Question Answering on SQuAD

Shasvat Desai
shasvatmukes@cs.umass.edu
Mili Shah
mashah@cs.umass.edu
University of Massachusetts Amherst
140 Governors Dr, Amherst, MA 01002

Abstract

Question answering (QA) is a well-researched problem in NLP. Traditionally, most research in this domain used conventional linguistically-based NLP techniques, like parsing and POS tagging. However, recent developments in neural network models including GRUs, LSTMs along with attention mechanisms have achieved state-of-the-art performance in QA. In this project, we explore combination of a linguistic feature - dependency parse, and deep learning architectures, including bi-directional attention layer and self-attention layers, for the task of reading comprehension question answering on SQuAD dataset[1]. We achieve a slight improvement with our architecture of combination of neural network and linguistic information, over the architecture with only a neural network.

1. Introduction

The ability to read a piece of text and then answer questions about it is called reading comprehension. Reading comprehension is challenging for machines, requiring both understanding of natural language and knowledge about the world. In this project, we aim to solve the problem of performing the task of Question-Answering on the SQuAD[1][1] dataset. SQuAD is a machine reading comprehension crowd-sourced dataset of more than 100,000 QAs, from more than 500 Wikipedia articles. The questions are posed by crowdworkers on the Wikipedia articles, and the corresponding answers are spans of text from the corresponding reading passages. The task then is, given a context passage and a question, predict the answer as a span from the given paragraph.

[1]https://rajpurkar.github.io/SQuAD-explorer/

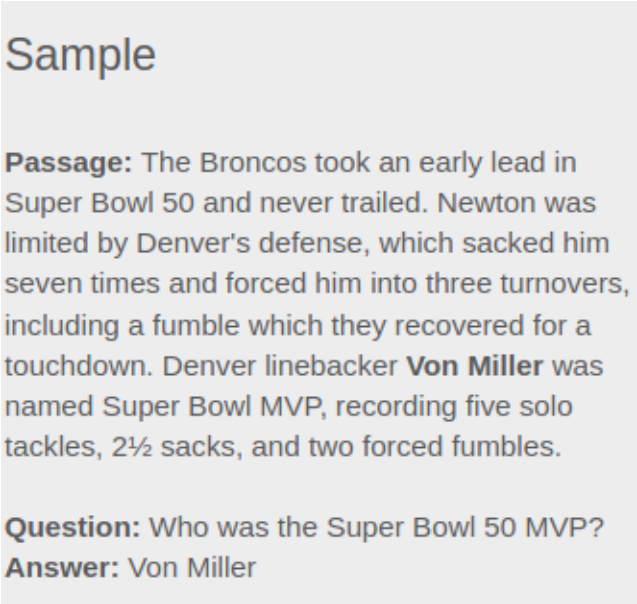

Figure 1. SQuAD dataset sample

1.1. Dataset

The original SQuAD dataset is divided into train, development and test sets. The test set is not publicly available, while the other two are. So, we split the train set into train and development sets with a 80:20 split ratio, and use the original development set as our test set. The resulting train set has 15,102 questions and 353 topics, while the development set has 3794 questions from 89 topics. The test set has 2067 questions from 48 topics.

1.2. Evaluation Metrics

The evaluation metrics we plan to use are F1 score and exact match(EM) score, as proposed by the authors of SQuAD.

F1 score: The F1 score measures the average overlap between the prediction and the ground truth answer. We

treat the ground truth and the prediction as bags of tokens, and compute their F1. We take maximum F1 over all the ground truth answers for a given question, and the average over all the questions.
EM score: This metric measures the percentage of predictions that match any one of the ground truth answers exactly.

For this task of machine reading comprehension, we explore a modification to a popular model for the Question Answering task : BiDAF[2], by using the Transformer[3] network(as described in Section 3.3.1), and the augmentation of dependency parse - a linguistic feature(as described in section 3.4), to BiDAF.

2. Background

Previously BiDAF[2] model has been used to perform this task of Question Answering on the SQuAD dataset. BiDAF includes character-level, word-level, and contextual embeddings, and uses bi-directional attention flow to obtain a query-aware context representation. We base our model on the BiDAF model.

Another model which we leverage to enhance the performance of our model is the Transformer Network[3]. Transformer Networks have been shown to be a powerful model in sequence to sequence learning. This network relies completely on attention mechanisms to draw global dependencies between the input and output. They allow full parallelization, thus significantly reducing the processing time relative to Recurrent Neural Networks.

Additionally, we build a dependency parse tree structure and provide it as additional features to our model. Dependency parse provides an approximation to the semantic relationship between predicates and their arguments, making it directly useful for the task of question answering. [4] employ techniques similar to ours for the task of Semantic Role Labeling.

3. Approach

We outline approaches taken for both - our baseline model, that is a simple Bag of words model, and also our new proposed architecture, that is based on BiDAF.

3.1. Baseline model

As our baseline, we implement a simple Bag of Words model. We do this using the following two methods:

N-grams: For each question and the corresponding context, we generate our candidate answers from the context paragraph by extracting all word n-grams i.e., 1, 2 and 3 grams from the tokenized paragraph. We take multiple windows of some size surrounding our candidate and do an intersection over union(IoU) to find the maximum IoU between the bag of words of each window and the question, assigning this maximum score to the given candidate answer. We select the candidate answer with the maximum score as our final answer.

Noun chunks: In this approach, instead of generating our candidate answers using n-grams, we follow a different path. As per the statistics on the SQuAD dataset given here[1], almost 80% of the candidate answers are nouns. Thus we use spaCy[2] to extract all noun chunks from the given paragraph and use them as our candidate answers. Rest of the method, to find the best candidate answer with the maximum score for a given question, and the evaluation method for our model, remain the same.

3.2. BiDAF

Bi-Directional Attention Flow network (BiDAF) is a hierarchical multi-stage architecture which models the representations of the context paragraph at different levels of granularity. BiDAF model consists of the following submodules:

1. **Word embeddings, character embeddings** : For both, passage and questions, word and character embeddings are obtained using GLOVE embeddings and 1D convolutions respectively.

2. **Contextual embedding (Phrase Layer)** : This layer employs sequence-to-sequence encoders, over passage and question separately, to embed the paragraph/sentence context in each word.

3. **Bi-directional attention layer**: This layer produces a question aware representation of the passage through coupling of the question and the passage.

4. **Modeling layer** : This layer contextually encodes the resultant encodings obtained from the output of the bi-directional layer.

5. **Span start predictor** : A dense network that assigns scores to answer span starts.

6. **Span end predictor** : An LSTM that assigns scores to all possible answer span endings. For each span start, this predictor only considers the tokens occurring after the span start token.

[2]https://spacy.io/

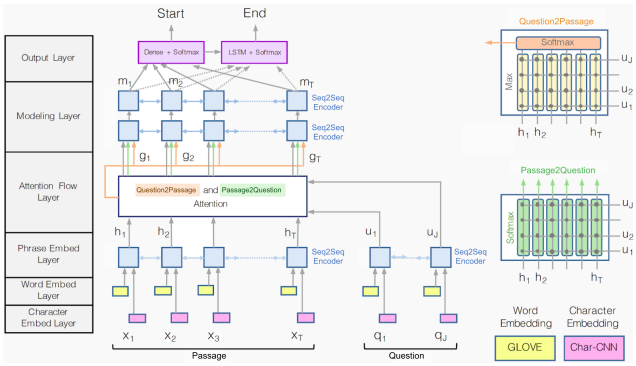

Figure 2. BiDAF model

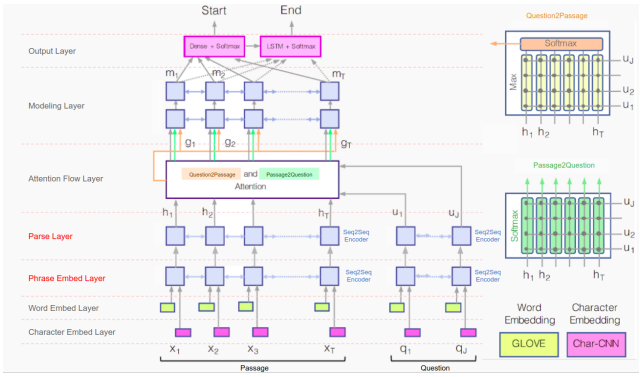

Figure 3. Our Model

7. **Softmax** : Softmax over all possible answer span starts, and softmax over all possible answer span ends.

The implementation of the BiDAF model is made available by the authors of the paper[3]. We build over this model to build a new architecture. In our model, we add another Layer over the phrase layer(which consists of Bidirectional LSTM's to generate contextual embeddings). Particularly, we leverage the Transformer Network to build our Parse layer and additionally use the Dependency Parse structure in our Transformer Network which enables our model to approximate the rich semantic information tween predicates and their arguments. In the following sections, we explain the motivation and the implementation using the Transformer Network in our model.

3.3. Our Model

Motivation: With the introduction of attention mechanisms to capture a richer context information, we have noticed a significant improvement in the sequence to sequence models, enabling the model to capture long term dependencies. Recently, a new architecture was proposed -

[3]http://allennlp.org/

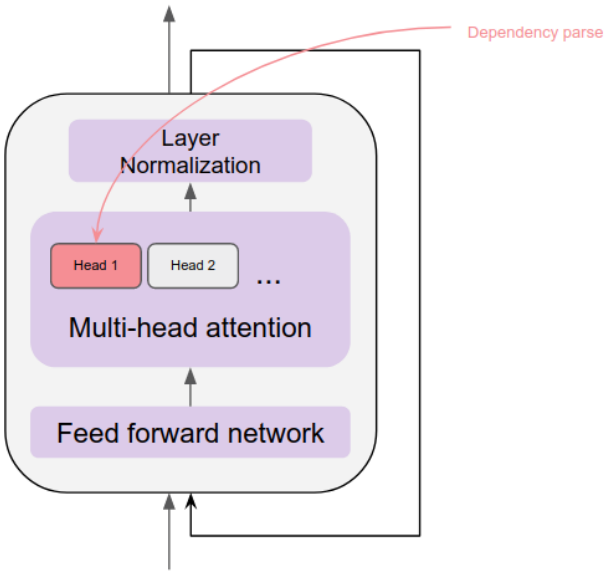

Figure 4. Transformer Network

the Transformer network which eschews the use of RNN and LSTM to capture the context information and relies solely on stacked layers of attention and feed-forward layers to capture the contextual information. Since our task is question answering, we additionally thought of leveraging the power of linguistic features and build a dependency parse structure which we embed as an integral part of our Transformer network to enable our model to capture higher level semantic information contained in the passage and the questions.

3.3.1 Transformer

The Transformer Network (Fig. 4) consists of stacked layers of self-attention (Illustration in Fig. 6) and feed-forward (or convolutions). The self-attention layers have multiple attention heads, all of which learn to attend differently to the input sequence, and their concatenated results are passed on to the feed-forward layers with residual connections.

Key Components of our Transformer Network are as follows:

Scaled Dot product attention: As show in the Fig. 4 The input to our Transformer Network is either word embeddings or output of seque We project this embedding using trainable weight matrices which are learned during backpropagation, into Query(of dimension d_q) ,Key(dimension =d_k) and Value(dimension=d_v), where each is of dimension d_{model} i.e. $d_v = d_k = d_q = d_{model}$

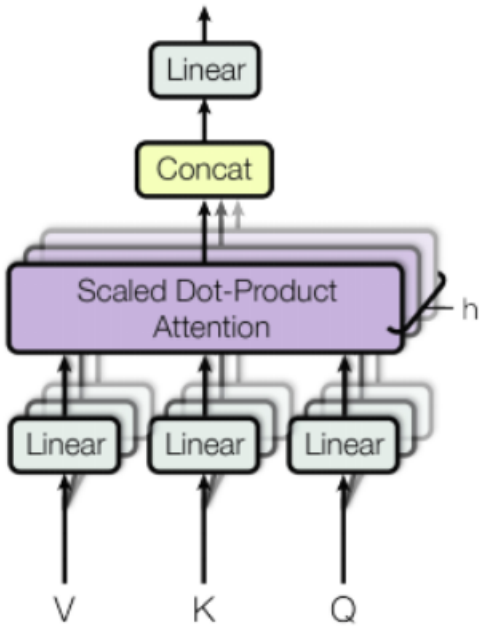

Figure 5. Attention head in Transformer Network

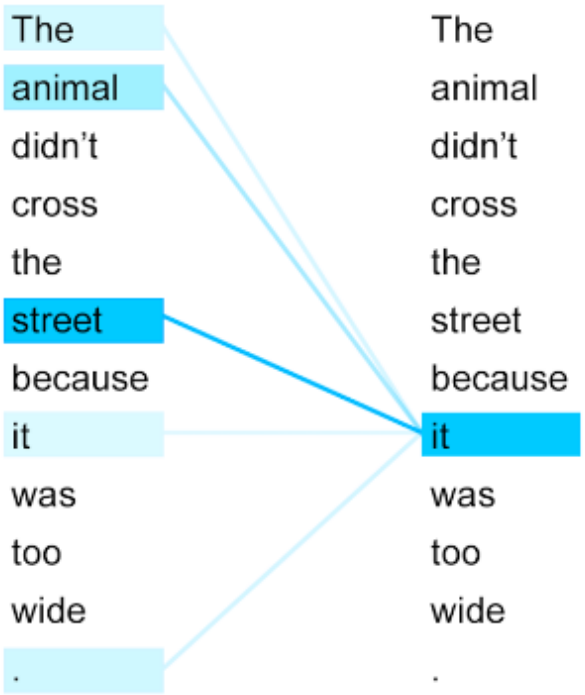

Figure 6. Self attention

The dimension d_{model} is our. Thus our projection weight matrices(W_q,W_k,W_v) are of each of dimension:$(embeddingdim * projectiondim)$, where our projection dimension is our d_{model}. The projection dimension is our hyperparameter with which we experiment. We compute the dot products of the query with all keys, divide each by d_k , and apply a softmax function to obtain the weights on the values. In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V We compute the matrix of outputs as:
$Attention(Q, V, K) = softmax(QK^T/\sqrt{d})V$
Dot-product attention is identical to our algorithm, except for the scaling factor of $1/\sqrt{d}$ .For large values of d_k , the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. To counteract this effect, we scale the dot products by $1/\sqrt{d}$. An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

Multi Head Attention: As shown in the figure Fig. 5 , we use multi head attention. Instead of performing a single attention function with d_{model} dimensional keys, values and queries, we compute the dot product attention in parallel. Thus we linearly project the queries, keys and values h times with different, learned linear projections to d_k , d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v -dimensional output values. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$MultiHead(Q, K, V) = Concat(head1...headh)W^O$
where $head_i = Attention(QW_i{}^Q, KW_i{}^K, VW_i{}^V)$

Hence if we employ h = 4 parallel attention layers, or heads and our $d_model = 256$, our key, value and query dimensions will be $d_k = d_v = d_{model}/h$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

FeedForward Networks: In each of our stacked layers, we pass the output obtained through the computation of self attention though a Feed Forward Network which has a single hidden layer. We use batch normalization after every layer which enables our model to become deeper We apply residual connections to the output of each of the above modules before they are passed through the next module.

Figure 7. Dependency parse of a sentence

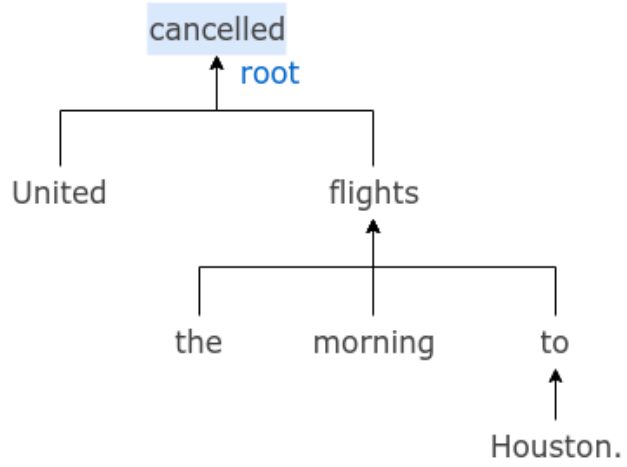

Figure 8. Dependency parse tree of a sentence

3.4. Dependency parse structure

Syntactic structure consists of lexical items, linked by binary asymmetric relations called dependencies. Traditionally, it has been seen that adding syntactic parse structures to our system allows us to effectively model and exploit the inherent structure contained in the text. An example of a dependency parse structure is shown above in the figure Fig. 7 Each sentence has a root - a word that is not dependent on any other word - usually the main verb. Each word (dependent) is connected to a head (the word it is immediately dependent on). Thus in the example sentence above, the root is a verb-'cancelled' which is the parent of nouns- 'United and 'flights' which in turn is parent of preposition 'to'. We leverage the ability of the dependency parse structure to model the higher level semantic information contained in our passages and questions to effectively answer the questions. We use spacy library to tokenize our input passages and questions and extract the POS tags along with the dependency head of each of the token. Thus after the process of tokenization ,we have the parent or the dependency head of each of the token from dependency parse structure. Specifically, we train with an auxiliary objec- tive on one attention head which encourages that head to attend to each token's parent in a syntactic dependency tree. We use the attention weights a_{th} between token t and each other token q in the se- quence as the distribution over possible heads for token t: $P(q = head(t)|X) = a_{thq}$ Now, We choose one of the layers from several stacked layers of our Transformer Networks and then embed this dependency parse information into one of its several heads. The task of each of the heads in the stacked layers is to compute self attention over the passages and the question. We designate one of the heads with the task of attending to each of the tokens parent obtained from the parse structure as shown in figure Fig. 4. This way during the training phase our network learns to incorporate the higher level semantic structure of the passage and the questions itself and thus generates embeddings which can effectively assist our model in making accurate predictions.

4. Experiments

4.1. Implementation details

- Deep Learning Package: PyTorch
- Programming Language: Python
- External Libraries: spaCy

4.2. BiDAF

We first trained the original BiDAF model, on which our model is based, on our dataset.
F1:71.49
EM:57.73

4.3. Transformer Network

Below we describe all the experiments done with the Transformer Network without the Dependency parse information embedded into the Transformer Network

4.3.1 Number of Transformer Networks

Here we experiment with different number of Transformer Networks in our model.
One Transformer Network:
We replace the Bidirectional LSTM used in the phrase layer in the original BiDAF model with our Transformer Network. our Transformer consists of 2 layers with 4 heads in each layer. Rest we keep the model intact. We observe that this model obtains comparable performance as the original model. Thus we conclude that a Transformer which only used self-attention and convolutions yields has same power as LSTM to capture long term dependencies with an additional advantage of faster training as parallel convolutions are much faster than LSTM. Due to varying server load and network bandwidth, we are unable to report the exact factor by which the Transformer Networks are faster than LSTM.
F1: 71.07
EM:57.75
Two Transformer Network: In addition to 1, we replace the span end predictor which uses LSTM with a Transformer.

AS a result we get a dip in the performance of the model. We suspect this might because the Transformer are unable to encode context information equivalent to 2 LSTM
F1: 43.54
EM:29.53

4.3.2 Number of stacked layers

We experimented with the number of stacked layers in the Transformer Network. Initially, we attempted to build a very deep network with 6 stacked layers and 8 heads in each layer to compute self attention. We had to reduce the batch size due to resource constraints. It failed to give good results. We found the reason for this was less data due to which the network was underfitting.
So next thing we tried with 3 stacked layers and 4 attentions heads in each layer but found to be in same situation as before wherein our model was underfitting.
3 stacked layers with batch size 10
F1: 23.00
EM:14.00
6 stacked layers with batch size 10
F1: 20.00
EM:12.00
Finally, we found 2 layers with 4 heads in each layer to give good results.

4.3.3 Number of attention heads in each layer

We experimented with $num_{heads} = 2$ to $num_{heads} = 8$ in each layer and found the $num_{heads} = 4$ works the best

4.4. Dependency parse

After carefully analysis of the results obtained from above, we, decided to keep all the layers of BiDAF as is and add an additional layer of Transformer over the Phrase layer as shown in figure Fig. 3 Additionally, we embed the Dependency parse structure information in the Transformer network as described above.

4.4.1 Joint Training

Here, we add an additional loss to train our model to learn each tokens parent in the dependency parse structure. The additional loss is given by : $\lambda * (1 - P(q = head(t)|X))$ This way we train the model to attend to each tokens parent. This model achieved comparable performance to BiDAF. We believe that since during the training, the model has access to poor representations , when training begins, the model sees random values of attention weights which do not contain any syntactic information, even in the head which will be trained to provide it with such information. Thus it was not able to improve much over BiDAF, as one would have expected it to.

Best Baseline Results					**F1 score (on test set)**	**Exact Match (on test set)**
Noun Chunks with a window size of 10					18.33	0.12
Our Experiments						
Description	**Phrase Layer**	**Modeling Layer**	**Dependency Parse Information**	**Hyperparameters tuned**	**F1 score (on test set)**	**Exact Match (on test set)**
BiDAF	LSTM	LSTM	-		71.49	57.73
Transformer in 1 Seq2SeqEncoder	Transformer	LSTM	-	Number of layers, number of attention heads	71.07	57.75
Transformer in 2 Seq2SeqEncoders	Transformer	Transformer	-	Number of layers, number of attention heads	43.54	29.53
Joint Training of dependency parse information and question answering	Transformer + Dependency Parse	LSTM	In Phrase layer transformer	Lambda	71.41	58.32
Gold parse labels in one attention head in transformer	LSTM	LSTM	(Transformer + Dependency Parse) as Parse Layer after Phrase Layer		**72.14**	**58.62**

Figure 9. Results

Window size	F1 score	EM score
10	0.1833	0.12

Table 1. Noun chunks with window size 10 results on the test set

F1: 71.41
EM : 58.32

Hence next we experiment with another method

4.4.2 Explicit Parsing information

This was our most successful experiment. Here we stacked a Transformer layer in top of the LSTM and equipped the Transformer with knowledge of each of the token parent which it should attend to. As a result we get a better performance that the BiDAF model on which our architecture is based.
F1:72.14
EM:58.62
In Figure 9, we summarize the results.

5. Our Contribution

In this project we proposed a new architecture for the purpose of Question Answering on SQuAD based on the BiDAF model. We combined the power of attention mechanism and the Dependency parse structures with the Bidirectional attention mechanism used in the BiDAF model to get improvement over the BiDAF model. The implementation of the BiDAF model and the Transformer Networks were made available by the authors of the paper.
Our two key contributions are:
1. We successfully extracted the dependency parse structure and embedded the information in the Transformer Net-

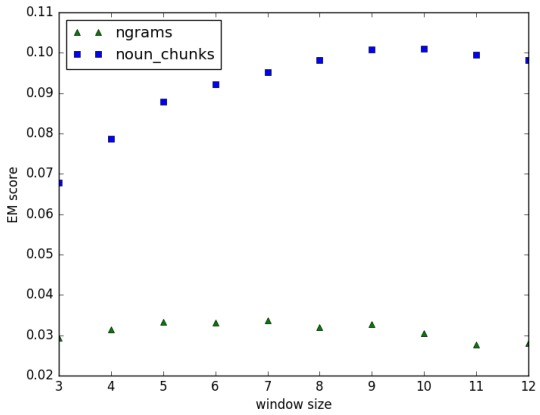

Figure 10. F1 score evaluation for baseline experiments

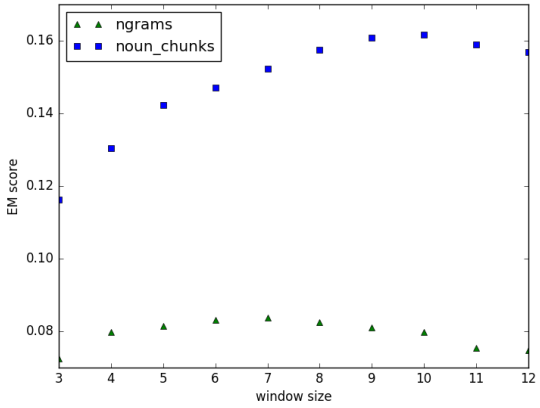

Figure 11. EM score evaluation for baseline experiments

works, thus imparting greater knowledge to the contextual embeddings computed by the Transformer by capturing the higher level semantic structure of the passage and the question.

2. Attention mechanism, that is the crux of our Transformer model, is known for capturing long term dependencies. We were successful in getting improvement over the performance of the BiDAF architecture by adding the Transformer Network equipped with the Dependency parse information on top of the Sequence to Sequence Encoder layer.

6. Conclusion

This model has a scope of improvement in various aspects. The promising results obtained by leveraging the Transformer Networks could be further explored by using more than two layers of stacked Transformer Network. One could also stack the Transformer Network after the bi-directional attention flow layer of the BiDAF network which could potentially improve the span prediction yielding better results. We could not explore these because of time and resource constraints. Inclusion of other linguistic features can also be explored.

References

[1] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," *CoRR*, vol. abs/1606.05250, 2016.

[2] M. J. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," *CoRR*, vol. abs/1611.01603, 2016.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[4] E. Strubell, P. Verga, D. Andor, D. Weiss, and A. McCallum, "Linguistically-Informed Self-Attention for Semantic Role Labeling," *ArXiv e-prints*, Apr. 2018.