Applications of Sequence to Sequence Models for Technical Support Automation

Ghodrat Aalipour*, Pranav Kumar†, Santosh Aditham*, Trung Nguyen*, Aditya Sood*
*Support Automation Team, Juniper Networks, Inc. Sunnyvale, CA
†Department of Computer Science, University of Colorado Boulder, Boulder, CO

***Abstract*—Juniper Networks, Inc. offers hardware products and software services to its enterprise customers. Due to the nature of it's business, Juniper Networks, Inc. is deeply invested in providing the best customer support and as part of the support automation team, our goal is to optimize the company's efforts towards it. For this purpose, alongside other initiatives, we leverage deep learning based sequence to sequence models wherever we see fit. In this paper, we discuss two such models: a conversational chatbot to help answer some technical questions for our customers, and a text summarizer to condense the large text in our support tickets and other articles. These two models are designed using bi-directional recurrent neural network (Bi-RNN) architectures for content understanding and were customized to fit the domain-specific nature of our data. First, we discuss our efforts towards data preparation. Then, we explain our model design, customization and evaluation mechanisms. Finally, we provide the preliminary results and share the potential impact our models will have on our business. Our initial results have BLEU score of 0.21 for text summarizer which is 16% better than our baseline model. Our chatbot passed the eye-tests of our subject matter experts.**

***Index Terms*—ChatBot, Text Summarization, Deep Learning, Seq2seq, Bi-LSTM, Attention Mechanism.**

I. Introduction

Juniper Networks, Inc. offers high-performance network solutions to help service providers, enterprises and the public sector create value and accelerate success. With a revenue of over 5 billion dollars, the scale of our customer service unit is significant. Customer support requests, or customer tickets, can be broadly classified into two classes - primary support that deals with the simpler use cases and, advanced support that deals with technical support requests. We receive more than 10,000 technical support tickets per month and we have more than 700 TAC (Technical Assistance Center) employees serving to such needs of our customers. This scale makes our TAC a multi-million dollar unit and optimizing the pipeline for resolving customer issues is a high priority for us. The two areas of growth that we have identified in this scope of optimizing our customer support efforts are: (a) helping our support engineers *quickly* reach to solutions for customer problems and, (b) helping our customers to get instantaneous answers to relatively straight-forward technical questions and popular business use cases. In this work, we focus on the later.

This work was supported by Juniper Networks, Inc. The first two authors worked on this project during their 2018 summer internship. They contributed equally to this work.

Today, chatbots are being used in many business contexts. Chatbots can be broadly classified to two categories: (a) QnA bots (b) conversation bots. The more popular QnA bots try to extract the intent and the entities from a customer question using Natural Language Understanding (NLU) and respond with one or more appropriate responses. These bots, while in their learning phase, respond with multiple answers to a question and gather feedback from customer to be able to learn and get accurate over time. Some popular QnA bots are: Erica [1] by Bank of America, Alice [2] that won the Loebner Prize. Conversation bots, also known as the generative chatbots, are the more advanced bots that are powered by artificial intelligence. Conversation bots aim to provide a human-like experience for the customer by preserving context while trying to resolve customer issue(s). At Juniper Networks, Inc., we want to build a conversation chatbot to help our customers get instantaneous answers to some of their technical issues. In this work, we describe the following in detail:

- *Objectives*: our choices of sub-projects to realize our chatbot goals and, neural network architectures that we used in our projects.
- *Challenges*: the various challenges we faced with data preparation, model evaluation, and impact measure of our projects on our business.

A. Objectives

The mean time to resolution (MTTR) for our TAC team is just above 10 days per technical support request. With hundreds of tickets per day, resolving each issue requires attention to detail, background knowledge and domain-specific skills. This is a major upkeep for us and suffers from coldstart problems. In this work, we propose leveraging state-of-the-art machine learning techniques to reduce the MTTR for our technical support tickets. The concept of *intelligence*, in this context, applies to several tasks such as job allocation, problem category detection, issue summarization and ultimately providing a resolution. Our focus for this work is on the application of sequence to sequence models that can be used to train conversational agents and summary generators as an early stage of intelligent problem resolution. An accurate model can significantly improve the customer success factor and save millions of dollars by helping support engineers understand the issue faster and better.

a) Chatbot: To be able to realize the afore mentioned goals, we propose to build a chatbot based on advanced

deep learning models that can provide resolutions to our customers for frequently occurring non-complex technical issues. The developed model learns the support engineer's jargon and responds to the imposed issue descriptions with the relevant resolution. Such an intelligent recommendation capability is learned by understanding relevance from the provided problem-resolution data with the help of the language modeling techniques based on modern machine learning algorithms.

b) Text Summarization: We define the criterias for success of our conversation chatbot as consistency and conciseness in response. For this purpose, we propose a text summarization tool that generates complete, meaningful and consistent paraphrasing of lengthy documents.

B. Challenges

Domain knowledge representation is a requirement when building machine learning models that try to automate some of the mundane tasks performed by people today. In this case, an intelligent model that can summarize or converse about domain-specific technical issues is expected to understand the technical terms in the literature at a very early stage. Support engineers have several years of grooming from education and/or work experience to learn these technical terms and their relations. But lack of well defined and documented relations among technical terms and ground truth was a major challenge for us when we took up these projects. Another challenging part, for many machine learning projects, is the quality and the size of the data. In our case, we worked on the TAC data from January 2016 to May 2018. The samples in the data needed extensive augmentation because the customer either did not have a technical background or did not give a concise description of their issues. Moreover, the support engineers were not given any set practices, procedures or patterns for documenting their work. Thus, data cleaning and preparation took a lot of time and we did not have any ground truth to begin with. Overall, the most important issues were:

- Highly unstructured overall data.
- Too few *labeled* samples for training the models.
- Biased distribution of the types of cases.
- Lack of evaluation data and metrics for conversation chatbots.

This paper is structured as follows: Section II gives additional background on machine learning in the NLP domain and talks about related work in this field. Section III describes our approaches for building chatbot and text summarization tools. In this section, we provide the details of our model architectures. Section IV shows the preliminary results and explains some of the observations and learnings we gathered while working on these projects. Finally, Section V concludes this work and presents the future directions and scopes that we would like to explore further.

II. Background and Related Work

The concept of chatbots dates back to 1976 when *Eliza* was created at the MIT Artificial Intelligence Laboratory [3]. Eliza simulated natural language conversations with then computers. It was based on keyword matching and substitution approaches for generating new dialogs, but it barely utilizes contextual information. Another early example of a chatbot is *Parry* that was created in 1995 to simulate a person with paranoid schizophrenia [4]. Ever since, there has been constant effort towards building intelligent chatbots. One of the more recent models, inspired by Eliza is A.L.I.C.E. (Artificial Linguistic Internet Computer Entity), also known as Alicebot [2] which converses with a human through a heuristic pattern matching algorithm. The next major advance in chatbot capability would be when deep learning was incorporated during the training phase.

Since late 1990s there have been many variations to traditional RNN architectures [5]. But a game changing turn to chatbot development took place in 2014 when Cho et.al. [6] proposed Sequence-to-sequence (seq2seq) encoder-decoder architecture for machine translation. Around the same time, LSTM was proposed [7] which addresses vanishing gradient problem that was plaguing RNN architectures. The same year, attention mechanism was proposed by Bahdanau [8] which eliminated the requirement to encode the entire input to a fixed length representation. All of these advances help popularize the idea of applying seq2seq models for text understanding. Yao et.al. proposed an encoder decoder model with additional network to model the structure and intention of the initial input phrase [9]. This model fits unlabeled data better. Later, a seq2seq model with new objective function which includes mutual information was proposed [10]. In 2016, a generative seq2seq model was proposed that generates and/or retrieves the best response for an input email [11]. This model helped in suggesting semantically diverse text within the responses. A traditional generative model for dialog systems that emphasizes on the importance of hierarchy within previous dialogs [12]. More recently, personalized responses started gaining more importance when building chatbots. [13] proposed a speaker model that dictates the personality of the response and an addressee model that generates the actual response. [14] proposed a model to cater response to the sentiment of the dialog. Model proposed in [15] rewrites the initial query to a standard query form a trained model generated from question-form queries, that is useful in generating appropriate responses. Lee et.al. [16], propose using different models for different scenarios although their models are similar to [13] in ways of personalizing the response.

In recent times, text summarization is another area being explored heavily by the research community. The model proposed by Nallapati et.al. [17] creates summary of a document using key phrases, named entities etc. According to the authors, these features help in encapsulating most of the information in the short summary created for a long document. In [18], the authors propose word attention and contextual gates in a seq2seq model for text summarization and neural machine translation. A survey conducted by Allahyari et.al. [19] is a good source to get a primer on various text summarization techniques.

Some of the key takeaways for us from all these prior works are:

- Having contextual similarity and consistency in responses are essential for production-level generative chatbots.
- Diversity in responses is needed to cater to the skewness in the training data.
- A method to evaluate the model-proposed response is mandatory. It helps in deciding whether to show the model-proposed response to the user or to get more information from the user before responding.
- When proposing new methods and architectures, we need to be aware of additional complexity, measure usefulness & understandability of responses, have domain-specificity and be goal-driven.

III. Proposed Approach

In this paper, we propose two applications of sequence-to-sequence models that help us in providing better automated support to our engineers and customers. We identified some precursors that are needed to be able to build these models, such as: a custom word embedding and extensive data preprocessing.

A. Custom Word Embedding

Picking the best representation of data, in a vector space, before feeding it to a neural network is a vital step towards the success of the model's performance in the text domain. The Natural Language Processing (NLP) community has come a long way from the initial idea of one-hot encoding of vocabulary to contextual word embeddings. A word embedding is a vector that retains information of the source word when represented in a vector space that corresponds to the corpus. Glove [20] and Word2vec [21] are two of the most popular word embeddings that are publicly available today. But the emebeddings for words in these pre-trained models carry the context of the word with respect to the corpus the models were trained on. For example, the publicly available Google's pre-trained word embedding on news dataset is an excellent source for the NLP community in general. But if the data for a particular use case is completely different from the news dataset, then these pre-trained word embeddings are of little use because of improper coverage and lack of context. For example, coverage of our custom word embedding model on our text summarization training data is clearly greater than the public models, as shown in Table I. Besides coverage, the lack of context can be observed in Table II below. This table shows the synonyms or closest words for a few sample words in our dataset using the publicly available pre-trained word embeddings from Stanford [20] and Google [21]. The table also shows the closest words for the same source words when the word embedding model is trained specifically on our internal dataset. One can notice that the nearest neighbors for domain-specific source words fall in the same context when using custom word embeddings. It is because of this reason, we trained a custom word embedding model on our domain-specific data using word2vec algorithm. Our model preserves the context of words and meaning of a word in our corpus better than any publicly available word embedding. We opted for skip-gram version of word2vec because it is not biased towards frequent words, and this is a useful feature for our dataset.

Model	Vocabulary Size	Coverage	Unknowns
Custom	1.15 M	91.51 %	15629
Stanford [20]	2.19 M	90.69 %	17138
Google [21]	3 M	85.73 %	26261

TABLE I: Table showing data coverage and number of unknown words of different word embedding models.

B. Data Preprocessing

To build effective models, we had to take several steps towards data preparation, cleaning, fine tunning the parameters and the model architecture.

1) Data Preparation: Our Technical Assistance Center (TAC) dataset consists of 19 features including synopsis, problem description and solution. The synopsis and the problem description are provided by the customers who might or might not have a full understanding of the problem to begin with. The synopsis field serves the purpose of *title*, it is supposed to deliver a precise but short summary of the actual problem. In some cases, this short summary was extracted from the problem description while in other cases it is abstract. The solution field is typically a culmination of the chain of communications between the customer and the support engineers.

The training data for the chatbot model should be in the format of question and answer (QnA) pairs, on which the model learns the appropriate language. We had to extract these QnA pairs from the raw, unstructured data of our TAC database. Due to how a service request ticket can evolve over it's lifecycle, the question and answer pairs for the chatbot can come from synopsis and solution or from communications between customers and engineers. This may be a better representation of what that problem actually is.

As for training data for our text summarizer, problem description forms the long text and the corresponding synopsis forms the short summary (label). Since there were never any set standards for these fields, the original data consisted of significant noise. For instance, in many cases, customers included the errors in terminal, log messages etc. in the problem description and that made it hard to extract the actual problem the case was about. Moreover, this noise significantly increased the length of the description. To filter out undesired noisy description, we looked at the distribution of lengths (i.e. the number of tokens) of the description and realized the noisy data have large number of tokens. We set two lower and upper thresholds for the lengths of the descriptions to apply filtering.

Some of the data pre-processing techniques we used are listed below:

- Using regular expressions to capture and normalize the required patterns.

Source Word	Model	Top 5 Synonyms
packet	Word2Vec [21]	envelopes, packages, tins, boxes, cartons
	GloVe [20]	packets, tcp, multicast, datagram, udp
	Custom Word Embbedding	multicast, dropped, server-bulk, control-queue, default-clas
loss	Word2Vec [21]	losses, loosing, lost, setback, defeat
	GloVe [20]	losses, losing, loose, weight, lost
	Custom Word Embbedding	rounded-down, round-trip, dp-ex, packet, alarm-receive
fpc	Word2Vec [21]	eaglefly, pr-citizen, verysian, at-dot-com, jm
	GloVe [20]	ffc, pcb, cct, midp, jedit
	Custom Word Embbedding	pic, pfe, log, errors, vsrt-pcore
bgp	Word2Vec [21]	gateway-protocol-bgp, ospf, bgp-routing, eigrp, excluding-osis
	GloVe [20]	ospf, eigrp, vrf, ip-address, ipv6
	Custom Word Embbedding	ebgp, ibgp, flaps, ospf, peers

TABLE II: Table of words and corresponding synonyms for each respective word embedding model.

Token	Description	Tag
Users/jdoe/Desktop/MX-680	A user directory	<USRDIR>
www.juniper.net	A URL	<URL>
127.0.0.1 8080	IP address and the port number	<IP> <PORT>
BUG1234	A known bug	<BUG>
MX-360 400.600.41	Device Version	<VR>

TABLE III: Table of tags and the corresponding description.

- Filtering all the special characters that have their ASCII values above 127.
- Removing all the punctuations in two ways, if they are not accompanied by spaces on either side the symbol then it is replaced with a space or else it is just removed.
- Ignoring all cases that involved languages other than English.
- Removing all cases related to RMA (Return Merchandise Authorization).
- Removing all cases where the resolution is directed towards an existing Knowledge Base (KB) article or Problem Report (PR).

2) Text Normalization: After data preparation, we applied several techniques to make the data consistent and help the model better understand the pattern in the data. We started with simple popular techniques such as lowercasing, stemming and lemmatizing and stop word removals. Besides applying these popular data cleaning steps, we identified a list of *generic tokens*, created *tags* for each of those tokens and added them to our vocabulary. The tags were represented using random word vectors in the range $[-1, 1]$. This was needed due to the domain-specific nature of the TAC dataset. In the description, we had several instances of a IP addresses, VPN configurations, user directorates, URL's, device version numbers, software version numbers, communication protocols etc. Hence, we mapped each instance to the corresponding category and created a tag for each category, in total there were 13 patterns that we identified. Then, we replaced each match from a specific category with its category tag. This approach maintains the semantic dependencies among the n-grams and improved the performance of our model. Some examples of tokens and their corresponding tags are delivered in the Table III.

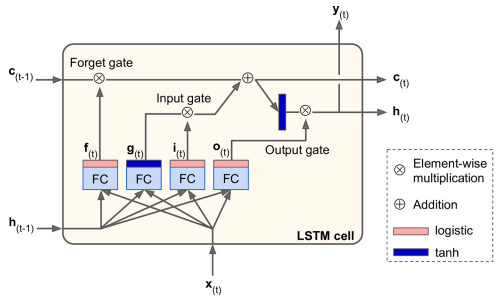


Fig. 1: An LSTM Cell, image taken from [22].

TABLE IV: LSTM Cell Equations

$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma(\mathbf{W}^t_{xi}\mathbf{X}_{(t)} + \mathbf{W}^t_{hi}\mathbf{h}_{(t-1)} + \mathbf{b}_i) \\
\mathbf{f}_{(t)} &= \sigma(\mathbf{W}^t_{xf}\mathbf{X}_{(t)} + \mathbf{W}^t_{hf}\mathbf{h}_{(t-1)} + \mathbf{b}_f) \\
\mathbf{o}_{(t)} &= \sigma(\mathbf{W}^t_{xo}\mathbf{X}_{(t)} + \mathbf{W}^t_{ho}\mathbf{h}_{(t-1)} + \mathbf{b}_o) \\
\mathbf{g}_{(t)} &= tanh(\mathbf{W}^t_{xg}\mathbf{X}_{(t)} + \mathbf{W}^t_{hg}\mathbf{h}_{(t-1)} + \mathbf{b}_g) \\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} \oplus \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\
\mathbf{h}_{(t)} &= \mathbf{o}_{(t)} \otimes tanh(\mathbf{c}_{(t)}) \\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)}
\end{aligned}
$$

C. RNN Model(s)

Sequence to sequence (Seq2seq) models are very versatile in that they can be trained on any paired sequence-type data, allowing us to apply the same architecture to both our proposed applications, although, they are generally used for Neural Machine Translation (NMT) based applications. These models usually involve an encoder-decoder architecture, where the encoder and the decoder are Recurrent Neural Networks (RNNs). These recurrent neural network models were initially proposed and implemented by [6] and [7]. The basic idea behind these implementations were to develope an encoder-decoder architecture using RNNs that can both train on and generate variable length sequences. The encoder network takes in a variable length input sequence and will steps in to each input time steps of the RNN and maps it to a fixed length vector called the *context vector*. The decoder later parses through the vector during each time steps and maps it back to a variable length target sequence. Nowadays, RNNs are widely being harnessed to handle variable length input sequences.

However, simple RNN models are not exactly learning what is a question and what is an answer, rather, they are

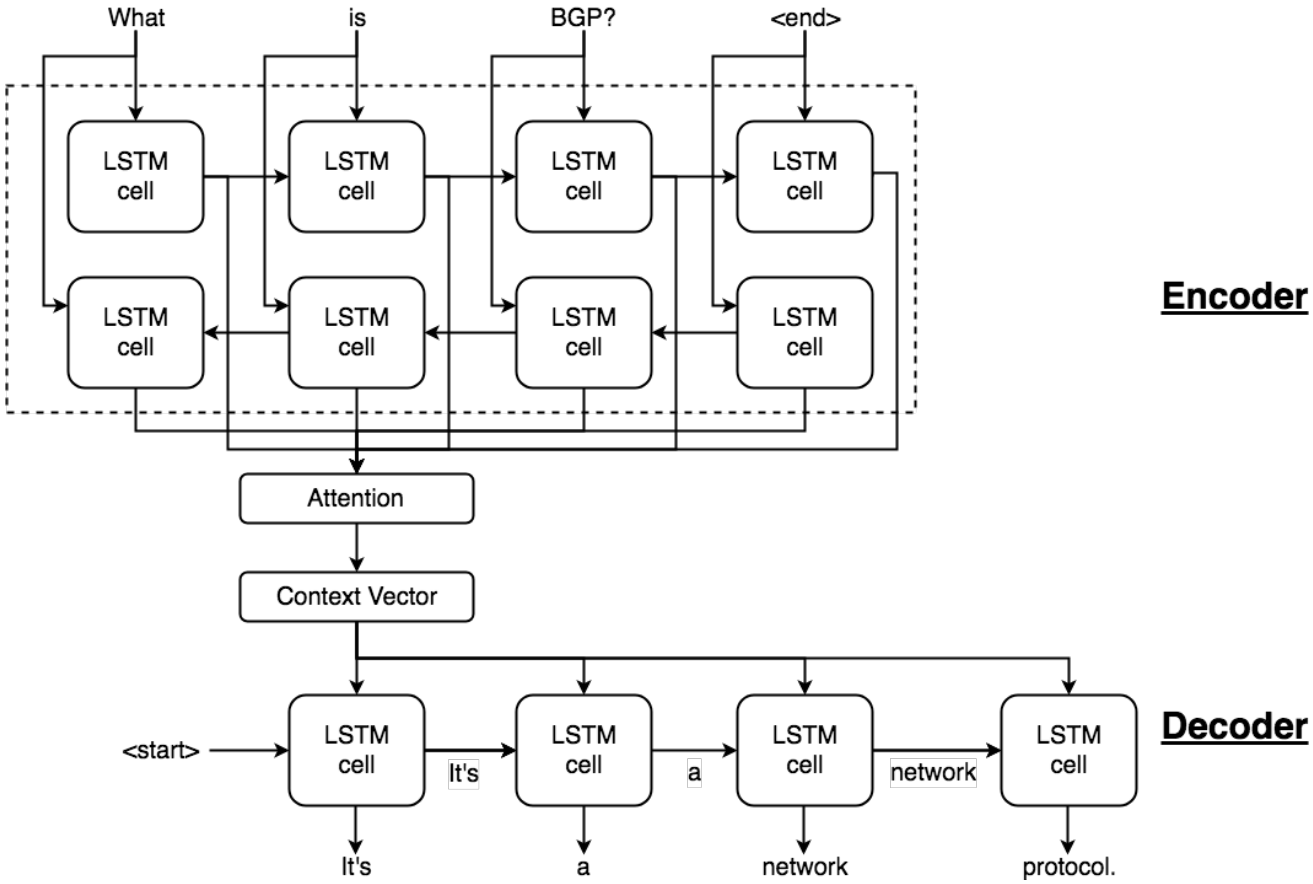


Fig. 2: The Encoder-Decoder Architecture for ChatBot and Summarization engine

learning properties of the training data such as grammar, classification etc. Such models are more suited for learning statistical patterns than semantic relations. Given that we don't have a well maintained and labeled dataset, this approach does not seem to fit our goals. From a machine learning perspective, another issue that comes with simple RNN models is the vanishing gradient and the fact that they become incapable of dealing with previous information as the document's length increases. To overcome this drawback, LSTMs (Long short-term memory) were introduced by [23]. These LSTM cells differ from traditional RNN cell in that they contain additional features such as input gate, output gate and forget gate to help retain long-term information through the network. A simple LSTM cell architecture can be observed in Figure 1 and Table IV. Here, $\mathbf{i}_{(t)}$ is the input gate that takes the current input, $\mathbf{X}_{(t)}$ at time t and the hidden state from the previous time step $\mathbf{h}_{(t-1)}$ and applies the sigmoid activation function σ. $\mathbf{W}_{xi}^{t}$ is the weights used by this gate and $\mathbf{b}_i$ is the bias. Information from the input gate is added to the forget gate $\mathbf{f}_{(t)}$ and the *tanh* gate $\mathbf{g}_{(t)}$ to get the new time step state $\mathbf{c}_{(t)}$. This new cell state $\mathbf{c}_{(t)}$ is then combined with the output of the forget gate $\mathbf{f}_{(t)}$ which computes how much information from the previous cell state $\mathbf{c}_{(t-1)}$ will be retained. This new cell state is used to generate the new hidden state output $\mathbf{h}_{(t)}$ with the help of a *tanh* activation function. This is also the actual prediction $\mathbf{y}_{(t)}$ of this LSTM cell for this time step. LSTMs does not focus on the contextual information from the future tokens, which may be helpful during it's training phase. To deal with this issue, we have used Bi-LSTM which processes the sequence backwards as well as forward, by which it exploits the future context. Here the output at each time-step will be the concatenation of the two output vectors from both directions.

Another technique that can enhance the model efficiency is attention. This mechanism in seq2seq models will help the network place higher emphasis on parts on the input that carries richer information. In an ordinary encoder-decoder model, the encoder reads the entire document and represents (memorizes) the entire document and delivers it to the decoder. Then, the decoder takes this numerical vector representation and will generate each output token in turn, this approach is different from how human translation is performed. Due to the challenge of trying to memorizes a new document and then translate it all at once, human translation is typically performed part by part. The translator will read a part of the document, translates it, and then go to the next part and perform the same actions. Mimicking this human translation approach, the attention mechanism helps identify what parts needs translation. In short sentences a regular encoder-decoder without attention mechanism would work well and they usually give high BLEU scores. But they can perform poorly on long documents which

may have dependencies on earlier parts of the document. The idea of attention mechanism was first appeared in an influential paper by Bahdanau et.al. [8]. It defined new parameters $\alpha_{t,j}$ as the amount of attention the model needs to give to j-th part to generate the output at time t. The *BahdanauAttention* class of TensorFlow implements this approach. First, they define a new hidden state h_j for every input $\mathbf{X}_j$ by concatenating the forward hidden states and backward hidden states from both LSTM layers and they call it the *annotation* for the input $\mathbf{X}_j$. Then, $\alpha_{t,j}$ is simply the weight to the annotation h_j at time t. Now, the new context vector c_t is the liner combination of the sequences of the annotations using these weights:

$$c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i \tag{1}$$

To determine these weights, we first need to normalize them so that they sum up to 1 at any time t and then we can set them to be a function of the previous hidden state and the current annotation state. To determine this function, Bahdanau et.al. defined a shallow network and let it fine-tune its own parameters. They call this latent function as the *alignment model* $e_{t,i} = a(s_{t-1}, h_i)$, which measures how much alignment the inputs at the neighborhood of i and the output at time t have. Here, s_t is an RNN hidden state at the decoder level for time t, computed by $s_t = f(s_{t-1}, y_{t-1}, c_t)$. Then, Bahdanau et.al. utilized this alignment score in an approach similar to softmax function to compute the weights $\alpha_{t,i}$. The denominator ensures that the sum of attention weights is one at any time t.

$$\alpha_{t,i} = \frac{exp(e_{t,i})}{\sum_{j=1}^{T} exp(e_{t,j})} \tag{2}$$

Figure 2 illustrates the architecture we used to build both models. Here *LSTM Cell* refers to the LSTM architecture explained in Figure 1. *Context Vector* refers to the fixed-length representation of the input, whether it is question for our chatbot or a document for our text summarizer. *Attn* is the attention given to specific parts of the input which is to be used alongside the previous *LSTM Cell* output to generate the next output token. Our encoder-decoder architecture based on Bi-LSTMs with attention mechanism also boasts the latest features such as dropout-wrappers, learning rate decay, early stopping, Adam optimizer and the previously specified domain-specific custom word embedding. Due to the extensive size and diversity of our datasets, creating a general architecture to furthur apply to our other applications (machine translation) was a point of emphasis.

IV. Experiments and Results

The infrastructure used for experimentation involves an Azure cloud virtual machine with the configuration of 6 VCPUs and 56GB RAM, powered by NVIDIA Tesla K80 GPU. We used python and Tensorflow [24] for coding our model architectures. Our text summarization model has 4 hidden layers where each layer has 512 hidden units. This model uses a range of training datasets from 44k documents with their summaries to 99k documents with their summaries, which were all extracted from our TAC dataset. It uses varying sizes of documents ranging from 9 words to 83 words (the size after cleaning). Our chatbot model has 3 hidden layers and uses 1024 hidden units per layer. This model uses a training dataset of 43k question and answer pairs that were extracted from our TAC dataset. It uses questions of varying lengths, ranging from 3 to 40 words per question. These thresholds guarantee only 10% data loss when compared to raw data but significantly increased the quality of the data. Unknown words in dataset account for 5-9%. Training time is in between 10 and 12 hours depending on the size of the model and the data.

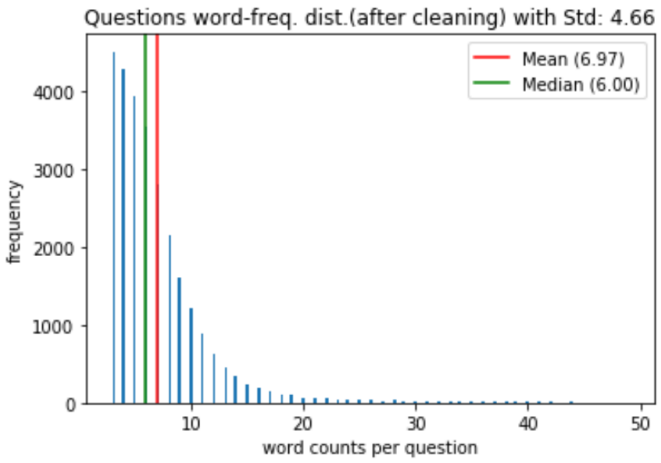
Model	Final Loss	BLEU	ROGUE
GRU	1.183	0.18	0.75
LSTM	0.754	0.19	0.59
Bi-GRU	0.563	0.18	0.69
Bi-LSTM	0.575	0.21	0.65

TABLE V: BLEU and ROGUE scores of different models for **Text Summarization.**

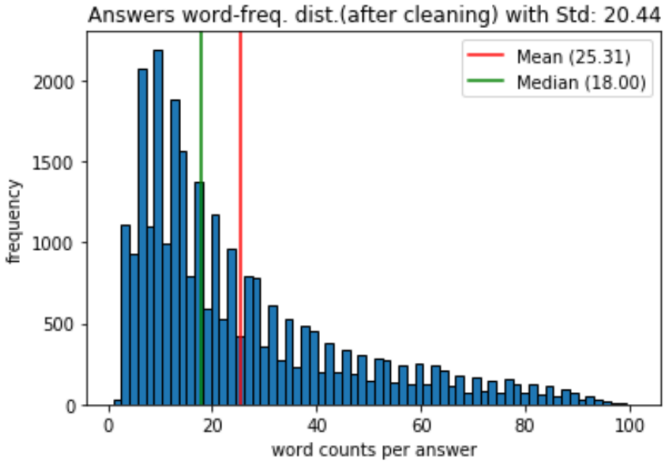
Some common features applicable for both models are as follows:

- Custom word embedding dimension size of 300 with vocabulary of 1.14 million.
- Word frequency minimum threshold of 20.
- Learning rate of 0.005.
- Optimization function was Adam.
- Loss function was weighted cross-entropy.
- Used dynamic Bahdanau attention.
- Keep probability of 0.75.
- If the update in loss does not decrease in 5 consecutive update checks, model stop training.
- Maximum number of training epochs was set to 20.

In this work we will only be evaluating our text summarization model. Two popular metrics have been applied to our text summarizer. The first metric is known as ROUGE and computes the portion of words in the generated summary to the number of words that were expected to generate. It finds the percentage of words that were expected to appear. The second metric is known as BLEU which computes the percentage of overlapping words to the total number of generated words. These definitions of ROGUE and BLUE dictate our n-gram size to 1. Our baseline summarization models built on uni-directional RNN networks had a mean BLEU score of 0.185 and a mean ROGUE score of 0.67 on this dataset. These architectures had relatively high training loss. But with the bi-directional RNN networks we could observe that the training loss decay was lower while the BLEU and ROGUE scores were higher. This can be observed from Table V. We attribute this to our task at hand, which is text summarization. Another interesting observation is that GRU network takes less time to train than its LSTM network counterpart because GRUs have fewer parameters and need less data to generalize. The greater expressive power of LSTMs typically yields better scores. For our particular use case, the BLEU scores matter

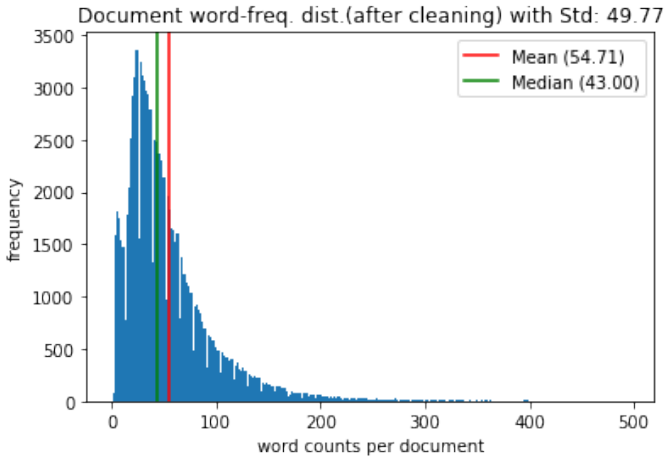


(a) Distribution of Question lengths

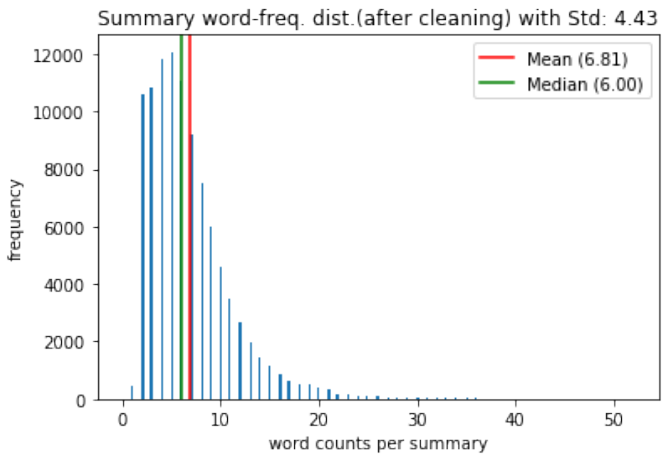


(b) Distribution of Answer lengths

Fig. 3: Query-Response data distribution for chatbot.

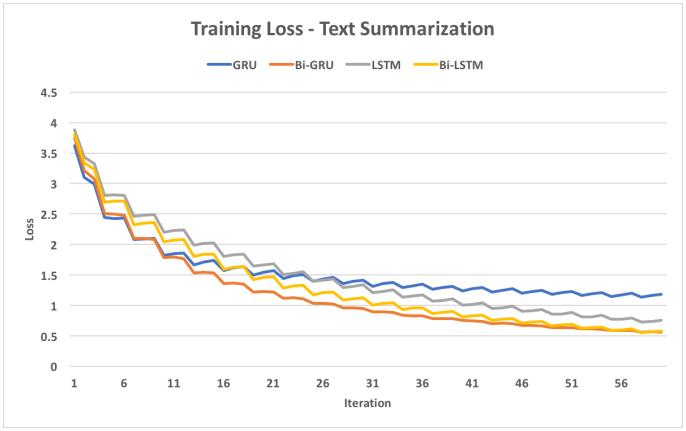


(a) Distribution of Document lengths

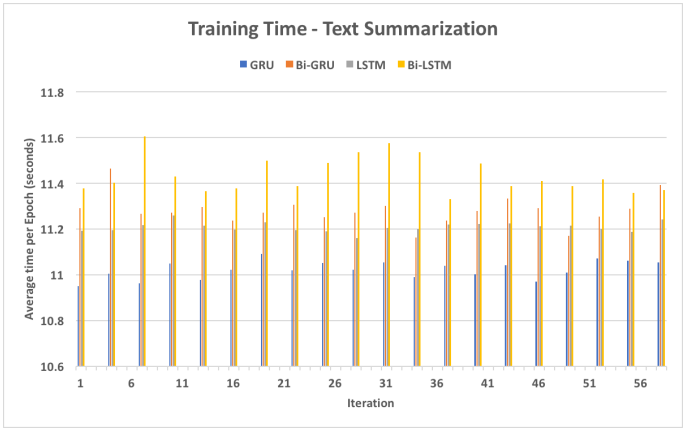


(b) Distribution of Summary lengths

Fig. 4: Document-Summary data distribution for text summarization.



(a) Error/Loss decay during training for text summarizer.



(b) Time taken during training for text summarizer.

Fig. 5: Training statistics for text summarizer.

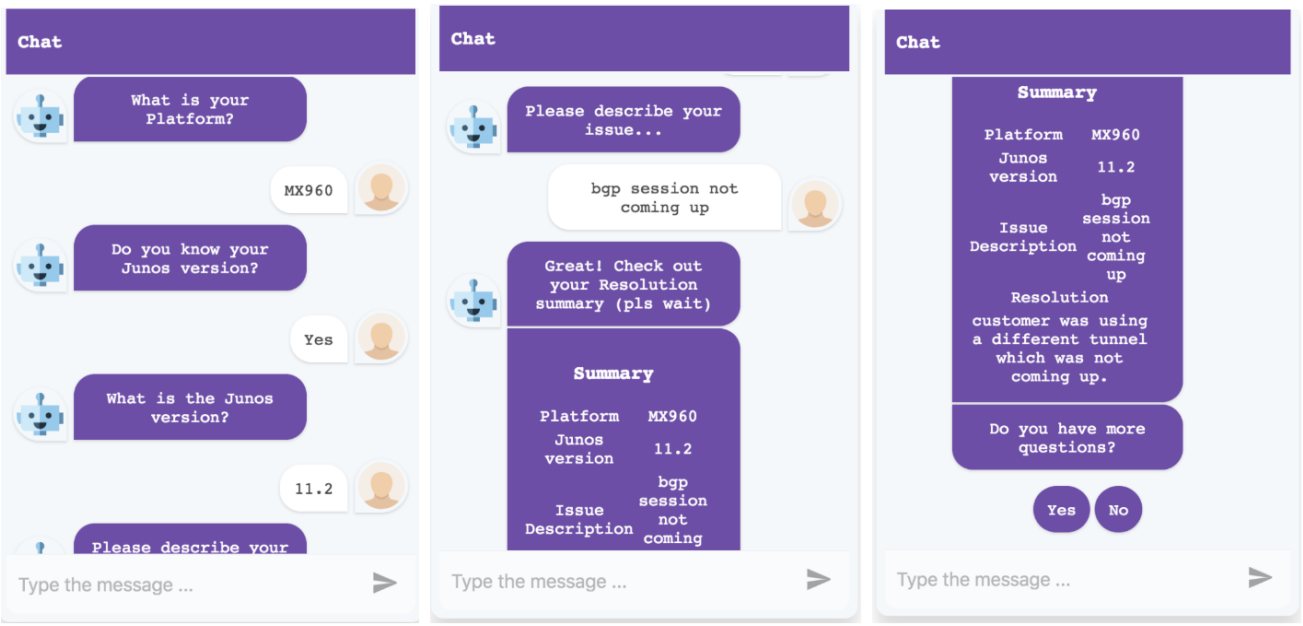


Fig. 6: Sample conversation from the developed **chatbot** model.

TABLE VI: Samples of machine-generated summary and human-given summary

Human Generated Summary	**Machine Generated Summary**
RPM configuration is not working	RPM probes issue
vLan is not connected to the correct domain	unable to ping the vLan
unable to access few sites from the external routing instances	unable to access sites from external routing
cpu spiking on fpc	cpu utilization issues
intermittent network outages and high core network cpu use	breaking network cpu

the most because they are more important with achieving an eye-test validation. Typically, the higher the BLEU score, the higher the eye-test satisfaction. Hence, we chose to use the bi-directional LSTM network architecture for our models.

Numerical evaluation metrics were helpful and necessary to us in evaluating progress while iterating on model design. But they were not sufficient for the business to evaluate the impact of proposed models. For this purpose, we needed to capture customer satisfaction and feedback. A stand-alone chatbot UI was developed using a template in React designed by Lucas Bassetti [25]. The backend was served by Flask in Python. Similarly for the text summarization we used Bootstrap for the front-end and Flask for serving the pages. These applications were released to a select set of domain specialists who were asked to give us regular feedback. So far, from eye tests and their self-evaluation, our target audience seem to be happy with our model performance.

The ideal way to evaluate our chatbot is by determining whether it satisfies its business purpose or not i.e., solving customer's issues. While the typical way of evaluating it is using set benchmarks. Unfortunately, it is not feasible for us to create such evaluation benchmarks now. This is due to lack of evaluation data in the form of past conversations. Also, it is really expensive to get large-scale human-judgment based scores for our chatbot. As explained by Liu. et al. [26], the standard metrics used for machine translation or text summarization do not correlate suitably with the human judgments over conversational agents. The metrics are primarily based on text matching, which is not the case in chatbots, because reasonable answers can have completely different set of words. Due to this lack of proper evaluation metrics for our chatbot, we had to restrict the evaluation of our chatbot to eye-tests from our subject matter experts. For this purpose we had to build a chatbot demo tool, as shown in Figure 6.

V. Conclusion

In this paper, we present the efforts of the support automation team at Juniper Networks, Inc. towards optimizing and automating our customer support with the help of sequence to sequence RNN models. Specifically, we explain the two bi-directional LSTM networks we developed to build our chatbot and summarization tool. Also, we cover the various preprocessing steps needed to implement these models, such as creating custom word embeddings and various data cleaning techniques. The proposed tools are still in their concept stage and we are actively working on them to add more features.

For future work, we are working on introducing practices within our organization that facilitate labeled dataset creation for evaluating machine learning models. We would also like to dissect the conversation aspect of chatbots in more detail.

For this, our bot has to understand the underlying context and preserve it throughout the session while being relevant in every response given to the customer. Finally, we want to explore other applications such as machine translation.

REFERENCES

[1] "Meet erica, your financial digital assistant from bank of america." [Online]. Available: https://promo.bankofamerica.com/erica/

[2] R. S. Wallace, "The anatomy of alice," in *Parsing the Turing Test*. Springer, 2009, pp. 181–210.

[3] J. Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation*. New York, NY, USA: W. H. Freeman & Co., 1976.

[4] S. Franchi and G. Guzeldere, "Constructions of the mind: Artificial intelligence and the humanities," 1995.

[5] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[6] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[7] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[9] K. Yao, G. Zweig, and B. Peng, "Attention with intention for a neural network conversation model," *arXiv preprint arXiv:1510.08565*, 2015.

[10] J. Li, M. Galley, C. Brockett, J. Gao, and B. Dolan, "A diversity-promoting objective function for neural conversation models," *arXiv preprint arXiv:1510.03055*, 2015.

[11] A. Kannan, K. Kurach, S. Ravi, T. Kaufmann, A. Tomkins, B. Miklos, G. Corrado, L. Lukacs, M. Ganea, P. Young *et al.*, "Smart reply: Automated response suggestion for email," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 955–964.

[12] I. V. Serban, A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau, "Building end-to-end dialogue systems using generative hierarchical neural network models." 2016.

[13] J. Li, M. Galley, C. Brockett, G. P. Spithourakis, J. Gao, and B. Dolan, "A persona-based neural conversation model," *arXiv preprint arXiv:1603.06155*, 2016.

[14] A. Xu, Z. Liu, Y. Guo, V. Sinha, and R. Akkiraju, "A new chatbot for customer service on social media," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 2017, pp. 3506–3510.

[15] Z. Yin, K.-h. Chang, and R. Zhang, "Deepprobe: Information directed sequence understanding and chatbot design via recurrent neural networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 2131–2139.

[16] C.-W. Lee, Y.-S. Wang, T.-Y. Hsu, K.-Y. Chen, H.-Y. Lee, and L.-s. Lee, "Scalable sentiment for sequence-to-sequence chatbot response with performance analysis," *arXiv preprint arXiv:1804.02504*, 2018.

[17] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang *et al.*, "Abstractive text summarization using sequence-to-sequence rnns and beyond," *arXiv preprint arXiv:1602.06023*, 2016.

[18] L. Wu, F. Tian, T. Qin, J. Lai, and T.-Y. Liu, "A study of reinforcement learning for neural machine translation," *arXiv preprint arXiv:1808.08866*, 2018.

[19] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut, "Text summarization techniques: a brief survey," *arXiv preprint arXiv:1707.02268*, 2017.

[20] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[21] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[22] A. Gron, "Hands-on machine learning with scikit-learn and tensorflow: Concepts, tools, and techniques to build intelligent systems," 2017.

[23] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," in *Advances in neural information processing systems*, 1997, pp. 473–479.

[24] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[25] L. Bassetti, "react-simple-chatbot," https://github.com/LucasBassetti/react-simple-chatbot, accessed 2018-10-06.

[26] C.-W. Liu, R. Lowe, I. V. Serban, M. Noseworthy, L. Charlin, and J. Pineau, "How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation," *arXiv preprint arXiv:1603.08023*, 2016.