An Analysis of The PointNet Architecture and Word-Dropout Applied To Question Answering

Mason Swofford
Computer Science
Stanford University
Stanford, CA 94305
mswoff@cs.stanford.edu

John Peruzzi
Computer Science
Stanford University
Stanford, CA 94305
jperuzzi@cs.stanford.edu

Abstract

The task of question answering is a fundamental problem in machine comprehension. Using a context paragraph, a model must learn to extract relevant information in order to answer a query question. In this paper, we explore novel improvements to the BiDAF model for question answering. We are using only non-PCE methods and evaluating on the SQuAD 2.0 dataset.

We have experimented with a number of techniques, most notably dropping out words from the context paragraph as a regularization technique, as well as applying the PointNet model from computer vision literature to the task of question answering. We also demonstrate a technique to integrate Self-Attention and character embeddings to the BiDAF model, and finally experiment with character-level convolutions with the word embeddings.

The character convolutions, as expected, slightly improved performance. Word dropout at our model size is shown to be certainly ineffective and PointNet and Self-Attention were not helpful in improving our model's performance.

1 Introduction

The task of question-answering has gained significant popularity is recent years. Given a query question and a context paragraph, the task is to predict the answer which is a continuous sequence of words from the context paragraph. While this task does not require knowledge retrieval, it does require significant reading comprehension skills. Effectively solving the reading comprehension problem is one major stepping stone in advancing language understanding and machine intelligence as a while. For our model, we experiment on the SQuAD 2.0 dataset, which is notably more difficult than the original SQuAD dataset, because many questions do not have answers in the passage [3]. While the original SQuAD dataset was difficult because many questions require a model to have a complex understanding of the context passage, the SQuAD 2.0 dataset is especially challenging, because it forces models to realize when they don't know an answer, and requires the model to be confident in its answer, or risk its overall performance.

Currently, the best performing models all utilize context embeddings, meaning that the embeddings of a word should depend on the context in which the word appears in the text. We elect not to use contextual embeddings like the high-performing BERT and ELMO do, and instead iterate on the BiDAF model, which is simpler but has yielded effective results. We chose this approach, because we were interested in trying novel techniques from other research and our own ideas. By testing on a simple model without context embeddings, we see how well these ideas apply to question answering.

We have proposed and experimented with three significant changes to BiDAF, two of which are highly original. Our most innovative idea is the use of PointNet and word convolutions to generate an alternative attention calculation. The PointNet model originates from the vision community and is

notable in that it provides a way to learn a global representation of an input, regardless of the ordering of the input features, and is effective for different sized inputs [5]. We apply a modified version of the PoinNet architecture to the query in order to learn a vectorized representation of it. We also learn a local representation of each context paragraph word by taking convolutions across the context paragraph. This effectively builds feature representations for each context paragraph word, taking into account the local words around it. We then combine the query representation and each context paragraph word's local representation, and us this as part of our context word attention.

We also experimented with dropping out words from the context paragraph as a regularization technique. Our research showed that this technique was used by [2], on tasks such as sentiment analysis and was shown to be an effective regularization technique. We independently decided that this form of regularization would be useful for our model after realizing that often the BiDAF would simply focus on just one or two key words from the query and find them in the paragraph in a context in which those words were unrelated to the query. We thus dropout words randomly to teach the model the importance of looking at more words in the context paragraph and not simply rely on finding a few of the question words.

We additionally experimented with self-attention, as described in R-Net [6]. Self-Attention is the idea of computing attention across the context paragraph, whereas the attention mechanism already incorporated into BiDAF only computes attention between the context and query. Self-Attention is thought to give the context words a better understanding of what the rest of the paragraph contains, and aggregates that information into a representation that supplements the original context word. The Transformer architecture showcases one example of how highly effective this idea can be [7].

Our final addition to the baseline BiDAF model was the addition of character-level embeddings, which supplement the word-level embeddings of the Stanford BiDAF model. The Stanford default implementation is notably different from the original BiDAF model in that the Stanford model only uses word-level embeddings of the question-words and context-words. [4] Character level embeddings are known to capture better the structural similarity of words, and are especially useful for understanding out of vocabulary words. This is very helpful in question answering, where out of vocabulary words often appear in the context and question and are often answers (names, cities, etc).

We evaluate the performance of these model additions to BiDAF to see which techniques best apply to the task of question answering.

2 Related Work

Our model is based on Stanford's implementation of the BiDAF architecture, which achieved state of the art results when it was published [1]. BiDAF was unique in that it utilized a novel Attention Flow Layer in order to capture both Query-to-Context and Context-to-Query attentions. This attention mechanism differed from previous attempts because it was computed for each context and each query word, instead of as a vectorized representation for the entire context and entire query. After the context and query words are supplemented with their attention representations, they are fed through a modeling layer which captures the interactions between the context words, conditioned on the query.

BiDAF notable contribution, the Attention Flow layer, is just one type of attention that can be used. R-Net employs a different form of attention with its Self-Matching Attention layer. Self-Matching Attention is one form of Self-Attention, where the model computes attention representations across the context paragraph, instead of between the context and query. By utilizing a Self-Attention mechanism that is agnostic to the actual distance between two words in the context, R-Net is able to capture dependencies effectively across the entire context, irrespective of locallity, while BiDAF relies on an RNN to capture intra-context dependencies. Although a BiDAF ensemble outperformed an R-Net ensemble on the SQuAD dataset, a single R-Net model outperformed a single BiDAF model. However, there is no reason to believe that these differing forms of attention cant be combined, and that a combined model may benefit from their differing attributes. For this reason, we chose to employ both an Attention Flow layer and a Self-Matching Attention layer.

Another recent work which describes a novel algorithm for modeling information across an input is Qi et al's PointNet architecture [5]. PointNet was designed to process 3D point clouds and build a feature representation for them that was agnostic to the ordering of the inputs, as point clouds do not have a natural ordering. This feature representation was then used to perform classification and

segmentation. The novel PointNet architecture introduced the idea of growing each 3D point's feature representation, in parallel, into a high dimensional representation, and then applying a symmetric function across this new high dimensional representation, in order to compute a representation for the entire point cloud, that was irrespective of the input ordering.

While this model architecture showed it could successfully model unordered 3D point clouds, there is little literature on it being applied outside of this domain. We wanted to see how this architecture could be applied to language tasks, and decided that we would use it to efficiently compute a representation of our query.

Although language has exploitable structure, unlike point clouds, we felt that by applying a PointNet style architecture to the query, our model would more easily pick up on the important words in the query, such as the type of question being asked (e.g. who/what/where/why etc.) regardless of where in the query the word appeared.

For example "After nightfall, who came home?" and "Who came home after nightfall?"

Finally, [2] showed that dropout out word embeddings can be an effective regularization technique for language processing tasks such as sentiment analysis. However, to the best of our investigative knowledge, this performance of this technique has not been tested on question answering tasks, making our evaluations of its performance contributions to the literature.

3 Approach

Our approach improves upon the BiDAF architecture presented by Stanford. Our first alteration was the addition of character-level embeddings for both the context and query words, following more closely the architecture presented in the original BiDAF paper, although with some distinct modifications [1]. On top of character-level embeddings, we have also added three additional features: Word-level dropout, Self-Attention, and an original PointNet Attention.

3.1 Baseline

The baseline for our model is Stanford's implementation of the BiDAF algorithm. As we are working on the default project, we will not explain the model in detail, but a detailed description of the implementation and code can be found in the original project handout. [1] [2]

3.2 Character-Level Embeddings

While we previously motivated why character-level embeddings are useful, we would also like to motivate our specific approach to utilizing them. As well, our implementation of character level embeddings differs from the original BiDAF paper's.

3.2.1 Character-Level Embeddings Motivation

We chose to have our final word embedding be a concatenation of the character-level embedding and the word-level embedding. This is for two reasons. First, for in-vocabulary words, the model can utilize both the relatedness aspect that word-level embedding captures, as well as the structural character-level information that word-level embedding contains. Second, for an out-of-vocabulary word, the model will be able to leverage the learned character-level features of the word, as well as the knowledge that the word is out-of-vocabulary, which is obtained from the "unknown" embedding that the word-level model yields. Possibly counter-intuitively, the "unknown" embedding is actually quite useful information for question answering, as out of vocabulary words such as names, places, and dates are often the answers to questions. Notably, this is also the approach taken by the original BiDAF paper [1].

[1] `http://web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf`

[2] `https://github.com/chrischute/squad`

3.2.2 Character-Level Embeddings Implementation

For a given word x_{padded} with "pad" character tokens appended to the end of the word as needed to ensure all words are the same length m_{word}, we look up a pretrained embedding of each character of size e_{char}, generating

$$x_{\text{emb}} \in \mathbb{R}^{m_{\text{word}} \text{x} e_{\text{char}}}$$

In parallel, we then apply 1d convolutions to x_{emb}, max pool the results, and then concatenate the resulting embeddings. Here, our model builds on the original BiDAF implementation. BiDAF simply uses one layer of kernel size 5.

Multiple Kernel Sizes

We hypothesize that convolutions of different window sizes will learn different relations between characters. For example, in the word "willingly," smaller convolutions of kernel size 3 or 5 may learn features for "gly" and "ingly" and capture the part of speech of the word, while a larger convolution of kernel size 7 can capture interactions between all characters and learn base words, such as "willing." Combining this information hopefully leads to a more diversified understanding of the word. Thus, we apply Conv1d layers with kernel size size of 3, 5, and 7 with 100, 150, and 200 filters, respectively, apply more filters to larger levels because we anticipate there being more features to learn. We get

$$\begin{aligned}
x_{\text{conv3}} &= \text{Conv1D}_{3,100}(x_{\text{emb}}) \in \mathbb{R}^{100\text{x}(m_{\text{word}}-2)} \\
x_{\text{conv5}} &= \text{Conv1D}_{5,150}(x_{\text{emb}}) \in \mathbb{R}^{150\text{x}(m_{\text{word}}-4)} \\
x_{\text{conv7}} &= \text{Conv1D}_{7,200}(x_{\text{emb}}) \in \mathbb{R}^{200\text{x}(m_{\text{word}}-6)}
\end{aligned}$$

We then apply a relu nonlinearity to each output followed by a max-pool, giving

$$\begin{aligned}
x_{\text{conv out3}} &= \text{MaxPool}(\text{ReLU}(x_{\text{conv3}})) \in \mathbb{R}^{100} \\
x_{\text{conv out5}} &= \text{MaxPool}(\text{ReLU}(x_{\text{conv5}})) \in \mathbb{R}^{150} \\
x_{\text{conv out7}} &= \text{MaxPool}(\text{ReLU}(x_{\text{conv7}})) \in \mathbb{R}^{200}
\end{aligned}$$

These vectors are concatenated to each other and to our word-level embedding (of size 300), giving

$$x_{\text{conv cat}} \in \mathbb{R}^{750}$$

We then project $x_{\text{conv cat}}$ to a smaller dimension through a linear layer, generating x_{proj} and then x_{proj} is passed through the highway and dropout layers in the given Stanford implementation of BiDAF.

Deeper Convolutions

In the hopes of learning more at the character level, we also experiment with applying two convolution layers sequentially to grow x_{emb} more slowly. This is instead of applying different kernel sizes and concatenating, to test if depth of the convolution is more important than more filters on the input layer. As well, by using two kernel sizes of 5 each time, we are able to achieve a large receptive field on the input. We choose to first grow to filter size of 128, and then 170, both with a window size of 5. These numbers are specifically chosen to match the number of parameters in the multi-kernel model, so we can compare depth vs breadth on models of comparable complexity. This model thus looks like

$$\begin{aligned}
x_{\text{conv hidden}} &= \text{ReLU}(\text{Conv1D}_{5,128}(x_{\text{emb}})) \in \mathbb{R}^{128\text{x}(m_{\text{word}}-4)} \\
x_{\text{conv out}} &= \text{MaxPool}(\text{ReLU}(\text{Conv1D}_{5,170}(x_{\text{emb}}))) \in \mathbb{R}^{170}
\end{aligned}$$

And we create $x_{\text{conv cat}}$ and then x_{proj} once again by concatenating the word-level embedding and then projecting to a lower dimensional space and then proceed to the highway layer.

3.3 Word-Level Dropout

Gal and Ghahrmani perform word embedding dropout by probabilistically setting the rows of an embedding matrix to 0 in a given train example. This ensures that the same word will be dropped out throughout the text, for example "The dog chased the cat" becomes "<> dog chased <> cat" if the word "the" is dropped out.

3.3.1 Word-Level Dropout Motivation

Analyzing where the original model made mistakes, we saw that often it often found key words from the question and honed in on them, even if the words were in the a part of the passage less related to the query. By dropping out words randomly, we wanted to force the model to not focus on specific words and become overly reliant on them, and instead read the other words and learn their relation and meaning to the query. We experimented with only dropping out words from the context paragraph. In the queries, due to their shorter length, each word may be extremely important, and without it, one may no longer understand the question being asked. Therefore, we only dropped out from the context paragraph, hoping that if a key word were dropped out, the model would have to learn to understand the broader meaning of the context paragraph from other words, in to figure out where the answer would be.

3.3.2 Word-Level Dropout Implementation

The original Gal and Ghahrmani paper dropped out words with a probability of $p_{\text{drop}} = .5$. We instead chose to drop out words with a probability of $p_{\text{drop}} = .1$, since from an intuitive perspective, one would not be able to find answer in a paragraph with half its words missing. The dropout must occur at the embedding matrix, and thus all other rows of the embedding matrix are multiplied by $\frac{1}{1-p_{\text{drop}}}$, as is standard in dropout techniques.

3.4 Self-Attention

Upon analyzing where our character-level BiDAF model was making mistakes on our development set, we noticed a large number of errors were occurring from our model not having enough understanding of the context paragraph. In order to help the model learn this, we decided to implement our Self-Attention and PointNet Attention layers.

3.4.1 Self-Attention Motivation

While the Stanford BiDAF architecture already features an Attention Flow Layer, this does not capture intra-context word dependencies. There are both local and global dependencies our model needs to recognize in order to comprehend the context paragraph. While the PointNet Attention hopes to capture local information, Self-Attention is a useful method for capturing global information across the entire context paragraph.

In order to better decide what is important to put in the global information, the Self-Attention needs to be conditioned on the query. To motivate this idea, we look to how a human may perform reading comprehension. They would likely read the query, and then read the context paragraph with that query in mind. When they reached what they thought was the answer, they may still skim ahead and back to make sure that the answer makes sense in the global context of the entire paragraph, as its possible a later sentence could actually be a better answer, or contradict their original answer.

3.4.2 Self-Attention Implementation

In order to benefit from knowledge about the query when performing our Self-Attention, we chose to add another bidirectional RNN on top of the BiDAF Attention Flow Layer, and then perform Self-Attention, before going through the BiDAF Modeling Layer. This allows our model to perform some initial modeling in our new bidirectional RNN, condition on the Self-Attention layer, and then refine its modeling process before going to the output layer. While our implementation generally follows the implementation in the R-Net paper, we will explicitly describe it, to prevent any confusion, using a simpler notation from the original paper.

If $v_t \in R^H$ is the hidden state of one context word from our new bidirectional RNN, our Self-Attention Layer output, $v'_t \in R^{2H}$ is computed as:

$$v'_t = [v_t, c_t]$$

where $c_t = att(v^P, v_t)$ is an attention-pooling vector of the whole context passage v^P:

$$s_j^t = \mathrm{v}^{\mathrm{T}} \tanh(W v_t + W' v_j)$$

$$a_j^t = \frac{\exp(s_j^t)}{\sum_{i=1}^{n} \exp(s_i^t)}$$
$$c_t = \sum_{j=1}^{n} a_j^t v_j$$

3.5 PointNet Attention

We applie the PointNet architecture to the query sentence to learn a global, vectorized representation of the query. We then apply a convolution over the context word embeddings to learn a representation of each word within its local context. We compare the similarity of each word in the context to the PointNet representation of the query, and add this similarity to our learned Bidirectional Attention output vector for each context word.

3.5.1 PointNet Motivation

We see our PointNet as another way for our model to learn which context words to focus on. Rather than computing the similarity between each individual query and context word, we treat the query as a single vector checking "all at once" how a context word is similar to the query representation. This could potentially learn completely new and useful information in comparison to the BiDAF attention method, which more indirectly compares a context word to a whole query, by first running an RNN on the query words, then computing individual word-to-word similarities, and then combining these.

Our approach is somewhat radical in that it treats the query as a "bag-of-words" when computing its representation, since PointNet is order-invariant. We justify this approach by the small size of the query sentences and the desire to learn "key words" from the query. Further, we wanted to investigate how models respond to this order-invariance, and if they could still learn useful features.

Finally, we convolve over the context words to learn their representation to mirror how the PointNet representations are computed. We use window sizes of 3 and 5 to pick up some of the surrounding context of a query word when learning its importance.

3.5.2 PointNet Implementation

The PointNet model computes a global vectorized representation of the query sentence, and convolved, local-context aware representations of each context paragraph word.

To generate the PointNet representation, for each passage query embedding $q_j \in R^{100}$, $j = 1, ..., M$ (100 is the hidden size we use for our embeddings) we apply two convolution filters to learn encodings of each query word $q_j^{enc} \in R^{400}$.

$$q_j^{enc} = CONV_{f=400}(CONV_{f=200}(q_j)) \in R^{400}$$

We then apply a symmetric max function across all the representations, to learn a final, 400 dimensional encoding of the query.

$$q^{rep} = MAXPOOL(q^{enc}) \in R^{400}$$

Next, we apply a series of convolutions over all the context embeddings $c \in^{N\text{x}100}$ to generate a representation of each context word $c_i \in R^{400}$ from $i = 1, .., N$. The kernels are of size 3 and 5. Padding must be added to ensure that every c_i gets a representation.

$$c^{rep} = CONV_{f=400,k=3}(CONV_{f=200,k=5}(c)) \in R^{N\text{x}400}$$

We compute the similarity of the between every c_i^{rep} and the query representation q^{rep}, and project down to a smaller dimension to get c_i^{att}

$$c_i^{sim} = q^{rep} \odot c_i^{rep} \in R^{400}$$
$$c_i^{att} = W_{proj} c_i^{sim} \in R^{200}$$

We then append these c_i to the output of the BiDAF attention flow layer (g_i in this version paper (CITE)).

4 Experiments

4.1 Data

We use the SQuAD (Stanford Question-Answering) 2.0 dataset [3]. The data is split into 129,941 train, 6,078 validation, and 5,915 test examples. The data comes in the form of a query question (a sequence of words), a context paragraph (another sequence of words), and an answer (a continuous sequence of words from the context). The SQuAD 2.0 dataset is especially difficult because many of the questions have no answer, which forces the model to only output an answer when it is confident.

4.2 Evaluation Metrics

There are two evaluation metrics we use, which are the standards for the default fianl project. The EM (exact match) score is contingent on exactly matching the output answer. With an EM score of 1 if the model outputs the exact labeled answer and a score of 0 if the output is even one word off. A less harsh metric is the F1 score, which is 2 * precision * recall / (precision + recall). The maximum EM and F1 score over 3 human answers is taken per query, and averaged over the entire dataset.

4.3 Experimental Details

Our initial experiments ran using the Stanford provided default configurations. However, we experimented with using a larger hidden layer, of size 300, and found that this caused training to be extremely slow, and partway through training, decided to stop the model as it didn't seem to be improving further. After that all experiments were run using the default configuration. However, our initial learning rate of .5 was decayed by 95% each epoch. We trained using two Tesla K80 GPUs for 40 epochs and selected the best model based on validation F1 score.

We trained each of our model configurations (PointNet, Self Attention, Word Dropout, and Character Embedding) separately.

4.4 Results

Our character encodings improved the F1 score by .69 and the EM score by .99 over the original model when we apply a series of different convolutions, but a drop in performance when we apply multiple layers of convolutions. However, the rest of our results showed a drop in performance. Our final test results were 56.771 EM and 60.825 F1.

We ran many ablation studies to determine which of our model additions would perform the best. Dropping out word embeddings, even at a very small rate of .1 lowered our EM and F1 scores by around 2.0. Adding the PointNet features alone to the model dropped our EM and F1 scores around 0.2. Adding just self attention gave only slightly lower scores by around .5 .

We not surprised in retrospect that dropping out words did not improve results. Training on less information could have helped the model be smarter and more robust, but we also knew it ran the danger of making it very difficult for the model to ever understand how sentences are actually structured and interfere with the attention flow as the model would have to learn which dropped out words to attend to, a task that does not appear at test time.

We were slightly more surprised that PointNet did not improve our attention calculations. This implies that learning from the onordered bag-of-words representation is very difficult and in fact was adding features to our attention vector which were confusing the model and not helping it learn.

The fact that self-attention did not improve our scores also surprised us. There were many locations in our model in which we could have added it, and we are still working and iterating to see if some improvement can be found, as it is not necessarily straightforward how to apply self-attention to the BiDAF model. However, self-attention is a proven technique for question answering and will hopefully lead to improvements, if we change the implementation specifications.

4.5 Analysis

We first ran the baseline model so that we could analyze the results and qualitatively look at where the model made mistakes.

From there, we decided that adding character-level word embeddings would help with the problem of out-of-vocabulary words. However, we did not know which of our two character-level embedding strategies would perform better, and decided to perform isolated testing of each model, where we only changed the character-level word embeddings. This allowed us to determine that using only one layer, of multiple filter sizes, was the best strategy moving forward. The multi-layer character level convolutional embeddings actually hurt our performance, which we think may be due to the filter sizes, and thus the receptive field, being too large.

We then again analyzed our models predictions on the development set and noticed our model seemed to not understand the context paragraph, and was simply looking for words which appeared in the query, even in less sensible contexts. This motivated dropout, to teach the model to not only search for certain words.

On dropout, we tested at multiple dropout sizes, and found that the less dropout we used, the better our model performed, but even at very small word drop-out rates, we still unperformed. This led to the conclusion that word-level dropout simply made it more difficult for the model to learn what the context meant. From an intuitive, human perspective, this makes sense, as it could be difficult to learn what the context passage meant if even a few necessary words were missing.

Pointnet and Self attention both reached accuracies comparable to the baseline model. However, upon analyzing the performance over time, we noticed that they took significantly longer to begin improving at all. This is expected, as they are more complex models. However, the fact that they never surpassed our baseline results meant that they were not providing our model with useful information. This lead us to the conclusion that the features learned by our PointNet model did not help when appended to the attention output vector, as they represented information that did not improve results. We can thus conclude that, as implemented, the PointNet features are not adding useful information. However, we believe that the idea to create a bag-or-words representation of the query, using the PointNet inspired architecture, may still be useful in another implementation; however, it clearly does not improve our specific model. We note that self-attention is a proven technique that as implemented did not improve our model, and thus we cannot take these reults as proof that the PointNet representation of a sentence has no value.

4.6 Conclusion

Since our primary goal was not climbing the leader-board with known techniques but analysis of the effectiveness of our novel question-answering techniques (especially PointNet and word-dropout), we did achieve the result of determining that as implemented, these techniques are not effective augmentations to the model. We also achieve the result of determining that character representation is best done with multiple filter sizes instead of multiple convolution layers. Finally, our self-attention, as implemented proved to be ineffective.

We chose the default project as a way to investigate applying novel techniques to the challenging domain of question-answering and we achieved that goal. While there exist many existing architectures for question-answering that perform well, it is hard to tell if other architectures have not been reported on because they do not work, or if it's because no one has thought of them or tried them yet. While our novel idea of utilizing the PointNet architecture did not improve our results, we think there are still possible variations we would like to try if we had more time. One possible avenue we wanted to try was using the learned feature representation from PointNet of the query as the initial hidden state for our query RNN and/or the modeling RNN. By doing this, we theorize we can "prime" the RNN with what it will be seeing, and allow it to better know what information will be important to hold in its hidden states.

Another avenue we would like to pursue further would be again applying word-level dropout, but using it in combination with Self-Attention. We believe that part of the reason word-level dropout was unsuccessful was because our model simply searched for important words, and did not have the capacity to understand the context paragraph. However, Self-Attention gives our model the necessary mechanism to "see," and therefore learn from, the entire context. We believe that under this pretense,

word-level dropout would actually help regularize and compound the benefits of our Self-Attention layer.

References

[1] Seo, M., Kembhavi, A., Farhadi, A. & Hajishirzi, H. *Bidirectional attention flow for machine comprehension*. 2016

[2] Gal, Y. & Ghahramani, Z *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks*. 2016

[3] Rajpurkar, P., Jia, R., & Liang, P. *Know What You Don't Know: Unanswerable Questions for SQuAD*. 2018

[4] Zhang, X & LeCun, Y *Text understanding from scratch*. 2015

[5] Qi, C., Su, H., Mo, M., & Guibas, L. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[6] https://www.microsof t.com/enus/research/wpcontent/uploads/2017/05/rnet.pdf

[7] Vaswani, Ashish, et al. " *Attention is all you need.* Advances in Neural Information Processing Systems. 2017.