Never Stop Learning

Lawrence Stratton
Unaffiliated
Columbia, MD 21029
deadbeef@stanford.edu, lawrence.stratton.ml@gmail.com

Abstract

Machine comprehension (MC) and question answering (QA) is the ability of a program to understand the nuances and concepts in a textual context or passage at a level where questions can be asked about the context and the program can generate an estimate of where in the context the answer can be found. The field of natural language processing has achieved recent success in MC by employing deep neural network-based architectures that incorporate temporal and hierarchical elements to compute answer predictions. In this paper, a mixture of models incorporating bidirectional recurrent neural networks (RNN), convolutional neural networks (CNN), and multiple attention mechanisms were used to attain an F1 score of 66.15% and EM score of 50.83%. The complex interaction between context, question, and answer is conjectured to be a function of word semantic similarity, hierarchical word semantic groupings, which might be perceived as ideas, long-term and short-term temporal relationships, and a filtering or attention mechanism that enables a model to focus on or ignore words, ideas, and relationships in a given example. This approximation of the complex interaction allows for an accurate prediction of the answer given a question about the context.

1 Introduction

Machine comprehension (MC) and question answering (QA) are experiencing dramatic improvements in capability and accuracy as the natural language processing community has shifted from expertly-constructed, structured feature-based systems to end-to-end neural network-based models that are trained on large datasets crafted for respective problems. The SQuAD dataset [1] was constructed using crowd-sourcing from human contributors and provides more complex question and answers than prior Cloze datasets, which enables more complex models to be built to approximate the inherent relationships. The dataset consists of 100,000 examples split into three sets – training, development, and test – with each example consisting of context, question, and answer triples.

In this paper, we started from the baseline model and progressively created more complex, larger models. The model complexity was increased by adding more sophisticated recurrent neural network architectures like Long-Short Term Memory (LSTM) [2] or Gated Recurrent Units (GRU) [3,4] and then adding stacked versions to create multi-layer LSTMs and GRUs. Another important variation was adding bidirectional versions of the LSTM [5] and GRU so that

temporal information could pass forward and backward to a timestep. Second, an attempt was made to create idea vectors from sequences of the word embeddings, which have their inspiration from the success of n-gram models that are also groupings of words. The core idea behind this approach was that although word embeddings encode semantic similarity between individual words [6], idea vectors might enable higher-level hierarchical concepts to be modeled by incorporating a similar metric among a hierarchy of the words. In this way, it may be possible to generate a knowledge base that could be used as a machine learning component like word embeddings, but for different tasks that might need a particular knowledge base, for example a financial corpus knowledge base versus a legal corpus knowledge base. Third, two attention mechanisms [7,8], Bidirectional Attention Flow (BiDAF) [9] and Dynamic Coattention (DynCoAttn) [10] were implemented to compare the differences and improvements that can be obtained from different forms of attention. Fourth, after the attention layer, either a secondary attention was applied like Self-Attention [11] or the blended features were concatenated with the attention features, which were then passed to a recurrent neural network modeling layer using either LSTM or GRU as the individual RNN units. These final features were then used to predict the starting and ending span predictions. An ablation study was provided to show the differences in performance with and without particular components of the architecture. Error analysis was provided to show the differences in models and where improvements could be made in future model architectures and experiments.

2 Background and Related Work

Rajpurkar et al. [1] created the Stanford Question Answering Dataset (SQuAD) using crowd-workers to answer questions on Wikipedia articles. They used a logistic regression model to develop an initial baseline F1 score of 51.0%, while human performance was listed as 86.8%. In Bahdanau et al. [7], they showed that it was possible to improve upon sequential models by incorporating an auto-aligning mechanism that was referred to as an attention mechanism. This attention mechanism was crucial to the model implemented in this paper and enabled the model to focus or respectively ignore features that may or may not have been relevant to a prediction at a particular time step in a question sequence.

In Seo et al. [9], a hierarchical approach composed of lower level features including character embeddings and word embeddings were progressively combined with contextual embeddings to form hierarchical features that were then passed to a unique attention layer. The CNN pipeline in this paper was inspired by this hierarchical feature set. The bidirectional attention layer was replicated in our model and enables attention to flow from the context to the question as well as from the question to the context. It is conjectured that this aids in capturing the mutual interactions between the information content in the question and the context, and the model is then able to replicate this interdependence succinctly. Finally, and most importantly to performance, the context features are passed along with the attention blended features to a modeling layer that incorporates another recurrent neural network. This layer will be shown to be indispensable in the experiment section.

In Xiong et al. [10], another approach was taken toward fusing or blending question and context features using a unique coattention mechanism. This coattention mechanism was viewed as comparable in model value to the BiDAF mechanism and was implemented in the model for comparison. In much the same way as BiDAF, the dynamic coattention (DynCoAttn) mechanism also required a subsequent modeling layer for fusing the context and blended attention features to produce substantially improved results. Lastly, they improved upon the prediction layer by implementing a dynamic decoder that iteratively computed improved start and end span predictions.

In Vinyals et al. [12], Pointer-Net was introduced as a means of using the attention mechanism idea as a way of dynamically picking an output from a variable length input. This approach is a profoundly useful technique and much time was spent unsuccessfully trying to incorporate this into the prediction layer. Vinyals solved computationally intractable problems like the traveling salesman with little to no tuning of hyperparameters, and this was most likely attributable to the exponentially efficient universal approximation properties of deep neural networks [13]. Vinyals' paper was incredibly compelling in that it cogently suggested that other problem domains could likely be dramatically improved if a dataset and a neural approach were designed to solve the particular domain problem.

In Wang and Jiang [14], the context and questions are preprocessed in an LSTM layer and then passed to their MatchLSTM layer. This layer performed an additive attention mechanism and then the blended representation was

concatenated with the original context where it was then passed to a modeling layer. This was repeated in the reverse direction to produce a bidirectional feature set. Finally, the output of the modeling layer was passed to an adaptation of the Pointer Net to produce predictions of the start and end span called Answer-Pointer. The more applicable version of the Answer-Pointer network was the boundary model, which only predicted the start and end indices instead of the entire sequence of indices from start to end.

In R-Net [11], a novel self-attention mechanism was introduced that helped to focus the attention from the context to the context as a means of maximizing the relevant features within the context. This was complemented with a gating mechanism to help the network determine if the features were relevant or irrelevant and might have been interpreted as a focus gate. Like previous models, the blended representations were then passed into a modeling layer, which was a bidirectional RNN. Like Wang and Jiang 2016, they chose to use a pointer network adaptation to help predict the start and end span predictions.

After taking inspiration from Quasi-Recurrent Neural Networks [15], which combines aspects of both convolutional neural networks and recurrent neural networks, a convolutional neural network pipeline was setup in parallel with the recurrent neural network pipeline. The CNN pipeline borrows inspiration from [16,17], where CNNs were used to model sentences. The underlying goal was to build idea vectors from the context and question separately, which would be hierarchical combinations of word embeddings from the question and from the context in order to efficiently capture the core ideas in a question or context.

Finally, although Memory Networks [18] expand neural architectures with computer architecture concepts and are promising in terms of merging the memorizing potential of modern computers with neural approaches, the approach was not implemented in this model. Similarly, although Adversarial Squad [19] brought the adversarial example approach to improving model robustness, adversarial example data augmentation was not used in this model to improve robustness though it seemed reasonable that if combined with a gating mechanism, these examples could help the model learn to ignore features better.

3 Approach

3.1 Problem Definition

The general question and answering problem is formulated as follows. There are 100,000 example tuples, where each tuple is composed of a question, the context, and an answer to the question from the context. The answer is specified by a start span and an end span, whereby the words between the start and end span are the answer to the question. For each example, each question, context, answer tuple (q^k, c^k, a^k) is composed of words, w_i. A question is defined as $q^k = \{w_t\}_{t=1}^{T=M}$ where a question is viewed as a sequence of M words. A context is defined as $c^k = \{w_t\}_{t=1}^{T=N}$ where a context is viewed as a sequence of N words. An answer is defined as an index or span representing the starting and ending position in the context of the answer or $a^k = [s^k, f^k]$. The overall objective is to develop a question-answering system that uses the 100,000 examples to approximate the underlying function $f(q^k, c^k) = a^k$. A deep neural network-based architecture composed of stacked recurrent network blocks, convolutional network blocks, and multiple attention mechanisms was developed as an approximation to this function. From [13], neural networks are universal, exponentially-efficient approximators, which enables accurate approximations to be developed given sufficient regularization, computational power, and dataset size.

3.2 Model

In approximating the function f, which represents the complex interactions between the question, the context, and the answer, it was assumed that the model would do this by systematically decomposing the interactions into the following three qualitative categories. First, the question and context were interpreted as a temporal word sequence, whereby information at multiple timesteps forward and backward from the current timestep were viewed as being relevant to the processing and understanding of the information content at the current timestep. Second, the words in the question and context alone provided information based on their word embeddings, but it was conjectured that word groups

could provide equally relevant information when amalgamated hierarchically among their adjacent neighbors in a sequence. This could be represented as a hierarchy whereby groups of words can be composed into ideas or higher-level concepts. Lastly, given dense information in the context and question, it was not sufficient to just have representative features of the question or context. The similarity and relevance between the features had to be identified so as to maximize what was relevant to a particular question or context. In short, a mechanism to focus on a particular set of ideas and concepts was needed to filter what was relevant and this was accomplished using an attention mechanism. From an adversarial standpoint, what was needed was not just an attention mechanism, but also a mechanism for ignoring what was not relevant. Since there are multiple ways of accomplishing the three categories, the architecture was broken down into the sub-architectures as shown in Figure 1.

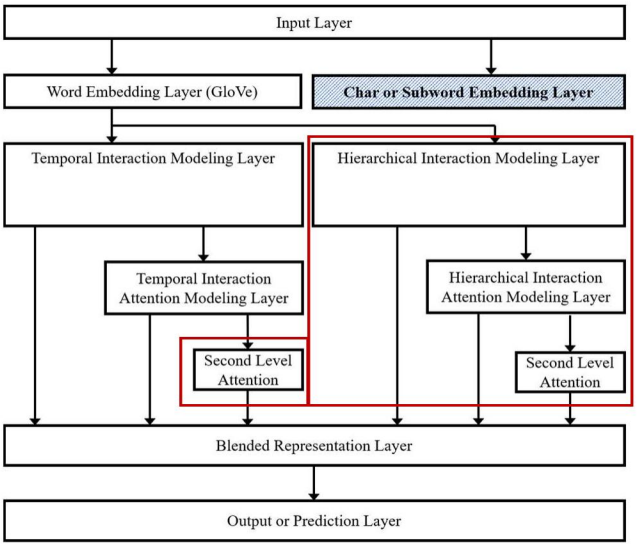

Figure 1. High-level modular architecture used to compute the approximation function $\hat{f}$. Red-boxed portions did not train well, were intractable computational bottlenecks, or produced very sub-par predictions when trained on small batch sizes.

This modular architecture enabled different model components to be swapped in to test whether one component was better than another. The modular variations that were possible included:

Temporal Interaction Modeling Layer: Basic Bidirectional RNN-GRU Encoder, Bidirectional RNN-GRU Encoder with N-Layers, Bidirectional RNN-LSTM Encoder with N-Layers.

Hierarchical Interaction Modeling Layer: 1-D Convolutional Neural Network of filter width f and layer depth d with k filters followed by max-pooling. Variation was primarily changed in the number of layers of the 1-D CNN, maxpooling sequence. The CNN layers were augmented with highway networks to enable increased depth, while minimizing training difficulty [20,21].

Temporal and Hierarchical Interaction Attention Layer: Basic Attention, Bidirectional Attention Flow, Dynamic Coattention

Second Level Attention: Self-Attention

Blended Representation Layer: Affine Relu Layer, Affine Maxout Layer [22]

Output or Prediction Layer: Softmax prediction

All blended features generated were then combined in the blended representation layer before being passed to the output or prediction layer. Although character embeddings and subword embeddings [23] can be used to improve the out of vocabulary (OOV) problem for the GloVe word embeddings, they were not used in this model. Part of speech

(POS) and named-entity recognition (NER) features were not used because they were shown to be ineffective at increasing performance in [14].

3.3 Temporal Approach RNN Equations

The following equations describe the best model that was trained during the experiments. This was the fully implemented bidirectional attention flow model.

$$q^k = \{w_t\}_{t=1}^{T=M}, \qquad c^k = \{w_t\}_{t=1}^{T=N} \tag{1}$$

$$h_k^q = StackedBiDirGRU(q^k, 3\ layers), \qquad h_k^c = StackedBiDirGRU(c^k, 3\ layers) \tag{2}$$

$$C2Q_k, h_k^c \circ C2Q_k, h_k^c \circ Q2C_k = BidirectionalAttentionFlow\left(h_k^q, h_k^c\right) \tag{3}$$

$$G = [h_k^c, C2Q_k, h_k^c \circ C2Q_k, h_k^c \circ Q2C_k] \tag{4}$$

$$M^1 = StackedBidirectionalGRU(G, 2\ layers) \tag{5}$$

$$M^2 = StackedBidirectionalGRU(M^1, 1\ layer) \tag{6}$$

$$z_s = {w_{(s)}}^T[G; M], \qquad p^s = softmax(z_s) \tag{7}$$

$$z_e = {w_{(e)}}^T[G; M], \qquad p^e = softmax(z_e) \tag{8}$$

4 Experiments

4.1 Implementation and Dataset

The SQuAD training dataset has the summary statistics shown in Figure 2. Based on the given statistics, limiting the answer span difference between start and end to 15 could have increased accuracy. The context was length was chosen to be 600, but it may have been possible to get similar or better accuracy using 300 due to computational limitations. The question length was set to 30.

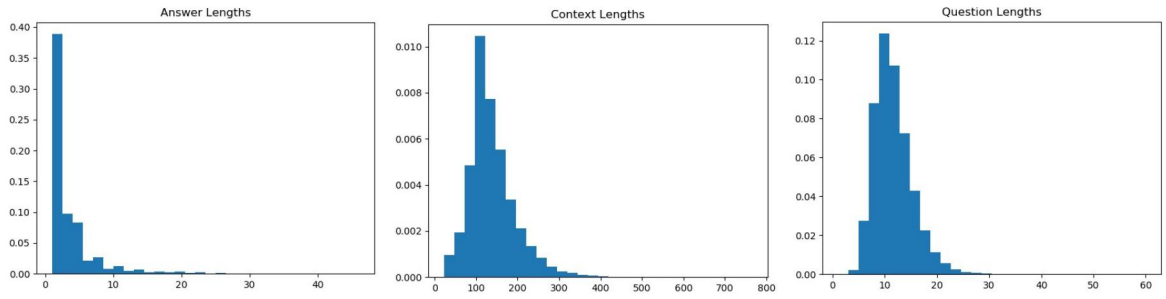

Figure 2. SQuAD training set summary statistics.

The experiments were performed using Tensorflow [24] and cuDNN [25] on Azure virtual machines ranging in hardware from one Nvidia Tesla M60 graphical processing units (GPU) to four Nvidia Tesla M60 GPUs per machine.

Models were trained using batch stochastic gradient descent with the Adam optimization algorithm [26] or the Adadelta optimization algorithm [27]. More complex models were trained with smaller batches to prevent out-of-memory (OOM) errors. The learning rate for Adadelta was set to 1.0 and was largely insensitive to learning rate changes. The learning rate for Adam was tuned using a hyperparameter search coupled with a random search as recommended in [28] then followed by a more focused grid search. The hyperparameter values for learning rate were calculated using a uniform random variable over the exponent and then raised to the power of ten. All recurrent networks had dropout applied to improve generalization from the training set to the development set [29,30]. The

dropout regularization parameter was randomly generated using uniform random generation between 0.2 and 0.3. The hyperparameter tuning is shown in Table 1 below with observations during training. Batch size and hidden units were set to 50 and 100 respectively.

Table 1. Hyperparameter search observations.

Model #	Learning Rate	Dropout	Observations
1	0.5000	0.2242	Loss explodes shortly after starting
2	3.2397e-2	0.2518	Loss grows, then trains slowly
3	3.4256e-4	0.2211	Trains well, fast
4	9.6060e-5	0.2388	Trains well, fast
5	1.1301e-5	0.2041	Trains well, slower than (4)
6	5.7779e-3	0.2328	Trains well

Training was stopped when fast learning was encounter and then the next hyperparameters were tested. If the training loss started to monotonically grow, training was immediately stopped. Hidden units were not changed based on training comments in [14], which stated that when more hidden units were added the improvements were small. Following the random search, a grid search was performed over the learning rate and dropout in the more limited range shown in Table 2.

Table 2. Grid search over learning rate for Adam algorithm.

Learning Rate	2.1097e-4	4.3763e-4	9.6060e-5	3.3519e-4	8.7968e-5

4.2 Results

The results are calculated based on two metrics, F1 and EM. F1 measures the overlap between the predicted words and the answer words. EM or exact match calculates if it is an exact string match with the answer. EM is a stricter measure. The results are shown in Table 3 and the training results are shown in Figure 4.

Table 3. Experiment results with ablation of BiDAF model.

Model	Train F1	Train EM	Dev F1	Dev EM
Baseline	0.5641	0.4419	0.3965	0.2895
BiDAF (basic)	0.6494	0.5370	0.4210	0.3059
BiDAF (full)	0.7656	0.6220	0.6475	0.4988
BiDAF (full, ens2)	**0.7922**	**0.6570**	**0.6615**	**0.5083**
DynCoAttn (full)	0.7614	0.6330	0.6327	0.4819

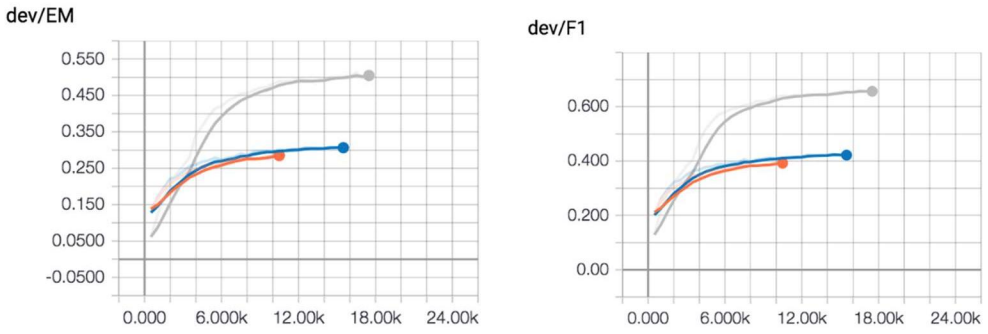

Figure 4. Basic baseline model in orange, bidaf model without modeling layer in blue, bidaf model with modeling layer in gray.

All RNN models produced the first encoder representations using three-layer deep LSTM or GRUs. The difference between LSTM and GRU units was negligible and GRUs were selected for reduced computational complexity in

subsequent models. The BiDAF basic model only implemented the bidirectional attention mechanism without a modeling layer. This improved the F1 accuracy roughly 3% over the dot product attention mechanism. By adding the modeling layer and adding another GRU RNN to the prediction layer, the F1 accuracy increased nearly 26% over the baseline model. In both the DynCoAttn model and the BiDAF model, adding the modeling layer dramatically improved the F1 and EM results. Consistently, the BiDAF implementation outperformed the DynCoAttn modelas shown in Figure 5, although the dynamic pointer decoder was not implemented from the full dynamic coattention network.

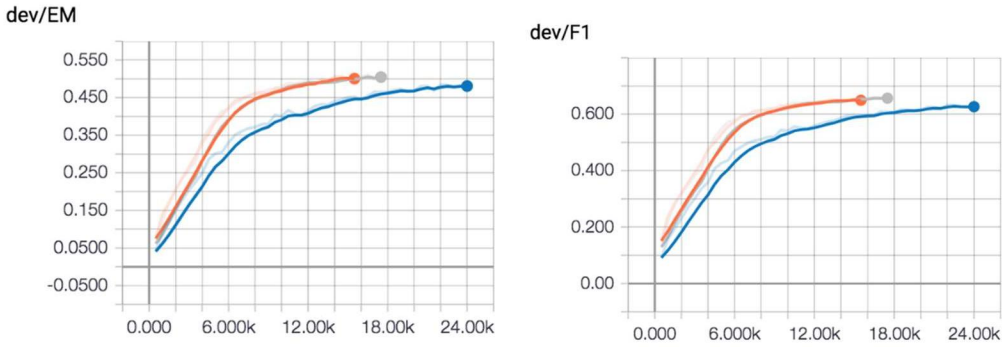

Figure 5. Bidaf model (full) orange, bidaf model (full, ens2) gray, dyncoattn model (full) blue.

The CNN models proved difficult to train and when trained produced sub-par results. This was disappointing because CNNs are efficiently implemented for GPUs and the hierarchical approach seemed reasonable. It was possible the CNN architecture could have been simplified to use simpler residual networks [31], instead of more complex highway networks, but ultimately this hierarchical approach requires further investigation.

The BiDAF and DynCoAttn models were augmented with a self-attention context feature, but this proved computationally costly and both augmented models could not be trained effectively.

4.3 Error Analysis

The best trained BiDAF model did well on when, what, and where questions. These particular examples had shorter answers, and that may have been the underlying factor in success. Successful examples are shown in Table 4. Longer what and how questions had much less successful predictions as shown in Table 5.

Table 4. Examples where the model did well.

Question	**True Answer**	**Predicted Answer**
when was the prime number theorem proven?	at the end of the 19th century	end of the 19th century
what did tesla incorrectly believe about x-rays?	x-rays were longitudinal waves	longitudinal waves
what legitimate dynasty came after the yuan	ming	ming
where did tesla go upon leaving gospic?	prague	prague
what did the greek root pharmakos imply?	sorcery or even poison	sorcery or even poison

Table 5. Examples where the model did poorly.

Question	True Answer	Predicted Answer
what is the function of the tardis?	time machine	firmly linked to the show in the public's consciousness
how big was the vertical assembly building?	130 million cubic feet	vab

5 Conclusion

Improving upon state of the art is difficult in machine comprehension, but with systematic decomposition, iterative improvements, and model diagnostics it seems eminently possible. The modeling layer and in particular applying an RNN model after computing blended representations proved to be invaluable for improving the accuracy of a model. Computational limitations can force early down-selections of model approaches, but it seems that more computational power can actually enable more expressive models to be built and tested. Learning rate annealing with ADAM at the very end enabled a small 1-2% boost using manual changes. Regularization was necessary to generalize well from the training set to the development set, but the tradeoff was that regularization made the model more difficult to train. These computational bottlenecks definitely encouraged learning new approaches to training a portion of the computational graph on multiple GPUs. When implementing models from research papers, although the high-level details are given, there was definitely a realization that the low-level training details or implementation nuances are crucially important even though they may be omitted or glossed over.

With computational bottlenecks, a desired future research direction is developing an information-theoretic metric for sorting examples in a batch. The impetus for this is to maximize the amount of information propagated through the network during training per batch, and it seems plausible that not all examples are equal in information content. Lastly, although the hierarchical convolutional neural network pipeline was largely a failure, the hierarchical notion of word embeddings as idea vectors still seems worth pursuing. While word embeddings are now almost a default element of deep neural network models for natural language processing problems, it would be a worthwhile pursuit to determine if idea vectors can be precomputed in a similar fashion as the next step up the conceptual chain.

References

1. Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.
2. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.
3. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
4. Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.
5. Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*(5-6), 602-610.
6. Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).
7. Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
8. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*(pp. 6000-6010).
9. Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H. (2016). Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*.
10. Xiong, C., Zhong, V., & Socher, R. (2016). Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*.
11. F. Wei, & M. Zhou. (2017) R-net: Machine reading comprehension with self-matching networks. Natural Language Computing Group, Microsoft Research Asia.
12. Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems* (pp. 2692-2700).
13. Montufar, G. F., Pascanu, R., Cho, K., & Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in neural information processing systems* (pp. 2924-2932).
14. Wang, S., & Jiang, J. (2016). Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*.
15. Bradbury, J., Merity, S., Xiong, C., & Socher, R. (2016). Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*.
16. Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.
17. Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.
18. Weston, J., Chopra, S., and Bordes, A. (2015) Memory networks. ICLR). arXiv preprint arXiv:1410.3916.
19. Jia, R., & Liang, P. (2017). Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*.
20. Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.
21. Zilly, J. G., Srivastava, R. K., Koutník, J., & Schmidhuber, J. (2016). Recurrent highway networks. *arXiv preprint arXiv:1607.03474*.
22. Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. *arXiv preprint arXiv:1302.4389*.
23. Mikolov, T., Sutskever, I., Deoras, A., Le, H. S., Kombrink, S., & Cernocky, J. (2012). Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*.
24. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). TensorFlow: A System for Large-Scale Machine Learning. In *OSDI* (Vol. 16, pp. 265-283).
25. Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., & Shelhamer, E. (2014). cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*.
26. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

27. Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.
28. Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*(Feb), 281-305.
29. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929-1958.
30. Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.
31. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).