NATURAL LANGUAGE PROCESSING: SUMMARIZATION USING RECURRENT NEURAL NETWORKS AND ATTENTION

Luna Skytte Hansen and Stine Bech Petersen *

Technical University of Denmark
Department of Applied Mathematics
and Computer Science
s144289@student.dtu.dk, s182909@student.dtu.dk

Alexander Rosenberg Johansen†

Technical University of Denmark
Department of Applied Mathematics
and Computer Science
aler@dtu.dk

ABSTRACT

This natural language processing project is inspired by the work of Abigail See et. al. [1], who proposed a model for abstractive summarization by implementing 1) a Sequence-to-sequence attentional model, 2) a Pointer-generator network and 3) a Coverage mechanism. Their model outperforms current state-of-the-art by 2 ROUGE points. The work of this project focus on the first part of the work of Abigail et. al., i.e. the attentional seq2seq-model. The model is implemented using pytorch and tested on data consisting of numerals generated for the purpose. The results compare the performance of a non-attentional and attentional model and it is confirmed that the later outperforms when dealing with longer sequences.

***Index Terms*—** Deep Learning, Neural Network, Natural Language Processesing, Abstractive Summarization, Attentional Sequence-to-sequence model

1. INTRODUCTION

Deep learning and neural networks are emerging topics that can be used to handle many data driven problems. Modern techniques provide solutions for image recognition, speech recognition and natural language processing. The concept behind neural networks is inspired by the biology of the neurons in the human brain and enables the computer to learn from observational data [2]. The computational model of Deep Learning involves multiple processing layers with parameters that are updated using a backprobagation algorithm [3]. Examples of recent state-of-the-art neural networks are many and involves everything from autonomous driving [4] and counting the number of people in a crowd [5] to computer security [6] and analysis sleep [7].

Natural language processing (NLP) aims at using the human language - either written or spoken - as data to train models to work as tools for many different cases. NLP is used in our everyday life for spell check, grammar, auto-correct and -complete. It is also used for conversational interfaces like Apple's Siri Service or Google Home [8].

One issue with NLP models is that the input data often comes in sequences of words. To accommodate for this a sequence-to-sequence model has been suggested by I. Suskever et al. [9], who proposes a multilayered Long Short-Term Memory RRN network along with word mapping and embedding.

A well established tool that applies NLP is machine translation, which refers to the translation of a text by a computer. Different approaches have been used to handle this task including encoder-decoder approaches [10] and attention [11].

Many machine language models are able to identify and process words but in recent studies it is also attempted to develop systems beyond identifying words. This entails models actually understanding the human language [12]. These systems bring Deep Learning technology one step closer to developing truly human-like AI-interfaces that can be used for more advanced problems.

Within the field of natural language processing *text summarization* refers to the application of deep neural networks to the task of creating meaningful summaries of long text pieces. When considering that the available amount of data is continuously growing it becomes clear why such algorithms can be extremely valuable. NLP text summarization algorithms can be described as being either extractive or abstractive and are usually build using either recurrent neural networks or convolutional neural networks. The extractive models use a copy-paste method where chunks of words from the original text are pasted into the summary. In contrast, the abstractive models are able to paraphrase the original text through segmentation, and addition and removal of chunks of text [13].

In 2017 A. See et al. [1] provided an abstractive text summarization model which outperformed the present state-of-art models by 2 ROUGE points. This model, and the article describing it, provides the guideline for the project described in this article. By attempting to recreate parts of See et. al's

*Students of MSc Engineering, Project Responsible
†Research Assistant, Project Supervisor

work it is desired to confirm the results as well as to learn from the process.

The approach for text summarization developed by A. See et al. includes three models; an attentional sequence-to-sequence network, a pointer generator network and a coverage mechanism. The sequence-to-sequence attentional network is the baseline of the final model and is therefore the main focus of this project. The other networks can be added to the baseline model in order to improve the performance of the model.

Beyond the work by A. See et. al [1] R. Nallapati et. al. developed two extractive summarization Recurent Neural Network (RRN) models [14]. One is based on classifying each sentence in the original text for use in the summary. The other model is based on selecting one sentence at a time in an arbitrary order to generate the summary. They found that the first approach using classification yielded the best results. R. Nallapati et. al also model abstractive summarization using Attentional Encoder-Decoder Recurrent Neural Networks [15]. The model they propose provides the foundation for the baseline model of A. See et. al [1].

The final network model proposed in this project implements the sequence-to-sequence model by A. See et al. The network consist of an GRU encoder and attentional GRU decoder. The focus of the project work has been to observe how the implementation of attention improves the network performance when data consist of long sequences.

2. THEORY

2.1. Reccurent Neural Networks

Recurrent neural networks (RNN) are a special type of neural networks in which the units form cyclical connections as depicted in Figure 1. The cycles formed in RNNs have a huge impact on the abilities of the network since they allow the network to have a memory of previous inputs which can influence the outputs [16]. This means that recurrent neural networks are especially good for processing sequential data such as speech or language, much like convolutional neural networks are specialized for processing matrix data such as images [17].

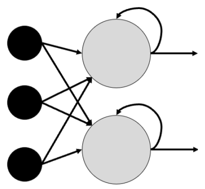

Fig. 1: Illustration of a vanilla recurrent neural network. The black circles represent the input layer and the grey circles represent the vanilla recurrent units.

The vanilla recurrent neural network unit takes two inputs, the input at time t, x_t, and the hidden state from the previous unit, h_{t-1}. The two inputs are concatenated and activated in a tanh-function and the result is the new hidden state, h_t [18]. Networks build from this simple unit suffer greatly from vanishing and exploding gradients which makes training it very difficult. The vanishing and exploding gradient refers to issue of the large decrease or increase of the gradient during training. This makes it difficult for the model to learn and update the weights of the network. This issue is a result of the serial architecture of the RNN [19]. To solve the issue of vanishing and exploding gradients the Long-Short Term Memory unit (LSTM) and Gated Recurrent Unit (GRU) can be implemented instead of the vanilla RNN cell. The models build in this project use a GRU which is depicted in Figure 2. In Figure 2.a it is illustrated how the input, x_i, at time, i is used as the input for a GRU which computes a hidden state, h_i. At the next time step, $i+1$, the input for the GRU will be x_{i+1} and h_i and the output will be the next hidden state h_{h+i}. In an encoder-decoder model the final hidden state, called the context vector, will be used as the input for the first time step in the decoder along with the decoder hidden state [20].

In Figure 2.b the internal workings of a GRU can be seen. This illustrates the mechanism that allows the GRU to have a memory of previous inputs. The GRU uses two gating functions, r_t, the reset gate and, z_t, the update gate. The reset gate is able to forget irrelevant parts of the input data. In summarization tasks this might be specific classes of word such as prepositions which don't convey important information. The update gate determines whether to retain the value of the previous time step or to update using the the new input [20].

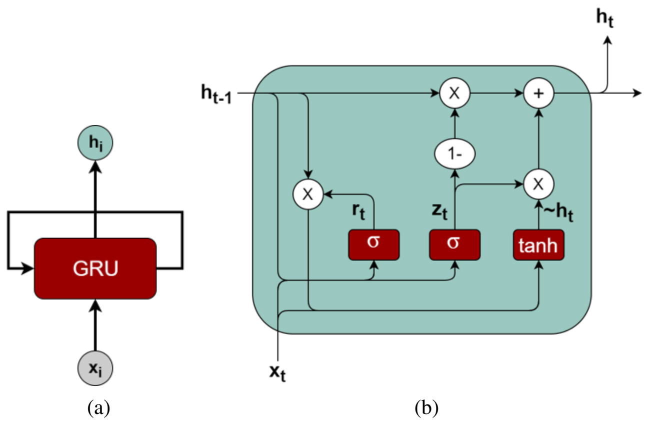

Fig. 2: The Gated Recurrent Unit (GRU). In (a): an illustration of a GRU-network, where each hidden state from one GRU serves as an input for the next GRU. In (b): The internal workings of a single GRU.

2.2. The model of See et. al

The work of A. See et. al [1], which defines the scope of this article, implements what is referred to as Pointer Generator Networks. 'Generator' in this case refers to the model being generative, which means that it applies probability distributions to evaluate the plausability of a large amount of candidate outputs [21]. 'Pointer' refers the use of probablity distributions to deal with out-of-vocabulary (OOV) words [1]. The Pointer Generator Network is subdivided into three increasingly complex models. Initially a sequence-to-sequence attentional model is presented followed by the pointer generator network which adds the pointer mechanism to the initial model. Finally a coverage mechanism to obtain the final model.

2.2.1. The sequence to sequence attentional model

The sequence-to-sequence attentional model by A. See et. al. is a many-to-many RNN based on LSTM units. In the model of See et. al. the encoder is bidirectional and the decoder applies attention which enables the model to handle long input sequences. The attention mechanism is directly translated to the model implemented in this project and is presented in **section 3 Model**.

2.2.2. The pointer-generator network

The pointer network uses the context vector from the attentional mechanism to calculate a generation probability, p_{gen} which is used to decide between generating a word from the vocabulary and simply copying a word from the original text. This enables the model to use OOV words in the summary instead of having to insert UNK tokens (unkown words) thereby potentially increasing the performance of the model. p_{gen} is calculated using a sigmoid function taking as inputs the context vector, h_t^*, the decoder hidden state, s_t and the decoder input, x_t as shown in Eq. 1. The parameters $w_{h^*}^T$, w_s^T, w_x^T and b_{ptr} are learned during training of the network.

$$p_{gen} = \sigma({w_{h^*}}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{ptr}) \qquad (1)$$

p_gen is used to calculate the probablity $P(w)$, which is the distribution from which the the output will be chosen. This probability relates to the *extended vocabulary* which contain both the model vocabulary and the words present in the input text.

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i^t \qquad (2)$$

2.2.3. The coverage mechanism

See et. al. [1] also apply a coverage mechanism to avoid a common problem arising in summarization algorithms - namely repetitions. Due to the time scope of this project the coverage mechanism has not been covered in detail. The basic point of the coverage mechanism is that a coverage vector consisting of the sum of all previous attention distributions is used to be able to penalize the model for repeatedly attending to the same input location [1].

3. MODEL

Our model is a Pytorch implementation of the sequence-to-sequence attentional model by A. See et al. [1]. The model proposed here take as input a batch of arrays of integers corresponding to the words index in the vocabulary.

3.1. Encoder

The encoder is illustrated in the data flow diagram of Figure 3. As can be seen the encoder takes two inputs, the input sequence of data and the previous hidden state. When there is no previous hidden state the implementation of the model simply feeds the network an initial hidden state of zeros. The input is padded with zeros using the Pytorch function, pad_sequence() which is necessary when the inputs are of unequal lengths. Following this the input sequences are embedded using the Pytorch function, embedding() which embeds the input into numbers in a specified number of dimensions. Finally the input is packed using pack_padded_sequence() which is necessary for the *gru()* module when using padded batches of inputs of unequal lengths. Finally the input and previous hidden states are fed to the GRU to produce two outputs, *output* which contains all hidden states and *hidden* which contains only the last hidden state. As visualized in Figure 4 the encoder is bidirectional which enables the encoder to take into account both the previous and future inputs when computing the hidden states. This was chosen because it seems reasonable to assumes that for natural language processing the input at one time step will depend on inputs at both future and previous time steps.

3.2. Attentional decoder

An illustration of our model and the attentional mechanism it applies in the decoder is shown in Figure 4 - it should however be noted that the numbers $1 - 4$ do not represent the actual size of the model. The decoder first applies embedding to the given input using the same dimensions as in the encoder. The embedded input is than passed through a ReLU activation before being fed to the GRU module. The attention mechanism is then applied to the output of the GRU.

From Figure 4 it can be seen how the model uses all encoder hidden states and not only the final hidden state. This is

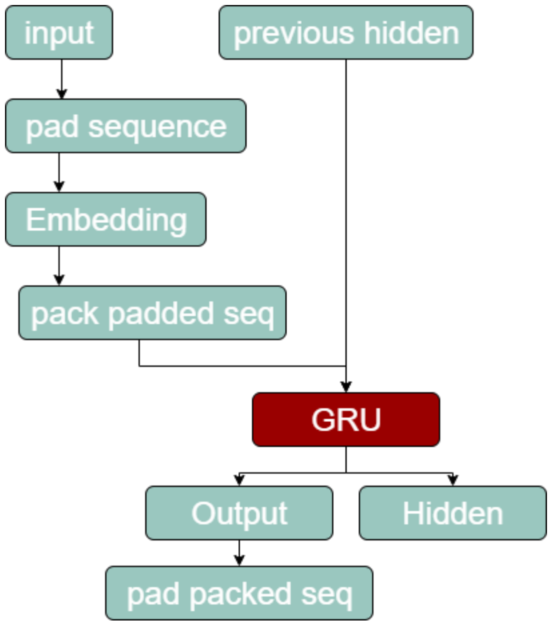

Fig. 3: Data flow diagram of the encoder as implemented in Pytorch.

the attentional mechanism of the model. When working with long input sequences is becomes increasingly difficult for the traditional sequence-to-sequence model to make correct predictions. This happens because it becomes increasingly difficult to contain all the information of the input in the single fixed-length context vector. Attention solves this problem by using the hidden states from the encoder at all time steps to compute context vectors for each decoder time step. This allows the model to focus on, or pay attention to, the relevant part(s) of the input sequence at a given time step of the output sequence.

Our implementation of attention follows that of See et. al. [1]. The content of the grey box in Figure 4 can be written as in Eq. 3 where the $tanh$ activation function corresponds to fc in the illustration which is a fully connected layer used to learn the attention weights, a_t.

$$a^t = softmax(v^T tanh(W_h h_i + W_s s_t + b_a ttn) \quad (3)$$

a_t is then used to compute a weighted sum of all the encoder hidden states to obtain the context vectors, c_t:

$$c_t = \sum_i a_i^t h_i \quad (4)$$

To obtain the final vocabulary distribution used to choose the next decoder output, the context vector is concatenated with the decoder state,s_t , and send through two linear layers.

$$P_{vocab} = softmax(V'(V[s_t, h_t^*] + b) + b') \quad (5)$$

3.3. Teacher-forced Learning

The model presented above is trained using teacher-forced learning. This means that the decoder input, s_t, is the previous output of the target. This is however only the case during training. For validation s_t is the previous output as predicted by the encoder which is what is illustrated in Figure 4. Using teacher-forcing forces the network to stay close to ground truth which can speed up training. A possible downside of using teacher-forced learning is that there is a discrepancy between what the model sees during training and prediction since there is no ground-truth available in prediction [22]. This could potentially result in bad model performance.

Optimization is done by updating all the weights of the network in each backpropagation using Stochastic Gradient Decent (SGD). The aim is to minimize the loss, which is obtained by computing the gradient of the loss function with respect to the weights. Masked cross-entropy is used as loss function in order to ignore the padding values.

4. DATASET AND EXPERIMENTS

A dataset used for testing the model was generated using the Python library *num2words*. The input vector of the datatset consist of a sequence of string numbers and the target vector consist of a sequence of numerals (word numbers). Two dataset was generated from this approach; one with order sequences and the target being the continuation of the sequence and the second being random sequences and the target being the numeral version of the input. The length of the input and target sequence is similar but the lengths vary for each input target pair up to a defined maximum length. The code for generating the data can be downloaded from GitHub.

Both the continuous and random dataset consist 50,000 samples partitioned into 33,500 training samples and 16,500 validation samples.

4.1. Experiments

The first iteration of the network included only the simple sequence-to-sequence model and therefor attention was yet to be implemented. The network was trained on the dataset of continuous sequences and the random ordered sequences. The performance of the network was tested for different maximum sequences length to observe the network behavior.

After implementing the attentional-decoder the network was further trained on the random sequence dataset for different maximum sequence lengths to observe the improvement from attention.

4.2. Training

The network was trained using 167,000 iterations (5 epochs) and a batch size of 5. The computational time for training

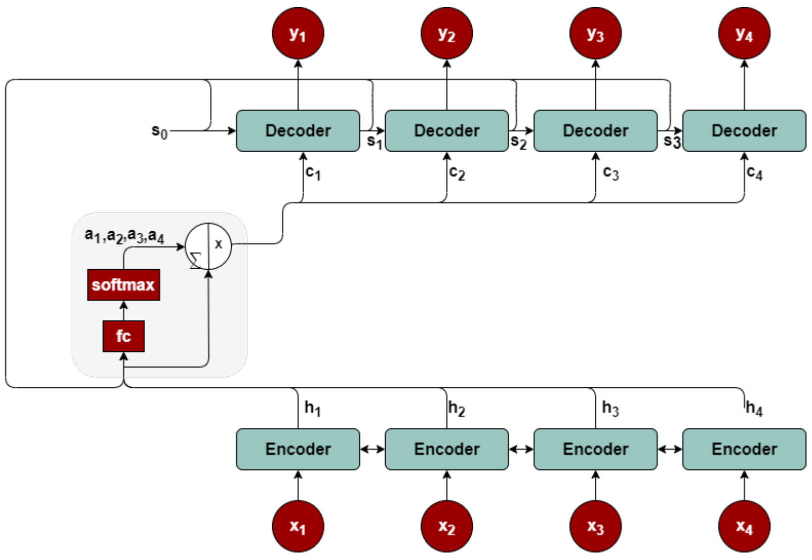

Fig. 4: Our attentional sequence to sequnce model. Double ended arrows imply bidirectionality.

was 3 minutes for the shortest sequence and up to 60 minutes for the longest sequences. The network was trained and tested on sequence of length 10, 15, 20, 25, 30, 35, 40 and 45. Stochastic Gradient Descent with a learning rate of 0.01 for optimization during backpropagation. Masked cross entropy was added to the optimizer to ignore padding values. An embedding dimension of 128 and a hidden dimension of 256 was used as in the work of A. See et. al. [1].

5. RESULTS

The performance of the simple sequence-to-sequence network was tested on both the continuous sequence dataset and the random order dataset. The accuracy of the validation on the trained network for different sequence lengths for both dataset is seen in Figure 5. The validation accuracy for both the continuous and random sequences are somewhat similar and decreases when increasing the sequence length. However, the training loss, seen in Figure 6, is significantly higher for the random sequence data when the sequence length increase.

Attention was later implemented in the network and the performance of the network was further tested on the random sequence data. The validation accuracy and training loss of the non-attentional and attention decoder model is seen in Figures 7 and 8 respectively. It is clear that the validation accuracy is higher and the training loss is lower for increasing sequence length in the attentional decoder model compared to the non-attentional decoder model. However, accuracy still decreases with higher sequence length in the attentional model and at a sequence length of 45 the accuracy is below 50%.

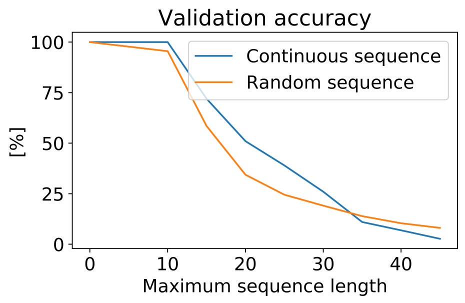

Fig. 5: Validation accuracy of model after training the network on continuous and random sequences of different lengths.

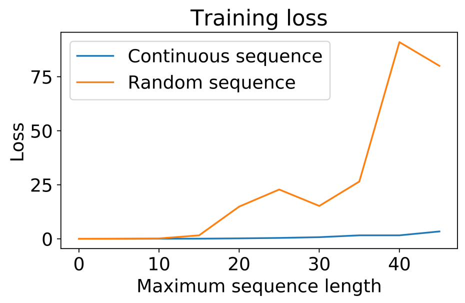

Fig. 6: Training loss of model after training the network on continuous and random sequences of different lengths.

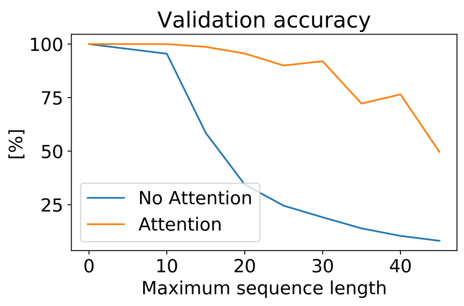

Fig. 7: Validation accuracy of model after training the non-attentional and attentional network on random sequences of different lengths.

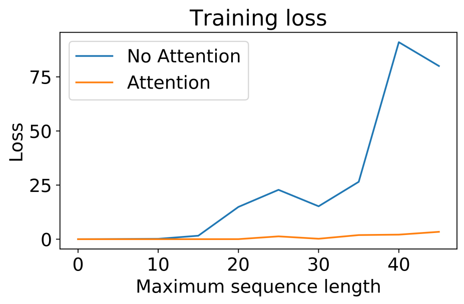

Fig. 8: Training loss of model after training the non-attentional and attentional network on random sequences of different lengths.

6. DISCUSSION

The datasets generated for testing and building the network are simple datasets representing some of the same issues and complexities found in text data. The datasets require the network to be able to remember previous inputs and process the current input based on this information. This is also true when working with text, as the meaning of a given word highly depends on the context of previous and future words of the sentence. The validation accuracy of the non-attentional network on the continuous sequence dataset follows the same curve as the random sequence dataset, see Figure 5. The network struggles with longer sequences and is not able to handle large amount of data to make valid predictions. This makes sense when considering that the network without attention has to contain information about the entire sequence in one context vector. The training loss for the random sequence dataset does however not follow the curve of the continuous sequence dataset, see Figure 6. The loss increases rapidly at the same sequence length as the validation accuracy decreases. The greatly increasing loss of the random sequence dataset indicates that the network is very unconfident about the predictions made.

The curve of the random sequence dataset is also compared to the performance of the attentional model. It is clear that the performance is highly improved by the implementation of attention both in terms of loss and accuracy, see Figures 8 and 7. The attentional network is tested and able to handle sequence of lengths up to 40 with acceptable accuracy of 75%. This confirms that the encoding of information into a single context vector was insufficient since the introduction of usage of all hiddens states from the encoder so greatly improves performance.

The datasets and the final network however does not represent the issue of OOV as all characters used to generated the sequence are presented in the vocabulary. The datasets also does not provide the challenge of learning inflection and conjugation of words, which is important for a summarization model.

The summarization model build for this project is inspired by the the work of A. See et al. However, the an GRU unit is used instead of an LSTM in the encoder and decoder modules of the network. The two units are similar in many ways and there is no clear agreement on which is better. The GRU is a later discovered method and is less complicated than the LSTM as the number of gates are limited. It is good practice to test both approaches, however, this was not within the scope and timeline of this project. The model of this project is also not the full implementation of the model by A. See et. al. [1]. The model is missing the pointer-generator and the coverage mechanism. The details of the the pointer-generator network is covered in **section 2.2.2.** But is yet to be implemented in the final model of this project due to time limitations.

Next step for improving the network is to test the performance on datasets of real text like the CNN or Daily Mail dataset which can be found on GitHub. An element of this implementation includes generating a vocabulary. The current model of this project uses all of the words used to generate the data as vocabulary. The approaches used by A. See et al. to generate the vocabulary is based on combining all of the words of the training data. The vocabulary could contain all words or only words with a frequency above a given threshold. This would also give reason to implementing the pointer generator to deal with OOV words that will appear from the validation data.

Recurrent Neural Network is not the only approach to solving a summarization problem. Recent work have shown significant results using Convolutional Neural Networks for extractive [23] and abstractive [24] summarization. The proposed method for extractive summarization uses CNN to classify each sentence in terms of importance. The proposed method for abstractive summarization uses a convolutional sequence-to-sequence model with attention similar to the model proposed in this project. Both methods achieves results comparable with state-of-the-art techniques. The CNN approach has potential similar to the RNN approach as it also enables the network to consider both future and past words.

7. CONCLUSION

The work of this project implemented the attentional sequence-to-sequence network of the summarization model proposed by A. See et al. [1]. This project has focused on the implementation of the attentional network which provides a solution to deal with longer sequences. A dataset of numerals was generated for the purpose. The performance of a non-attentional model was compared to the attentional model. It was clear that the accuracy of the network decreases very fast with increasing sequence length for the non-attentional model. The attentional model however was able to handle longer sequences of numerals with acceptable accuracy.

8. REFERENCES

[1] Abigail See, Peter J. Liu, and Christopher D. Manning, "Get to the point: Summarization with pointer-generator networks," *CoRR*, vol. abs/1704.04368, 2017.

[2] Michael A Nielsen, *Neural networks and deep learning*, vol. 25, Determination press San Francisco, CA, USA:, 2015.

[3] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[4] Pin Wang, Hanhan Li, and Ching-Yao Chan, "Quadratic q-network for learning continuous control for autonomous vehicles," 2019.

[5] Sarkar Snigdha Sarathi Das, Syed Md. Mukit Rashid, and Mohammed Eunus Ali, "Cccnet: An attention based deep learning framework for categorized crowd counting," 2019.

[6] Yoon-Ho Choi, Peng Liu, Zitong Shang, Haizhou Wang, Zhilong Wang, Lan Zhang, Junwei Zhou, and Qingtian Zou, "Using deep learning to solve computer security challenges: A survey," 2019.

[7] Karan Aggarwal, Swaraj Khadanga, Shafiq Joty, Louis Kazaglis, and Jaideep Srivastava, "A structured learning approach with neural conditional random fields for sleep staging," *2018 IEEE International Conference on Big Data (Big Data)*, Dec 2018.

[8] Bernard Marr, "5 amazing examples of natural language processing (nlp) in practice," *Forbes*, 2019.

[9] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[10] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[12] James L. McClelland, Felix Hill, Maja Rudolph, Jason Baldridge, and Hinrich Schütze, "Extending machine language models toward human-level language understanding," 2019.

[13] Wajdi Homaid Alquliti, Norjihan Binti, and Abdul Ghani, "Convolutional Neural Network based for Automatic Text Summarization," , no. January, 2019.

[14] Ramesh Nallapati, Bowen Zhou, and Mingbo Ma, "Classify or select: Neural architectures for extractive document summarization," *CoRR*, vol. abs/1611.04244, 2016.

[15] Ramesh Nallapati, Bing Xiang, and Bowen Zhou, "Sequence-to-sequence rnns for text summarization," *CoRR*, vol. abs/1602.06023, 2016.

[16] Alex Graves, *Supervised Sequence Labelling With Recurrent Neural Networks*.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

[18] Andrej Karpathy, "The Unreasonable Effectiveness of Reccurent Neural Networks," .

[19] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio, "On the difficulty of training recurrent neural networks," 2012.

[20] Micheal Nguyen, "Illustrated Guide to LSTM's and GRU's: A step by step explanation," 2018.

[21] Ilya Sutskever, *Training Recurrent Neural Networks*, Ph.D. thesis, Univeristy of Toronto, 2013.

[22] Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio, "Professor Forcing : A New Algorithm for Training Recurrent Networks," , no. Nips, pp. 1–9, 2016.

[23] Yong Zhang, Meng Joo Er, and Mahardhika Pratama, "Extractive document summarization based on convolutional neural networks," *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pp. 918–922, 2016.

[24] Yizhu Liu, Zhiyi Luo, and Kenny Q. Zhu, "Controlling length in abstractive summarization using a convolutional neural network," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, 2018, pp. 4110–4119.