You Just Want Attention

Stanford CS224N Default Project | Track: SQuAD
Mentor: Ethan Chi

Pranav Jain
Department of Computer Science
Stanford University
`jpranav@stanford.edu`

Swastika Dutta
Department of Computer Science
Stanford University
`swastika@stanford.edu`

Abstract

Question Answering is a major domain of interest in the intersection of information retrieval and natural language processing, mainly due to its wide application in search engines, chat systems, robotics, medical and literature. In this project, we build a QA model based on one of the most popular NLP datasets, Stanford Question Answering Dataset (SQuAD) 2.0.

We implement a robust QANet model from scratch, and experiment with its parameters and architecture. We analyze the effect of different embeddings, and compare the efficiency of attention-based QANet and BiDAF (Bi-Directional Attention Flow) based models. Finally, we will develop a weighted majority voting-based ensemble model to improve the overall performance of different base model variations. We achieve a F1/EM score of 71.63/68.85 in the dev set (Rank 5), and 69.42/66.46 in the test set (Rank 6) of the CS224N IID SQuAD Track leaderboard.

1 Introduction

Complex tasks like contextual question answering caps the representational efficacy of present NLP models. In this paper, we implement and analyze QANet model, and introduce improvements to the baseline model. The QANet model handles the long term dependencies and context in the input for complicated tasks like question answering and hence, it tackles a very important NLP problem. This approach serves as a faster alternative to its counterpart — RNN and reduces time overhead [1]. Further, QANet uses different components which can be parallelized and the output layer can be modified to suit different transformer use cases such as sentiment analysis [2], machine translation [3], and other NLP tasks.

Even though the QANet has remarkable performance, we notice certain drawbacks of the original research work. This model assumes that end of the span is independent of start. Other researches, such as DCR [4], show good results with alternate approach. The conditional output, i.e., end position is considered a function of start, might be particularly helpful for handling unanswerable questions since the model predicts a reasonable overall span of the 'answer', and not start and end indices independently. Also, this paper does not discuss the limitations of the model, nor analyze wrong predictions made by the model. Such analysis helps to probe systematic errors made by the model and formulate potential improvements. Further, GloVe leverage count-based global co-occurance to generate embeddings, i.e., it assumes similar words occur in similar contexts. fastText uses n-gram model to generate embeddings and can handle out-of-context vocabularies better. Thus, it might perform better than GloVe for open-domain contextual questions like SQuAD 2.0.

In brief, we adopt the strong features of QANet architecture and aim to improve its shortcomings to build a QA model for SQuAD 2.0 dataset.

2 Background and Related Work

The original QANet paper [5] proposed a novel architecture to effectively handle automated context-based question answering.
The previous works in this domain had leveraged the use of RNNs due to inherent sequential nature of text. Specifically, even the state-of-the-art neural models, including BiDAF [6], r-net [7] and ReasoNet [8] have been built on Recurrent Neural Networks (RNN).
There are predominantly two important drawbacks of RNN-based models:
(1) Time-consuming training and inference, and prevent parallel computation due to sequential nature
(2) Inadequate performance in modeling long term dependencies
Some proposals to improve efficiency of RNNs are Gated Recurrent Units [9], Long Short Term Memory [10] and Reinforcement Learning methods [11]. However, their ability to handle complicated tasks such as question answering remain questionable.

To tackle these problems, full convolution architectures had been proposed for sentence classification [12] and sequence to sequence learning [13]; and full attention architectures such as Transformers[14] and DiSAN [15] for encoding-decoding, handling dependencies and other tasks. These models have gained popularity for their fast and highly parallelizable computation, flexibility in modeling dependencies and precision. The QANet builds on these approaches to develop a new robust feed-forward model with parallelized data loaders. Its architecture comprises solely of convolution to capture local interactions and self-attention to capture global interactions between word pairs.
Data augmentation is widely used in NLP to fabricate slightly modified synthetic data to regularize model parameters and train robust generalizable models. Some prominent efforts are synonym–based [16], type swap [17] and backtranslation [18] data augmentation methods. The original QANet paper uses backtranslation technique to introduce syntactical diversity in the enhanced data and achieves better results than its counterparts. However, in our reimplementation, we introduce diversity by simplifying the complex questions, as discussed in Analysis Section.

Finally, the original QANet model handles start and end indices of answers as separate entities. However, we take inspiration from the work on Match-LSTM and Answer Pointer [19] and develop a conditional outer layer where the end index is a function of the start index.

QANet achieves impressive scores in SQuAD 1.1 dataset and serves as a great launching pad for handling SQuAD 2.0. In this paper, we extend QANet to handle unanswerable questions in SQuAD 2.0, improve its output layer and incorporate active learning training methods.

3 Approach

We build two classes of models: BiDAF (sourced from base code given in CS224N [20]) and QANet (based on research work of Adams et al.[5].)

3.1 BiDAF

This model is based on [6] and serves as strong baseline. We implement and compare the following variations of this model:
1. Word Embeddings: We train on the model on another popular non-contextual word embedding, fastText [21] and compare the results.
2. Character embedding: We incorporate 200-D character embedding in the model architecture.
3. Conditional Output Layer: We examine the effect of conditioning end prediction on start prediction [22], by drawing inspiration from Match-LSTM and span representations [4].

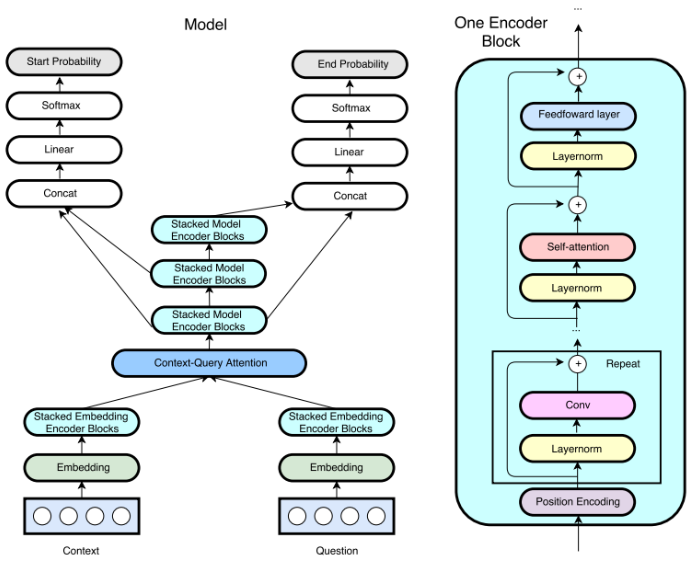
3.2 QANet

We have implemented the QANet model architecture from scratch, based on the research work QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension [5]. We adopted some auxillary functions from [20] and took inspiration from GitHub Repository [23].

QANet inputs context paragraph (C) of n words and query sentence (Q) of m words, and outputs the answer to the question (span S from context paragraph).

It is comprised of five major components:

Figure 1: Architecture of QANet Model



(1) Input embedding layer — Each word w in context and query is represented by concatenation of its GloVe word embedding and convolution output character embedding. This vector is passed through a two-layer highway network to produce x. Highway networks [24] have gated architecture which regulate flow of information through network and eases gradient-based training of very deep networks. They allow unimpeded information flow across several layers on "information highways".

(2) Embedding encoder layer — It is a stack made of depthwise separable convolution layer [25], multiheaded self-attention layer[26] and feed-forward layer, as shown in Figure 1.

Each of these building blocks is placed inside a residual block for better gradient conduction, and layer normalization [27] is applied for hidden state stabilization. This layer is found in input embedding and in the core of the model, as shown in the QANet architecture.

(3) Context-query attention layer — Similarity matrix S stores the trilinear similarities [28] between each context-query word pairs. Context-to-query attention $A = \bar{S}.Q^T$, where $\bar{S}$ is the normalized softmax of S. Following BiDAF and DCN [29] strategy, query-to-context attention $B = \bar{S}.\bar{\bar{S}}^T.C^T$, where $\bar{\bar{S}}$ is column normalized S. This layer evaluates the interrelation and information flow from the context to corresponding question.

(4) Model Encoder layer — This layer inputs $[c, a, c \odot a, c \odot b$, where a and b are rows of attention matrices A and B respectively. The parameters are similar to those of embedding encoder layer, however there are more convolution layers in this stage.

(5) Conditional Output layer — This layer calculates the probabilities for start and conditional end positions of answer span as $p^1 = softmax(W_1[M_0; M_1])$ and $p^2 = softmax(W_2[W_1; M_0; M_2])$, where W_1 and W_2 are two trainable variables and M_0, M_1 and M_3 are the outputs of the encoders. The score of a span is $p^1.p^2$ and thus, the model outputs s, e in linear time, such that $s \leq e$ and $p^1_s.p^2_e$ is maximized.

Due to time and resource constraints, we could not implement entire Match-LSTM layer. However, we show that our simple conditional layer shows improvements in the overall madel performance.

Finally, we experiment with the following variations of the model:
1. Word and character embeddings: We train and compare the results of fastText and Glove embeddings.
2. Conditional Output layer: As in BiDAF model.
3. Learning Rate: We will implement learning rate annealing and run our model on varying learning rates.
4. Larger QANet We examine the effect on increasing the model size on the performance of the base

model. We will also experiment with different types of regularizations, namely L1, L2 and dropout, to prevent overfitting in these models.
5. Active Learning and Data Augmentation: Described in further details in Analysis Section.

3.2.1 Emsemble Model

Finally, we will develop an ensemble model to achieve a better compounded performance than the trained models. We will use majority voting for our model, with each base model having one vote. In case of ties, we use majority weighted voting as follows: the vote of each model is weighed by their confidence score (which is proportional to dev F1 score), and then we return the vote having highest sum of weights from all base models.

4 Experiments

4.1 Data

We use the custom dataset provided in the course, which is based on SQuAD 2.0 consisting of both answerable and unanswerable (context, question, answer) triples. Specifically, we have training set of 129,941 examples, dev set of 6078 randomly selected examples and test set of 5915 examples.

The input to our model is a paragraph and a question which may or may not be answerable based on the context. Our model should output span of text from context paragraph as answers, or, classify the question as unanswerable.

4.2 Evaluation method

The performance of our model is measured by two broad metric systems: (1) Exact Match: Binary output based on exact match with ground truth , (2) F1 score: Harmonic mean of precision and recall. We take maximum of F1 and EM scores across the three human-annotated ground truths for each question, and then average these for the entire dev or test datasets.

Specifically, we use these metrics to navigate through experiments:
(1) Overall EM and F1 scores (as described in handout)
(2) EM scores for answerable questions
(3) F1 scores for answerable questions
(4) Answer vs. No Answer (AvNA).

Unanswerable questions are new additions to SQuAD 2.0 and forms a major portion of the dataset. Computing scores for unanswerable (Metric 4) and answerable questions (Metric 2 and 3) separately helps us to see the relative performance of our models for these sub-groups and fine-tune the model accordingly. On the other hand, overall scores (Metric 1) determines the final performance of the resultant model.

4.3 Experimental details

We trained the following models configurations with given specifications:
1. BiDAF model with Glove Embeddings w/o character embedding: The parameters in our model is exactly the same as that of handout repository [6]. It was trained for approximately seven hours on Azure NC6_v3 (30 epochs) with batch size 16 and hidden dimension 128, learning rate 0.5, dropout rate 0.2, EMA 0.99 and Adadelta optimizer.

2. BiDAF model with Glove Embeddings w character embedding: We use the same parameters for our model with 200D character embeddings with character filter 16. As in the original BiDAF architecture [6], character-level embedding layer is expected to provide more flexibility, better handling of out-of-vocabulary words and morphology.

3. BiDAF model with FastText Embeddings w/o character embedding: We use similar configurations for all the BiDAF models for fairer comparison of the models architecture and embeddings.

4. BiDAF model with FastText Embeddings w character embedding: Similar to the previous model.
4. BiDAF model with FastText Embeddings w character embedding and Conditional Output Layer: Similar to the previous model.

5. QANet base model with FastText Embeddings: After observing better results with fastText vectors and character encoding in BiDAF models, we extend its experimentation in QANet models. We use character encoding in all our QANet models, as recommended in [5] and ablation method to evaluate our outer conditional layer. Our model takes around 16 hours to train on Azure NC6_v3 GPU.
We train our QANet on various hyperparameters with batch sizes (16 32, 64), hidden dimensions (64, 128), encoder layers (2, 4, 6), exponential step warm up learning rates (0.001 to 0.2), dropout probability (0.01 to 0.1), attention heads (4, 8), Adam optimizer and L2 regularization of 3e-7 on bigger QANet models.
5. Ensemble Model: We select 7 best-performing QANet models and 2 BiDAF models and develop an ensemble method on resultant output start and end indices.
We select the 'best' models in the following manner: We evaluate the F1*, EM* and AvNA scores of each trained QANet variant. Finally, we choose two models having highest F1*, one model with highest EM*, two models with highest overall F1 and two models with highest AvNA score. Finally, we include two character-embedded BiDAF models with conditional output, where one is trained on GloVe embeddings and other is trained on fastText embeddings.

4.4 Results

GloVe vs fastText embeddings on BiDAF model:

The Figure 2 shows the tensorboard visualizations for training on BiDAF with the two different embeddings. We notice that the train negative log-likelihood (NLL) is approximately similar for both embeddings. However, the fastText embeddings show better results in dev set, and hence, can be said to be more generalizable. Specifically, it shows slightly better performance for answerable questions in EM, F1 and NLL as inferred from Table 1 and Figure 2.

Figure 2: Tensorboard visualization of dev AvNA, EM, F1 and NLL for BiDAF models with GloVe (orange) and fastText (blue) embeddings

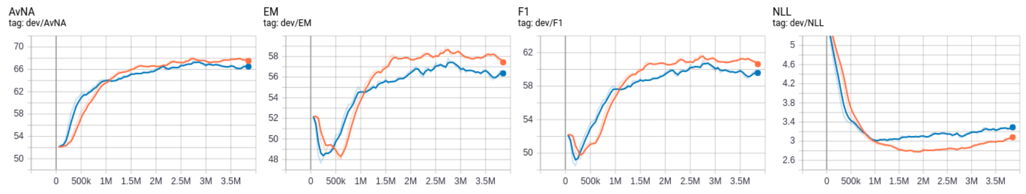


Table 1: Comparison of GloVe vs fastText on SQuAD dev set

Model	AvNA	Overall EM	Overall F1	EM *	F1 * [1]
BiDAF with GloVe	66.5	56.2	59.2	59.1	57.5
BiDAF with fastText	68	57.8	61.3	59.9	60.3

BiDAF w character embeddings vs BiDAF w/o character embedding:

We use ablation method to decide the necessity of character embedding in our model. Overall, we see that the BiDAF architecture with character embeddings outperforms the other variant in both GloVe and fastText models, as in Table 2.

Table 2: Analysis of character embeddings in BiDAF

Model	AvNA	Overall EM	Overall F1
BiDAF w character embedding	68.1	58.1	61.4
BiDAF w/o character embedding	66.5	56.3	59.5

BiDAF w conditional output vs BiDAF w/o conditional output:

[1]EM* and F1* refer to EM and F1 for answerable questions in SQuAD 2.0 dev set

We changed the output layer described in [20] to handle conditional output span, as described in approach section. We compared the performance of BiDAF model with fastText embedding with conditional output, as summarized in Table 3.

Table 3: Analysis of conditonal output layer in BiDAF

Model	**AvNA**	**Overall EM**	**Overall F1**
BiDAF w conditional output	68.1	58.2	61.3
BiDAF w/o conditional output	68.7	58.3	61.5

From analysis of BiDAF models, we observe the improvement of model by addition of character embeddings, final conditional output layer and use of fastText embeddings. We extend these learnings to our QANet models. To verify this in constrained timeline, we used ablation method on QANet models with smaller training datasets.

Variations of QANet: We analyze TensorBoard graphs for different QANet models, and we observe that QANet models with higher hidden size and batch size perform better than its counterparts. Also, we notice minor deflections in performance due to changes in number of attention heads and number of encoder blocks. We also note that learning rate is an important factor and model performance decreases when learning rate deflects from 1e-7.

Table 4: Results of different QANet models

Model	**AvNA**	**Overall EM**	**Overall F1**
QANet with 128 hidden size, batch size 64	75.2	65.18	69.88
QANet with 128 hidden size, batch size 64, L2	**75.53**	**65.78**	**70.13**
QANet with 128 hidden size, batch size 32	75.13	64.9	69.11
Larger QANet with 128 hidden size, batch size 32	74.5	64.5	68.99
QANet with 128 hidden size, batch size 32, 4 attention heads	74.2	64.3	69.33
QANet with 64 hidden size, batch size 64	72.3	64.3	68.99
QANet with 64 hidden size, batch size 32	72.9	64.1	68.79

Ensemble:

We use weighted majority voting ensemble model on the selected models, as described in Experimental Details Section.

Table 5: Analysis of Ensemble Models

Model	**AvNA**	**Overall EM**	**Overall F1**
Vanilla Majority Voting	76.13	67.13	71.11
Weighted Majority Voting	76.55	68.85	71.63

5 Analysis

We analyzed the data in dev set and the structure of questions. Some questions were in passive voice. while others were active (started with why, what, and other question types).

Table 6: Analysis of question types in dev set

Question type	What	How	Where	Why	Who	When	Others
Simple examples starting with question type	907	68	37	4	75	106	4881
Total examples with question type	3726	590	257	91	688	427	289

The number of examples were small to make any concrete conclusion. However, we observed that our QANet model performs better (by approximately +1.5 F1 score) on most 'simple' question types in comparison to 'complicated' (or passive) questions. To improve the performance of our model on such question, we use **data augmentation**. We ‘simplify’ such questions by using nltk parsing, with limited success. We took inspiration from GitHub Repository [30] for this exercise, and add the successful simplifications to our dataset with same uuid. While predicting answers, we choose the answer span from context with lower NLL loss.

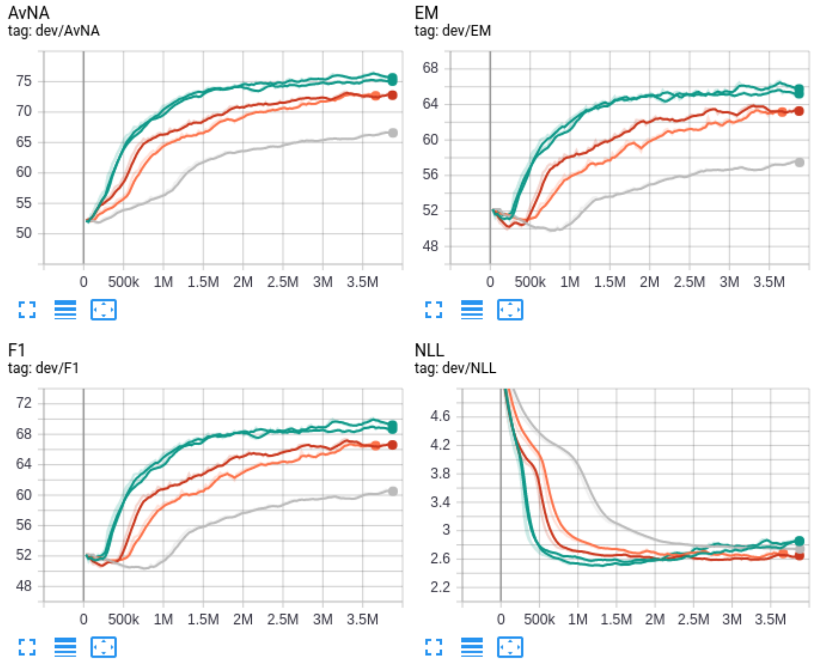
Table 7: Analysis of answer positions in dev set

Answer position	No Answer	First line of context	Later lines of context
Number of examples	3168	918	1992

On further analysis, we find that our model has better F1 score on examples where the answer is present in first line of context, irrespective of question type. Since this ‘difficulty’ is dependent on the context rather than the question, there is a correlation between batch NLL and difficult contexts. To handle these type of inputs, we use **active learning** during our training. We maintain 1000 batches with maximum batch NLL during each epoch in training, and feed these batches to our model again at the end of the epoch. We tested this method on smaller train datasets and noticed that this method introduced initial randomness in our train NLL loss. However, our dev NLL loss improved after approximately 5 epochs and hence, we extended this method in final training over complete train dataset.

Finally, we analyze the dev TensorBoard curves for our QANet models with different hyperparameters. As evident from Figure 3, impact of hidden size and learning rates dominates over the impact of other factors such as number of encoder blocks, regularization, attention heads and others. In fact, most of the base models in our final ensemble model had hidden layer 128. Two QANet models with 64 hidden size performed well on EM* and AvNA metrics, and were included in the ensemble to provide more variety in votes.

Figure 3: Effect of hidden size and learning rate dominates other factors such as L2 regularization, attention heads, encoder layers. Models with hidden dimension 128 performs similarly (corresponding to green lines). Models with hidden dimension 64 has lower maximum performance cap. Grey line shows model trained with hidden size 128 and learning rate 3e-4



6 Conclusion

In this project, we implement a QANet model with conditional output layer, and train it with active learning and data augmentation. We analyze the impact of different hyperparameters of the model and evaluate our models on comprehensive evaluation metrics.

Our QANet model achieves a promising scores: 70.13/67.31 dev set F1/EM. Our weighted majority voting ensemble model improves the base QANet performance to 71.63/68.85 dev set F1/EM (Rank 5 on leaderboard), and 69.42/66.46 test set F1/EM (Rank 6 on leaderboard).

One of the main drawbacks of our current implementation is the absence of data augmentation by backtranslation. We could not incorporate backtranslation due to time constraints, however it will serve as an inspiration for future work. It would also be interesting to test our model with BERT embeddings and gauge its performance against other state of the art models. Finally, our conditional output layer can be further improved further by adopting the output layer from Match-LSTM [19].

References

[1] Shigeki Karita, Nanxin Chen, Tomoki Hayashi, Takaaki Hori, Hirofumi Inaguma, Ziyan Jiang, Masao Someki, Nelson Enrique Yalta Soplin, Ryuichi Yamamoto, Xiaofei Wang, Shinji Watanabe, Takenori Yoshimura, and Wangyou Zhang. A comparative study on transformer vs rnn in speech applications. *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Dec 2019.

[2] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. Disan: Directional self-attention network for rnn/cnn-free language understanding, 2017.

[3] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data, 2016.

[4] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering, 2018.

[5] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension, 2018.

[6] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[7] Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. May 2017.

[8] Yelong Shen, Po-Sen Huang, Jianfeng Gao, Weizhu Chen, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. In *23rd SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, August 2017.

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

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

[11] Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end answer chunk extraction and ranking for reading comprehension, 2016.

[12] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[13] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning, 2017.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[15] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. Disan: Directional self-attention network for rnn/cnn-free language understanding, 2017.

[16] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification, 2016.

[17] Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. Neural question generation from text: A preliminary study, 2017.

[18] Li Dong, Jonathan Mallinson, Siva Reddy, and Mirella Lapata. Learning to paraphrase for question answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 875–886, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[19] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *ArXiv*, abs/1608.07905, 2017.

[20] GitHub Repository. michiyasunaga/squad. `http://https://github.com/michiyasunaga/squad/`, 2022. [CS224N Default Project IID Track code].

[21] GitHub Repository. Facebook Research. `https://github.com/facebookresearch/fastText`. [fastText embeddings].

[22] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer, 2016.

[23] GitHub Repository. BangLiu. `https://github.com/BangLiu/QANet-PyTorch/`.

[24] Rupesh Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. 05 2015.

[25] Lukasz Kaiser, Aidan N. Gomez, and Francois Chollet. Depthwise separable convolutions for neural machine translation, 2017.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[27] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[28] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[29] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering, 2018.

[30] GitHub Repository. pass2act. `https://github.com/DanManN/pass2act`.