Server-Less Rule-Based Chatbot
Using Deep Neural Network

Santhosh Kumar Nagarajan
Abstract

Server-Less Rule-Based Chatbot Using Deep Neural Network

Santhosh Kumar Nagarajan

Customer support entails multi-faceted benefits for IT businesses. Presently, the business depends upon on conventional channels like e-mail, customer care and web interface to provide customer support services. However, with the advent of new developments in Scania IT, different IT business units is driving a shift towards automated chatbot solutions to provide flexible responses to the user’s questions. This thesis presents a practical study of such chatbot solution for the company SCANIA CV AB, Södertälje.

The objective of the research work presented in this thesis is to analyze several deep learning approaches in order to develop a chatbot prototype using serverless Amazon Web Services components. The proposed bot prototype includes two main Natural Language Understanding (NLU) tasks: Intent classification and Intent fulfilment. This is a two-step process, focusing first on Recurrent Neural Network (RNN) to perform a sentence classification (intent detection task). Then, a slot filling mechanism is used for intent fulfilment task for the extraction of parameters.

The results from several neural network structures for user intent classification are analyzed and compared. It is found that the bidirectional Gated Recurrent units (GRU) were shown to be the most effective for the classification task. The concluded model is then deployed on the designed AWS stack. They demonstrate that the bot behaves as expected and it places more insistence on the structure of the neural network and word embeddings for future advancements in order to find an even better neural network structure.
Acknowledgements

I would first of all like to thank Uppsala University and my supervisor ArunKumar Krishnamurthy from SCANIA CV AB for providing a unique opportunity to work at the forefront of Scania IT technology solutions.

I take this opportunity to express my sincere gratitude and thanks to my reviewer Dr. Carl Nettelblad, for his extensive support and guidance throughout the thesis research work.

I would also like to thank my thesis coordinator Dr. Justin Pearson for his time and support.

Finally, a special thanks to my family and friends for their love, support and encouragement.
Table of Contents

Acknowledgements I
List of Tables IV
List of Figures V
List of Acronyms VII

1 Introduction 1
1.1 Chatbot: A definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.3 Goal for the master thesis . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.4 Guidelines and Constraints . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.5 A brief chronicle about chatbots . . . . . . . . . . . . . . . . . . . . . . . . 3
1.6 SCANIA CV AB . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2 State of the Art 5
2.1 High-level view of Chatbots . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.2 Chatbot Conversation Framework . . . . . . . . . . . . . . . . . . . . . . 6
2.3 Natural Language Understanding . . . . . . . . . . . . . . . . . . . . . . . 7
2.3.1 Recurrent Neural Network . . . . . . . . . . . . . . . . . . . . . . . . 8
2.3.2 Cell types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
2.3.3 Word embedding . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

3 Problem Formulation and Use Case scenarios 15
3.1 Problem description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
3.2 Identified Internal customer problems . . . . . . . . . . . . . . . . . . . . 15

4 System architecture 17
4.1 System modules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
4.1.1 Intent classifier . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
4.1.2 Content analyzer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
4.1.3 Business-logic executor . . . . . . . . . . . . . . . . . . . . . . . . . . 18
4.1.4 Dialogue manager . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
4.1.5 WSO2 API manager . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
4.2 Serverless AWS components . . . . . . . . . . . . . . . . . . . . . . . . . . 19
4.2.1 Amazon S3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

© Uppsala University
Santhosh Kumar Nagarajan
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.2 API Gateway</td>
<td>19</td>
</tr>
<tr>
<td>4.2.3 Lambda</td>
<td>19</td>
</tr>
<tr>
<td>4.2.4 SageMaker</td>
<td>19</td>
</tr>
<tr>
<td>4.2.5 DynamoDB</td>
<td>20</td>
</tr>
<tr>
<td>5 System Implementation</td>
<td>21</td>
</tr>
<tr>
<td>5.1 Programming language and libraries</td>
<td>21</td>
</tr>
<tr>
<td>5.2 Custom components</td>
<td>21</td>
</tr>
<tr>
<td>5.2.1 User Interface design</td>
<td>21</td>
</tr>
<tr>
<td>5.2.2 API design</td>
<td>22</td>
</tr>
<tr>
<td>5.3 Conversational agent responses</td>
<td>22</td>
</tr>
<tr>
<td>6 Experiments</td>
<td>23</td>
</tr>
<tr>
<td>6.1 Dataset generation</td>
<td>23</td>
</tr>
<tr>
<td>6.2 Preprocessing</td>
<td>24</td>
</tr>
<tr>
<td>6.3 Neural network configurations</td>
<td>25</td>
</tr>
<tr>
<td>6.3.1 Base network structure</td>
<td>25</td>
</tr>
<tr>
<td>6.3.2 Single LSTM layer topology</td>
<td>26</td>
</tr>
<tr>
<td>6.3.3 Reversed input sequence LSTM topology</td>
<td>27</td>
</tr>
<tr>
<td>6.3.4 Single GRU layer topology</td>
<td>29</td>
</tr>
<tr>
<td>6.3.5 Bilayer GRU topology</td>
<td>30</td>
</tr>
<tr>
<td>6.3.6 Bidirectional GRU topology</td>
<td>31</td>
</tr>
<tr>
<td>6.4 Epochs study</td>
<td>32</td>
</tr>
<tr>
<td>6.5 Optimal neural network</td>
<td>33</td>
</tr>
<tr>
<td>7 Results</td>
<td>34</td>
</tr>
<tr>
<td>7.1 NLU evaluation</td>
<td>34</td>
</tr>
<tr>
<td>7.2 Bot prototype mockup dialogues</td>
<td>35</td>
</tr>
<tr>
<td>8 Conclusion and Future work</td>
<td>37</td>
</tr>
<tr>
<td>8.1 Summary</td>
<td>37</td>
</tr>
<tr>
<td>8.2 Future work</td>
<td>38</td>
</tr>
<tr>
<td>Appendix</td>
<td>39</td>
</tr>
<tr>
<td>Literature</td>
<td>42</td>
</tr>
</tbody>
</table>
List of Tables

Table 1.1: Customer preferences for Chatbots over Apps . . . . . . . . . . . . . . 2
Table 3.1: List of Internal and External user problem scenarios . . . . . . . . . . 16
Table 6.1: Neural network performance summary . . . . . . . . . . . . . . . . . . 33
List of Figures

Figure 1.1: Scania CV AB .................................................. 4
Figure 2.1: Chatbot schema ........................................... 5
Figure 2.2: Chatbot Conversation Framework .................. 6
Figure 2.3: Temporal unfolding operation on Basic RNN structure ...... 8
Figure 2.4: RNN - The back Propagation through time .............. 9
Figure 2.5: Simple recurrent unit cell with single layer ............ 10
Figure 2.6: An unrolled cell of LSTM [17] .......................... 12
Figure 2.7: The Gated Recurrent Unit cell .......................... 13
Figure 2.8: One-Hot Encoding ......................................... 14
Figure 4.1: Serverless Chatbot Architecture ....................... 17
Figure 4.2: WSO2 API Manager ..................................... 18
Figure 4.3: Serverless AWS components ............................. 19
Figure 6.1: Internal and External user problem scenarios .......... 23
Figure 6.2: Internal and External user problem scenarios excluding Other Class 24
Figure 6.3: Base neural network structure .......................... 25
Figure 6.4: Single LSTM neural network topology ................. 27
Figure 6.5: Single LSTM neural network accuracy ................ 27
Figure 6.6: Reversed input sequence LSTM topology ............. 28
Figure 6.7: Reverse input sequence LSTM neural network accuracy .... 28
Figure 6.8: Single GRU neural network topology ................ 29
Figure 6.9: Single GRU neural network accuracy ................ 29
Figure 6.10: Bilayer GRU neural network topology .............. 30
Figure 6.11: Bilayer GRU neural network accuracy ............... 30
Figure 6.12: Bidirectional GRU topology .......................... 31
Figure 6.13: Bidirectional GRU network accuracy ................. 32
Figure 6.14: Bidirectional GRU network accuracy over 300 epochs .... 32
Figure 7.1: Normalized confusion matrix for Bidirectional GRU intent classifier 34

Figure A.1: Single LSTM neural network loss ...................... 39
Figure A.2: Reverse input sequence LSTM neural network loss .... 39
Figure A.3: Single GRU neural network loss ....................... 40
Figure A.4: Bilayer GRU neural network loss ...................... 40
Figure A.5: Bidirectional GRU neural network loss .............. 41
List of Figures

Figure A.6: Chatbot UI .................................................. 41
### List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short Term Memory</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>NLU</td>
<td>Natural Language Understanding</td>
</tr>
<tr>
<td>NER</td>
<td>Named Entity Recognizer</td>
</tr>
<tr>
<td>GRU</td>
<td>Gated Recurrent Unit</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>DRL</td>
<td>Deep Reinforcement Learning</td>
</tr>
<tr>
<td>CI/CD</td>
<td>Continuous Integration and Continuous Deployment</td>
</tr>
</tbody>
</table>
1 Introduction

This chapter formally introduces the context and a definition of chatbot in section 1.1. Then, the motivation, goal and design guidelines of the master thesis are explained in section 1.2, 1.3 and 1.4 respectively. Afterwards, a brief history of conversational agents is presented in section 1.5.

1.1 Chatbot: A definition

As per the Oxford English Dictionary, a chatbot is formally defined as follows:

chatbot (Noun) – "A computer program designed to simulate conversation with human users, especially over the Internet."

A chatbot most commonly referred to as a smartbot, chatterbot, artificial conversational agent or interactive agent.

The underlying concept behind every chatbot is to convincingly simulate how a human would interact as a conversational partner. In earlier 1950, Alan Turing defined a simple test referred to now as the Turing test [1], was to evaluate a machine’s ability to exhibit intelligent behaviour. Turing proposed that a computer can be said to possess artificial intelligence and passed the Turing test when the human evaluator cannot reliably distinguish the machine responses from the human. This testing standard is widely criticized over the years. For a chatbot, it is not necessarily required to have enormous open domain knowledge and it can spotlight on specific topics such as for instance assisting users to reserve a meeting room at the office.

Therefore, the adopted definition of a chatbot just "for this thesis" is a computer program that communicates by text in a humanlike manner providing services to end-users in order to accomplish a well-defined goal.

1.2 Motivation

Several studies have been conducted to research client preferences concerning customer services. To shed light on how chatbots are reshaping today’s online experiences, the teams at Drift, a digital conversational marketing platform in association with Survey-
Monkey Audience, Salesforce, and myclever have conducted a recent survey in 2018 to determine the customer sentiments around chatbot services [2]. According to the survey, 34% of the customers experience difficulties to navigate websites, and 31% says that they cannot get answers to simple questions through traditional communicational channels. On the other hand, customers see potential benefits such as 24-hour service (64%), instant replies (55%) and desired answers to simple questions (55%) in conversational agents. Chatbots are much preferred over other communication channels especially based on the survey results as represented in Table 1.1, chatbots outperformed apps in the following five benefits categories.

<table>
<thead>
<tr>
<th>Benefits categories</th>
<th>Chatbots</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick answers to simple questions</td>
<td>69 %</td>
<td>51 %</td>
</tr>
<tr>
<td>Getting 24-hour service</td>
<td>62 %</td>
<td>54 %</td>
</tr>
<tr>
<td>Quick answers to complex questions</td>
<td>38 %</td>
<td>28 %</td>
</tr>
<tr>
<td>Ability to easily register a complaint</td>
<td>38 %</td>
<td>28 %</td>
</tr>
<tr>
<td>Getting detailed / expert answers</td>
<td>28 %</td>
<td>27 %</td>
</tr>
</tbody>
</table>

Table 1.1: Customer preferences for Chatbots over Apps

1.3 Goal for the master thesis

The main purpose of this master thesis focuses on developing prototype/architecture for the server-less interactive chatbot application using deep learning and NLP. It should serve as a communication vector in SCANIA AB customer service and let customers send text queries regarding the most common issue they encounter regarding the company’s contract and resolve it.

Ideally, the chatbot should:

- Provide a chat interface to input user messages.
- Assist users with a subset of questions/problems that they often face.
- Deploy a scalable machine learning model for user intent prediction in the AWS environment.
- Recognize custom-defined business entities within a user utterance.
- Ask the user for further information in order to fulfil their requirement.
1.4 Guidelines and Constraints

The few guidelines enforced by SCANIA AB and chatbot design constraints are listed below:

- Be designed only using AWS Server-less components.
- The chatbot implementation must abide by the security principles of Scania CV AB.
- Be able to reply to users in real-time and provide 24 x 7 service.
- The developed software should be straightforward and technically documented.
- The developed machine learning model must consume an acceptable time for training.

1.5 A brief chronicle about chatbots

The term chatbot was coined by Mauldin [3] to represent systems aiming to pass the Turing test. The first instance of a chatbot released to a larger audience was ELIZA [4] in 1966. ELIZA was widely accepted as the first computer program capable to pass the Turing test. ELIZA was a program that simulated a psychiatrist and paraphrased the user input using natural language processing techniques [5]. Despite being relatively simple, ELIZA created the illusion of understanding the user with the help of predefined, randomly chosen human-like responses thereby making people think that they were talking to a real person.

Even after several decades, the later chatbots heavily followed ELIZA’s approach with some add-ons such as emotions management and speech synthesis. Then in 2001, came to a chatterbot agent named SmarterChild [6] developed by Colloquiss that operated on AOL and MSN messenger. Inspired by the rise of instant messaging, SmarterChild was focused to provide instant access to news, weather forecasts, store notes, trigger timed Instant Messenger reminders, sports etc. The important feature that SmarterChild held was, connected to an information repository and detained useful information about its user. The performance of the SmarterChild was hindered by the technical limitations of NLP at that time.

Watson is a question-answering computer system capable of answering questions posed in natural language. It combines artificial intelligence (AI) and sophisticated analytical software for optimal performance as a “question answering” machine. IBM Watson was specially developed to compete with the champions of the game of Jeopardy [7]. Since the competition was won by Watson in 2011; IBM Watson offers “Cognitive Computing” services to build conversational agents for various domains which can
Introduction

process a huge amount of data.

Later in the early 2010s came to the ascent of virtual assistants like Apple’s Siri, Google’s Google assistant, Amazon Lex, Microsoft’s Cortana etc. All of these above mentioned AI agents can assist in handling the mobile device without physically accessing it. Recently, they have incorporated enhanced capabilities like image-based search [8].

As inferred from the chatbot history, a lot of progress has been made in the field of AI and NLP. This does not imply that current conversational agents are flawless.

Deep Learning, a subset of machine learning composed of algorithms that permits flexibility for deep neural networks in learning latent structures. RNN provides systems with greater capability to find meaning and maintain state when processing sequences of data, such as words in sentences. RNN variant architectures help to overcome the limitation of context/intent recognition, an essential part of NLP thus allowing the development of robust chatbots.

1.6 SCANIA CV AB

The master thesis was done under the supervision of SCANIA CV AB which is a major Swedish manufacturer of commercial vehicles – specifically heavy trucks and buses. Apart from commercial vehicle manufacturing, Scania IT is an integral part of the entire Scania business and extends from infrastructure and development to maintenance and support. Scania IT solutions are crucial for the Scania business to work. Much of the company’s production is governed by IT and by placing Scania IT business internally, valuable proximity to Scania’s other business units is created. This forms the basis for a proactive approach.
2 State of the Art

This chapter reports some related works in the field of Conversational Agents and then formally elucidates the state-of-the-art Deep Learning (DL) and Natural Language Understanding (NLU) techniques used that are crucial in developing a system interacting using natural language.

2.1 High-level view of Chatbots

Conceptually, a conversational agent comprises a number of distinct components that work together to accomplish a task/goal. Figure 2.1 depicts the high level relations between each chatbot component architected as part of thesis work.

The user utterance is first processed by the Intent classifier module which infers the intent of the user. Afterwards, the recognized intent and slot entities combined with the inferred information retrieved from back-end systems helps to formulate an appropriate response or sequence of actions. If the intent is judged to be unclear, the conversational
agent may ask the user to provide additional details in order to interpret the context. Finally, the action handler module receives the selected action and executes it.

2.2 Chatbot Conversation Framework

A Chatbot provides responses to questions or performs tasks the user asks for. Questions can be either:

Open Domain

The chat conversations in open domain can go anywhere, users can ask for any kind of information. It is not necessary to achieve a well-defined goal or intention. The presence of infinite topics and factual information makes this a very challenging problem.

Closed Domain

A closed domain conversation focuses on solving a specific task/problem and usually, it focuses on one particular sector. Closed domain bots provide limited functionality addressing concrete business needs.

Once a question is asked, the Chatbot needs to provide an answer and potentially do a task. There are two main ways to generate a response:

Retrieval based system

Retrieval models follow a heuristic method to retrieve a predefined response/answer. The heuristic could be a simple rule-based expression or a complex machine learning classifier. It is easier to implement these models since they do not require huge data [9]. The prospect for using retrieval models is that it will not make grammatical mistakes and provides rigid business dialogue flows.
Generative based system

Generative models do not rely on pre-defined responses. Generative models are solely based on Machine Translation techniques, but instead of translations between languages, it translates from an input utterance to an output response using "sequence-to-sequence ML models". The lack of data conversations constraint at the company makes the generative model harder to implement.

2.3 Natural Language Understanding

NLU is a branch of artificial intelligence (AI) that understands the input made in the form of sentences and turns it into structured information. NLU involves two tasks: intent identification/classification and slot filling. Natural Language Processing (NLP) is the ability of a computer program to make machines understand the natural language. Natural Language Understanding belongs to the NLP family which comprises various components for text processing.

A NLP tokenizer splits the input utterance into tokens that correspond to words and punctuation symbols (if punctuation is irrelevant to the problem, it can be filtered out). NER labels sequence of words in an utterance which are the name of things, such as Person, Organization, Currencies, Date, Time, Numbers, Location and Country [10]. The NER task is much similar to Slot filling task, and the main difference is that NER provides no technique to recognize custom words in the text.

Both NER and Slot filling use sequence-tagging as an approach, but the goals are different. Slot filling [11] looks for specific pieces of information to fill the required infoboxes. For instance, from a user utterance ‘show flights from Boston to New York today’, NER tags ‘Boston’ and ‘New York’ as location whereas in Slot filling ‘Boston’ and ‘New York’ are the departure and arrival cities specified as the slot values in the user’s utterance.

Other widely used components of NLP are stemmers and lemmatizers. The role of both stemming and lemmatization is to reduce the inflectional and derivationally related forms of a word to its common base form.

This thesis work mainly focuses on two specific NLU tasks (intent classification and slot filling) using Tokenizer, Stemmer, Lemmatizer and Named Entity Recognizer NLP components.
2.3.1 Recurrent Neural Network

RNNs or Recurrent Neural Networks are neural networks especially designed to handle sequences of data. In essence, RNN are neural networks with loops to retain sequential information. A RNN cell contains an internal state $C_t$ at time $t$, which is fed back as input to neuron at next timestep. The neuron outputs a value $h_t$ at each timestep. The main drawback of RNN implementations is that they suffer from the exploding or vanishing gradient [12] problems as proven by Sepp Hochreiter and Jurgen Schmidhuber in 'Long short-term memory' [13]. The same network is used in different timesteps. Analysis of RNN is usually performed on the unfolded version of the network (see Figure 2.3). In this way, a recurrent neural network is transferred into a multilayer feed-forward network thus keeping the same weights on unrolled elements arising from the same recurrent cell [14].

![Figure 2.3: Temporal unfolding operation on Basic RNN structure](image)

Backpropagation

Backpropagation is shorthand for "backward propagation of errors" [15]. It is a method used for updating weights, i.e. training, neural networks, including RNN. It is an algorithm for decreasing a loss function, which can be the error relative of the actual output produced relative to some expected output for the given inputs.

Formally, the backpropagation algorithm is the following:

- For a given input pattern, propagate it through the network to get an output.
- Backpropagate the error between the predicted outputs against the desired outputs.
- Calculate the error function derivatives with respect to network weights.
- Adjust the weights to minimize the error.
- Repeat the above steps for other training samples.
The modified algorithm Back Propagation through time (BPTT) is basically the standard backpropagation on an unrolled RNN. The key difference is that, since the layers correspond to various time steps of the same cell, the weight updates are summed up in each instance.

As per Figure 2.4 the loss function for each observed error affects the present and earlier timesteps with partial derivatives. For instance, if we consider the gradient of the error $E_3$ with respect to inputs $x_0$, $x_1$, $x_2$ and $x_3$ we can apply the following chain rule:

$$\frac{\partial E_3}{\partial x_3} = \frac{\partial E_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial x_3} \quad (2.1)$$

$$\frac{\partial E_3}{\partial x_2} = \frac{\partial E_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_2} \cdot \frac{\partial s_2}{\partial x_2} \quad (2.2)$$

$$\frac{\partial E_3}{\partial x_1} = \frac{\partial E_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_2} \cdot \frac{\partial s_2}{\partial s_1} \cdot \frac{\partial s_1}{\partial x_1} \quad (2.3)$$

$$\frac{\partial E_3}{\partial x_0} = \frac{\partial E_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_2} \cdot \frac{\partial s_2}{\partial s_1} \cdot \frac{\partial s_1}{\partial s_0} \cdot \frac{\partial s_0}{\partial x_0} \quad (2.4)$$

2.3.2 Cell types

This section explains several cell types used as part of this thesis work. These cells are the basic units in neural network architecture.

Simple RNN

A simple RNN cell consists of two inputs and two outputs. As per Figure 2.5, $x_t$ is provided as input to the current time step $t$ while the other input comes from the previous step input and merges that information with the current input. Thus a recurrent...
cell captures some correlation data between the current data step and the previous one. The RNN cell produces an output $y_t$ that is the next layer of computation. The decision at time step $t - 1$ affects the decision taken at time $t$.

The two above mentioned inputs are concatenated and passed through a single feed forward layer (e.g. tanh, $\sigma$). The next state $h_t$ is computed as

$$h_t = \sigma(W \cdot [h_{t-1}, x_t] + b)$$  \hspace{1cm} (2.5)

The square brackets in equation 2.5 denotes the concatenation operation of $h_{t-1}$, $x_t$, and $W$ and $b$ are weights and biases respectively. Since the recurrent network loops back the output as inputs, the network faces two kinds of problems, namely exploding and vanishing gradients [16].

Vanishing gradient – In a RNN the gradients are backpropagated in time all the way to the initial hidden layer. The gradients from the deeper layers need to go through continuous matrix multiplications because of the chain rule, and they shrink exponentially for any value smaller than 1 and make it impossible for the network to learn anything.

Exploding gradient – Gradient values larger than 1, on the other hand, will grow exponentially, crashing the model, hence the name “exploding gradients” for this phenomenon.

The Simple RNN is not able to manage long-term dependencies due to the gradient problems. This problem has been analyzed in detail, leading to the proposal of other cell types.
Long Short-Term Memory

LSTM [13] is a complex cell that enhances the ability of RNN to store longer-term temporal information [13] and overcomes the limitation of decaying error backflow in RNN for storing information over extended time intervals. LSTM is an efficient gradient-based method and it has three gates namely Forget gate, Input gate and Output gate that decide how much of the previous cell state to retain, and how much of the current input is considered to calculate current state and current output.

Let \( x = (x_0, x_1, ..., x_t)^T \) be the input sequence, and \( h = (h_0, h_1, ..., h_t)^T \) be the hidden state provided by LSTM layer, then relation between \( x \) and \( h \) is given by the following equations:

- **Forget gate** – It decides how much of the previous hidden state to retain.
  
  \[
  f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)
  \]  
  (2.6)

- **Input gate** – It decides how much of the present input to consider.
  
  \[
  i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)
  \]  
  (2.7)

- **Cell state** – It simply combines the effects of the forget and input gates.
  
  \[
  C_t = f_tC_{t-1} + i_t \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)
  \]  
  (2.8)

- **Output gate** – It decides how much of the hidden state exposed to the output.
  
  \[
  o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)
  \]  
  (2.9)

- **Hidden state** – At the end, the new hidden is computed using the output gate.
  
  \[
  h_t = o_t \cdot \tanh c_t
  \]  
  (2.10)
The gates mentioned above are implemented with single layer feedforward networks. LSTMs are designed to remember the input state for a longer time than an RNN, hence allowing long sequences to be processed accurately. LSTM is a fundamental part of voice assistant architectures from Google, Apple, Amazon, Microsoft and other tech companies [18].

The LSTM solves the problem of vanishing gradients. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms. By truncating the gradient, LSTM learns to bridge minimal time lags in discrete time steps by enforcing constant error flow. Therefore, the backpropagation gradient neither vanishes nor explodes when passing through, but it remains constant. When the forget gate is ON having activation close to 1.0, then the gradient does not vanish. The forget gate activation never exceeds 1.0, therefore the gradient does not explode.

**Gated Recurrent Units**

The Gated Recurrent Unit (GRU) introduced by Kyunghyun Cho [19] is a variant of LSTM where the forget gate and input gate are combined into a single update gate. The hidden state and cell were merged together replacing the output gate. The reset and update gate controls how much the hidden unit remembers or forgets the information while processing a sequence. The major advantage of GRU over LSTM is that fewer parameters are used to tune thereby shorter training time for GRU.

Firstly, the GRU adds a reset gate $r_t$ that determines how much of the previous state $h_{t-1}$ is passed to the tanh layer. The update gate $i_t$ determines the weighting between the tanh and the previous state: if the values are near to 1, the cell is similar to simple RNN which suffers from vanishing gradients, but as the values of this gate decline the cell tends to retain its state unchanged.
Basic components of a GRU:

- Update gate \( i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \)
- Reset gate \( r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \)
- Current memory content \( \tilde{h}_t = \tanh(W \cdot [r_t \times h_{t-1}, x_t]) \)
- Final memory at current time step \( h_t = (1 - i_t) \times h_{t-1} + i_t \times \tilde{h}_t \)

The \( \sigma \) notation allows a GRU to carry forward information over many time periods in order to influence a future time period. In other words, the value is stored in memory for a certain amount of time and at a critical point pulling that value out and using it with the current state to update at a future date. Nonetheless, both the LSTM and the GRU seem to have comparable performance.

2.3.3 Word embedding

The bases of the recurrent network have been discussed in the earlier part, the other important part of the neural network is its input choice.

For the Intent classifier, the task seems to look at a piece of text (sentence) and predict the correct intent category the user talking about. However, all neural networks handle numerical inputs, therefore the input sentence has to somehow be converted into a sequence of numerical vectors. In the deep learning frameworks such as TensorFlow, Keras, this part is usually handled by an embedding layer which stores a lookup table to map the words represented by numeric indexes to their dense vector representations [20].
For categorical variables where no such ordinal relationship exists, the integer encoding is not enough. In fact, using an encoding where the neural network would assume an inherent ordering of categories tends to result in bad performance and unexpected results.

The most naive approach in word embedding is to consider “one-hot” code vector of words. A “one-hot” encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

![One-Hot Encoding](image)

Figure 2.8: One-Hot Encoding

The representation of one-hot encoding is straightforward to implement but it has some limitations, it highly depends on the input dictionary: the length of a hot coded vector is the same as the length of the dictionary. This implementation suffers from a big problem: two words with largely similar meaning will be considered as disparate words, therefore, any parameters learned from one word will not be applicable to the other.

Other techniques such as Stemming or Lemmatization try to generalize the words by stripping out suffixes and normalize the words to their root form. Stemming applies some rules brutally to remove the suffixes, whereas Lemmatization finds the lemma with linguistically correct rules. Despite the benefits of the above-mentioned techniques, certain informative details are removed that could be useful in some ways.

Using word embedding as inputs to RNN has the following advantages [21]:

- No “curse of dimensionality” thanks to the reduced size of the input.
- Semantic and syntactic relationship between associated words.

The deep neural network takes the sequence of embedding vectors as input and converts them to a compressed representation. The compressed representation effectively captures all the information in the sequence of words in the text. The deep network part is usually an RNN or some forms of it like LSTM/GRU.
3 Problem Formulation and Use Case scenarios

This chapter starts with a short description of the problem in section 3.1. Then, section 3.2 articulates the business use cases with accompanying description, as agreed with Scania CV AB.

3.1 Problem description

Presently, users report their issues in the web interface or contact the customer support through one of the supported communication vectors (E-mail, customer care phone call) to create an online incident and get a unique reference number to identify a particular incident report. The submitted reports follow a special schema which holds meta-data such as the date and time of the reported issue, description of the issue, tags used for filtering issues, etc.

A customer support agent then analyzes the recorded incidents through the web interface tool. Then the agent responds to those incidents by referring to company manuals or databases for specific information and close the incident tickets. If the incident needs to update the status or modify some current information pertaining to the contract, then the agent might have access to perform certain operations in some cases or escalate the incident ticket to the respective department.

Mostly, the recorded incidents refer to a subset of common questions and the vocabulary used by a customer to update their contract status is much different than a customer wishing to change their contact details. Therefore, such patterns could be exploited to automate certain tasks and can reduce the load on the customer support team. Good integration of chatbots in the customer experience can provide customers with timely and accurate responses thereby overcoming the limitations of a human agent.

3.2 Identified Internal customer problems

The foremost question to address is "Which customer issues should the conversational agent be able to address?". This is an obvious issue since the bot requires enough training
samples to classify user problems.

A set of user scenarios have been identified, with the help of the product owner at Scania, as adequate for this thesis. They are summarized in Table 3.1.

<table>
<thead>
<tr>
<th>User problem scenario</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Deployment details</td>
<td>Respond to frequently asked questions such as server configurations, deployment stages (development, testing and production) URL by referring to the service manuals.</td>
</tr>
<tr>
<td>Service availability check</td>
<td>The user would like to check whether a service is available in a particular country or not.</td>
</tr>
<tr>
<td>List product markets</td>
<td>Lists all markets that support a particular product/service.</td>
</tr>
<tr>
<td>Display contract details</td>
<td>The user would like to display all their contracts or a particular contract.</td>
</tr>
<tr>
<td>Contract modification</td>
<td>The user would like to modify their contract status or any other details pertaining to that contract.</td>
</tr>
<tr>
<td>Update contact information</td>
<td>The user would like to update their contact details such as E-mail, cell number etc…</td>
</tr>
</tbody>
</table>

For the above-mentioned use cases, the customer agent tags the respective department or refer product manual to fulfil the customer intent. An additional assumption is made that users will discuss only one problem at a time per incident raised.
4 System architecture

This chapter will talk about the overall architecture of the conversational agent.

![Serverless Chatbot Architecture](Figure 4.1: Serverless Chatbot Architecture)

4.1 System modules

4.1.1 Intent classifier

A classifier is a method to recognize pieces of data – in this case, identifying a sentence across diverse categories much alike how humans will classify data. One of the prime tasks of the conversational agent is Intent recognition. The Intent classifier module is responsible for inferring the user intent based on the user utterance/message. It is a multi-class classification problem. The intent classifier is based on a deep neural network model which associates the utterance against a label/intent.

4.1.2 Content analyzer

Every intent has a unique template (or frame) which may require certain slots (entities) to be filled in. The content analyzer parses the utterance and extracts the required slot
values from it. The extracted features are mapped against the slot templates [22]. If the bot does not hold enough information to process the request, then the lacking slots values are requested from the user in order to fulfil the request.

4.1.3 Business-logic executor

After the intent is recognized, its respective business logic is triggered. When the content analyzer completes the template with the necessary information, then the respective business rules are executed. This might include sending FAQ answers or updating the database details through an API call.

4.1.4 Dialogue manager

This module handles the information flow across the conversation. The problem behind slot filling implementation is that the bot needs to manage all kinds of information about the conversation like *What did the user just ask? Which slots need to be filled in? Which missing slot details the bot can ask for next?*. This context is built while conversing with the user. It will contain all the slot values needed in order to fulfil the user intention.

4.1.5 WSO2 API manager

The WSO2 API manager is a generic platform built at Scania using WSO2 [23]. This platform can be leveraged if there is a need to publish or subscribe to any on-premise APIs. For the thesis implementation, in order to access the on-premised database from the AWS environment, the bot subscribes to a published API using the WSO2 platform.

![Figure 4.2: WSO2 API Manager](image-url)
4.2 Serverless AWS components

4.2.1 Amazon S3

Amazon Simple Storage Service (Amazon S3) [24] is an object storage service that offers scalability, data availability, security, and performance. Amazon S3 provides easy-to-use management features whereby the data can be configured by finely-tuned access controls to meet specific business and compliance requirements. The Chat UI files are stored in an Amazon S3 bucket.

4.2.2 API Gateway

AWS API Gateway [25] is a service provided by AWS, that makes it easy for developers to create, publish, maintain, monitor and secure REST and WebSocket APIs dynamically. APIs act as an entry point for applications to access data or any functionality from the back-end services such as services running on Amazon EC2 functions, AWS Lambda, Static host web applications or any real-time monitoring applications.

4.2.3 Lambda

AWS Lambda [26] is a compute service by the AWS cloud platform, it is fully managed and there is no need to provision or manage servers. Basically, in AWS Lambda, the AWS user can upload a function which can receive data through either event invoked by other AWS services or even via an HTTP call with AWS API gateway. Lambda service is based on the principle "Pay only for the compute time you consume". AWS Lambda supports code in Java, Python and Node.js, and the service can launch processes in languages supported by Amazon Linux (including Bash, Go and Ruby). The business logic part of the chatbot application is implemented in AWS Lambda.

4.2.4 SageMaker

Amazon SageMaker [27] is a fully managed service that allows developers and data scientists to build, train, and deploy machine learning models. The Build module provides a hosted environment to work with your data, experiment with algorithms,
and visualize your output. The Train module allows for one-click model training and
tuning at high scale and low cost. The Deploy module provides a managed environment
for you to easily host and test models for inference securely and with low latency. The
deep neural network model for user intent classification task is deployed in SageMaker
and performs real-time predictions to identify user intention.

4.2.5 DynamoDB

Amazon DynamoDB [28] is the most prominent NoSQL cloud database provided by
Amazon Web Services. It is a completely managed NoSQL cloud database platform to
store, process and access data to support high performance and scale-driven applications.
It supports both document and key-value store models. DynamoDB is used to record the
conversation history, dialogue states and user-activity logs of the chatbot application.
5 System Implementation

5.1 Programming language and libraries

Having considered the scope of the thesis project and Scania’s software environment, Python 3.6 was chosen as the main programming language to implement the chatbot. Python is known for its simplicity and prolific machine learning libraries and frameworks. Python is a trade-off for building a quick yet functional prototype and reduce the cognitive overhead for future developers. The company predominantly uses the .NET framework (C# language) for the majority of their projects. Within the scope of this project, Python was much appropriate in order to build and iterate a minimum viable product at an accelerated pace.

The following libraries are used to implement the chatbot in practice.


2. Pandas [31] is a commonly used Python library providing for general data structures with high performance and great flexibility for manipulation and analysis.

3. NumPy [32] is extensively used in data science to perform mathematical and scientific operations. It supports large, multi-dimensional arrays and matrices.

4. Scikit-Learn [33] handles complex data algorithms with data visualization techniques. The Scikit-Learn’s train_test_split method helps to split the dataset into random train and test subsets.

5. NLTK [34] is a popular library for natural language processing (NLP). It provides linguistic features like tokenization, stemming, lemmatization, text classification, named entity recognition (NER), recognizing parts of speech (POS) etc.

5.2 Custom components

5.2.1 User Interface design

To interact with the bot business logic, a separate front-end has been designed to input the user utterance. The UI component is hosted on the Amazon S3 storage under
'static website hosting' [35]; it just passes the user input utterance to AWS Lambda and returns the response received. This reusable component acts as an interface between the front-end and the back-end component. The chat UI developed using Scania Bootstrap, Javascript and JQuery provide a visually appealing experience that enables the natural conversation flow between your chatbot and end users.

5.2.2 API design

The chatbot service is targeting a legacy product and its data centres are located at Scania on-premise. Since the chatbot platform is hosted in the AWS serverless environment and the bot needs to access the on-premise database for the fulfilment of the business logic; this task can be accomplished with the help WSO2 API manager as mentioned in section 4.1.5. A custom API has been developed using .NET Core 2.2 and published under the WSO2 API Manager.

An Access Token [36] is a credential that can be used by an application to access an API. Access Tokens can be either an opaque string or a JSON web token. They inform the API that the bearer of the token has been authorized to access the API and perform specific actions specified by the scope e.g. the HTTP verbs GET, POST etc. The published API incorporates Access Token authorization for the secured access of published API for a limited duration (5 minutes) [37].

5.3 Conversational agent responses

Owing to the issues considered in section 2.2, the approach considered for generating responses is retrieval-based. The module responsible for retrieving the responses based on the identified intent is the Business-logic executor, it holds all the logic pertaining to user intent. Certain services provided by chatbot are accessible based on the user access privileges. For instance, updating critical information related to the contract can be performed by the service owner, not the product owner/customer. The responses provided by chatbot have a structure that is similar to what a customer support agent might have replied.
6 Experiments

This chapter extensively focuses on the experiments performed to identify a suitable neural network structure for intent classification and discusses the validation results obtained.

6.1 Dataset generation

The data obtained from the incident reports cannot be directly fed as input to the classifier. The prime issue with the dataset is that most of the recorded data are tagged under the other category. Fortunately, the required amount of data to perform the experiment is collected and it identifies the most critical user problem scenarios.

The data-set comprises of 110000 reports. figure 6.1 shows the Internal and External user problems distribution. It can be observed that the deployment details and service check classes are prominent user problems. The figure 6.2 captures the problem distribution excluding the other class. Based on the filtered data, train-validation-test subsets are generated and performance analysis on the intent classifier is performed.

Figure 6.1: Internal and External user problem scenarios
6.2 Preprocessing

The raw textual data from the real world cannot be directly fed into the model. To clean the dataset following methods are applied.

1. Strip characters – Remove unimportant punctuation and special characters such as !"#$%&’()*/+,-.;::<=>?@[\]^_`{|}~
2. Tokenization – Split the sentence into words named as tokens.
3. Letter case – Lower case all the tokenized words.
4. Lemmatization – Lemmatization in linguistics, is the process of grouping together the inflected forms of a word as a single item.
5. Input encoding – Convert each token into indexes based on the texts’ vocabulary list.
6. Input padding – Pad the input sequence to make them equal by a fixed length of 50.
7. One-Hot encoding – After indexing the output class, perform one-hot encode them as discussed in section 2.2.3
8. Train, Validation and Test set – Split the dataset into three subsets of corresponding size 60/25/15.
6.3 Neural network configurations

The foremost NLU task in this thesis is intent detection task. The intent classification is considered as a multi-class classification problem; Neural networks model inputs the words contained in the sentence and outputs a vector of class probabilities. The class with the highest probability is considered as the intent category, a trait value of the input sentence.

This section elaborates different neural network topologies for the user intent detection and their performance.

6.3.1 Base network structure

Each varied RNN topology has the same base network structure as represented in figure 6.3. The input ‘X’ represents the pre-processed tokens. Each word in the sample is considered as a separate token and each token is index mapped against the training vocabulary list. The sequence of indexes is transferred into dense vectors of fixed size by an embedding layer [38]. This sequence of embedding vectors is parsed to the recurrent neural network structure. At last, the RNN’s output is passed through a softmax layer using the softmax function to show the probability distribution of output classes.

The Softmax function [39] widely referred to as normalized exponential function takes as input a vector of \( m \) real numbers and returns \( m \) probabilities. Prior applying to softmax layer, RNN’s output vectors could be negative or greater than 1; but the post
probability distribution of softmax layer components lies in between the interval \((0,1)\) and the components will add up to 1. The standard softmax function is defined as

\[
S(Z)_j = \frac{e^{Z^T W_j}}{\sum_{i=1}^{m} e^{Z^T W_i}} \tag{6.1}
\]

where ‘w’ is the weight vector of the connection between the RNN layer and softmax layer and ‘m’ is the output size of the softmax layer representing the number of classes. The sum of the resulting probability distribution from the softmax layer is defined as

\[
\sum_{j=1}^{m} S(Z)_j = 1 \tag{6.2}
\]

Categorical cross entropy [40] is the loss function used for the multi-class intent classification problem whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverge from the actual label. For multi-class classification, we calculate a separate loss for each class label per observation and add up the result.

\[
-\sum_{c=1}^{m} y_{o,c} \log(p_{o,c}) \tag{6.3}
\]

Where ‘m’ is the number of classes, ‘y’ is the binary indicator (0 or 1) if the class label ‘c’ is expected classification for observation and ‘p’ is the predicted probability observation of class ‘c’.

The Adam optimizer [41] is used for optimization in neural network training. It is a specialized gradient-descent algorithm that uses the computed gradient, its statistics, and its historical values to take small steps in its opposite direction inside the input parameter space as a means of minimizing a function.

### 6.3.2 Single LSTM layer topology

In the first network structure configuration, the RNN substructure in Figure 6.3 is replaced by an LSTM layer. Figure 6.4 represents the single LSTM network’s structure.
Figure 6.4: Single LSTM neural network topology

Figure 6.5 depicts that increase in the number of epochs elevates the accuracy of both training and validation set. During the training phase, the accuracy of the model fluctuates greatly. However, it still fits the learning curve of correct classification and records a maximum accuracy of 86.56% on the validation set. The cumulative training time was 56 minutes.

Figure 6.5: Single LSTM neural network accuracy

6.3.3 Reversed input sequence LSTM topology

The next structure implements a minute alteration to the single LSTM layer topology by reversing the order of the input sequence. Figure 6.6 represents the Inverse LSTM network’s structure.
Experiments

The LSTM is much capable of solving long term dependencies problem and learns much better when the input sequences are reversed but the target sentences are not reversed [42]. The phenomenon discussed is caused by the introduction of many short term dependencies to the dataset. Reversing the input sequence increases the memory utilization of LSTM. The training time (57 minutes) did not vary much than the single LSTM structure. Reversing the input sources recorded better validation accuracy of 93.99 % and did much better on long sentences than LSTMs trained on the raw source sentences, which suggests that reversing the input sentences results in LSTMs with better memory utilization.

Figure 6.6: Reversed input sequence LSTM topology

Figure 6.7: Reverse input sequence LSTM neural network accuracy
6.3.4 Single GRU layer topology

For the second neural network configuration, the RNN substructure in Figure 6.3 is replaced by a GRU layer. Figure 6.4 represents the single layer GRU network’s structure.

The training and validation set accuracy plots of single GRU layer is shown in Figure 6.9. The single GRU network has fluctuations almost similar to the single layer LSTM topology during the training phase. The maximum validation accuracy observed during model training is 87.85 %. The key difference between the above-mentioned topologies is the total training time; It took only 38 minutes to train the single GRU layer topology.

![Figure 6.8: Single GRU neural network topology](image)

![Figure 6.9: Single GRU neural network accuracy](image)
6.3.5 Bilayer GRU topology

The next RNN type considered is bilayer GRU. Another possible configuration to improve the network accuracy is to stack two GRU layer on top of each other as shown in Figure 6.10 for better capture of input information.

The accuracy of the above-mentioned topology is represented in Figure 6.11. It is evident from the graph that bilayer GRU performance is really stable with better accuracy and less jitter in loss value. The configured model captured a validation accuracy of 89.91 % but the captured total network training time is double the training time of a single GRU layer structure.
6.3.6 Bidirectional GRU topology

The other RNN topology considered is bidirectional GRU. Bidirectional wrapper combines the outputs of the forward and backward RNNs. Bidirectional structure leverages the network knowledge by feeding the input sequence in normal time order for one network, and in reverse time order for another; it allows the networks to have both backward and forward information about the sequence at every time step. Figure 6.12 represents the configuration of Bidirectional GRU topology.

The accuracy of the above-mentioned topology is represented in Figure 6.13. It is clear from the graph that the bidirectional GRU network performance is much better in terms of network accuracy and loss. The configured model captured a validation accuracy of 95.4 % with a training time of 53 minutes.
An epoch is defined as a single pass through your entire training set while training to have an erudite machine learning model. One epoch for the large dataset is much huge to feed at once. Therefore the epoch is processed through small batches. There are 32 batches in the devised neural network configurations. The neural network is trained for 300 epochs to ensure that the optimum does not surpass a window size of 30 epochs.
Figure 6.14 represents the neural network training accuracy over 300 epochs.

As shown in Figure 6.14 the model fits well during the initial learning phase. The validation accuracy tends to decrease over the number of epochs and the model lacks to generalize well. It is an evident sign of overfitting in this case. Generally, model overfitting could be avoided by adding noise and dropouts. In the era of big data, 60/25/15 dataset split for train, validation and test are no longer appropriate. It would be better to divide in 98/1/1 ratio. Thus, it can be intuitively understood that significant decrease in validation accuracy, there is no point training the model over more epochs as it has reached its maximum achievable performance.

6.5 Optimal neural network

Having considered the different network structures, the optimal network can be chosen to handle the intent classification task for effectiveness. Table 6.1 represents the neural performance table.

<table>
<thead>
<tr>
<th>Network topology</th>
<th>Validation accuracy</th>
<th>Validation loss</th>
<th>Network training time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single LSTM</td>
<td>86.56 %</td>
<td>0.71</td>
<td>56</td>
</tr>
<tr>
<td>Reversed Input LSTM</td>
<td>93.99 %</td>
<td>0.26</td>
<td>40</td>
</tr>
<tr>
<td>Single GRU</td>
<td>87.85 %</td>
<td>0.77</td>
<td>38</td>
</tr>
<tr>
<td>Bilayer GRU</td>
<td>89.91 %</td>
<td>0.31</td>
<td>73</td>
</tr>
<tr>
<td>Bidirectional GRU</td>
<td>95.41 %</td>
<td>0.22</td>
<td>53</td>
</tr>
</tbody>
</table>

Table 6.1: Neural network performance summary

The final network structure chosen is Bidirectional GRU; it provides substantial validation accuracy (higher the accuracy value is better) and validation loss (lower the loss value is better) with a considerable amount of network training time compared to other proposed neural network configurations.
7 Results

In this chapter of the thesis, the performance of the system was assessed quantitatively and discusses further the implemented bot prototype.

7.1 NLU evaluation

The confusion matrix captures the performance of the intent classifier by displaying expected values against predicted values. As shown in figure 7.1, the diagonal counts depict the elements where the predicted values are par with expected values. The off-diagonal counts depict the elements where the intent classifier got the prediction incorrect.

![Figure 7.1: Normalized confusion matrix for Bidirectional GRU intent classifier](image)

Figure 7.1: Normalized confusion matrix for Bidirectional GRU intent classifier
The agent seems to identify most of the classes correctly. However it confuses some product market elements for the deployment details elements. This could possibly because the input sentence did not have enough information to clearly distinguish between product market and deployment details class or the sentences are not marked correctly. Nonetheless, other classes are predicted as expected and the comprehensive performance is satisfactory. Finally, the test accuracy of 91.6 % is achieved using the bidirectional GRU neural network classifier.

7.2 Bot prototype mockup dialogues

The following are the sample dialogue scripts exchanged between the chatbot and the user during the bot prototype implementation for different use case scenarios.

The below-mentioned dialogue flow captures the Contract details user problem and it requires chassis number slot in order to fulfil the requirement.

Bot: Welcome to Scania Bot!! It’ll be my pleasure to assist you! How may I help you please?
User: Fetch the contract details for chassis A111
Bot: Kindly find the below provided contract details for chassis A111.
Status Id - 1, Market Code - SE, Service Object No - A111, Reg No - TEST, Contract No - A111, Start Date - 2016-01-01 00:00:00, End Date - 2020-12-31 00:00:00

The dialogue flow scripted below does not provide chassis information, which is mandatory for retrieving the contract details.

User: May I know the start date on the chassisport contract for Chassis
Bot: I need a valid chassis number in order to complete the given request
User: A1234567890
Bot: It’s not a valid local contract.

The next example deals with the Service check user problem. This task needs product / service and country slot values to fulfil the request.

User: Is Product-A deployed in Switzerland?
Bot: Yes, Product-A is available in Switzerland.

The third mockup case belongs to Product markets user problem. This tasks fetch product name as slot value from user utterance to list the product market countries.

User: list me all product name markets?
Bot: The product name is available in Sweden, Finland and Germany.

The fourth mockup case belongs to Contract modification user problem. It requires the existing contract type and new contract type as slot values. The adjective position (ADP) like from and to nuances are identified with the help of POS tagger.

User: Update the contract A12345 from type A to type B.
Bot: Shall I proceed further to update the contract A12345 from type A to type B?
User: yes
Bot: Thanks for your confirmation! your contract A12345 is updated.

The fifth mockup case belongs to Contact modification class. It requires the e-mail/phone number and contract number as slot values.

User: modify the e-mail address of the contract A12345.
Bot: Sure! Can I have your e-mail address to proceed further with your request.
User: petter@yahoo.com
Bot: Thanks for the information! Shall I proceed further to update your contract e-mail address to petter@yahoo.com?
User: ok
Bot: I have updated your A12345 details.

The last representative case belongs to Deployment details class. It requires the deployment environment name, product name and country as slot values.

User: provide the production stage details of Product-A deployed in Switzerland
Bot: Product-A Production details - DNS name, DB server name and DB credentials.
8 Conclusion and Future work

In this last chapter of the thesis, some conclusions are drawn by resuming some achievements and presenting some possible future works.

8.1 Summary

At first, the gist about the thesis goals and guidelines are presented followed by a quick review of the conversational agents, highlighting the theory and techniques behind those chatbots were laid down on paper in Chapter 1. The fulfilment of the thesis scope revolves around the below-mentioned goals:

i) A study of deep learning approaches that better suit the creation of a conversational agent.

ii) Analyze and select AWS server-less components needed for the bot architecture.

iii) Built a bot prototype using identified approach on AWS platform.

Secondly, In Chapter 2, an extensive study on neural network structures and word embeddings are discussed followed by the problem description and use case scenarios in Chapter 3. Then in Chapter 4, a scalable serverless AWS architecture was designed and moreover, the bot architecture was disintegrated into several system modules for handling a particular design task. Afterwards, experiments were carried out on the identified network structures like LSTM and GRU for user intent detection in Chapter 6. Finally, the bot prototype results were shown in Chapter 7. The neural network accuracy and loss were captured for each neural network configuration examined, and the developed bot prototype was tested by the potential user.

The developed bot architecture solution satisfies the defined goals and supports easy development and deployment. It followed the continuous integration and continuous deployment (CI/CD) pipeline for porting the developed solution across development, testing, staging and production environment. Furthermore, the developed bot adheres to the security principles and guidelines of the company. Additionally, the bot prototype provides scope for integration of additional user problems. Even though the proposed solution does not facilitate auto-generative responses, but still it works fine and provides responses as expected considering the requirements provided.
8.2 Future work

After all the devised solution is not an elixir, several possible enhancements could be implemented to improve the efficiency of the conversational agent. The following list provides suggestions to advance the prototype further.

- At present, the bot supports for the English language, other languages use cases could be explored.

- Ideally, the model could be trained automatically after a certain period of time.

- Deep reinforcement learning algorithms (DRL) [43] could be explored to train the model and reward the agent when desired results are produced.

- A combination of CNN and LSTM architecture could be explored. As per the publication [44], this architecture reduces the model complexity and training time.

- The bot can avail AWS services like Amazon Polly [45] that turns text into lifelike speech allowing to provide Text-to-Speech and Speech-to-Speech services.

- Multi-level context management shall be retained to support rich dialogue flows.

- Advanced NLU components like Spacy [46] and Stanford NLP [47] can be integrated for better natural language processing.

- The bot could provide better user experience through the user profile personalization strategy.
Appendix

Figure A.1: Single LSTM neural network loss

Figure A.2: Reverse input sequence LSTM neural network loss
Figure A.3: Single GRU neural network loss

Figure A.4: Bilayer GRU neural network loss
Figure A.5: Bidirectional GRU neural network loss

Figure A.6: Chatbot UI
Literature


