CNN and RNN Hybrid Reading Comprehension Systems

Xiaohua Liang
SCPD
Stanford University
xiaohual@stanford.edu

Siyu Yang
SCPD
Stanford University
siyu@stanford.edu

Abstract

In this project, we examine two approaches to the reading comprehension task on the SQuAD 2.0 dataset. We implement QANet, make improvements to our baseline BiDAF model, and train two hybrid models mixing recurrent and convolutional layers, while experimenting with different embedding, self-attention and discretization configurations. On the dev set, our QANet model achieves 63.6% F1, 60.5% EM, and our BiDAF model achieves 65.9% F1, 62.9% EM (non-PCE).

1 Introduction

Reading Comprehension (RC) is a form of a Question Answering task, where the model predicts the answer to a question by selecting a segment in the context paragraph. A pivotal dataset that spurred much recent work on RC is the Stanford Question Answering Dataset (SQuAD) [1], consisting of questions posed on a set of Wikipedia articles. The newer version, SQuAD 2.0 [2], additionally contains questions that cannot be answered based on the provided paragraph.

A recent neural network model developed for the RC task, the QANet [3], was proposed as an improvement over the earlier Bi-Directional Attention Flow (BiDAF) model [4], both evaluated on the original SQuAD dataset. QANet uses convolution and self-attention exclusively in its embedding and model encoders, forgoing any recurrent structures. The two models however share a high-level architecture: word and character embeddings of the context paragraph and the question are fed to the embedding encoder layer, followed by a bi-directional attention flow layer, a model encoder layer and an output layer (Figure 1).

In this project we first implement the QANet following the same high-level structure as the baseline BiDAF model. We train it on SQuAD 2.0 with unanswerable questions, as well as on the original SQuAD dataset. We then explore the effect of mixing RNN-based and CNN-based encoders for the embedding and model encoder layers in this shared high-level architecture by training two hybrid models. We also discuss a number of other experiments involving character embeddings, self-attention in the QANet's encoder block, and the discretization step at the end of the output layer. Our best QANet model achieves 63.6% F1, 60.5% EM on the dev set, and 60.1% F1, 56.2% EM on the test set; our BiDAF model achieves 65.9% F1, 62.9% EM on the dev set, and 64.5% F1, 61.5% EM on the test set.

2 Related work

Previously, complex NLP tasks such as Machine Reading Comprehension (RC) and Question Answering (Q&A) are tackled by stacks of recurrent neural networks (RNNs) composed of Gated Recurrent Unit (GRU) [5] or Long Short Term Memory (LSTM) units [6]. In contrast, convolutional neural networks (CNNs) found more success in tasks where a representation of the meaning of the sentence as a whole is sufficient, such as part of speech tagging and sentiment analysis. Since

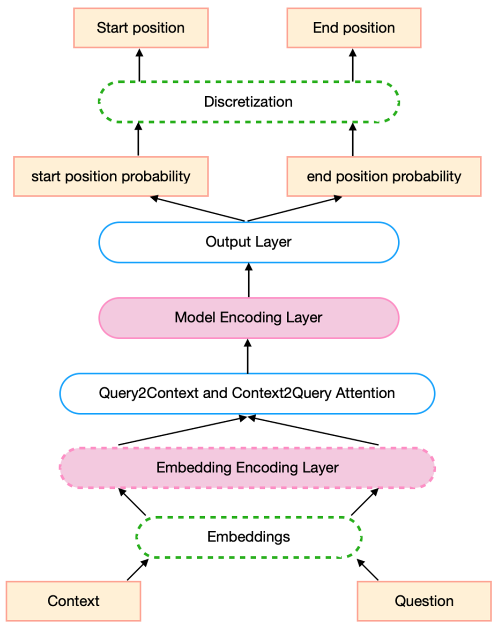

Figure 1: Overview of the high level architecture shared between BiDAF and QANet. Boxes in blue are parts that are unchanged from the baseline model or original architecture; boxes in pink are modules that we experimented with creating our hybrid models; boxes in green with dotted line outline are other parts of the model that we experimented with.

the release of the first and second versions of the SQuAD dataset [1] [2], many RNN-based RC systems have been developed, incorporating a variety of attention mechanisms: BiDAF introduces bi-directional context-to-query and query-to-context attention flow [4]; Gated Self-Matching Networks (R-NET) adds a gate to the network and uses a form of self-attention on the context passage [7]; Reinforced Mnemonic Reader incorporates dynamic-critical reinforcement learning as the optimization method to mimic how humans read a passage multiple times in order answer a question [8].

Despite the popularity of RNNs and their alignment to the sequential nature of text and speech, RNNs are slow to train since they cannot take advantage of parallel computation that modern GPUs support. To overcome the training and inference speed limitation, researchers have explored fully convolutional or fully attention-based models as alternatives to RNNs. In particular, the introduction of transformers for Machine Translation tasks by Vaswani *et al.* [9] demonstrates that recurrent structures (and even convolutions) can be replaced by a combination of different attention mechanisms. Extending the success of non-recurrent networks to the task of RC, Yu *et al.* [3] was able to surpass previous models in both speed and accuracy on the original SQuAD dataset.

3 Approach

We implemented QANet while conforming to the high-level architecture of the baseline BiDAF model, adding character embeddings to both. We then mixed RNN-based and CNN-based encoder blocks (summarized in Table 1) to understand the performance of such hybrid models.

3.1 Baseline model: BiDAF

We trained our baseline BiDAF model with the provided implementation and hyperparameter values, obtaining 60.0% F1 and 57.6% EM on the dev set after convergence in training loss.

Table 1: The four combinations of RNN-based and CNN-based encoder blocks for the embedding and model encoder layers. Combinations marked by a question mark are the hybrid models we build.

	Model layer		
		RNN	CNN
Embedding layer	RNN	BiDAF	?
	CNN	?	QANet

3.2 QANet

We implemented QANet in PyTorch, following [3] closely. We used the same bidirectional attention flow layer from BiDAF, and modified the multi-headed self-attention block in the official implementation of the Transformer network by Vaswani *et al.* [9]. In all sections below, positions of `<pad>` are masked by 0 after each layer.

Word and character embedding layer For each word the output of the embedding layer is the concatenation of the word and character embeddings $[x_w; x_c] \in \mathbb{R}^{p_1+p_2}$, where $p_1 = 300$; p_2 was initially 200 but reduced to 64 after finding no significant difference training for 10 epochs. The character embeddings are randomly initialized from the provided initial values and are not frozen, unlike the word embeddings. The character and word embeddings are concatenated before going into another layer of the Highway Network (with a projection to the hidden dimension of 128 or 96 for BiDAF) [10].

QANet encoder block The encoder blocks are used for both the embedding encoder and model encoder layers. Firstly, position encoding (PE) is added to each component of the input [9] with a phase difference between even and odd positions in the embedding:

$$\text{PE}_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$\text{PE}_{(pos,2i+1)} = \sin(pos/10000^{2i/d_{\text{model}}} + \pi/2)$$

where pos is the position in the text sequence, i is the dimension in the embedding, and d_{model} is the length of one embedding vector. The input is then fed to a number of sublayers of residual blocks with layernorm: $f(layernorm(x)) + x$. The sublayers are a number of depthwise separable convolutions [11], a multi-headed self-attention and a feed forward layer. The multi-headed self-attention is a projection of the concatenation of h scaled dot product attention outputs, where all of the keys K, values V and queries Q are the position encoded input $x + PE(x) \in \mathbb{R}^{\text{seq}\times\text{emb}}$.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$,

and $\text{Attention}(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$, where $d_k = d_{\text{model}}/h$.

Embedding encoder layer The embedding encoder layer consists of one encoder block described above, with kernel size 7 for convolution and 4 convolutional sublayers.

Model encoding layer The model encoding layer consists of 7 encoder blocks, with kernel size 5 for convolution and 2 convolutional sublayers. Three stacked layers are used.

Output layer and loss function For QANet, the output start and end indices are predicted without a LSTM between them:

$$p^1 = softmax(W_1[M_0; M_1]), \qquad p^2 = softmax(W_2[M_0; M_2])$$

where M_i is the output of the i^{th} encoder layer in the model encoding layer. The loss function is the negative sum of the log probabilities of the predicted distributions at positions of the true start and end indices y^1 and y^2 averaged over all N training examples:

$$L(\theta) = -\frac{1}{N}\sum_i^N \left[log(p^1_{y^1_i}) + log(p^2_{y^2_i})\right]$$

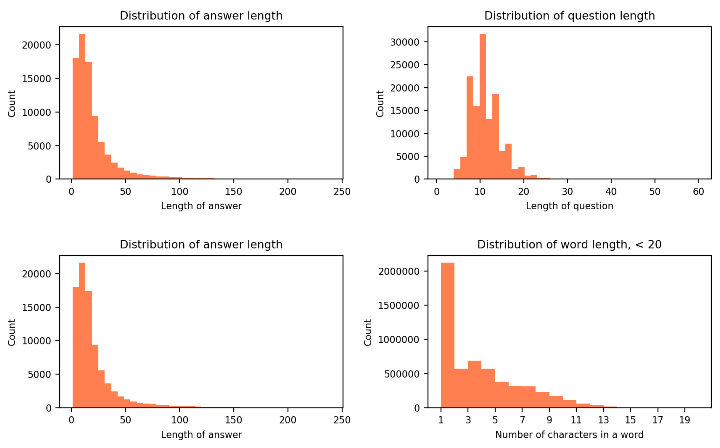

Figure 2: Histograms of context, question, answer and word lengths in the training split.

Stochastic depth Stochastic depth (layer dropout) [12] as a regularization and training time saving method was implemented by keeping a count of the depth of the sublayer in each encoder layer, where sublayer l has survival probability $p_l = 1 - \frac{1}{L}(1 - p_L)$, where L is the number of conv layers in that block, and $p_L = 0.9$ as described in the Yu *et al.* [3]. This means that the entire residual block would be dropped during training according to the layer's survival probability p_l, sampling from a Bernoulli random variable.

4 Experiments

4.1 Data

We use the provided SQuAD 2.0 training and dev sets for training and validation before submitting results on the test set. Splits are made on the article level so that how well the model generalizes can be fairly assessed. SQuAD 2.0 contains 478 articles, 442 in the training set and 16 in the dev set. There are a total of 100,000 answerable questions and 50,000 unanswerable questions, 130,319 (87%) of which are in the training set.

To accommodate SQuAD 2.0's unanswerable questions, a `<unk>` token is prepended to each context so that unanswerable questions' predicted start and end positions of the answer should be 0. The distribution of context, question, answer and word lengths for the training set is shown in Figure 2. Any context paragraphs longer than 400 tokens are discarded during training and validation; this leaves us with 129,941 training examples. Long context paragraphs are capped at 400 words during evaluation on test set.

4.2 Evaluation method

The evaluation criteria are F1 score (word level after lower casing and removing punctuation, articles and extra whitespace) and Exact Match (EM) count. We also manually review a sample of predicted answers. In addition, we visualize the activation of the multi-headed self-attention sublayers in the embedding encoder layer for a subset of the validation data to ensure that masking is applied correctly (see example visualizations in Figure 4) and verify that the multiple heads are attending to words in the text differently.

4.3 Experimental details

Our implementation used PyTorch 1.0 and ran in eager mode. The BiDAF model was trained using batch size 64 and surprisingly both learning rate 0.5 and 0.001 gave equally good results, attesting to BiDAF's insensitivity to learning rate, in drastic contrast to QANet. Various configurations of the QANet were mostly trained with a batch size of 32, and learning rate from 0.0003 to 0.001

worked reasonably. QANet often showed signs of overfitting on the dev set, and increasing dropout probability did not always help.

The training speed for a number of model configurations are shown in Table 2. Note that we had to employ a number of different GPU SKUs. QANet takes longer to train per epoch, but [3] argues that it converges more quickly so that the total training time to reach a certain performance is shorter compared to BiDAF.

Table 2: Summary of the training speed for different models and configurations.

Model	GPU	Training speed (minutes per epoch)
BiDAF baseline	Half M60	15.9
BiDAF with char emb	Half M60	28.6
Full QANet	K80	75.3
Small QANet (2 heads, 96 hidden size)	V100	7.2

4.4 Results

In this section we discuss our experiments, their results and other observations made during the training process. Apart from our hybrid model experiments, we neglect to report on their performance in full since we did not train all of them to convergence to save time, often stopping a run after a few epochs once the learning curve does not show improvement from previous runs.

4.4.1 QANet trained on SQuAD 1.1

To verify our implementation of the QANet, we trained it on SQuAD 1.1 with character embeddings and a smaller hidden size of 96 instead of 128 (to save time), and was able to achieve an F1 score of 77.8 after only 15 epochs, which reproduces the F1 score of 77.0 reported in [3], obtained after 13 epochs. This indicates that our implementation is sound.

4.4.2 Hybrid models

Our main experiment involved creating two hybrids between BiDAF and QANet: one with LSTMs from BiDAF as the embedding encoder layer and CNN and self-attention blocks as the model encoder layer (called BiDAF-QANet in Table 3), and vice versa (QANet-BiDAF). Their performance is summarized in Table 3.

It would be more fair to tune the hyperparameter for each model configuration so that the performance is optimal in each case. We did not carry out hyperparameter sweeping and kept all hyperparameters the same instead; this is reasonable given how insensitive BiDAF is to learning rate (both 0.5 and 0.001 worked well). Optimizing the learning rate for each hybrid could be a good first step in future work so that the potential of the hybrid models can be evaluated. Under our setting, neither hybrid models has superior performance; the BiDAF-QANet hybrid is better than the QANet-BiDAF hybrid by about 2 points in F1 score, perhaps suggesting that LSTM layers are better at capturing the contextual information in a piece of text and so are more important for the embedding encoder layer.

Table 3: Performance of our BiDAF, QANet and two hybrid models with the same configuration and hyperparameters. The RNN blocks used are LSTMs.

Name	Embedding encoder	Model encoder	F1	EM
BiDAF	RNN	RNN	65.0	61.4
QANet	CNN + atten	CNN + atten	61.9	58.1
BiDAF-QANet	RNN	CNN + atten	62.0	58.5
QANet-BiDAF	CNN + atten	RNN	59.9	56.3

4.4.3 Trainable `<unk>` and character embeddings

Adding character embeddings improved the F1 score of both models. For BiDAF, F1 improved from 60.0% to 63.4%, and EM from 52.8% to 57.6%. In an earlier model, we neglected to make the word embedding of the `<unk>` word trainable, and once it became trainable, performance improved by about 2 points in F1 score. However, we observed that the effect of adding character embeddings and making `<unk>` trainable are not entirely orthogonal, and so these two mechanisms are somewhat interchangeable. This is because the trainable character embeddings can capture information about out of vocabulary words the same way a trainable `<unk>` word embedding can.

4.4.4 Hidden dimension

To reduce the computation load we decreased the hidden size dimension of the QANet from 128 as specified in the paper to 96, and observed no significant difference in performance in the first ten epochs.

4.4.5 Number of heads in multi-headed self-attention

We experimented with using 2, 4 and 8 heads in QANet's multi-headed self-attention sublayers, and observed no significant difference in performance after about 7 epochs. Our final and hybrid models use 4 heads.

4.4.6 Input size for questions

We conducted an experiment in the hope to reduce the amount of computation and therefore training time by capping the length of questions at 50 (decided based on question lengths distribution shown in Figure 2). This did not lead to comparable performance.

4.4.7 Weight on probability of no-answer

Upon observing that the majority of confusion in the model's decision for whether to answer a question comes from answering unanswerable questions (see section 5.2), we experimented with two solutions. An obvious approach to address the models' bias towards answering questions too liberally is to amplify the penalty for that behavior in the negative log likelihood (NLL) loss function by scaling up the loss when no-anwer cases are incorrectly predicted. This decreased the speed at which the model learnt and did not yield better performance. We did not do any further tuning on the scaling factor after preliminary experiments with scale factor of 1.5, 2, and 4 all show negative results.

A second approach was adding a post-processing step in the discretization procedure, where we simply multiplied the probability score of the zeroth position p_0 by a scaling factor s. This biases the model towards choosing not to answer a question, and increased the F1 score by 1.7 for QANet and 0.96 for BiDAF (intuitive since QANet suffered from the issue more acutely). We chose the value of s based on dev set performance, which is summarized in Table 3.

s	F1	EM	Ans. vs no ans.
0.0	65.0	61.4	71.2
1.25	65.2	61.64	71.3
2.0	65.6	62.38	71.5
4.0	65.9	62.9	71.1
8.0	65.7	62.9	70.4

(a) BiDAF

s	F1	EM	Ans. vs no ans.
0.0	61.9	58.1	69.8
4.0	63.0	59.6	70.0
6.0	63.4	60.0	70.1
8.0	63.6	60.5	70.1
10.0	63.5	60.4	69.8

(b) QANet

Figure 3: Dev set performance for using different values of the scaling factor s for p_0.

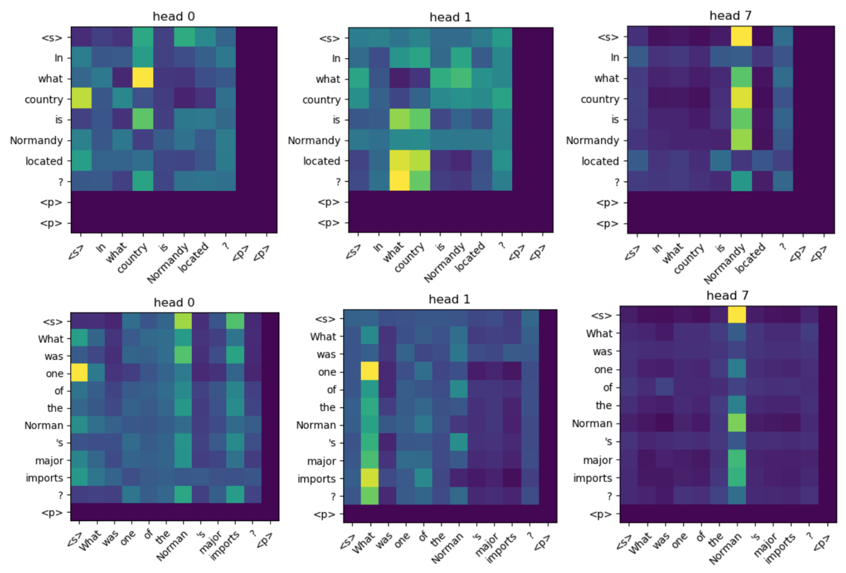

Figure 4: Visualizing 3 out of the 8 heads in the multi-headed self-attention for two questions in the validation set, in our QANet implementation. Lighter colors denote higher values. In the first row is a question with possible answers, and the second row is an unanswerable question. One way to interpret a self-attention matrix is that it represents how much attention each word down the rows is paying to all the words in this sentence across the columns.

Table 4: Comparing the performance of the models on all questions and on the subset of answerable questions that the models also chose to answer.

Model	All questions		Answered answerable questions	
Name	F1	EM	F1	EM
BiDAF	65.0	61.4	83.7	74.5
QANet	61.9	58.1	80.6	71.3

5 Analysis

5.1 Activation of multi-headed self-attention

In QANet, each attention head acts as a feature selector and attends to different aspects of the text [9]. We visualized the self-attention matrices in the embedding encoder layer for both context and questions during evaluation on the dev set, and show samples for an answerable and an unanswerable question in Figure 4. Cell $c_{i,j}$ in the matrix represents how much word w_i is attending to word w_j in the same question text, and the lighter the color, the higher the activation. We see that different heads indeed become activated differently during evaluation. Question words such as "what" is often highly activated in one of the heads.

5.2 Error analysis

The question answering task on SQuAD 2.0 can be viewed as deciding whether a question has an answer and predicting the span if it does. Although both sub-tasks are predicted by the same softmax output, their consequence on the F1 metric is different: answering an unanswerable question affects the F1 score more than answering an answerable question imperfectly.

To gain a sense of how much our performance is negatively affected by mistakenly answering an unanswerable question and by not answering an answerable question, we show the performance of the models on the subset of answerable questions that the model also chose to answer on the validation set in Table 4. This should reflect the performance of the models in SQuAD 1.1, where not answering a question is never a favorable option. Eliminating all mistakes made about whether to answer a

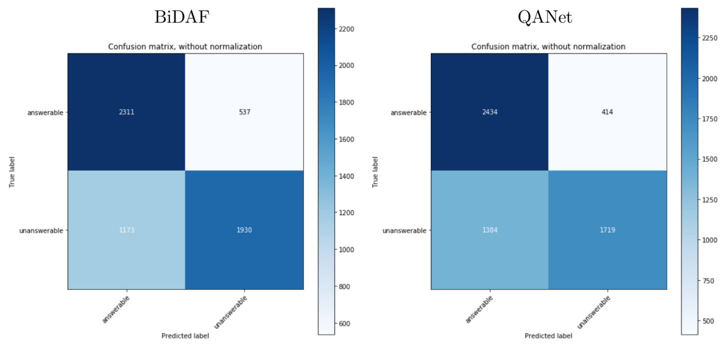

Figure 5: Confusion matrices for BiDAF and QANet, for the model's decision to answer a question.

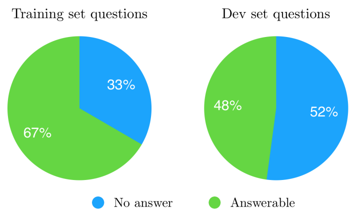

Figure 6: Proportion of answerable questions in the training and dev splits.

question, the F1 score reached 83.7 for BiDAF and 80.6 for QANet, which suggests that the most important aspect to improve on is correctly deciding whether to answer a question.

To further understand the role of each type of mistakes (answering an unanswerable question, and not answering an answerable one), confusion matrices are plotted for both models in Figure 5. Answering an unanswerable question was the greatest source of confusion. This observation inspired us to reweight the probability of no-answer in the discretization layer, explained in section 4.4.7.

5.3 Share of unanswerable questions in the training and dev sets

The above observation also led us to check the distribution of answerable questions in the training and dev set. The dev set has proportionally many more unanswerable questions compared to the overall distribution and that of the training set (Figure 6). The difference in distribution could partially explain why our models did not decline to answer often enough, since in the training set there are many fewer unanswerable questions. It could also explain to some extent that the reweighting of the no-answer probability in section 4.4.7 did not lead to comparable F1 score on the test set.

6 Conclusion

This project compared the performance of RNN-based BiDAF model and CNN and self-attention based QANet on the Reading Comprehension task on SQuAD 2.0, which is challenging since one third of the questions do not have an answer in the context paragraph. We implemented QANet following the same high-level architecture as the BiDAF model, so that we can switch out the embedding encoder and model encoder layers of the networks. We then built and trained two hybrid models, observing that the hybrid with BiDAF's encoder blocks as the embedding encoder achieved better performance than the counterpart configuration. We analyzed the major cause of the low F1 scores, concluding that answering unanswerable questions was the type of mistake that hurts the F1 score the most. We then reweighted the probability of the start and end word probabilities favoring not answering, which increased the F1 score by 1.7 for QANet and 0.96 for BiDAF on the dev set.

A first step in future work could be to carry out hyperparameter sweeping for the two hybrid models so that their optimal performance can be compared, instead of using one set of hyperparameters across all four models. The imbalance of proportion of unanswerable questions in the dev set could also be addressed so that the dev set distribution reflects the overall distribution better.

References

[1] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, *Squad: 100,000+ questions for machine comprehension of text*, 2016. eprint: `arXiv:1606.05250`.

[2] P. Rajpurkar, R. Jia, and P. Liang, *Know what you don't know: Unanswerable questions for squad*, 2018. eprint: `arXiv:1806.03822`.

[3] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, "QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension," *arXiv.org*, arXiv:1804.09541, Apr. 2018. arXiv: `1804.09541 [cs.CL]`.

[4] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional Attention Flow for Machine Comprehension," Nov. 2016. arXiv: `1611.01603`.

[5] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *NIPS 2014 Deep Learning and Representation Learning Workshop*, 2014.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: `10.1162/neco.1997.9.8.1735`. eprint: `https://doi.org/10.1162/neco.1997.9.8.1735`. [Online]. Available: `https://doi.org/10.1162/neco.1997.9.8.1735`.

[7] W. Wang, N. Yang, F. Wei, B. Chang, and M. Zhou, "Gated self-matching networks for reading comprehension and question answering," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2017, pp. 189–198.

[8] M. Hu, Y. Peng, Z. Huang, X. Qiu, F. Wei, and M. Zhou, "Reinforced mnemonic reader for machine reading comprehension," in *International Conference on Learning Representations*, 2017.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[10] R. K. Srivastava, K. Greff, and J. Schmidhuber, *Highway networks*, 2015. eprint: `arXiv:1505.00387`.

[11] F. Chollet, *Xception: Deep learning with depthwise separable convolutions*, 2016. eprint: `arXiv:1610.02357`.

[12] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, *Deep networks with stochastic depth*, 2016. eprint: `arXiv:1603.09382`.