Machine Reading Comprehension

Stuti Sakhi*, Yerania Hernandez†
Department of Computer Science and Engineering
Texas A & M University
*Email: stuti@tamu.edu
† Email: hernandez.yerania@tamu.edu

***Abstract*—Machine Reading Comprehension (MRC) allows the opportunity for a system to understand a given context and provide an answer based on query on this context. In this paper, we explore a variety of methods that have been developed to provide high F1 and exact match scorings on the SQuAD data set with the goal of implementing a multi-layer model that combines a variety of these features. We provide flexibility in configuring a number of parmeters in order to analyze the different configurations of networks. Our evaluation and results demonstrate that our best model is a combination of GLoVe embeddings, an LSTM approach to RNN, a BiDAF weight attention, and a smart span method to provide the best answer possible.**

I. Introduction

Machine Reading Comprehension (MRC) provides the ability for computers to read and understand natural language text and it can be considered as one of the necessary abilities for artificial intelligence [1]. In order to train a system in answering a query about a given paragraph, the model will need to be able to create complex interactions and connections between the context provided and the query. Being able to extract such information definitely has its caveats, but could be beneficial for a number of domains, including improving search engines in documents that are domain-specific and providing support when answering customer service inquiries. Systems from previous works have achieved promising results, but are typically characterized by using attention weights on summarized context, the attention weights are computed and dependent on previous steps, and attention layers are computed unidirectional, usually from query to context.

In this paper, we focus on exploring and analyzing a variety of networks in order to find the best model to provide an adequate answer to the query given. The generic idea of our network is to provide an embedding layer, an encoder layer, an attention layer, and the final output layer. Each of these layers has a variety of features that were evaluated, including comparing word and character embeddings, the addition of highway layers, comparing Gated Recurrent Units (GRU) and Long Short Term Memory (LSTM) performance, using a unidirectional attention layer versus a Bi-Directional Attention Flow (BiDAF) layer, and finally using softmax to predict the span based on maximizing the probability distribution. Each of these features has demonstrated promising results individually and therefore, we combine a number of these features in order to develop the best model to solve the MRC challenge. Our final model is able to outperform a number of previous approaches from the SQuAD test set leader board. The rest of the paper is dedicated to describing a few previous approaches, further details on the data set we used and the approach we took for each layer, the evaluation metrics we used, and the final results for the number of models we used along with future work for this task.

II. Related Work

In order to tackle the MRC challenge, a major contributor to developing these models has been the number of large datasets that have been developed and made available. In 2013, MCTest was too small of a dataset in order to be able to train an end-to-end model [8]. In the recent years, industry has provided additional attention to artificial intelligence and as result machine comprehension has become a strong factor in understanding natural text, which lead to CNN/DailyMail dataset in 2015 [5]. However, by 2016, the Stanford Question Answering Dataset (SQuAD) was published providing an extensive dataset with a large number of questions, answers, and a wide-ranging of topics [6].

Typical end-to-end architectures have used a variety of attention mechanisms, one of which depends on the previous step for attention weights. Bahdanau et al. uses this approach referred to as dynamically updating the weights [3]. However, Hermann et al. and Chen et al. demonstrate that the accuracy of the model can increase if a bilinear term is used in order to compute the weights [5]. Furthermore, Kadlec et al. feeds the attention weights after only computing them once into the output layer without depending on previous weights [7]. Each of these approaches are only unidirectional, where the answer is derived from the correlation between the query to context and consider the answer is a single token, which is not appropriate for the SQuAD data set. The BiDAF research paper describes the ability of performing an attention layer from context to question and question to context, which demonstrates a better performance than the previous unidirectional approaches [9]. In addition, Wang & Jiang explored an LSTM architecture for natural language inference (NLI) and demonstrated how the network provides emphasis on word-level matches and relationships between words [12]. Inspired from LSTM networks, highway layers have proven to overcome the difficulty of training a network the deeper it increases considering deepness is crucial to a network's success [10]. Based on these features that have demonstrated improvements to current machine comprehension tasks, we

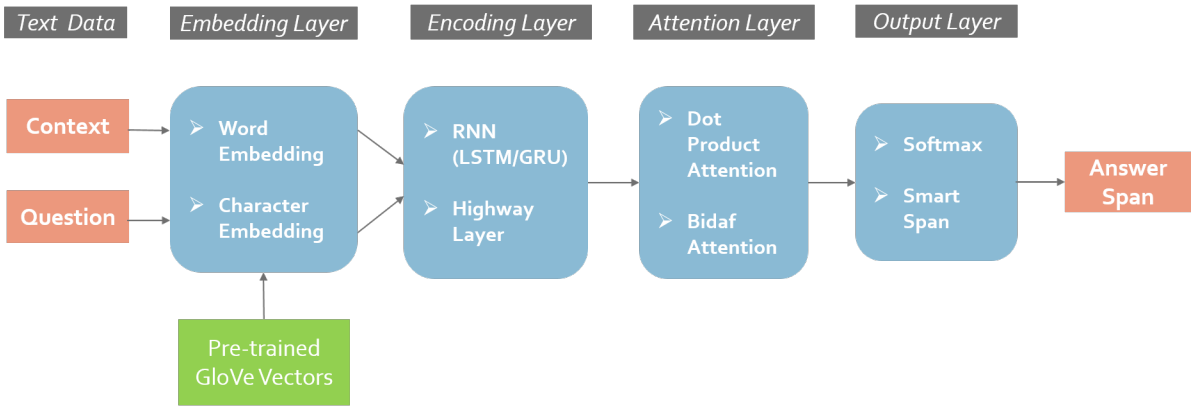

Fig. 1. An overview of the model approach and features implemented within each layer

implemented a model that provided flexibility in using a combination of these aspects in each layer.

III. Approach

Our MRC model is a multi-layer network that consists of four layers as shown in Figure 1: embedding layer, encoding layer, attention layer, and output layer. Each of these layers has a variety of features and complexity that we have implemented based on previous works in order to improve the performance of the model.

A. Dataset Analysis *[Yerania Hernandez]*

The data set used for this model was the Stanford Question Answering Dataset (SQuAD). This data set was created with the help of crowd workers on a set of Wikipedia articles. The questions were asked by the crowd workers and the answers can be found in a given segment of text. With over 100,000 question and answer pairs, it provides our team a large enough data set to train and test our model. Unlike other data sets, SQuAD provides a diversity of questions considering it is larger than most existing data sets and it does not take the simple approach of just providing answer choices from where the machine picks from. Instead, the system needs to use logical inference in order to answer the questions from the passages provided. Considering SQuAD uses Wikipedia articles, the data set is clean compared to other document systems, having no spelling mistakes and the answer is always in the passage.

With further analysis on the train data set, we were able to select our hyperparameters when initializing our network. In the train context data set, we discovered that 95th percentile of the distribution in the context length was 245 words and 99th percentile of the context length is 325 words. As a result, our context length for the network was 300 words. In the train question data set, we observed that that the distribution of the length of questions ranged from 18 to 23 in the 95th and 99th percentile, respectively. However, when analyzing the question length with respective to the number of questions, the majority of questions range from 20 to 40 words. As a result, our question length for the network was 30 words. In the train answer data set, we further analyzed the length of answers and the distribution of the answer span based on the start index. This provided further insight in selecting an answer span length for our final output considering the 95h percentile and 99th percentile of the distribution was between 11 to 21 words. In order to obtain this final answer, we further describe our output layer in later sections.

B. Preprocessing *[Yerania Hernandez]*

Before adding and describing the layers we created for our model, we filtered through the dataset in order to separate the context, the questions, the answers, and the answer spans into separate files. This preprocessing aspect of our dataset was essential in order to standardize all the data, such as additional symbols or punctuations and the aligning the indexes of the span with the actual context. After verifying that the answer from the context based on the answer spans correlates with the actual answer provided from the documentation, we were able to create tuples of the context, question, answer, and answer span. Each tuple was further divided into its respective file for further usage in the actual network.

C. Embedding Layer *[Stuti Sakhi]*

The first layer in the model is the embedding layer. The embedding layer converts text into numerical vectors. This step is crucial because most of the machine learning techniques, including Neural Networks require numbers as input in order to perform any sort of job. There are various techniques existing for embedding including word embedding as well as character embedding. Word embedding directly convert word to vectors, while character embedding combine the embedding of each character in the word to get the word embedding. Below, we discuss the techniques we used in detail.

1) Word Embedding: Word Embedding is used to generate vector representations for the words in the vocabulary. There are two broad categories of word embedding techniques - frequency based and prediction based. Frequency based techniques decompose the word co-occurrence matrix to derive the vector representations. It majorly relies on the global count statistics to arrive at the vectors. On the other hand, prediction

based techniques take only local context into account and learn word embedding which capture meaning in the vector space. Considering the global count statistics and learning dimensions of meaning both are equally crucial in deriving word embedding. Keeping this in mind, we went ahead with using GLoVe Vectors for the word embedding as it combines the best of both worlds. The GloVe vectors are learned by optimising a loss which uses global count statistics information. We used the pre-trained GloVe vectors of dimension 100 for our project. So each word in the dataset(context and question) was converted to a vector using these pre-trained GLoVe vectors.

2) Character Embedding: Character embedding is another way to convert text to vectors. Here we have a vector for each character in our dataset. To get the word embedding we combine the character embedding of the constituent characters. Character embeddings have less dimension as the number of characters are limited. Moreover, character embeddings help us to utilise the internal structure of the word and also handle out of vocabulary words. We used the character level Convolutional Neural Network(CNN) [9] to learn the character embeddings.

For the character level CNN, we start with representing each character with trainable character embeddings c_1, c_2, ..,c_l. Now each word can be represented as e_1, e_2,.., e_k using it. Now, these word representations are input into a 1 dimensional CNN to get the hidden state embedding of the words.The idea is that, each hidden vector is a combination of a window of characters due to the convolution. Both the embedding and filter for the CNN are learned during the training of this model.

D. Encoding Layer ***[Stuti Sakhi]***

Once we had the vector representation of words, our next step was to make each word aware of the words before it and after it. This is a very important step as words individually only give partial information. It is the sentence which actually holds the complete meaning. To make each word aware of its context, we used a bidirectional Recurrent Neural Network. We also used the highway layer as a part of the encoding in some of the cases to model higher complexity. Now, we discuss each of these in detail

1) Recurrent Neural Network: A recurrent neural network(RNN) which is capable of handling sequential data. The hidden state in the RNN, remembers the previous instances and hence can handle sequential data. Figure 2 shows an unrolled RNN. Here, we can view inputs, X_1, X_2,.., X_n as the word embedding of a sentence (context and answer in or case). Each of the hidden state H_t is found using the present input Xt and previous hidden state H_{t-1}. Equation 1 below shows this relation. W_{hh} and W_{xh} are learned during the training.

$$H_t = tanh(W_{hh}H_{t-1} + W_{xh}X_t) \tag{1}$$

RNN, in theory can remember information which was processed long back. In reality however, they face an issue of diminishing gradient which does not allow them to do so. For this reason we decided to work with two modifications of RNN, Long Short Term Memory(LSTM) and Gated Recurrent

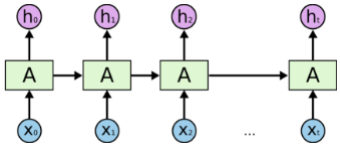

Fig. 2. Recurrent Neural Network

Unit(GRU). Both of these add extra gates to the RNN, thus making it easier for information from past to pass.

A GRU uses an update gate and a reset gate. The update gate decides on how much of information from the past should be let through and the reset gate decides on how much of information from the past should be discarded. On the other hand, LSTM has three gates forget gate, update gate and output gate. Hidden state is computed using the forget gate and update gate. The output gate determines how much much representation the hidden state must have in the output. LSTM and GRU are complex models when compared to a vanilla RNN. However, the added computation allows us to capture information from long sequences. Generally LSTM captures more information from past when compared to GRU. We compared the performance of both of these to find a better fit for out data set.

To make sure words are aware of other words both from left as well as right, we used a bidirectional LSTM/GRU. Bidirectional LSTM/GRU just concatenate the the hidden states of two LSTM/GRU in opposite directions.

2) Highway Layer: Since this is a complex problem, we decided to add more non linearity to the model by adding fully connected layers. However, as our model was already deep, a fully connected layer would not perform well and we might end up losing valuable information from the starting layers. For this reason we decided to add in a highway layer [4]. A highway layer is inspired by LSTM. It has 2 gates transform gate and the carry gate. The transform gate decides how much of the input must be represented by the non linear transformation while the carry gate decides how much of the input must be passed as it is to the next layer. Lets suppose x is the input to the highway layer, then the output y can be written as shown in equation (2). Here, H(tranformed input) is a function of W_h, T(tranform gate) is a function of W_t and C(carry gate) is a function of W_x. All the three W_h, W_t and W_x are learned during the training.

$$y = H(x, W_h).T(x, W_t) + x.C(x, W_x) \tag{2}$$

We tried out various combination in this layer GRU and LSTM with and without the highway layer.By the end of this layer, we have the hidden state vectors for both context and the question.

E. Attention Layer ***[Stuti Sakhi]***

Now that we have both the context and question hidden layers, our next step would be to understand which part of the context is relevant to the question. This layer is called attention as it decides where in the context we need to give attention in

order to answer the given question. Intuitively speaking, this layer will output a vector which assigns weights to each word in the context according to their relevance with the question. We tried out two attention mechanisms, Dot Product Attention and Bidirectional Attention flow in our implementation.We discuss the two in detail below.

1) Dot Product Attention: Dot Product Attention is one of the most primitive attention mechanisms. Lets denote question encodings as q_1,q_2,..,q_M and context encodings as c_1, c_2,...c_N. We evaluate the attention distribution ^{i} for each context state c_i as follows.

$$e^i = [c_i^T q_1 \;\; c_i^T q_2 ... c_i^T q_M] \tag{3}$$

$$\alpha^i = softmax(e^i) \tag{4}$$

Now we take the weighted sum of the question hidden states q_i to get the attention output ai for each context state c_i.

$$a_i = \sum_{j=1}^{M} \alpha_j^i q_j \tag{5}$$

Then finally we concatenate each of these a_i to c_i to get the final attention output hidden state b_i

$$b_i = [c_i; a_i] \tag{6}$$

There is no learning involved in this Dot product attention.

2) Bidirectional Attention Flow Mechanism: Bidirectional Attention Flow is a high performing attention mechanism [9]. It is based on the idea that attention must flow both ways, from question to context as well as context to question. Lets denote question encodings as q_1,q_2,..,q_M and context encodings as c_1, c_2,...c_N. We evaluate each element S_{ij} of the similarity matrix S such that

$$S_{ij} = W_s^T [c_i; q_j; c_i \cdot q_j] \tag{7}$$

Here W_s is learned during the training. Now, first we perform Context to Question attention. For each context hidden state c_i we evaluate a_i according to the following equations.

$$\alpha^i = softmax(S_{i:}) \tag{8}$$

$$a_i = \sum_{j=1}^{M} \alpha_j^i q_j \tag{9}$$

Now, for the Question to Context, firstly for each i from 1 to N we find m_i. All the m_i are concatenated to get m.Then c', the question to context coefficient is evaluated.

$$m_i = max_j(S_{ij}) \tag{10}$$

$$\beta = max(m) \tag{11}$$

$$c' = \sum_{i=1}^{N} \beta_i c_i \tag{12}$$

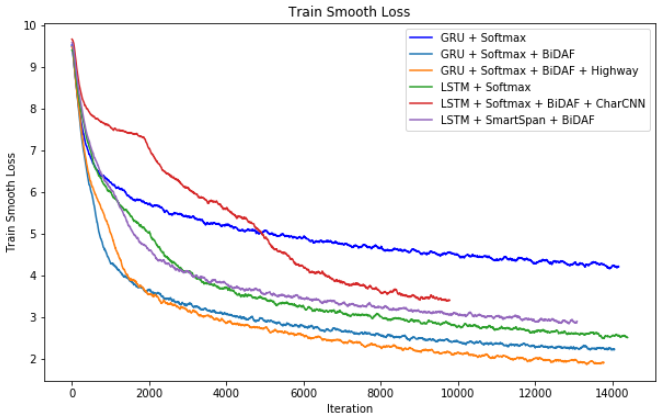

Fig. 3. The loss of the training set for the different models that were configured as the number of iterations increase.

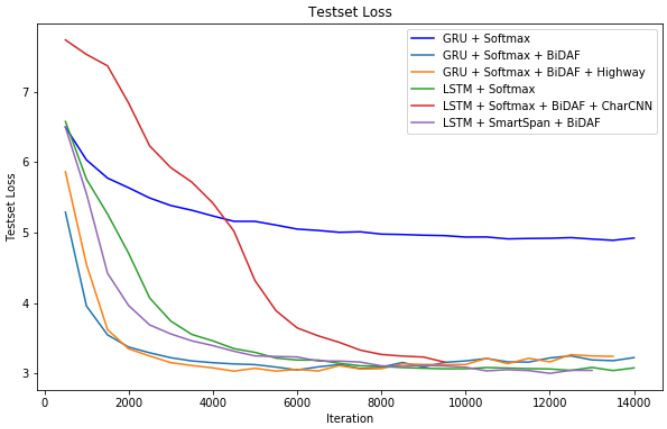

Fig. 4. The loss of the testing set for the different models that were configured as the number of iterations increase.

Finally the Question to Context attention and Context to Question attention are combined to get the final attention out b_i for each context hidden state c_i.

$$b_i = [c_i; a_i; c_i \cdot a_i; c_i \cdot c'] \tag{13}$$

The attention layer takes in question encoding and context encoding and outputs a single vector which is our attention output.

F. Output Layer ***[Yerania Hernandez]***

The final layer of our network consists of what we refer to as the output layer. The main purpose of this layer to predict the span of the answer, from the start index to the end index. From the previous layers, we are able to produce the hidden state layer for the context and the vector for the attention, which is concatenated together as a representation of the context hidden layer and the attention output. This combined layer serves as the input to a TensorFlow fully connected layer. With a fully connected layer, our output layer will be able to compute the start index and end index based on the probability distribution vector produced from a softmax calculation. With these probability distributions, we evaluated

TABLE I
RESULTS FOR SQUAD TRAIN AND TEST SET

Embedding Layer	Encoding Layer	Attention Layer	Output Layer	Dropout	F1 Score		EM Score	
					Train	Test	Train	Test
Glove	GRU	Dot Product	Softmax	0.85	57.41	36.47	45.30	25.95
Glove	GRU	BiDAF	Softmax	0.85	85.01	64.28	72.70	49.11
Glove	GRU + Highway	BiDAF	Softmax	0.85	89.41	65.47	77.30	50.46
Glove	LSTM	BiDAF	Softmax	0.75	81.29	64.83	66.70	50.18
Glove + Char CNN	LSTM	BiDAF	Softmax	0.70	69.07	61.95	55.50	46.89
Glove	LSTM	BiDAF	Smart Span	0.75	77.66	**66.43**	64.90	**50.56**

two different methods in order to obtain the final start and end indexes. The basic method obtains the maximum index probability from the start and end distribution vectors and returns this as the start and end index. The second method, which we refer to as a smart span method, is based on the analysis we had done on our data set. Considering we know that answers span between 11 to 21 words, we selected 15 as the maximum words that the answer should span. As a result, we are able to maximize the probability through the product of the start and end distribution vector in order to obtain the best start and end index. After all these layers have been initialized, we add a loss layer, which simply computes the cross-entropy loss for the start and end index predictions while also using the Adam optimizer in order to minimize the loss across each batch.

IV. EVALUATION [STUTI SAKHI]

We use two evaluation criteria to evaluate our model.

1) Exact Match (EM) : It is a binary measure which checks if the predicted answer matches exactly the actual answer.
2) F1 : F1 is a harmonic mean of precision and recall. Precision the percentage of words in the predicted answer which are present in the actual answer. Recall is the percentage of words from the actual answer which are present in the predicted answer.

V. RESULTS [YERANIA HERNANDEZ]

Our entire network was developed with a variety of parameters in order to allow us the flexibility of testing different configurations. These different networks were implemented using Tensorflow 1.11 and Python 3.6, while training on Google Colab due to the available Tesla K80 GPU. The results of the different model configurations we experimented with are summarized in Table I. As shown, the model with the best configuration uses GLoVe vectors, the LSTM network in the encoding layer, BiDAF attention in the attention layer, and the smart span method in the output layer. The F1 score of this model on the test set is 66.43 and the EM score is 50.56, which performs comparable to the scores displayed on the leader board of the SQuAD data set. In addition, Figure 3 demonstrates the loss of the train set of each of the configurations as the number of iterations increases. The most basic configuration consists of using the GRU network with a softmax as part of the output layer. As the graph displays, the loss is much higher than any of the other configurations. Furthermore, this loss is also replicated in the test set, as shown in Figure 4. All the other configurations demonstrate low losses along with reaching this stage at a much quicker pace. Each of these configurations were run from ten epochs to fifteen epochs, causing them to run for over 14,000 iterations. However, after a certain number of iterations, it is visible that the loss converges.

In addition to the different features we explored within each layer, we also analyzed the effects of certain hyperparameters. After training a few configurations, we observed that accuracy scoring between the train and test had a difference of over twenty points. This concerned us due to the fact that it could be possible that our network was overfitting the results. As a result, we began decreasing the dropout ratio in order to reduce this difference between the training and test set. As is shown in Table I, the decrease in dropout reduced the difference between the train and test set to about ten points. This difference seemed more reasonable considering these results were more correlated to each other and therefore we used 0.75 as the dropout ratio for our final model. In addition, we analyzed the decrease of the learning rate and the effects it had on the scoring as well. However, we only had the opportunity to configure two different learning rates and although it did increase the scoring of our configurations, we cannot make a complete conclusion on the actual effect it had on our results. Our final model used a 0.0008 learning rate instead of the initial 0.001 used for the base model considering previous works focused on the importance of having a low learning rate.

VI. Conclusion

In this paper, we focused on exploring a variety of approaches to develop an MRC model to analyze the SQuAD data set. Our results demonstrate that our best model uses a combination of GLoVe word embedding, LSTM network, BiDAF attention, and smart span to finalize the appropriate answer based on the query and context provided. The analysis on these different configurations provides an insight on the different aspects that have been taken into account to solve the MRC challenge. As a result, we obtained a model that performed similar to some of the methods published on the SQuAD data set leader board. Future work consists of exploring hyperparameters and other configurations, which could possibly increase the F1 and EM score. In addition, due to the flexibility we add to our model, we could further extend this model to implement other types of recurrent neural networks.

References

[1] "2018 NLP Challenge on Machine Reading Comprehension." *2018 NLP Challenge on Machine Reading Comprehension*, 2018, mrc2018.cipsc.org.cn/.

[2] Dwivedi, and Priya Dwivedi. "NLP - Building a Question Answering Model – Towards Data Science." *Towards Data Science*, Towards Data Science, 29 Mar. 2018, towardsdatascience.com/nlp-building-a-question-answering-model-ed0529a68c54.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

[4] Fleming, Jim. "Highway Networks with TensorFlow – Jim Fleming – Medium." *Medium.com*, Medium, 29 Dec. 2015, medium.com/jim-fleming/highway-networks-with-tensorflow-1e6dfa667daa.

[5] Karl Moritz Hermann, Tomas Kocisk y, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *NIPS*, 2015.

[6] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *In EMNLP*, 2016.

[7] Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text understanding with the attention sum reader network. *In ACL*, 2016.

[8] Richardson, Matthew, et al. "MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text." *2013 Conference on Empirical Methods in Natural Language Processing*, 2013.

[9] Seo, Minjoon, et al. "BI-DIRECTIONAL ATTENTION FLOW FOR MACHINE COMPREHENSION." *ICLR 2017*, 2017, doi:https://arxiv.org/pdf/1611.01603.pdf.

[10] Srivastava, Rupesh Kumar, et al. "Training Very Deep Networks." 2015.

[11] Wang, Shuohang, and Jing Jiang. "Learning Natural Language Inference with LSTM." *Cornell University Lab*, American Physical Society, 10 Nov. 2016, arxiv.org/abs/1512.08849.

[12] Wang, Shuohang, and Jing Jiang. "MACHINE COMPREHENSION USING MATCH-LSTM AND ANSWER POINTER." *ICLR 2017*, 2017, doi:https://arxiv.org/pdf/1608.07905.pdf.