Understanding Attention for Question Answering models

Stanford CS224N {Default-IID Squad} Project

Akshatha Kamath
School of Education
Stanford University
kamathak@stanford.edu

Abstract

In this work, we explore and compare the performances of non Pre-trained Contextual Embeddings (PCE) based approaches - Bidirectional Attention Flow (BiDAF), Dynamic Coattention Network (DCN), and FusionNet and also consider ensemble based approaches of FusionNet to get an idea of attention focused models' effectiveness on the SQuAD dataset[1]. We aim to reproduce the papers by implementing the network architectures using the PyTorch[2] framework. Reproducing the DCN paper, we obtain an F1 score of 49.67% on the Stanford Question Answering Dataset (SQuAD2.0). We obtain an F1 score of 64.49% using the FusionNet single model.

1 Key Information to include

- Mentor: Kamil Ali
- External Collaborators (if you have any): N.A.
- Sharing project: N.A.

2 Introduction

Machine Comprehension involves a short paragraph (context) and a question (query) with a goal to output the location of the answer within the given text. There have been rapid improvements in performance in QA and the complexity of the problem space has been constantly evolving. question. The problem is considered AI-complete and requires both natural language understanding[3], as well as general world knowledge[1].

One can categorize QA models into two main classes: (1) pre-trained contextual embedding (PCE) models, or (2) advanced encoding models (non PCE models). PCE models have been highly successful at the question answering task [4, 5, 6, 7] but require a large number of parameters and correspondingly, the associated computation costs. Non-PCE based methods are known to perform poorly on the SQuAD2.0 dataset, which contains over 50,000 adversarial questions that could be answered.

Xiong and Zhong [8] proposed a "coattention model" to give attention to both the question and the context at the same time, building a codependent representation of them simultaneously which they use to predict the starting and ending point of the answer within the context. Second, they introduced a dynamic point decoder, which iteratively moves through the context to determine which start and endpoints in the context satisfy the question best. We determined that the dynamic coattention model would provide a much more robust and accurate question answering model upon which we could experiment with different adjustments to attempt to optimize for speed of training.

FusionNet was influenced by the Dynamic coattention networks. It is a non-PCE reading comprehension model built on top of DrQA [9], a simple model encoding with RNN features like pre-trained

word vectors, term frequencies, part-of-speech tags, name entity relations, and whether a context word is in the question or not, and predicting the start and end of an answer with a PointNet-like module [10]. PointNet is used to learn a global, vectorized representation of the query sentence, followed by a convolution over the context word embeddings to learn a representation of each word within its local context, i.e. which context words to focus on by treating the query as a single vector and checking "all at once" how a context word is similar to the query representation.

3 Related Work

3.1 Bi-Directional Attention Flow(BiDAF)

The BiDAF network is a multi-stage hierarchical process that represents the context at different levels of granularity utilizes and combines Context-to-Question (C2Q) attention and Question-to-Context (Q2C). BiDAF, alongside its attention layer, implements character embedding, word embedding, encoder, modeling, and output layers.

Before describing the full model, we briefly introduce the different modules we have used to build the Dynamic Coattention networks for question answering.

3.2 Long Short Term Memory (LSTM) Networks

Hochreiter and Schmidhuber [11] introduced a recurrent neural network(RNN) architecture to deal with the vanishing gradient problem of the traditional RNN, particularly in cases where the co-occurence of two events is important to the problem despite the two events being far apart (in language sentences or time series). A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

3.3 Co-attention

Coattention was first used in the neural translation model [12] where they present an encoder-decoder architecture. The decoder in this architecture weighs the encoding components based on an 'alignment score' between the hidden decoder state and the encoding component. Attention thus helps the decoder to attend to the most relevant parts of the input.

3.4 Pointer Networks

[10] introduced the concept of pointer networks where discrete token positions of an input sequence are used to learn the conditional probability of an output sequence. Pointer networks solve the problem of variable sized output dictionaries by using neural attention.

3.5 Highway Networks

For our DCN implementation network we will be using the highway max-out network. The concept of 'highways', introduced by [13] allows network layers to interpolate between the standard perceptron layers and the identity layers. They show evidence that the highway networks are easier to optimize than plain networks, converge faster, and have more robustness to weight initialization.

3.6 Maxout Networks

The activations in Maxout networks[14] are parameterized piecewise linear convex functions that are themselves learned. The DCN+ paper [15] uses maxout nodes to improve the question answering task performance by aggregating predictions from multiple models, and enhance the model averaging properties of dropout training.

4 Approach

4.1 Dynamic Coattention Networks

Figure 1 is the diagramatic representation of the implemented appraoch. In this approach, DCN comprises of a Co-attention Encoder and a Dynamic Pointer Decoder. The first part of this comprises of an **LSTM encoder** where the module encodes the contexts and questions using the same LSTM. LSTM. Further, the question encoding is passed through an extra layer (linear transformation followed by tanh non-linearity) to allow for a difference between context and question encoding spaces. Then, before passing it to the coattention module, the two encodings append one learned sentinel at the end to give the model the choice to not give attention to any specific word.

Co-attention encoder fuses the question and the passage together to generate a question-oriented passage representation . The idea of coattention is that the question and context encodings attend to each other, and thus are multiplied to arrive at an affinity matrix L. We then get the column-wise and row-wise softmaxes of L, namely the attention weight matrices A^D and A^Q. We then obtain a summary C^Q of the context corresponding to each word of the question as DA^Q. We then concatenate Q and C^Q and multiply with A^D to arrive at the coattention context. This context is word order insensitive as it builds upon LSTM encodings that are order sensitive. In the final step of this module, we use a bidirectional LSTM to combine the word order with the coattention context, to produce coattention encoding U.

The coattention encoding generated in the previous step is fed to the **Dynamic Pointer Decoder** [16][17]. The goal of this is to iterate across many different potential output points and choose the best one. The Dyanamic Pointer Decoder is composed of a Highway Maxout network . The Decoder outputs two values which are the start index and the end index of the answer it has predicted in the passage. This decoder works iteratively by producing a proposed answer (two pointers indicating the start and end indices, aka, a substring of the context). We then use two highway maxout networks to decode the answer predictions from the decoder LSTM's hidden state. At each iteration, each network takes the non-linear projection of the coattention encodings of the estimated start and end tokens (initially $(0, 0)$), the coattention encoding U itself, and the current decoder LSTM's hidden state as input. It then produces m+1 scores over every context word as a prediction of start or end indices of the answer. The LSTM then computes a new hidden state to be used in the next iteration. The highest scoring start and end indices generated by the networks are used as the inputs to these highway maxout networks for the next iteration. Note that while the start pointer network has the same architecture as the end pointer network, they share no parameters. Finally, we also try to implement another improvement made in literature by merging the coattention outputs of the current layer with the residual outputs of the previous layer.

The Loss : Our loss is the sum of the two losses for the predicted start and end indices of the answer. As in the original paper, we use a softmax cross entropy loss on the entire output score weights for each subloss produced by each HNM, where the labels are a one-hot encoding of the actual positions for the start and end of an answer span. The loss is averaged over each batch.

4.2 FusionNet

As mentioned in FusionNet[18], the above methods don't fully capture the full information in the context or the question, which could be vital for complete information comprehension. Thus, in the paper, they conjecture an attention scoring function utilizing all layers of representation with less training burden. This leads to an attention that thoroughly captures the complete information between the question and the context. With this fully-aware attention, the paper then puts forward a multi-level attention mechanism to understand the information in the question, and exploit it layer by layer on the context side. The key components of this model are high level fusion, word-level fusion, alternative high level fusion, self-boosted fusion and alternative self-boosted fusion as shown in figure **??**.

We explain an overview of the architecture here, but recommend looking at the original paper for detailed mathematical equations. First we transform every word in the context and questions to an

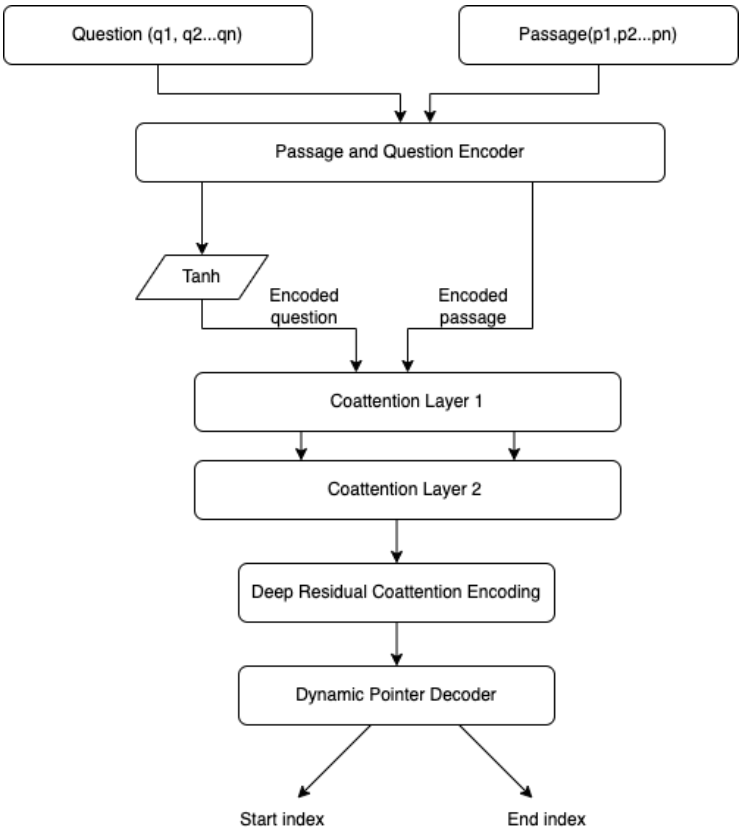

Figure 1: Dynamic Coattention network module

input vector using the 300 dimensional GloVe embeddings, a 600-dim contextualized vector[19], a 12-dim part of speech (POS) embedding[9], 8-dim named entity recognition(NER) embedding and a normalized frequency term[9].

Fully-Aware Multi-level Fusion: word level : We feed the GloVe contexualized embeddings to a standard two layer bidriectional LSTM (MT-LSTM). The attention-based fusion is presented below: $g_i^C = \sum_j \alpha_{ij} g_j^Q, \alpha_{ij} \propto exp(S(g_i^C, g_j^Q)), S(x,y) = ReLU(W_x)^T ReLU(W_y)$.

Reading: We use another bidirectional LSTM to form both, high level (h^{Ch}, h^{Qh}) and low level (h^{Cl}, h^{Ql}) concepts for the context and the question. For **question understanding** we apply a different bidirectional LSTM that takes in h^{Ql} and h^{Qh} to arrive at a final understanding vector for the question U_q.

Fully-aware Multi-level Fusion: Higher-level: For every h_i^A, we do the following when we fuse body B to body A. First, we compute an attention score S_{ij} with history of words HoW_i^A and HoW_j^B. We then form attention weights by computing the softmax of the attention score, and then concatenate $h_i^A$with the summarized information. A new BiLSTM is applied to obtain the representation for C fully fused with information in the question Q.

Fully-Aware Self-Boosted Fusion: Self-Boosted Fusion uses fully-aware attention on the history-of-word to consider distant parts in the context, as follows:
$HoW_i^C = [g_i^C; c_i^C; h_i^{Cl}; h_i^{Ch}; h_i^{Cl}; h_i^{Ch}; u_i^C; v_i^C]$.
The final context representation U_c is then obtained by applying a bidirectional LSTM to a concatenation of v^C and the fully aware attention v_1^C.
$U_c = u_1^C, ..., u_m^C = BiLSTM([v_1^C; v_1^(\wedge C)], ..., [v_m^C; v_m^(\wedge C)])$

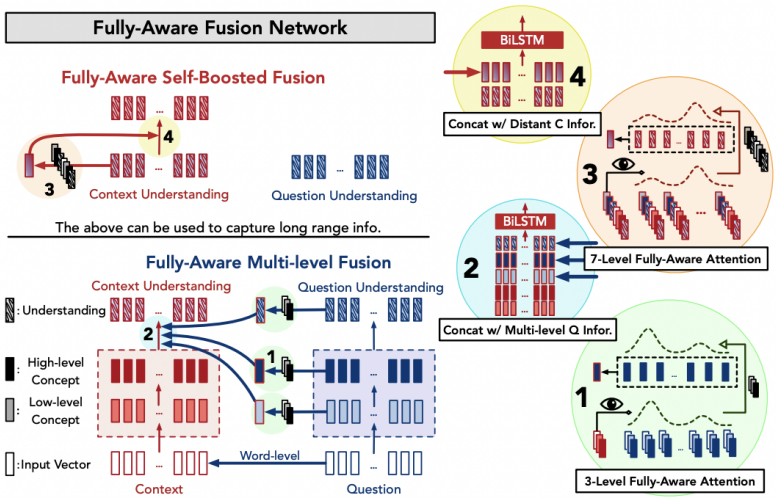

Figure 2: FusionNet architecture (taken from the original paper)

5 Experiments

Our experiments mainly just consisted of training our model and seeing how it performed on unseen dev sets of data; we trained 3 different models after our various adjustments described in our approach to the model and saw their performance on the F1, EM and AvNA metrics provided by Tensorboard. We use the following code bases[1][2][3] as reference for our implementation.

5.1 Data

We use the SQuAD 2.0 dataset provided for the course project with the following three splits: train (129941 examples), dev(6078 examples) and test(5915 examples). Note that these are different from the three splits in the official dataset.

5.2 Evaluation method

Our evaluation consisted of the previously discussed metrics of the F1 and EM on the dev and test set of data. EM measures whether our answer exactly matches one of the 3 true gold answers. F1 score takes each gold answer as bags of words and doesn't require choosing the exact same span as human's, which is believed to be more reliable. AvNA (Answer vs. No Answer), a measure of the classification accuracy of our model when only considering its answer vs. no-answer predictions, is mainly used as a debugging tool.

5.3 Experimental details

5.3.1 Co-attention

We use pre-trained GLoVe word vectors [20] as the initial encoding for question and context tokens. The set contains 2.2M word vectors with dimension 300 from a 840 billion token crawl. We then used Stanford CoreNLP[21] to tokenize the data. We converted the character-indexed answer spans from SQuAD2.0 to token indexing. The conversion was simplified by disabling Penn Treebank token transformations to prevent having some tokens replaced by longer placeholders. Despite this, there were still special cases in which the conversion could not be completed, which we excluded from training. We ended up excluding 953 out of 86821 spans from the training set. Finally for the preprocessing step, we used batches containing contexts and questions of varying lengths. A context

[1] https://github.com/atulkum/co-attention
[2] https://github.com/asmitapoddar/question-answering-SQuAD2.0-dcn
[3] https://github.com/SebastianHurubaru/cs224n_squad_2

length of 400 was sufficient since there were very few samples above that size(as seen in figure 3. Similarly, we set a maximum question length of 30.
Replicating the original paper, we used the Adam optimizer with learning rate 0.01. The weights

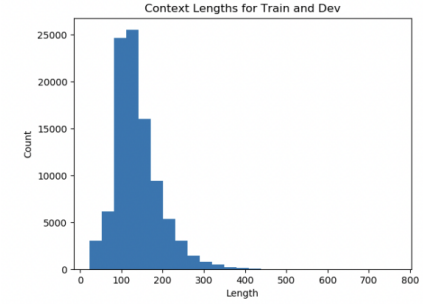

Figure 3: Length of context statements in train and dev sets

and biases and hidden states were initialized to zero whereas the encoding sentinels were randomly initialized. The start and end indices were initialized to the beginning of the context. We used LSTM hidden layers of size 200, BiLSTMs thus produce hidden states of dimension 400. We used a maxpool size of 16. We also did not use L2 regularization. We stopped training after 100000 steps since the model's F1 score no longer increased on the dev set after that point(as seen in figure 4). While the original paper does not provide us with a baseline we can use here, we observed particularly poor performance with our implementation.

5.3.2 FusionNet

In contrast to the original paper that is evaluated on the Squad version 1, we also need to consider the case where an answer is not found. In this case, we prepend a OOV (Out of Vocabulary) token to the beginning of each context when the question is unanswerable. The model still outputs the start(p_Start) and end(p_end) soft-predictions in this case, but when discretizing a prediction, if $p_start(0) \cdot p_end(0)$ is greater than any predicted answer span, the model predicts no-answer. Otherwise the model predicts the highest probability span as usual. We train our model for 30 epochs and select the one with the highest F1 score. The following hyperparameters give the best results: Learning rate 0.003, adamax optimizer, Exponential moving average decay rate 0.999, maximum gradient norm for gradient clipping 5.0, and a dropout probability 0.35. We also did not use L2 regularization. We then tried to use 6 models from 6 runs with slightly different hyperparameters (36 models in total), giving us a dev F1 of 67.735 and EM score of 65.23. We also do not have the results to the original FusionNet method since it was implemented on the version 1 of the dataset.

5.4 Results

We compare our results to the baseline model based on Bidirectional Attention Flow (BiDAF) [22]. The original BiDAF model uses learned character-level word embeddings in addition to the word-level embeddings. Unlike the original BiDAF model, the baseline implementation does not include a character-level embedding layer. After 25 epochs, using the baseline model, we get Dev NLL: 02.98 F1: 61.809 EM: 58.310 AvNA: 68.56. We use the non-PCE leaderboard values as reference since both coattention and FusionNet are non-PCE methods

With coattention, we obtain surprisingly low values of 49.67 and 38.39 as the F1 and EM scores which are worse than expected. We wonder if the low results are because of choosing a low decoder iteration limit(MAX_ITER) of 1 to reduce training time- whereas the paper provides results for both MAXITER= 4 and MAXITER= 1. We also could not find the ADAM hyperparameters in the original paper and used lr = 0.001 in our case. The initialization of most of the network and LSTM parameters to zero could also be a potential cause. Finally, no datset reshuffling was used for different epochs. We ran the FusionNet single model (without ensembling) and obtained the following metrics after 3853754 steps: Dev NLL: 06.21, F1: 64.49, EM: 60.56, AvNA: 72.00.

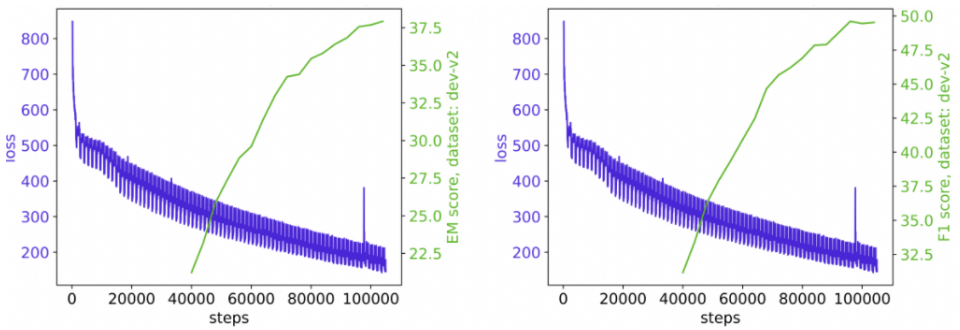

Figure 4: The training loss, F1 and EM scores of our coattention model.

Model evaluated on DEV	Dev NLL	F1	EM	AvNA
BiDAF baseline	02.98	61.81	58.31	68.56
Coattention	-	49.67	38.39	51.34
FusionNet single	06.21	64.49	60.56	72.00
Fusion Net ensemble	05.81	64.38	61.79	74.13

Note: Missed logging the dev NLL for the coattention module.

6 Analysis

In this section, we leave out the coattention network due to its poor performance. While we did try changing the hyperparameters aka, the sizes of the LSTMs, the learning rates, etc. we observed no significant patterns explaining the performance drop.

We then proceed to evaluate how tolerant our model is to changes in its inputs, such as typos. The goal here is to look at changes in model performance upon adversarial attacks. We explored various attacks presented in research [23, 24, 25, 26] to find a simple but useful technique for our purpose. As shown in the figure 5 we change the dev set by substituting two words in a sentence and by replacing words by random words to explore how our model behaves. We see a nearly clear linear correlation between the number of values changed and the drop in performance. While there is a performance drop, the linear relations also show some level of robustness to changes in the input sentences.

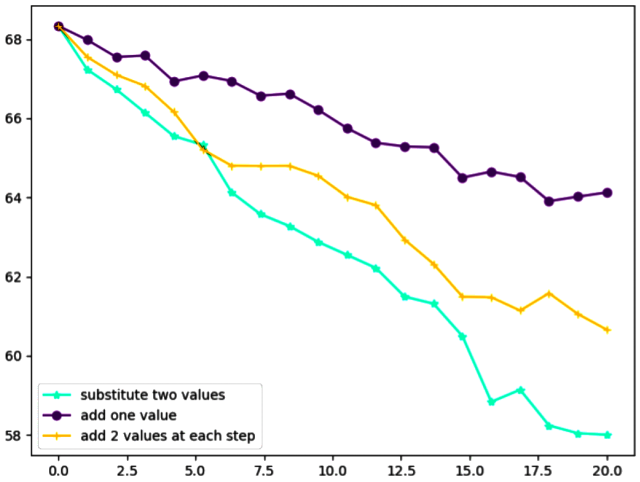

Figure 5: Robustness of the FusionNet single model to adversarial attacks of changing words in the sentences)

7 Conclusion

We tried to replicate the coattention network with a similar behaviour as shown in the paper, but resulted in significantly decreased performance. The FusionNet single and ensemble modules boost the performance on the question-answering tasks and show some level of robustness to adversarial attacks. While this paper limited to reproducing the original work, we considered the following as possibilities for future work:

- **Lower training time**: Both, the coattention and FusionNet models, took significantly longer than the baseline to train. Hence, we wish to speed the implementation of the DCN so it takes lesser time to execute so as to allow me to try multiple tweaks and hyperparameter settings.
- **Combination of different types of attention**: Interesting future work could involve developing a model structure combining the different types of attention layers either internally in the model or externally by ensembling to synergize and overcoming each method's shortcoming.
- **Initialisation techniques**: For our experiments, we initialised the parameters as described in the paper. We could also explore the effect of alternative initialisation schemes, such as Xavier initialisation for our LSTMs [27].
- **Training alterations to improve robustness to adversarial attacks**: We could explore models with improved deliberation and reasoning to avoid a significant decrease in performance upon adversarial attacks, particularly those caused by small typos. Perhaps the next version of Squad might include context with misspelled works or grammatically incorrect sentences, but ones that still contain an answer to the question.

References

[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[3] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[6] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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

[9] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

[10] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[13] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[14] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International conference on machine learning*, pages 1319–1327. PMLR, 2013.

[15] Caiming Xiong, Victor Zhong, and Richard Socher. Dcn+: Mixed objective and deep residual coattention for question answering. *arXiv preprint arXiv:1711.00106*, 2017.

[16] Yanshu Hong, Yiju Hou, and Tian Zhao. Coattention answer-pointer networks for question answering.

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

[18] Hsin-Yuan Huang, Chenguang Zhu, Yelong Shen, and Weizhu Chen. Fusionnet: Fusing via fully-aware attention with application to machine comprehension. *arXiv preprint arXiv:1711.07341*, 2017.

[19] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *Advances in neural information processing systems*, 30, 2017.

[20] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[21] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[22] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[23] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*, 2018.

[24] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Semantically equivalent adversarial rules for debugging nlp models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

[25] Max Bartolo, Tristan Thrush, Robin Jia, Sebastian Riedel, Pontus Stenetorp, and Douwe Kiela. Improving question answering model robustness with synthetic adversarial data generation. *arXiv preprint arXiv:2104.08678*, 2021.

[26] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. *arXiv preprint arXiv:1804.06059*, 2018.

[27] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.