that a very concentrated attention at the start token suggests that the passage likely has an answer. The Answer-Pointer is able to make use of this attention knowledge to predict less likely tokens from the context, likely causing end token attention distributions to become more concentrated as well. Yet this means that the OOV token, which is used to help predict N/A, will have less attention weight, by comparison, in the end token layer, which means that our model is less likely to predict N/A in general.

Character-level embeddings alone significant improved the BiDAF baseline. This was expected, since the BiDAF model leverages character embeddings. Since these embedding learn morphology rather than semantics, we expected adding this layer to improve the performance of the model, and it did. However, the failure of the B' model to improve on B was worthing noting. B Prime's metrics were slightly worse than B's. The answer pointer layer utilizes a probability distribution based on the attention scores to select an start and end tokens such that the end token is conditionally more likely when combined with the start token. This improvement should be orthogonal to the morphology level benefit provided by character embeddings, yet it detracted from B'. We conclude that information from the morphological information from character-level embeddings must also be inferred indirectly by the answer pointer output.

With respect to model C', self-attention, character-level embeddings and a chocolate Answer-Pointer output layer working together produce the biggest improvement to the model besides raw character embeddings. C' also has the highest positive gradient in metrics over C of any of the model pairs, providing evidence that under this architecture, the Answer-Pointer provides the most benefit. The Answer-Pointer layer and the self-attention modeling layer both rely on self-attention over context tokens. This dependency allows the self-attention layer and the Answer-Pointer layer to learn distinct facets of the context from the same information, resulting in layer synergy. It is worth noting, however, that models B and C' performed quite similarly in terms of metrics, achieving EM and F1 scores within 1 of each other, and that B' was only slightly worse than B. Thus, C Prime's improvement over C may be anomalous. After examination of the question/answer pairs which differed from the true answers, it is clear that the Answer-Pointer model does a good job of understanding text when the question text is directly in the context paragraph or the wording is close to the input question. However, if the question incorporates a word or abbreviation that the model has not seen, even the models which incorporate character embeddings have a difficult time recovering.

Although model D showed an increase in performance over the baseline, adding the Answer-Pointer layer slightly decreased the model's performance. However, qualitative examination of the model showed that the D' model was actually able to identify answers that the other models - including the Answer-Pointer output models - struggled with. For example, the question in Figure 2. was answered by all of the models as some version of "the object's weight", but D' was able to extract the correct interpretation of the question.

- **Question:** What equals the spring reaction force on an object suspended on a spring reaction scale?
- **Answer:** Gravity
- **C' Prediction:** equals the object's weight
- **D' Prediction:** gravity acting

This implies that combining an answer pointer based output and a coattention provides the model greater power in parsing indirect answers from the context.

7 Conclusion

We performed an ablation study on Answer-Pointer output with other common SQUaD improvements. After a comprehensive set of test on a myriad of models largely built from scratch, which implemented a character-level embedding layer, self-attention, and co-attention, we conclude that while our chocolate Answer-Pointer factored into our model with the highest merics, it provided inconsistent benefit, especially to more sophisticated models: in models B' and D', the inclusion of a chocolate answer pointer slightly hampered their performance over the baseline.

We uncovered how Answer-Pointer output layers qualitatively change SQUaD models as well, hypothesizing that these output layers are able to use start token distributions with concentrated attention to predict end token distributions with concentrated attention, leading to answer predictions that are longer and less inclined to predict N/A. Thus, we propose that aggressively regularizing the answer pointer RNN, perhaps applying dropout on the start token inputs, is crucial in combating this weakness.

Although we have identified which models do not interact well with the Answer-Pointer layer, we have no direct evidence to support why these specific models were not helped by this layer. We are inclined to conject that more sophisticated models, such as B, C, and D already capture self-attention information that the answer pointer could provide in predicting the output, leading to its effects being dampened in B', C', and D'. Yet the large improvement of C' over C prevents us from making this conclusion. If we'd had more time, we'd have liked to see if C consistently underperforms C' with different modeling seeds.

Ablation testing has shown that although Answer-Pointer layers are not suited to all types of models, the BiDAF model with character-level embeddings and self-attention benefits from the addition of this probability based layer. Using this combination of improvements allows the model to learn the question answering task much more accurately.

References

[1] N. L. C. Group, "R-net: Machine reading comprehension with self-matching networks," May 2017. [Online]. Available: https://www.microsoft.com/en-us/research/publication/mcr/

[2] M. J. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," *CoRR*, vol. abs/1611.01603, 2016. [Online]. Available: http://arxiv.org/abs/1611.01603

[3] S. Wang and J. Jiang, "Machine comprehension using match-lstm and answer pointer," *CoRR*, vol. abs/1608.07905, 2016. [Online]. Available: http://arxiv.org/abs/1608.07905

[4] C. Xiong, V. Zhong, and R. Socher, "Dynamic coattention networks for question answering," 2018.

[5] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," *CoRR*, vol. abs/1606.05250, 2016. [Online]. Available: http://arxiv.org/abs/1606.05250

[6] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NIPS*, 2015, pp. 2692–2700. [Online]. Available: https://arxiv.org/pdf/1506.03134.pdf

[7] C. staff. Cs 224n default final project: Building a qa system (iid squad track). [Online]. Available: http://web.stanford.edu/class/cs224n/project/default-final-project-handout-squad-track.pdf

[8] A. Petrushko. R-net-pytorch/outputlayer.py at a6ed4a02b0cf68bade9e9a43a93ec290a3b6fabd · tailerr/r-net-pytorch. [Online]. Available: https://github.com/tailerr/R-NET-pytorch/blob/a6ed4a02b0cf68bade9e9a43a93ec290a3b6fabd/source/nn/modules/output_layer.py

[9] Slashcom, "R/pytorch - pytorch multi-head attention module." [Online]. Available: https://reddit.com/r/pytorch/comments/c2u6g5/pytorch_multihead_attention_module