Towards a Data-Driven Automation of Customer Support

Ricardo Costa Dias Rei

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Dr. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Dr. João de Almeida Varelas Graça

Examination Committee

Chairperson: Prof. Dr. Nuno João Neves Mamede
Supervisor: Prof. Dr. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Members of the Committee: Prof. Dr. Bruno Emanuel Da Graça Martins

May 2019
Acknowledgments

First, I would like to thank Professor Luísa Marques da Silva Coheur for her guidance during this dissertation. The countless meetings we had and all the exchanged emails were crucial for the elaboration of this thesis. Also, the friendship developed along this period was important for keeping me motivated and happy throughout all the work.

Secondly, I would like to thank João Graça and all the Unbabel research team for stimulating my interest in the field and teaching me so much.

A special thanks to you to my family, my girlfriend, and my friends for always being present.
Abstract

Customer support is essential for user satisfaction and critical for business. Its automation is a current research challenge. Yet, due to a lack of available data regarding this scenario, academic research has been mainly focused on community support, which is different from the typical customer support, where human agents are paid to give support to clients under a standardized language style. In this work, we analyse how retrieval- and generative-based deep learning models behave in a customer support scenario via email. Also, in an attempt to unify both approaches, we develop a simple hybrid model and a Generative Adversarial Network to jointly train both approaches. Our results suggest that a generative approach, based on a Sequence-to-Sequence model with attention, might be more suitable than a more complex retrieval-based approach. In addition, our human evaluation suggests that combining generation with retrieved answers (our hybrid approach) can improve the final answer quality.

Keywords

Customer Support, Automatic Reply, Natural Language Processing, Deep Learning, Customer Support Automation
Resumo

Apoio ao Cliente é essencial para a satisfação do utilizador e fundamental num mercado cada vez mais competitivo. Como tal, a sua automação é atualmente um desafio de grande interesse industrial e académico. No entanto, devido à falta de dados disponíveis sobre este cenário, a investigação tem-se focado no Suporte baseado na Comunidade, que difere do Apoio ao Cliente tradicional, em que, agentes humanos, são pagos para dar suporte a clientes numa linguagem uniforme. Neste trabalho, analisamos o comportamento de modelos de aprendizagem profunda baseados em recuperação de informação e baseados em geração num cenário de Apoio ao Cliente por email. Além disso, na tentativa de unificar as duas abordagens, desenvolvemos um modelo híbrido e uma rede neuronal Geradora e Adversária que pode ser usada para treinar conjuntamente estes modelos. Os resultados obtidos sugerem que, uma abordagem generativa simples, baseada num modelo sequência-para-sequência com um mecanismo de atenção, pode ser mais adequada do que uma abordagem baseada em recuperação de informação. Além disso, a nossa avaliação humana sugere que combinar ambas as abordagens pode melhorar a qualidade da resposta final.

Palavras Chave

Apoio ao Cliente, Aprendizagem Profunda, Automação do Apoio ao Cliente, Aprendizagem Automática, Processamento de Língua Natural
# Contents

1 Introduction ............................................. 1  
   1.1 Goal and contributions ............................ 3  
   1.2 Document Overview ................................. 5  

2 Background .............................................. 7  
   2.1 Supervised Machine Learning ....................... 9  
      2.1.1 Perceptron .................................. 10  
      2.1.2 Multi-Layer Perceptron ...................... 11  
      2.1.3 Recurrent Neural Network .................. 12  
      2.1.4 Sequence-to-sequence Models ................. 15  
      2.1.5 Attention Layer ............................ 16  
   2.2 Unsupervised Machine Learning ................... 17  
      2.2.1 K-Means .................................. 17  
      2.2.2 Generative Adversarial Networks .......... 18  
   2.3 Text Representation Models ....................... 19  
      2.3.1 Sparse Models ............................ 19  
      2.3.2 Dense Models ............................ 21  
         2.3.2.A Single Value Decomposition .......... 21  
         2.3.2.B Neural Word Embeddings .............. 21  
         2.3.2.C Contextualized Word embeddings ... 23  

3 Related Work ........................................... 27  
   3.1 Available Email/Customer Support Corpus ........ 29  
   3.2 Automatic Email Answering ......................... 30  
      3.2.1 Rule-Based Automatic Email Answering .... 30  
      3.2.2 Corpus-based Automatic Email Answering ... 31  
   3.3 Dialog Systems .................................. 33  
      3.3.1 Retrieval-based Systems .................... 34  
      3.3.2 Generative-based Systems .................. 36
List of Figures

1.1 Overview of the proposed Answer Suggestion system pipeline. .......................... 3

2.1 Example of a K-Nearest Neighbors (KNN) from Adi Bronshtein tutorial. The circle in the middle is a new input to be classified. If the algorithm is considering only 1 neighbour, the new input is classified as belonging to class 1 (blue square). Taken from: https://tinyurl.com/ybrvz4og .......................................................... 9

2.2 Comparison between a mini-batch gradient descent updates and the Stochastic Gradient Descent (SGD) updates for finding the minimum of a function. Taken from: https://engmrk.com/mini-batch-gd/ ............................................ 11

2.3 Graphical representation of an Multi-Layer Perceptron (MLP) with only one hidden layer from Matt Mazur blog post. Taken from: http://tinyurl.com/j5vk5stu ............................. 12

2.4 Graphical representation of a Recurrent Neural Network (RNN) and the respective development of the underlying computations. Taken from: http://tinyurl.com/q6dcybc .......................................................... 12

2.5 Graphical representation of a vanilla RNN cell. Taken from: http://tinyurl.com/y4hjswmn .......................................................... 13

2.6 Graphical representation of a Long Short-Term Memory Network (LSTM) cell. Taken from: http://tinyurl.com/yy79zmxo .......................................................... 14

2.7 Graphical representation of a sequence-to-sequence encoder decoder with LSTM neural units. Taken from: https://tinyurl.com/be3neh5 .......................................................... 16

2.8 Example of 3 clusters created by the K means algorithm with k=3. Taken from: http://arun-aiml.blogspot.com/2017/07/k-means-clustering.html .......................................................... 17

2.9 Graphical representation of the Continuous Bag-of-Words (CBOW) and the Skip-Gram models proposed by Mikolov et al. (2013a). .......................................................... 22

3.1 Example of an email labelled with the corresponding intent and the triggered template answer. .......................................................... 31

3.2 Prototype-and-edit model architecture proposed by Wu et al. (2018). .................. 38

4.1 Siamese-Like Dual Encoder model inspired in (Lu et al., 2017) architecture. .... 46
4.2 Bidirectional Dual Encoder with Dual Attention. .......................... 47
4.3 ELMo Dual Encoder model architecture. ................................. 47
4.4 Sequence-to-Sequence (Seq2Seq) model with attention. ............... 48
4.5 Sampling example for partial sequences from (Yu et al., 2017). In this example 4 full sequences are generated by sampling from the softmax output and the average reward is used as reward for the third token. ........................................ 50

5.1 Loss curve comparison over epoch for the ELMo Dual Encoder model trained from scratch with Twitter data (left side) and adapted from Ubuntu Dialog corpus to the Twitter data (right side). In both figures the green line represents the training loss while the blue line represents the validation loss. ........................................ 57
List of Tables

3.1 State-of-the-art results for the Ubuntu dialog corpus (single model) 36
4.1 Email Corpus Statistics 41
4.2 Twitter Corpus Statistics 43
4.3 Ubuntu Dialog Corpus Statistics 43
4.4 Customer context/question plus the following answers across the 3 different corpus 44
5.1 Ubuntu Dialog corpus ranking task results 56
5.2 Twitter Customer Support corpus ranking task results 57
5.3 Private Email corpus ranking task results 57
5.4 Answer generation results for the Ubuntu Dialog corpus v2.0 58
5.5 Answer generation results for the Twitter Customer Support corpus 58
5.6 Answer generation results for the private email corpus 58
5.7 Customer question plus the answers given by our models plus the ground truth answer 59
5.8 ELMo Dual Encoder Attention weights over a short context sentence 60
5.9 ELMo Dual Encoder Attention weights over a long context with multiple turns 60
5.10 Seq2Seq context/answer attention alignments during decoding. In this Figure, dark blue squares represent strong attention values over the column token while decoding the row token 61
5.11 Scores given by human annotators on the different models. CI stands for confidence interval 62
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVD</td>
<td>Single Value Decomposition</td>
</tr>
<tr>
<td>LSI</td>
<td>Latent Semantic Indexing</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>CBOW</td>
<td>Continuous Bag-of-Words</td>
</tr>
<tr>
<td>SGD</td>
<td>Stochastic Gradient Descent</td>
</tr>
<tr>
<td>ELMo</td>
<td>Embeddings from Language Models</td>
</tr>
<tr>
<td>Seq2Seq</td>
<td>Sequence-to-Sequence</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>TF-IDF</td>
<td>Term frequency-inverse document frequency</td>
</tr>
<tr>
<td>EOS</td>
<td>end-of-sentence</td>
</tr>
<tr>
<td>POS</td>
<td>Part-of-speech</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machines</td>
</tr>
<tr>
<td>tanh</td>
<td>Hyperbolic Tangent</td>
</tr>
<tr>
<td>NB</td>
<td>Naive Bayes</td>
</tr>
<tr>
<td>AERS</td>
<td>Automatic Email Responding System</td>
</tr>
<tr>
<td>KNN</td>
<td>K-Nearest Neighbors</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolution Neural Network</td>
</tr>
<tr>
<td>GAN</td>
<td>Generative Adversarial Network</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-Term Memory Network</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>MRR</td>
<td>Mean Reciprocal Rank</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>MLE</td>
<td>Maximum Likelihood Estimation</td>
</tr>
<tr>
<td>NLL</td>
<td>Negative Log-Likelihood</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-Layer Perceptron</td>
</tr>
<tr>
<td>LCS</td>
<td>Longest Common Subsequence</td>
</tr>
</tbody>
</table>
Introduction

Contents

1.1 Goal and contributions ........................................ 3
1.2 Document Overview ........................................... 5
Customer support is considered essential on the ramping technology-driven market, as every company that supplies a service needs to provide information to its customers, being that the effectiveness of this interaction is crucial for user satisfaction.

Companies around the globe have reached the same conclusions, this is, as their services/products reach a great number of users, customer support becomes critical for business competition with other companies in the same marketplace. Several studies (Hennig-Thurau and Klee, 1997; Negash et al., 2003) show that establishing a successful relationship with a customer is a key factor in obtaining a long-term customer and ultimately, the company’s success. The outcome of this interaction can be determinant for customer retention, as a successful one is directly linked to the increase of trust by the customer and its perception of commitment to the company.

1.1 Goal and contributions

This thesis goal is, thus, to build a system capable to assist human agents that perform customer support, by automatically providing them answers to the users’ requests. These answers can be used directly or as suggestion (Figure 1.1). We hope in this way, to help reducing the human effort and costs per user request\(^1\).

![Figure 1.1: Overview of the proposed Answer Suggestion system pipeline.](https://aircall.io/blog/customer-service-automation/)

Maybe due to a lack of publicly available data, most of the previous work in customer support is done with small in-house datasets, not taking advantage of the recent advantages in Natural Language Processing (NLP) considering deep learning. Actually, taking into account the huge amounts of data from email conversations between customers and support agents collected daily by companies\(^2\), we believe that deep learning models perfectly fit our scenario. Therefore, we will make use of modern deep learning methods in NLP that, to the best of our knowledge, have not been properly explored to automate email answers. In this work, we will explore recent results within two deep learning approaches: retrieval- and generative-based. From an architecture point of view, for the retrieval-based approach we will test Dual Encoder Neural Networks; for the generative-based approach, we will make

---

\(^1\)According to Radicati email statistics report\(^3\), in 2021 the total number of business and customer emails exchanged is expected to be of 319.6 billion.

---
use of Sequence-to-Sequence (Seq2Seq) models. In both approaches we will incorporate attention mechanisms and leverage the advantages of transfer learning. In addition, and following the approach presented in Yu et al. (2017), we will take advantage of the obtained retrieval and generative models, and jointly improve them with a Generative Adversarial Network (GAN), in which the Dual Encoder works as a discriminator and the Seq2Seq model as a generator. Also, we will test an hybrid model that combines the generative and the retrieval approaches, ensuring that the returned answers are fluent and grammatically correct even when the generative model is not.

In order to evaluate our work, we will use a private email corpus provided by Unbabel. In addition, in order to benchmark our models with a publicly available corpus, we will make use of the Twitter Customer Support Corpus from Kaggle. This is a good example of a real customer support scenario but, in contrast to an email scenario, contains short messages with long conversations. Also, and considering that recent academic research has been focused in Community Support via chat, we will also test our models in the Ubuntu Dialog Corpus, which is a well-known community support corpus. It should be clear, however, that community support is different from customer support, as, in the latter, human agents, engaged by a company, perform the support to their clients following a standardized language style, which is not visible in the former. In fact, by analyzing emails sent by customers and replied by human agents, we have verified that many emails are replied with the same templates. In addition, and in contrast to chats, emails are characterized by long texts with only a few messages exchanged between customer support agent and client.

At first, we believed that Dual Encoder models could provide better results since they rely in previously seen answers. After our experiments, we concluded that, in contrast with other domains, the standardized language and template usage benefits Seq2Seq models, making them able to generate grammatically correct answers. We also show that adapting a Dual Encoder model from a community support scenario to a much more standardized domain such as the customer support, yields to significant performance gains. Finally, we present a set of failed experiments with GANs that can help future researchers understand the behavior of generative model when different reward policies are applied. We also perform a human evaluation and concluded that, in contrast to the obtained automatic results, mixing generation with retrieved answers can improve the final answer quality.

Finally, we should say that during the development of this thesis, we have submitted a paper to ACL Student Research Workshop\(^4\) taking place in conjunction with ACL 2019 in Florence, Italy.

\(^4\)https://sites.google.com/view/acl19studentresearchworkshop/home?authuser=0
1.2 Document Overview

This document is organized as follows: Chapter 2 presents the background, and Chapter 3 the related work. In Chapter 4 we describe the proposed models. In Chapter 5 we evaluate our work. Finally, Chapter 6 presents the main conclusions from this work and possible ideas to implement in a future work.
# Background

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Supervised Machine Learning</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Unsupervised Machine Learning</td>
<td>17</td>
</tr>
<tr>
<td>2.3 Text Representation Models</td>
<td>19</td>
</tr>
</tbody>
</table>
In this Chapter we will start by presenting the underlying machine learning concepts of this thesis both supervised (Section 2.1) and unsupervised (Section 2.2). Then, we will present how we can transform natural language texts into a machine-interpretable representation (Section 2.3).

2.1 Supervised Machine Learning

Supervised Machine Learning is a category of algorithms that learn from externally supplied instances to produce a mapping function \( f(x) \) between a set of variables \( X = \{x_1, x_2, ..., x_n\} \) and a set of variables \( Y = \{y_1, y_2, ..., y_m\} \). Essentially, these algorithms uncover general hypotheses from the supplied data (training data) that can then be used to make future predictions in unseen data (test data).

Supervised learning problems can be further grouped into regression and classification depending on \( Y \) values. If \( Y \) is composed of continuous variables of real values, then we have a regression problem, if \( Y \) is composed by a finite set of categories, then we have a classification problem. With that said, one of the simplest classification algorithms is the K-Nearest Neighbors (KNN) (Fix and Hodges, 1951). This algorithm is non-parametric, meaning that it does not make any assumptions on the underlying data distribution. Considering a set of features, that is, individual measurable properties or characteristics of a phenomenon being observed, this algorithm assigns a label to a new input based on feature similarity to the training examples (Figure 2.1). Since the decision boundary of the KNN is formed by locally linear segments but, together, these segments can form complex shapes, this algorithm is considered non-linear in contrast to linear methods in which the decision boundary on the feature space is a linear function.

![Figure 2.1: Example of a KNN from Adi Bronshtein tutorial. The circle in the middle is a new input to be classified. If the algorithm is considering only 1 neighbour, the new input is classified as belonging to class 1 (blue square). Taken from: https://tinyurl.com/ybrvz4og](https://tinyurl.com/ybrvz4og)

Linear methods are usually the first algorithms people learn for estimate \( f(x) \) in predictive modelling and can provide strong baselines in many problems. At their core, these methods aim to define the function \( f(x) \) as a linear combination of observed features. Linear methods can either be generative or discriminative. In generative methods (e.g. Naïve Bayes classifier) \( f(x) \) is a probability function that is computed by counting and normalizing events. Typically a label is assigned to a new input based on the
maximum a posteriori decision rule. In contrast, discriminative approaches involve computing a set of parameters \( \theta_{\text{model}} \) that minimize an error function (e.g., the sum of squares or hinge loss). The computed parameters \( \theta_{\text{model}} \) can then be used to make predictions in unseen data.

Artificial Neural Networks are a class of discriminative models that are loosely inspired in the way biological neural networks process information. McCulloch and Pitts (1988) proposed a simplified model of the human neuron as a computing element that could be described in terms of propositional logic. Depending on the task at hand, there are many different types of neural networks (e.g., Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Multi-Layer Perceptron (MLP), etc.).

Neural networks have been showing impressive results in a wide variety of NLP tasks, such as, machine translation, structured prediction, and question-answering. These achievements have been mainly obtained with deep neural architectures, that is, neural networks composed of multiple nonlinear transformations.

Deep neural networks have been around for several years now, but, until recently, training this kind of architectures was a difficult task. Fortunately, nowadays with the increase of the computational power, the amount of available training data, the usage of new non-linear functions, etc., this kind of architectures became easier and accessible to almost everyone which has sparked a variety of new models and methods for NLP.

In the next sections, we will explain the basic concepts and all the neural network architectures that will be used during this thesis.

2.1.1 Perceptron

The building block of a neural network architecture is the Perceptron. Formally, the Perceptron algorithm can be defined as follows:

\[
\text{Perceptron}(x) = g(xW + b)
\]  

Where \( x \) is an input vector containing \( n \) numeric features, \( W \) is a weight matrix and \( b \) a bias term. Essentially, when the Perceptron receives an input \( x \) it applies a linear transformation to \( x \) via \( W \) and, then, the resulting values are summed with the bias terms to create a score \( z = xW + b \). Finally, \( z \) is passed through an activation function \( g(z) \) that allows the Perceptron to learn a non-linear transformation. Commonly used \( g \) functions are: sigmoid, Rectified Linear Unit (ReLU), Hyperbolic Tangent (tanh), softmax, among others.

One way to train the Perceptron and approximate its function \( \text{Perceptron}(x) \) to \( f(x) \) is to define a loss function and apply the Gradient Descent algorithm. The Gradient Descent is a first-order iterative optimization algorithm for finding a function minimum. In this case the function that we want to minimize is
a parameterized loss function $L(\hat{y}, y; \theta_{model})$ that computes the difference between the model prediction $\hat{y}$ and the expected value $y$, given the current model parameters $\theta_{model} = \{W, b\}$. The gradient descent algorithm can be described as follows:

**Algorithm 1:** Gradient Descent Optimization

\[
\theta_{model} \leftarrow \text{any point in the parameter space}
\]

\[
\text{while } L(\hat{y}, y; \theta_{model}) > \epsilon \text{ do}
\]

\[
\text{for } w_i \in \theta_{model} \text{ do}
\]

\[
w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} L(\hat{y}, y; \theta_{model})
\]

\[
\text{end for}
\]

\[
\text{end while}
\]

The parameter $\alpha$ in Algorithm 1 is usually called the learning rate. $\alpha$ is an hyper-parameter, that is, a parameter whose value is set before the learning process begins, and it defines how much we want to change $\theta_{model}$ per update. Also, in the preceding algorithm, we can compute the loss function with a different number of training examples. Stochastic Gradient Descent (SGD) is a commonly used version of algorithm 1 that uses one sample at a time. One benefit of SGD is that it is computationally faster, but it as the disadvantage of having a noisier gradient. Most of the applications nowadays use mini-batches of size $k \ll n$. The mini-batch version can be efficiently computed in a GPU with parallel computation and creates more stable gradients.

![Figure 2.2: Comparison between a mini-batch gradient descent updates and the SGD updates for finding the minimum of a function. Taken from: https://engmrk.com/mini-batch-gd/](image)

### 2.1.2 Multi-Layer Perceptron

A MLP is a class of feed-forward neural architecture that consists of, at least, three layers: an input layer, an hidden layer and an output layer. Each layer can be seen as a set of several Perceptrons organized together to create arbitrarily complex functions.

In Figure 2.3 we can see a simple graphical representation of such architecture where blue circles represent activation functions and green circles represent input features.
Essentially, the MLP represented in Figure 2.3 receives an input vector with only two feature variables and passes it from layer to layer in order to produce a final output. Between each layer, a set of weights $W^i$ apply a linear transformation to the inputs and a bias term $b^i$ is added. Finally, the transformed input is passed through a set of activation functions to produce the layer output. The computed output is then used as input features for the next layer until a final output is produced.

In the described architecture the model parameters $\theta_{model}$ is defined by the set of weights $W^i$ and $b^i$ used in-between layers, thus, as in the Perceptron algorithm, $\theta_{model}$ can be optimized via gradient descent.

### 2.1.3 Recurrent Neural Network

A conspicuous limitation of the described models is their inability to process sequential data such as sequences of words. Unlike the referred models, RNN architectures have an internal state/memory that makes it possible to process sequential inputs (Figure 2.4).

![Figure 2.3: Graphical representation of an MLP with only one hidden layer from Matt Mazur blog post. Taken from: http://tinyurl.com/j5vkStu](http://tinyurl.com/j5vkStu)

![Figure 2.4: Graphical representation of a RNN and the respective development of the underlying computations. Taken from: http://tinyurl.com/q6dcybc](http://tinyurl.com/q6dcybc)

An RNN receives, at each time step $t$, an input $x_t$ together with a state vector $s_{t-1}$ and computes a hidden state $h_t$. In the simplest RNN architecture (Elman, 1990) the produced $h_t$ is used as state vector $s_t$ for the next input $x_{t+1}$. Formally this simple RNN architecture is defined as:
\[
h_t = g(x_t W^x + s_{t-1} W^s + b)
\]
\[
s_t = h_t
\]

In Equation 2.2, \(W^x\) and \(W^s\) are weight matrices that respectively transform \(x_t\) and \(s_{t-1}\) linearly. Together with a bias term \(b\), these linear transformations are summed and passed through a non-linear function (e.g. ReLU) to produce the next hidden state \(h_t\). This internal computations are illustrated in Figure 2.5.

In the described RNN architecture the model parameters \(\theta_{\text{model}}\) are the matrices \(W^x\) and \(W^s\) together with the bias term \(b\), thus, as in the previous neural networks, we can optimize these parameters via gradient descent.

Theoretically, after training, the described model should be able to represent information of an entire sequence \(x = \{x_1, x_2, ..., x_k\}\) in the last hidden state \(h_k\). In practice, due to vanishing gradients\(^1\), the described model is not able to model long-term dependencies and the final hidden state \(h_t\) will only contain information from the entire input sequence.

To address the vanishing problem, Hochreiter and Schmidhuber (1997) proposed the Long Short-Term Memory Network (LSTM) neural architecture. LSTMs are explicitly designed to deal with long-term dependencies and avoid vanishing gradients. The two core concepts behind an LSTM model are the cell/memory state, and the various gates. The cell state carries relevant information throughout the processing of the sequence receiving only some minor updates. The gates can be seen as individual Perceptron’s with sigmoid activation’s that decide which information should held-out. Formally, a LSTM architecture can be defined as follows:

\(^1\)The vanishing gradient problem occurs when the values of the gradients are too small and the model stops learning or takes to much computations to learn.
\[ f_t = \sigma(W_f[h_{t-1}; x_t] + b_f) \]
\[ i_t = \sigma(W_i[h_{t-1}; x_t] + b_i) \]
\[ \hat{c}_t = \tanh(W_c[h_{t-1}; x_t] + b_c) \]
\[ c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \]
\[ o_t = \sigma(W_o[h_{t-1}; x_t] + b_o) \]
\[ h_t = o_t \odot \tanh(c_t) \]  \hspace{1cm} (2.3)

This architecture, at each time step \( t \), receives a new input \( x_t \), the previous cell state \( c_{t-1} \) and the previous hidden state \( h_{t-1} \). Given \( x_t \) and \( h_{t-1} \) the forget gate \( f_t \) decides what values from the cell state should be erased, and the input gate \( i_t \) decides what values from the input should be kept. Afterward, a vector of new candidate values \( \hat{c}_t \) is created through a \( \tanh \) layer. We then multiply the forget gate \( f_t \) with the previous cell state \( c_{t-1} \) to erase information and we sum the result with the multiplication between the input gate \( i_t \) and the candidate values \( \hat{c}_t \), thereby storing new information. Finally, the output for the time step \( t \) is given by \( o_t = \sigma(W_o[h_{t-1}; x_t] + b_o) \) and the hidden state by \( h_t = o_t \odot \tanh(c_t) \). Figure 2.6 illustrates the described computations.

As we have seen with simple RNN models, LSTM can be optimized via gradient descent, however instead of simply having 2 matrices and a bias term we will have the following set of parameters: \( \theta_{model} = \{W_f, W_i, W_c, W_o, b_f, b_i, b_c, b_o\} \).

---

**Figure 2.6:** Graphical representation of a LSTM cell. Taken from: http://tinyurl.com/yy79zmxo
Also, when modelling context, sometimes is useful to look forward in a sequence and use that information to make more informed decisions. So far, the described RNN architecture consider only past information. To solve this problem, bidirectional RNN architectures were proposed. A bidirectional RNN is composed of two independent RNN models that process the input sequence in opposite directions (forward and backward). This way for each time step \( t \) we can merge the forward and backward outputs in order to combine past and future information.

### 2.1.4 Sequence-to-sequence Models

Neural Sequence-to-Sequence models have been proposed by Sutskever et al. (2014) as a general framework to adapt a RNN to a setting where the output sequence may have a different length from the input sequence. This as been applied to a wide variety of tasks such as machine translation, speech recognition (Prabhavalkar et al., 2017) and dialog systems (Vinyals and Le, 2015).

In most cases, this framework consist of two RNN models, the encoder and the decoder (Figure 2.7). The encoder RNN maps a variable-length source sequence into a fixed-length vector \( v_q \) that encodes relevant information from the source. The decoder RNN is then initialized with \( v_q \) and maps it back into a variable-length target sequence. The two RNNs are jointly trained via gradient descent in order to maximize the conditional probability of the target sequence given the source sequence, which can be formally expressed by minimizing the following objective function:

\[
J_\theta = -\frac{1}{|Y|} \sum_{i=1}^{n} \log(P(y_i|x_1, \ldots, x_m, y_1, \ldots, y_{i-1}))
\]  

(2.4)

Once training is complete, the described model can produce answers by finding the most likely sequence of tokens:

\[
\hat{Y} = \arg\max_Y P(Y|X, \theta)
\]  

(2.5)

Yet, since the answer space is intractable, computing the most likely sequence becomes impractical. For that reason, beam search is usually used. In a beam search decoding, at each step, we keep track of the top \( k \) most likely partial answers until \( k \) candidate answers are fully generated. This approach will lead to a near-optimal solution with a reasonable computation effort. Interestingly, it is commonly observed that, a simple beam of size 1 (Sutskever et al., 2014), also called greedy search decoding, performs well for downstream tasks such as machine translation.
2.1.5 Attention Layer

A potential limitation of the described architecture is that it has to encode all the information of the source sequence in a fixed-length vector. To address this problem Bahdanau et al. (2015) proposed an attention layer that selectively retrieves relevant information from the source sequence during decoding. The attention layer can be seen as a simple feed-forward neural network that learns to score words in the source given the current decoder state. Formally, the attention layer is defined as follows:

\[
\text{score}(h^d_t, h^e_k) = v^T a \tanh(W_a[h^d_t; h^e_k])
\]  

(2.6)

In the previous equation \(h^d_t\) refers to the decoder hidden state of the \(t^{th}\) decoding step and \(h^e_k\) stands for the encoder hidden state of the \(k^{th}\) word in the source. Given that, during decoding, we will pair \(h^d_t\) with all the encoder hidden states and compute a softmax distribution over the source sequence words:

\[
\alpha_t(i) = \frac{\exp(\text{score}(h^d_t, h^e_i))}{\sum_j \exp(\text{score}(h^d_t, h^e_j))}
\]  

(2.7)

With the computed softmax distribution we create a weighted sum of the encoder hidden states in order to retrieve relevant information from the source.

After Bahdanau et al. (2015) various other forms of attention scores have been studied. Among those, the dot product (Luong et al., 2015) is, without a doubt, the most popular one.

Finally, one possible drawback of the softmax distribution computed after the scoring function is that, for long sequences, the computed weights are scattered along the entire source sequence. For that reason Martins and Astudillo (2016) proposed an attention mechanism in which the softmax function is replaced with a sparsemax function. Sparsemax is a softmax transformation that, in contrast to the softmax distribution, may exactly assign zero probability to some of its output variables. Consequently, with a sparsemax we are able to obtain a more selective and compact attention focus.
2.2 Unsupervised Machine Learning

Unsupervised machine learning differs from supervised machine learning in that the categories/labels are not known and the learning consists in using the inputs from \( X \) to find the structure or relationship between different inputs.

In this section we will present two important unsupervised learning algorithms that will be central pieces for this thesis, namely K-Means and GANs. More precisely, we will describe how we can group similar documents with a K-Means algorithm and how we can generate synthetic text with a GAN algorithm.

2.2.1 K-Means

The term “K-means” was first used by Macqueen (1967). In this algorithm, the observations from \( X \) are clustered into \( k \) groups, where \( k \) is provided as an input parameter. The algorithm starts by initializing \( K \) centroids to represent each cluster. After initializing the centroids, the algorithm computes the distance of each observation to each centroid, in order to assign each observation to the closest centroid. The centroids are then recomputed by averaging the observations inside a cluster. The process of assigning clusters to observations and recomputing the centroids is repeated in a loop, until convergence or a max number of iterations is reached. Fig. 2.8 illustrates a final result of the algorithm.

![K-Means diagram](http://arun-aiml.blogspot.com/2017/07/k-means-clustering.html)

**Figure 2.8:** Example of 3 clusters created by the K means algorithm with \( k=3 \). Taken from: [http://arun-aiml.blogspot.com/2017/07/k-means-clustering.html](http://arun-aiml.blogspot.com/2017/07/k-means-clustering.html)
2.2.2 Generative Adversarial Networks

GAN models were proposed by Goodfellow et al. (2014) and allow two models to improve each other in a zero-sum game frame- 
work. In a GAN setup we have a generative model (G), that learns the data distribution in order to produce new samples, and a discriminator model (D), that tries to identify if a training sample comes from the training data or if it was produced by G. By combining the previously described supervised algorithms we can build a GAN system in which we have a classifier working as D and a sequence-to-sequence model working as G. In this scenario, in principle, we end up improving the generative model by encouraging it to generate valid sequences that can fool D, and we improve the discriminator model by providing additional synthetic sequences. However, for discrete outputs like text it is not trivial to apply a GAN, since the output layer is not differentiable. When dealing with discrete outputs techniques such as Reinforcement Learning (RL) must be applied to tune the generator’s parameters.

Yu et al. (2017) proposed a sequence generation framework, called SeqGAN, to train a discrete generator. In their work the generator was trained using the REINFORCE algorithm (Williams, 1992), to overcome the impossibility of differentiating discrete outputs. The RL reward signal comes from the discriminator judged on a complete sequence, and is passed back to the intermediate state-action steps using Monte Carlo search, that is, for each partial sequence, \( n \) possible full sequences are generated by sampling from the softmax output and the reward of each token is estimated by the likelihood that it would produce a sequence that can fool the discriminator model. The seqGAN training procedure consists in:

1. Pre-training the generator model, using Maximum Likelihood Estimation (MLE) on the training set.
2. Pre-train the discriminator via minimizing the cross entropy, with synthetic sequences sampled from our pre-trained generator.
3. Iteratively train the generator via policy gradient and the discriminator via cross entropy.

To test the efficacy of the proposed framework the authors created the oracle dataset. The oracle dataset contains 10,000 sequences of length 20 that were generated by a randomly initialized LSTM, aka, the oracle. The oracle LSTM mimics a real language model by generating sequences token by token \( p(x_t|x_1,...,x_{t-1}) \). In the experiments the authors initialize another LSTM as a generator and a CNN as discriminator. The goal of their experiments is to approximate the language model of the generator LSTM to the language model of the oracle LSTM. The authors show that over 250 training epochs the Negative Log-Likelihood (NLL) loss of the generator is always decreasing, however after 250 epochs it is still over 8 indicating that the generator is still far from the oracle model. The authors

\[^2\] A mathematical representation of a situation with at least two players, in which, each participant gains and losses are balanced by the other’s participants gains or losses.
also experimented with 3 real world datasets; a Chinese poetry corpus\(^3\), Obama political speech\(^4\) and a music corpus\(^5\). In all this corpora the authors showed that the proposed training framework is able to improve a pre-trained LSTM decoder in terms of BLEU.

### 2.3 Text Representation Models

In Section 2.1 and 2.2 we have seen how to develop algorithms that uncover general hypothesis from data that can be used to make future predictions. However, in order to apply the described algorithms to NLP tasks, we need to represent textual information in a vector form. In this section, we explain how to represent text as vectors, starting with sparse vector models and then explaining how to represent the same text into a dense space.

#### 2.3.1 Sparse Models

The simplest approach for representing documents in NLP is by using a bag-of-words. A bag-of-words is a representation of text that describes the occurrence of words within a document. In this type of representation, a document is seen as a vector in which each entry corresponds to a certain word \(w\) belonging to a fixed vocabulary \(\Gamma\) with a finite size \(n\). Formally, a document \(d_i\) belonging to a collection of documents \(D\) is represented by its feature vector \(v_i = \{w_{1,i}, w_{2,i}, ..., w_{n,i}\}\) where \(w_{j,i}\) corresponds to a feature weight indicating the importance of the word \(w_j\) in the document \(d_i\) in \(D\).

One of the simplest ways to compute the weights \(w_{j,i}\) is to consider binary values, where \(w_{j,i}\) is 1 if a certain word is present in \(d_i\) or 0 if it is not. The main disadvantage of using binary values as features is that all words in a document will be given the same weight when in practice some words are more important to the semantic meaning of the document than others. In order to avoid such phenomenon, it is common practice to use Term frequency-inverse document frequency (TF-IDF) features.

TF-IDF is an information retrieval technique that weighs a term’s frequency (TF) and its inverse document frequency (IDF). Each word has its respective TF and IDF score. The product of the TF and IDF scores is called the TF-IDF weight. Intuitively, TF-IDF weights, measure how important a given word \(w\) is to a document \(d \in D\). Equation 2.8 states a way to compute the described features.

\[^3\]http://homepages.inf.ed.ac.uk/mlap/Data/EMNLP14/
\[^4\]https://github.com/samim23/obama-rnn
\[^5\]http://www-labs.iro.umontreal.ca/~lisa/deep/data/
\[ \text{TFIDF}(w, d, D) = \text{TF}(w, d) \times \text{IDF}(w, D) \]  

(2.8)

where:

\[ \text{TF}(w, d) = \frac{f_{w,d}}{\max\{f_{w',d} : w' \in d\}} \]  

(2.9)

\[ \text{IDF}(w, D) = \log \frac{|D|}{|\{d \in D : w \in d\}|} \]  

(2.10)

Regardless of the weighting scheme, with a bag-of-words model we can evaluate the similarity of two documents \( d_i \) and \( d_j \) as a distance between the respective vectors \( v_i \) and \( v_j \). A commonly used distance metric is the cosine similarity, given by Equation 2.11.

\[ \cos(v_i, v_j) = \frac{v_i \cdot v_j}{||v_i|| \cdot ||v_j||} = \frac{\sum_{i=1}^{n} v_{i,j}}{\sqrt{\sum_{i=1}^{n} (v_{i,i})^2} \sqrt{\sum_{i=1}^{n} (v_{j,i})^2}} \]  

(2.11)

The described representation models can also be used with \( n \)-grams sequences instead of single words. An \( n \)-gram is a contiguous sequence of \( n \) words belonging to a document \( d_i \). By considering \( n \)-grams as features we change the scope of the vocabulary and we allow the bag-of-words model to capture some structure from the documents.

The described representations allow us to represent entire documents, but vector space models can also be used to represent words. The simplest way to represent an individual word is by using a one-hot vector. In the one-hot vector representation a given word \( w_i \in \Gamma \) is represented as a vector \( w_i \) with all entries equal to 0 except the position that represents the word \( w_i \) itself (that will have value 1). However, this rather simplistic approach does not capture intrinsic notions of semantic similarity between words.

One way to capture the semantic similarity between words is to create a term-document matrix \( M \) in which each row \( i \) represents a word \( w_i \in \Gamma \), each column \( j \) represents a document \( d_j \in D \) and each entry \( M_{i,j} \) counts how many times \( w_i \) appears in \( d_j \). After creating such matrix we can extract word representations by looking to row vectors. Naturally, since similar words tend to appear in similar contexts, these representations will capture the intrinsic semantics of words.

Finally, rather than creating term-document matrix we can create a term-term matrix. Each entry in such matrix represents how many times two words appear together in a context. The context can be defined as a fixed size sliding window or even an entire document from the training corpus. After building such matrix the vector representing a word \( w_i \in \Gamma \) is given by the row or column \( i \) of the term-term matrix.

In sum, the described models allow us to represent similar documents with closer vectors because similar documents tend to have similar words. This same principle can also be applied to words: similar words have closer vectors because they tend to occur in similar contexts (Jurafsky and Martin, 2000).
2.3.2 Dense Models

In Section 2.3.1, we have seen how to represent words in a sparse vector space. In sparse spaces each dimension corresponds to a word in a vocabulary $\Gamma$, consequently, each vector has the same length as the entire vocabulary $\Gamma$. Additionally, most entries in a document/word vector are zero and even with a term-term matrix drawing semantics between similar words can be challenging.

Fortunately, using dense representations, we are able to represent textual information with much fewer parameters. According to Jurafsky and Martin (2000) words in a dense space, also called embedding space, have two main advantages; 1) They provide a better generalization and 2) They require fewer parameters by machine learning algorithms.

2.3.2.A Single Value Decomposition

Furnas et al. (1988) was the first to introduce a Single Value Decomposition (SVD) method called Latent Semantic Indexing (LSI) for generating embeddings from a term-document matrix.

In linear algebra, the SVD is a factorization of a real or complex matrix. Generally speaking, the axes from a matrix are rotated into a new space, so that the new axes capture the most variance. In the new space, we can represent data by only using a small number of dimensions (the ones that capture more variance).

LSI is an SVD applied to a term-document matrix, thus, a matrix $M_{n \times m}$ where $n$ is the size of the vocabulary $\Gamma$ and $m$ the number of documents, is factorized into a matrix $\hat{M}_{n \times k}$ where $k$ represents the $k$ dimensions with higher variance. Each row $i$ then acts as an embedding with size $k$ for a document $d_i \in \Gamma$. This approach can also be applied to term-term matrices to create word dense representations (Schütze, 1992).

2.3.2.B Neural Word Embeddings

Another approach for representing words into an embedding space is by learning such representation with neural networks. Broadly speaking, these neural networks are trained to minimize the distance between the representation of words that occur together in a context and maximize the distance between words that do not occur in the same context.

Neural word embeddings have been exceptionally successful in many NLP tasks. In fact, in many NLP architectures, traditional representations such as bag-of-words have been replaced by neural word embeddings\(^6\).

\(^6\)http://ruder.io/word-embeddings-1/
In recent years, several different approaches have been proposed for producing these representations (Bengio et al. (2001); Mikolov et al. (2013a); Pennington et al. (2014)). For simplicity, during the rest of this thesis, we will address neural word embeddings as word embeddings.

Among various methods, Word2Vec\(^7\) is arguably one of the most popular ones. In the original paper Mikolov et al. (2013a) proposed 2 different methods: The Continuous Bag-of-Words (CBOW) and the Skip-Gram.

The CBOW model essentially aims to predict a word \(w_t\) in the middle of a context defined as the \(i^{th}\) words preceding \(w_t\) and the \(i^{th}\) words after \(w_t\). The objective function of the CBOW model can be defined as:

\[
J_\theta = \frac{1}{T} \sum_{t=1}^{T} \log p(w_t | w_{t-i}, w_{t-i+1}, \ldots, w_{t+i+1}, w_{t+i+2}, \ldots, w_{t+i})
\] (2.12)

The Skip-gram model is similar to the CBOW model but it turns the objective on its head. Given a word this model learns to predict the surrounding words by maximizing the following objective function:

\[
J_\theta = \frac{1}{T} \sum_{t=1}^{T} \sum_{-i \leq i \leq T, i \neq 0} \log p(w_{t+i} | w_t)
\] (2.13)

The bottleneck of the described models lies on the softmax output (more precisely, the normalization term). The cost of computing the softmax is proportional to the number of words in the alphabet, which is typically on the order of hundreds of thousands or millions. For that reason Mikolov et al. (2013b) proposed the Negative Sampling model. In this model instead of predicting target words given a context, the model will receive pairs of words and predict if they belong to the same context (e.g. considering

\(^7\)https://radimrehurek.com/gensim/models/word2vec.html
a context phrase “I want a glass of orange juice”, the input “orange” and “juice” has an output target 1 while the input “orange” and “king” has an output target 0. The network architecture is similar to what we have seen in Word2Vec models. The main difference is that the output layer computes a sigmoid function instead of a softmax. Using a sigmoid output layer and redefining the problem to be a logistic regression between 2 input words produces a significant speed-up in computations.

Overall, the main insight that we take from the models proposed by Thomas Mikolov (Mikolov et al. (2013a); Mikolov et al. (2013b)) is that what really matters is the distance between two words and the intrinsic statistical properties between them in a given corpus. This intuition is not different from what we have seen when we talked about representing words via term-term matrix, the only difference is that these properties are learned in an embedding space defined in a projection layer of a neural network.

Pennington et al. (2014) took a step back and proposed a method to do explicitly what Word2Vec does implicitly via Gradient Descent, the GloVe embeddings\(^8\). To achieve this, the GloVe algorithm goes through the following steps:

1. Collect word co-occurrences in a form of a term-term matrix \(X\) in which each entry \(X_{ij}\) represents how often the word \(w_i \in \Gamma\) appears in context of the word \(w_j\).

2. Define soft constraints for each word pair according to the formula:

\[ w_i^T w_j + b_i + b_j = \log(X_{ij}) \]

3. Finally, we define a cost function \(J\) that directly aims to minimise the difference between the dot product of two vectors representing two words and the logarithm of their co-occurrences:

\[ J_\theta = \sum_{i=1}^{n} \sum_{j=1}^{n} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2 \]  \(2.14\)

In Equation 2.14 \(f\) is a weighting function which help us to prevent learning only from the most common word pairs.

### 2.3.2.C Contextualized Word embeddings

Word embeddings are dense word representations that can be learned by exploiting vast amounts of textual data. These representations have proved to be exceptionally successful in many NLP tasks capturing the semantic meaning of words and more complex phenomenon such as word arithmetic’s (e.g. the representation of the words king - man + woman \(\approx\) queen). However, these representations are fixed and do not take into account word semantics given the context.

\(^8\)https://nlp.stanford.edu/projects/glove/
Word polysemy is the ability of a word to have different meanings. An example of such word is “orange”. The “orange” word can either refer to fruit or to a colour depending on the context. Consequently, when we represent this word with a fixed representation we are stripping down different meanings which, in the end, can impact our NLP system.

To address this word property a number of approaches have been attempted (Li and Jurafsky (2015); Neelakantan et al. (2014); Pilehvar and Collier (2016)) but among those approaches, contextual word representations derived from pre-trained bidirectional language models, have been showing significant improvements to the NLP state-of-the-art (Peters et al. (2018); Devlin et al. (2018)).

Peters et al. (2018) introduced a new type of deeply contextualized word embeddings derived from a bidirectional language model, the so-called Embeddings from Language Models (ELMo). In contrast to widely used word embeddings (e.g. GloVe Pennington et al. (2014) or Word2Vec Mikolov et al. (2013c)), ELMo embeddings are function of the entire input sequence. The proposed word representations are computed from a model consisting of a two-layer LSTM with word-level embeddings computed through a CNN over the characters. The internal representations of the described model are then optimized with a bidirectional language model.

Language modeling is the task of assigning a probability to sentences in a language. Formally a forward language model can be defined as:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1}) \quad (2.15)$$

Given the forward language model, we can think of a backward language model as a forward language model with the input sequence reversed. With that said, the neural architecture proposed by Peters et al. (2018) learns to combine a forward and a backward language model by jointly maximizing the log likelihood in both directions:

$$\sum_{k=1}^{N} [\log p(t_k | t_1, ..., t_{k-1}) + \log p(t_k | t_{k+1}, ..., t_N)] \quad (2.16)$$

The described setup (model + bidirectional language model task) is then trained at a large scale and fine-tuned after in downstream tasks in a semi-supervised manner. For inclusion in downstream tasks, the authors frozen the model parameters and fine-tuned 4 task-specific weights that learn to combine different model layers:

$$E L M o_k^{task} = \gamma^{task} \sum_{j=0}^{3} s_j^{task} h_{k,j}^{LM} \quad (2.17)$$
With the described approach, Peters et al. (2018) were able to achieve impressive results in a broad range of NLP tasks showing that deeply contextualized word embeddings effectively encode semantic and syntactic information that fixed embeddings did not.

It is worthwhile to mention that, recently, Devlin et al. (2018) proposed the BERT model. This model will not be addressed in this thesis but, in short, BERT is designed to learn a bidirectional language model by jointly conditioning on both left and right context in all layers with a transformer model. The paper results show that this pre-trained model can be fine-tuned to downstream tasks and achieve state-of-the-art results.
3

Related Work

Contents

3.1 Available Email/Customer Support Corpus ........................................ 29
3.2 Automatic Email Answering ................................................................. 30
3.3 Dialog Systems .................................................................................. 33
In this Chapter, we start by presenting available corpora related to our problem setting. Then we present works that are directly related to Automatic Email Answering. Finally, due to the fact that Automatic Email Answering is similar to a conversational system, we present relevant work from the dialog systems research area.

### 3.1 Available Email/Customer Support Corpus

As we said before, since there is no publicly available customer support email data, we will make use of an internal email corpus provided by Unbabel. Nonetheless, in order to benchmark our result we decided to investigate publicly available data that could be similar to our scenario, this is, data from an email domain or customer support data from another text platform.

To the best of our knowledge, the following corpus are the only available email corpus:

- **The OMQ corpus** (Eichler et al., 2014) consists of a set of emails and online requests received by the OMQ company\(^1\). Each email was manually assigned to one or more categories representing several issues reported by the clients. The categories are not exclusive, hence they are organized in groups according to the issue similarity. The corpus contains a total of 627 emails and 41 categories arranged in 20 groups. The major problem about this corpus is the absence of the replies.

- **Enron email raw corpus**\(^2\) has 619,446 messages from 158 employees and it was made public during an investigation to the Enron Corporation. Many versions of the original corpus are available. In Klimt and Yang (2004) version there are only 200,399 messages from the 158 original users. Each message belongs to one or several folders and each folder to a specific user. Since the corpus is organized by user-defined folders (e.g: personal, work, events, etc.) this corpus has been widely used to study automatic classification of emails into user-defined folders. The disadvantage of this corpus is that the messages are mostly from senior management employees and not customer support related.

As said before, an alternative is using a customer support related corpus from other text platforms. The following corpus are examples of such:

- **The Ubuntu\(^3\) Dialog Corpus** (Lowe et al., 2015) consists of approximately one million conversation threads about technical support for Ubuntu-related problems. The corpus was collected from Ubuntu-related chat rooms where each chat room has a particular topic. Users typically post a

---

\(^{1}\)https://www.omq.de/en/
\(^{2}\)https://www.cs.cmu.edu/~enron/
\(^{3}\)https://www.ubuntu.com/
question in these rooms and after a while, a more experienced user replies with a possible solution. Although the context is similar to customer support, the main disadvantage of this corpus is the lack of common structure and standardization in the responses, due to the fact that, instead of having customer support agents answering, we have a community formed by other Ubuntu users, making it difficult to extract templates from such responses.

- The Twitter Customer Support Corpus from Kaggle\(^4\) is a large corpus of tweets and replies that intend to encourage research in natural language understanding and conversational models applied to the customer support scenario. With almost 3M tweets from 20 major brands such as Apple, Amazon, Xbox, Playstation, Spotify, and so forth, this is the largest publicly available real customer support corpus and a great fit to our study. One disadvantage is that the great majority of replies redirect the questions to direct messages (e.g: AppleSupport: Thanks for reaching out to us. We are always happy to help. Send us a DM so we can look into this together.).

Although the presented corpora are not exactly what we are looking for, we can use them to train and benchmark our algorithms. As we will see later, the same models developed for dealing with emails can also be applied to long conversations with several messages exchanged between customer and support agent (e.g: the Ubuntu Dialog corpus or the Twitter Customer Support corpus) and the features learned by these models in such domains can be beneficial to the email domain via transfer-learning.

### 3.2 Automatic Email Answering

An Automatic Email Answering system tries to deflect customers with template answers. These answers can either solve problems directly or provide pointers to support materials.

We can divide Automatic Email Answering systems into two-classes: Rule-based and Corpus-based. In a rule-based approach, the developer, tries to use domain expertise to create well defined rules to map a question into a predefined answer. In contrast, corpus-based approaches mine previously seen data in order to find the most appropriate answer.

#### 3.2.1 Rule-Based Automatic Email Answering

Sneiders (2010) explored the use of manually crafted text patterns (similar with regular expressions) in order to match client emails with FAQs\(^5\). Experiments with 2 different languages (Swedish and Latvian) and in two different domains (insurance and telecom) showed that 90% of the times the correct answer was retrieved.

\(^4\)https://www.kaggle.com/thoughtvector/customer-support-on-twitter/
\(^5\)online documents that poses a series of common questions and answers on specific topics
In their most recent work, Sneiders et al. (2016) used the same approach in order to assign a template to new emails messages. In this work, the authors, observed that the initial inquiries are normally divided into two parts: i) a description of the problem context; ii) a response trigger. The description can be seen as a set of keywords from a specific context. Response triggers can be a call for information or a call for action and is important for the system to distinguish them. Also, response triggers can be implicit or explicit and it is important for the system to capture them. The referred author believes that until here, automated answering systems were only trying to capture the context, and, according to that, replying with a standard response. Nevertheless, in order to produce high-quality responses, the propose of an email must be resolved. In their experiments, a Support Vector Machines (SVM) was used as a baseline and the manually crafted text patterns slightly outperformed SVM for a dataset of 9663 email messages to the Swedish Social Insurance. It is important to note that this approach is highly domain dependent and does not handle well new languages since the patterns are designed for a specific language.

3.2.2 Corpus-based Automatic Email Answering

To the best of our knowledge, all corpus-based Automatic Email Answering systems in the literature are retrieval-based. For that reason, in this section we will address the corpus-based approaches as retrieval-based, although as we will see in the next section generative approaches are also corpus-based.

One way to reply to new messages is to use supervised learning algorithms to classify those into a set of predefined intent categories with predefined template answers. In that sense, finding automated answers can be seen as a multi-label document classification task, where the system receives a question $q$, classifies it into a category $c$ and answers with the template answer $a^c$ (Figure 3.1).

![Figure 3.1: Example of an email labelled with the corresponding intent and the triggered template answer.](#)

In (Bonatti et al., 2016), the authors used a private corpus from Fundação Estudar with 11,410 email/answer pairs in which the answers were based on 120 different templates. In order to label answers with the correct templates, the referred authors simply compared each answer with the possible templates and assigned the template that better matched the answer. After labelling all the email/answer pairs the authors tested different classifiers and features. Among the several experiments with lemma-
tization and Part-of-speech (POS) filtering the authors concluded that lemmatization did not help, but a POS filter leaving only verbs, nouns, adjectives and adverbs improved the results. In terms of classification algorithms the authors concluded that SVMs and Naive Bayes (NB) were the most robust ones, with SVMs achieving a 87% precision, recall and F1, slightly outperforming NB.

Yang and Kwok (2012) created the Automatic Email Responding System (AERS), a system that classifies new emails into several intents with templates that include web page links to support manuals. It is important to notice that this type of responses does not solve the problem directly, as it forces the user to make further research. Nonetheless, a total of 3015 emails were manually labeled with 200 possible classes and each email was represented in a bag-of-words space that did not include stop-words\(^8\) or numbers. Several classification algorithms (K-means++, NB and KNN) were tested in a test set consisting of 1611 emails. AERS correctly classified 96.2% of them using K-means++ algorithm.

In Mota et al. (2013), the goal was to retrieve a reply from a corpus of email/reply pairs, so that with a maximum of 10% edit would be used by the human agent to answer a new email. In order to do such, the authors found a set of possible intents/templates, since they only had email/reply pairs without any intent labels or templates associated. To do such, a cluster-based approach was proposed consisting in using a modified Jaccard distance\(^9\) to compute the edit distance between replies. The algorithm idea is to loop over all email/reply pairs and create clusters containing pairs that have a maximum 10% edit distance between them. In this approach the authors considered that if a reply \(R_i\) has less than 10% edit distance to a reply \(R_j\) and if \(R_j\) has less than 10% edit distance to a reply \(R_k\) them even if \(R_i\) and \(R_k\) had more than 10% edit distance they belong to the same cluster. After finding the clusters, the authors removed the clusters with size 1 and tested with a 5-fold cross validation several classification algorithms (SVM, KNN, Logistic Regression, among other linear classifiers) for the templates retrieval task. It was considered that any answers inside a cluster could be used as a reply if the new email was classified as belonging to that cluster. The SVM and KNN achieved the best performances with a 44% and 38% precision respectively. Since the KNN algorithm was much faster than the SVM algorithm (\(\approx 30\) minutes) and the precision difference was not significant (0.06%) the authors decided to do further tests with the KNN algorithm, namely testing different values of \(k\) and using different evaluation metrics such as the precision@10 and Mean Reciprocal Rank (MRR)@10. The higher precision@10 value and MRR@10 achieved was 62% and 45% respectively with a \(k\) value of 1. Increasing the value of \(k\) did not help improve the results.

In all the previously described works the Automatic Email Answering system was modelled as a multi-label classifier under a supervised learning paradigm. However, a retrieval-based system can simply search for answer by transforming a question into a vector space model and looking for the most similar answer in that same space. In the work presented by Alfalahi et al. (2015), the authors,
translated new client emails into a shadow answer, i.e., a bag-of-words containing the expected terms in the answer. As an example, if the input email contains the word “password” then it is expected that the answer contains terms such as “account”, “request”, “new”, etc. thus the shadow answer will be a bag-of-words containing those terms. In order to know which are the expected words in an answer to an email, a co-occurrence matrix is used. The co-occurrence matrix was created beforehand as a preparation phase, measuring the strength of words present in an email and words present in an answer. In the answer retrieval phase, the shadow answer becomes a query to Lucy\textsuperscript{10}, an open source information retrieval system with a TF-IDF-based ranking, where a ranked list is returned. The performance measure used in the experiments was the rank of the original response (e.g: In a perfect scenario the original response to a test email should appear in rank 1). Using 1431 email/answer pairs from Swedish Pension Authority\textsuperscript{11}, the system showed that the best average rank of the original question was 66, which the authors considered lower than expected.

### 3.3 Dialog Systems

Closely related to Automatic Email Answering systems are dialog systems. A dialog system is a computer system intended to converse with a human in natural language. The main difference between an Automatic Email Answering system and a dialog system is that, the last, is designed to handle extended conversations instead of single messages.

There are two main criteria which are used to describe dialog systems: task-oriented and general purpose. In a task-oriented dialog system the goal is to extract information in order to perform a specific task. This includes for example conversational agents that converse with customers in websites to answer questions or address problems. On the other hand, general purpose conversational agents are designed for having unstructured conversations like humans do. This type of systems are often deployed for entertainment reasons, but can also be used for practical purposes such as therapeutic reasons (Weizenbaum, 1983). During this thesis, since we are not going to cover general-purpose dialog systems, we will refer to “task-oriented dialog systems” as “dialog systems”.

In the next Sections we will present relevant works that studied dialog systems in support scenarios. Since most of these systems stem from retrieval-based or generative-based approaches we will present them according to that division (Section 3.3.1 and Section 3.3.2). Finally, in Section 3.3.3 we will present a relevant work that tries to combine both approaches under an hybrid model.

\textsuperscript{10}https://lucy.apache.org/
\textsuperscript{11}https://tinyurl.com/y9cho6jc
3.3.1 Retrieval-based Systems

Following the recent trends in deep learning, Lowe et al. (2015) proposed the use of a Siamese Network, also known as a Dual Encoder, to learn to score context/answer pairs. More formally the input to this model is a context $C$ (a conversation snippet) and a potential answer $a$. The model outputs a score for the answer $a$ given the context $C$. In order to find a good answer we need to calculate the score for multiple answers and choose the one with the highest score. In that paper the authors also present a new corpus of technical support extracted from chat logs from the Ubuntu channels on a public IRC network, the previously mentioned Ubuntu Dialog Corpus V1.0. From the presented corpus negative samples are created by taking a context and randomly assigning an answer. The model is then trained to distinguish between correct and incorrect pairs in a training set with a ratio of 1:1 between positive and negative pairs. In their experiments, the authors compare this model with a TF-IDF baseline in a ranking task in which the model is asked to rank 10 candidate answers given a context. From these candidates only one is the original answer, and the goal is to see how high the model ranks the original answer. The evaluation metric used was the $\text{recall}_{10}@k$ (Section 5.1). From their results we can observe that the proposed model outperforms the TF-IDF baseline with a $\text{recall}_{10}@1$ of 0.604, a $\text{recall}_{10}@2$ of 0.745 and a $\text{recall}_{10}@5$ of 0.926 against 0.410 $\text{recall}_{10}@1$, 0.545 $\text{recall}_{10}@3$ and 0.708 $\text{recall}_{10}@5$, respectively.

Right after Lowe et al. (2015) presented the first version of the Ubuntu Dialog Corpus, Kadlec et al. (2015) introduced a preliminary research and experiments with that corpus and reported new state-of-the-art results for it. In their experiments the authors implemented three different Dual Encoder models: a CNN model, a bidirectional LSTM model, and an LSTM model with the same architecture as the one presented by Lowe et al. (2015). The CNN model had 400 filters of length 1, 100 filters of length 2 and 100 filters of length 3 followed by a max-pooling layer to get a fixed-length sentence embedding. The bidirectional LSTM model concatenated the last hidden state from the forward pass with the last hidden state from the backward to create a bidirectional sentence embedding. In all models, the authors shared the parameters between the answer and the context encoders, and both sentence embeddings are combined to produce a final score in the same way as in (Lowe et al., 2015). In their experiments the CNN model achieved 0.549 $\text{recall}_{10}@1$, the LSTM model achieved 0.638 $\text{recall}_{10}@1$ and the bidirectional LSTM model achieved 0.630 $\text{recall}_{10}@1$. The best results (0.683 $\text{recall}_{10}@1$) was achieved with an ensemble model that averaged the predictions of multiple models.

Similar to Lowe et al. (2015), Lu et al. (2017) used one year of Amazon customer service chat transcripts to train a siamese-like Dual Encoder. The documents were tokenized, anonymized, and converted to lowercase. From that corpus, positive pairs were extracted by considering customer messages ending with question marks and the agent answer that followed. Negative samples were created by

\footnote{In a siamese Dual Encoder the parameters from the RNN that encodes the context and the parameter from the RNN that encodes the answer are shared, while in a siamese-like Dual Encoder the parameter are not shared.}
taking a question and randomly assigning an answer from the corpus. The final corpus consisted of 3.3 million training samples with a positive to negative ratio of 1:2. After this, a Dual Encoder was trained to learn low dimensional embeddings for the questions and the answers. The embeddings of the answer were then used to cluster all the answers with a K-Means++, and, for each cluster, a template was created by an human agent. During inference only those templates define our candidates pool and the top-k templates were presented to agents. For automatic evaluation the proposed method was tested on an answer ranking task with 10k randomly sampled customer message/answer pairs extracted from the original corpus vs a TF-IDF ranking baseline. The metrics used were MRR and \( \text{recall}_{10} \) and the proposed method achieved 0.685 and 0.80, respectively, while the TF-IDF ranking only achieved 0.562 and 0.652 respectively. An end-to-end human evaluation was also performed in which a recruited pool of agents had to give a score between 1 and 3 to the top-3 answers, retrieved by each method. From 100 randomly selected questions, the average relevance score for the TF-IDF model was 1.66 while the average relevance score for the proposed model was 2.08.

To improve upon previous neural retrieval-based models for multi-turn dialogs, such as the Ubuntu Dialog Corpus, Wu et al. (2017) proposed the Sequential Matching Network. In this model the context is divided into a sequence of sentences \( S = \{s_0, s_1, ..., s_m\} \), where \( m \) denotes the number of sentences in the context. Then, using a look up table, each sentence \( s_t \) was transformed into a set of word embeddings \( s_t = \{e_1, e_2, ..., e_n\} \), where \( n \) denotes the length of the \( s_t \) sequence. The same process was applied to a candidate answer \( \hat{a} \) and a set of term-term matrices \( C = \{c_1, c_2, ..., c_m\} \) was computed by applying a dot product between word embeddings in \( s_t \) and word embeddings in \( \hat{a} \). Also, \( \hat{a} \) is passed through an RNN to produce a set of hidden states \( H^{\hat{a}} = \{h_1, h_2, ..., h_k\} \), where \( k \) denotes the size of the answer. Then, each sequence \( s_t \) in the context was passed through the same RNN to produce the hidden states \( H^{s_t} \) and a sequence-answer similarity matrix \( A \) was computed with the dot product between the hidden states in \( H^{\hat{a}} \) and in \( H^{s_t} \) (\( A_{i,j} = h_i^{s_t} h_j^{\hat{a}} \)). Finally, each \( c_i \in C \) was paired with \( A \), and a vector \( v_i \) was computed by applying a 2D convolution operation followed by a max pooling. The computed \( v_i \) vectors were then concatenated and transformed into a final score with a linear transformation. Intuitively, this architecture matches important information in the context-answer pair at a word level and segment level, and, after that, this information is refined with a CNN that produces a pair-score.

In their experiments, with the Ubuntu Dialog Corpus, the referred authors were able to achieve 0.726, 0.847 and 0.961 as \( \text{recall}_{10} \), \( \text{recall}_{20} \) and \( \text{recall}_{50} \), respectively, significantly outperforming the previous state-of-the-art.
Table 3.1: State-of-the-art results for the Ubuntu dialog corpus (single model).

<table>
<thead>
<tr>
<th>Ubuntu Dialog Corpus V2.0</th>
<th>$R_{10}@1$</th>
<th>$R_{10}@2$</th>
<th>$R_{10}@5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kadlec et al. (2015)†</td>
<td>0.638</td>
<td>0.784</td>
<td>0.949</td>
</tr>
<tr>
<td>Lowe et al. (2015)</td>
<td>0.552</td>
<td>0.721</td>
<td>0.924</td>
</tr>
<tr>
<td>Wu et al. (2017)</td>
<td>0.726</td>
<td>0.847</td>
<td>0.961</td>
</tr>
</tbody>
</table>

†results available only for the first version of the corpus.

3.3.2 Generative-based Systems

As it has been pointed out, Seq2Seq models have been proposed as a general framework to adapt a RNN to a setting where the output sequence may have a different length from the input sequence.

One of the first examples of a Seq2Seq model applied to a support scenario was the work of Vinyals and Le (2015). In this work the authors compare a vanilla Seq2Seq model trained with open subtitles\(^{12}\) data with a popular rule-based bot (CleverBot\(^{13}\)) using human evaluation on a set of 200 open-domain questions. During human evaluation if three out of four human judges agreed, the score was recorded; otherwise discarded. The Seq2Seq model was preferred in 97 out of 200 questions whereas the CleverBot was only preferred in 60 out of 200 questions. In the remaining questions 20 were tied and 23 were discarded. The authors also tested this model with a domain-specific IT helpdesk dataset. The vanilla Seq2Seq model was able to achieve a perplexity of 8, whereas an n-gram retrieval model achieved 18. The authors also provide several conversation examples for this dataset in order to support the conclusion that the model can provide useful answers to the user.

Lowe et al. (2017), the same authors that introduced the Ubuntu Dialog Corpus, in a more recent study, have experimented with answer generation rather than answer retrieval for the previously proposed corpus. In the referred work, the authors started by presenting a Seq2Seq model in which, the encoder, is used to encode all the context messages exchanged between users until a certain point in the conversation. Then, a decoder, sharing all the parameters with the encoder, is used to decode an answer. As stated by the authors, for long sequences with several messages exchanged between users, it becomes difficult to model long-term dependencies, even with LSTMs. For mitigating that problem, the authors also experimented a Hierarchical Encoder-Decoder architecture (Sordoni et al., 2015). In such architecture, two different encoders are used: one that works at the sentence level and another that works at the document level. The first encoder, receives a sentence $s_i$ representing a user message and encodes it into a fixed size vector $h^{s_i}$. This vector, along with all the other sentence embeddings $h^{s_1}, h^{s_2}, ..., h^{s_m}$, where $m$ stands for the number of sentences in the context, are passed as input to the second encoder to produce a document level embedding vector $h^d$. Intuitively, this document level

---

\(^{12}\)http://opus.nlpl.eu/OpenSubtitles.php  
\(^{13}\)https://www.cleverbot.com/
embedding provides contextualized information about the complete context, thus, it is then passed to
the decoder RNN to guide the next answer generation. In order to compare the generated answers with
the ground-truth, word embedding distances were used, namely, embedding average, greedy matching
and vector extrema (Section 5.1). The Hierarchical Encoder-Decoder model outperformed the simpler
Seq2Seq model with shared parameters in all metrics achieving a 0.617, 0.452 and 0.408 embedding
average, greedy matching and vector extrema distance to the ground-truth, respectively.

Recently Hardalov et al. (2018) explored the ability of generative models in the customer support
domain. In their work the authors used the Twitter Customer Support corpus from kaggle. This corpus
contains 3 million tweets from 20 big companies such as Amazon, Apple, Uber, etc. Since there is
a big variability in the answers given by different companies the authors decided to use only Apple
tweets. All answer that redirected the customer to other support channels were ignored leaving only
49626 context/answer pairs that the authors split into train and testing leaving 45582 context/answer
pairs for training and 4044 context/answer pairs for testing. After a preprocessing phase, Hardalov et al.
(2018) experimented with 2 generative models; a Seq2Seq model with attention and a transformer model
(Vaswani et al., 2017). This two models where also compared with a TF-IDF retrieval-based baseline.
For evaluation the authors used word-overlapping metrics and embedding-based metrics to capture
not only the ability of the models to provide answers word-wise similar to the ground-truth, but also
semantically close. In their experiments the Seq2Seq model is able to outperform the other 2 systems in
all the metrics used. The transformer model is ranked second by three of the evaluation metrics, namely
Greedy matching, Vector Extrema and ROUGE-L, but for the BLEU-2 and for the embedding average
the retrieval systems performed slightly better than the transformer model. The authors considered
the results unexpected since the same exact model achieved the state-of-the-art for neural machine
translation.

3.3.3 Hybrid System

A commonly reported problem of Seq2Seq models is that they tend to produce generic or ungrammatical
answers (Wu et al., 2017). Nevertheless, Lowe et al. (2017) compare the performance of a retrieval-
based approach with generative models and by looking into word embedding distances the authors
concluded that, generative models are better at staying at the topic. In an attempt to unify this two
approaches Wu et al. (2018) proposed an hybrid model that, first retrieves a prototype answer from
previously seen conversations, and then edits the prototype answer according to the differences between
the current context and the prototype context.

In practise, the proposed model is a Seq2Seq model augmented with an edit vector, that encodes
the differences between the prototype context and the current context. In such model, when a new

14https://www.kaggle.com/thoughtvector/customer-support-on-twitter
context $C$ is received, a retrieval system is used to fetch the closest context/answer pair $\{C', A'\}$ from a set of previously seen conversations. Then a bidirectional encoder RNN is used to encode $A'$, and the differences between $C'$ and $C$ are computed by an attention mechanism that creates a weighted average between insertions and deletions to form an edit vector storing relevant differences between contexts. Figure 3.2 illustrates the described model.

In their experiments Wu et al. (2018) only considered single turn answers. The described authors collected over 20 million human-human context-answer pairs (with contexts containing only 1 turn) from Douban Group\textsuperscript{15}, a forum-like social media chinese platform. Among several baselines the authors used the Seq2Seq with attention model from OpenNMT (Klein et al., 2017) and a retrieval-based model that used Lucene\textsuperscript{16} to retrieve 20 candidate answers and the Dual Encoder model from Lowe et al. (2015) to re-ranked the 20 candidates.

![Figure 3.2: Prototype-and-edit model architecture proposed by Wu et al. (2018).](image)

For evaluation, the authors used 1000 contexts from the test set and measured the answers produced by the different models in terms of relevance, diversity, originality and fluency. For measuring relevance the authors used word embedding distance metrics such as embedding average, greedy matching and vector extrema. For measuring diversity the authors measured the ratio of distinct unigrams and bigrams in the generated answers. For originality the authors calculated the percentage of times that the generated answer could be found in the training set (exact matches only). Finally, for fluency three native speakers were used to give scores with the following criteria: +2 if the answer is fluent and grammatically correct, +1 if there are a few grammatical errors in the answer but readers could understand it, 0 if the answer is totally grammatically broken, making it difficult to understand. In terms of relevance the proposed model scored higher than the others across all metrics, in terms of diversity the Lucene + Dual Encoder retrieval baseline scored higher, in terms of originality the proposed model outperformed all others, and finally, in terms of fluency, the Seq2Seq model with attention scored higher.

\textsuperscript{15}https://www.douban.com/group

\textsuperscript{16}http://lucene.apache.org/
4

Methods and Techniques

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Corpora</td>
<td>41</td>
</tr>
<tr>
<td>4.2 Retrieval-based Models</td>
<td>44</td>
</tr>
<tr>
<td>4.3 Generative-based Model</td>
<td>48</td>
</tr>
<tr>
<td>4.4 Hybrid Model</td>
<td>49</td>
</tr>
<tr>
<td>4.5 Generative Adversarial Networks</td>
<td>49</td>
</tr>
</tbody>
</table>
Following recent trends presented in Chapter 3, we explored corpus-based, retrieval- and generative-based approaches that were previously applied to dialog systems in community support domains. Namely, as retrieval-based models we developed several different Dual Encoders and, as generative-based, we developed a Seq2Seq model with attention. Lastly, we combined both approaches in two ways; with an hybrid model that uses the generated answer to retrieve a template answer, and with a GAN that jointly trains a Dual Encoder and a Seq2Seq models. As corpora, we will make use of the publicly available Ubuntu Dialog Corpus, the Twitter Customer Support corpus and a private email corpus provided by Unbabel.

In this Chapter, we will start by presenting the corpora and all the preprocessing steps applied to it. Section 4.2 explains the developed retrieval methods. Section 4.3 presents the developed generative model. Section 4.4 presents our hybrid model and finally, Section 4.5 explains our GAN model.

4.1 Corpora

Our main goal is to study the feasibility of deep learning models applied to customer support email data. As previously said, we will use a private customer support email corpus provided by Unbabel, but, in order to benchmarks our results with publicly available data we will also use the Twitter Customer Support Corpus from Kaggle and the well-studied Community Support Ubuntu Dialog Corpus v2.0.

4.1.1 Private Email Corpus

This corpus was collected during 2016 and contains 42k email/answer pairs exchanged between customer and support agent. For each email the following preprocessings were applied: lower-casing, names removed, links replaced with a special URL token, Unicode characters converted to UTF-8, text split into tokens (using NLTK\textsuperscript{1} tokenizer). In addition, all pairs containing at least one sequence with more than 300 tokens were removed from the corpus. After preprocessing the email pairs we ended up with 39244 email/answer pairs that we split into train/validation/test according to the following ratios 90%, 5% and 5% respectively, leaving 35319 pairs for training, 1963 for validation and 1962 for testing. Some statistics are presented in Table 4.1.

<table>
<thead>
<tr>
<th></th>
<th>Average Question Length</th>
<th>Average Answer Length</th>
<th>Nº of pairs</th>
<th>Vocabulary Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50.41</td>
<td>83.73</td>
<td>39 244</td>
<td>30 267</td>
</tr>
</tbody>
</table>

Table 4.1: Email Corpus Statistics.

Similar to what we have seen in Section 3.3.1, with the future intent of experimenting Dual Encoders,

\textsuperscript{1}https://www.nltk.org/
all the extracted pairs were labeled as positive samples. Negative samples were created by pairing each customer-email with a randomly selected agent-answer that had less than 0.7 cosine similarity from the original answer in a TF-IDF feature space. The reason we selected only answers with less than 0.7 cosine similarity is because, in our domain, we have many similar answers and by blindly creating negative pairs we will end up with valid pairs with negative labels. For the validation and test sets we created, then, for each email, a set of 10 possible answers, including the ground truth. This is later used to test the models in a ranking task, as in (Lowe et al., 2015).

Finally, with all the unique answers from our training set, we created 1000 clusters in a TF-IDF feature space using a K-Means++ algorithm. Then, for each cluster, we selected the document closer to the centroid and created a list of 1000 possible template answers to be used later by retrieval-based models. We use a value of K equal to 1000 because, in this way, we guarantee a good answer coverage, and, at the same time, retrieval-based models are able to compute in a few seconds what is the best candidate answer, to a given question.

4.1.2 Twitter Customer Support Corpus

As we previously said, the Twitter Customer Support Corpus from Kaggle is a large, corpus of tweets and replies that intend to encourage research in natural language understanding and conversational models applied to the customer support scenario and, thus, a great fit to our study. With that said, all the preprocessing steps that will be described next were inspired by the work done by Hardalov et al. (2018).

Firstly, we selected all the tweets from the Apple support team. From those messages we excluded support answers that redirected customers to other support channels, as long as, disclaimers saying that Apple only offers support in English. After this data selection step we ended up with 49k Apple support answers. In order to build context/answer pairs we searched in the original corpus for the previous tweets that originated those answers. The context was defined by concatenating all the previous tweets, in reversed order, until a maximum of 150 tokens. For each document the following preprocessings were applied: lower-casing, text split into tokens (using NLTK Twitter tokenizer), ids anonymized, and links replaced by the URL token. After these preprocessings we ended up with 49007 pairs that we split into train/validation/test by using all the pairs in which the answer was given in the last 5 days of the corpus for validation and test, and the remaining data for training. This lead to a corpus with 45844 pairs for training, 1581 for validation and another 1581 for testing. Table 4.2 presents some corpus statistics.

As done with the previous corpus, extracted pairs were labelled as positive, negative samples were built, and a ranking set was created from the validation and test sets. Lastly, an answer pool of 1000 candidate was created using a K-Means++ algorithm.

---

2https://www.kaggle.com/thoughtvector/customer-support-on-twitter/
4.1.3 Ubuntu Dialog Corpus V2.0

The data from the Ubuntu Dialog Corpus (Lowe et al., 2015) comes fully tokenized, stemmed and lemmatized and entities like names, locations, organizations, URLs, and system paths were replaced with special tokens, thus no pre-processing was needed. Also, the training set provided is composed of triplets in the form (context, candidate answer, label), such that:

- The context is defined by all previous utterances in the conversation up to the point in which the next answer is to be found.
- A candidate answer will follow the given context.
- The label denotes whether that candidate answer is a true answer to the context (label = 1), or a randomly drawn answer from elsewhere in the corpus (label = 0).

In addition to the training set, the authors provide a validation set and a test set, but these sets instead of having triplets as explained for the training set, are composed by series containing a context and 10 possible answers including the ground truth. During their experiments they evaluate their models in the ability of ranking those 10 candidate answers.

In our work, in order to be consistent between different corpora, we split the training set into 3 parts (99%/0.5%/0.5%) to create our own training/validation/test sets. With these sets instead of evaluating the ranking ability of our model between epochs we looked into its accuracy. In Table 4.3 we present some corpus statistics.

<table>
<thead>
<tr>
<th>Average Context Length</th>
<th>Average Answer Length</th>
<th>N° of pairs</th>
<th>Vocabulary Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>86.34</td>
<td>17.25</td>
<td>1 000 000</td>
<td>91 623</td>
</tr>
</tbody>
</table>

Table 4.3: Ubuntu Dialog Corpus Statistics.
In Table 5.7 we display a set of examples that show the differences between the 3 described corpus. Note that, for the Ubuntu Dialog corpus, the authors only made available a fully pre-processed version. Also, for the Private Email corpus, we could not show a more representative sample because for longer messages it becomes easy to identify the company behind the data.

### 4.2 Retrieval-based Models

In a typical Retrieval-based formulation the system receives a query $q$ (e.g. a conversation up to a certain point in which, an answer is to be expected or a client email message) and $n$ candidate answers $\{a_1, a_2, ..., a_n\}$ from which an answer $a_k$, $1 \leq k \leq n$ is chosen. If the selected answer $a_k$ is inside the ground-truth set, question $q$ is considered to be answered correctly, otherwise incorrectly. From this formulation we can view such a system as a scoring function $s$ between $q$ and $a_k$. 

4.2.1 TF-IDF baseline

A commonly used retrieval-based baseline (Lowe et al., 2015; Hardalov et al., 2018) is to model the scoring function $s$ as a cosine similarity between the query $q$ and a candidate answer $a_k$ in a TF-IDF feature space extracted from the training data. The motivation here is that answers tend to share words with the respective query for which they were created, thus, among several candidate answers, the best candidate $a_k$ will be the one with a similar vectorial representation.

We have implemented this baseline in Python 3.6.6\(^3\) with the Sklearn 0.19.2 toolkit\(^4\). To compute the TF-IDF features we have considered both the answer and the queries. Also, in order to reduce the feature space we have excluded all the words in the scikit learn stop-words list.

4.2.2 Siamese-Like Dual Encoder

As previously said, Lu et al. (2017) proposed a simple Dual Encoder model to distinguish between positive and negative pairs and, later, to rank candidate answers. The proposed model consists in a LSTM that encodes the query and another LSTM that encodes the answer. Both vectors are then concatenated into a single vector and passed to a MLP with a single hidden layer with 3 hidden units. The output layer consists in a softmax function that will determine if the question matches the answer or not.

Our second retrieval-based baseline will consist in a similar model, but after some preliminary experiments we decided to use a bidirectional LSTM instead of a unidirectional, and an MLP hidden layer of size 300 instead of 3. We also replaced the softmax output with a sigmoid function for simplicity. In Figure 4.1 we illustrate our model.

This model was initialized with pre-trained GloVe\(^5\) word embeddings, which were frozen during the first training epochs and fine-tuned during the last epochs. The model was trained to minimize the binary cross-entropy loss function using Adam optimizer (Kingma and Ba, 2014).

All the code was developed using Python 3.6.6 with the help of the PyTorch 0.4.1\(^6\) toolkit.

4.2.3 Dual Encoder with Dual Attention

This Dual Encoder model consists of two bidirectional LSTMs with shared parameters that are used to encode the query and the answer by summing the forward and backward passes. The final hidden state of the query is then used to compute a soft-attention (Luong et al., 2015) over the answer's hidden states. Then, this attention-weighted answer is concatenated with the final query hidden state and passed

---
\(^3\)https://www.python.org/downloads/release/python-366/
\(^4\)https://scikit-learn.org/
\(^5\)https://nlp.stanford.edu/projects/glove/
\(^6\)https://pytorch.org/
through a feed-forward tanh layer to create a contextualized query representation. Symmetrically, the same process is applied to the answer and the two resulting vectors are concatenated and passed through a MLP with one hidden layer and a sigmoid output.

The model was trained to minimize the binary cross-entropy loss function using Adam optimizer and, similar to the Siamese-Like Dual Encoder baseline, it was initialized with pre-trained GloVe word embeddings, which were frozen during the first training epochs and fine-tuned during the last epochs. Figure 4.2 illustrates the described model. Also, like the Siamese-Like Dual Encoder, this model was implemented using Python 3.6.6 and PyTorch 0.4.1.

4.2.4 ELMo Dual Encoder

Similar to the previously presented architecture, this model consists of two bidirectional LSTMs with shared parameters. The first encoder is used to encode a query by concatenating the forward and backward passes. Similar to the Dual Encoder with Dual Attention model, the final hidden state of the query is used to compute a soft-attention (Luong et al., 2015) over the answer’s hidden states, or vice-versa depending on the corpus. Finally, all the encoding vectors, namely, the query encoding, the answer encoding, and the attention weighted average are concatenated and passed to a MLP with two hidden layers and a final sigmoid output.

Yet, the important piece of the described architecture is the embedding layer. In contrast to the previously describes models, this Dual Encoder uses contextualized ELMo embeddings concatenated with fixed GloVe embeddings. This will not only produce contextualized information as well as fixed information about input words. Figure 4.3 illustrates the described architecture.

---

7For the Ubuntu Dialog Corpus and for the Twitter Customer Support corpus since the contexts are much longer than answers we compute attention over the context instead.
Figure 4.2: Bidirectional Dual Encoder with Dual Attention.

Figure 4.3: ELMo Dual Encoder model architecture.
4.3 Generative-based Model

In contrast to retrieval-based models, generative models, such as Seq2Seq models, can generate completely new answers when a new question is received. However, it is a common believe that generative models do not work well for a real setup, as they tend to produce generic or ungrammatical answers. For that reason, retrieval-based approaches are more common (Wu et al., 2017). Nonetheless, taking into consideration that, in customer support scenarios, the answer space in narrow, we believe that Seq2Seq models can obtain a good performance, even for long documents such as emails.

We have implemented a Seq2Seq model with attention consisting into two LSTMs with shared embeddings: an encoder and a decoder. Both LSTMs are composed of 2 stacked layers. For each decoding step, an weighted averaged of all the encoder’s hidden states is calculated using Luong’s attention mechanism (Luong et al., 2015) and concatenated to the current decoder hidden state in order to help the decoder stay on topic. Figure 4.4 illustrates the described model.

![Seq2Seq model with attention.](image)

The embedding layer is initialized with pre-trained GloVe embeddings and then fine-tunned during training. We also applied teacher forcing with a probability of 80%. This means that, the input symbol given to the decoder at each training step, is the ground truth instead of the one previously outputted by the decoder. This helps to guide the decoder when learning, but may leave it unprepared to generate whole sequences or to recover from small mistakes during test time.

For testing this model 2 different decoding strategies were implemented: greedy decoding and beam search. The greedy decoding is the same strategy that is used during training, meaning that at each time the decoder chooses the most likely token, without considering implications of this choice in future steps. In contrast, the beam search keeps track of the $k$ best hypothesis, thus at each time step, for these $k$ hypotheses we have $n$ new possible tokens. This makes a total of $k \times n$ new hypothesis to consider from
which we only keep the \( k \) best ones. Once every hypothesis reached the end-of-sentence (EOS) token, we return the hypothesis with the highest score. With this strategy the model is able to recover from bad greedy choices that will end up increasing the error rate in the future.

Similar to all the other described models, this model was implemented using Python 3.6.6 and PyTorch 0.4.1.

### 4.4 Hybrid Model

As previously said, Alfalahi et al. (2015) formulated the concept of a shadow answer, i.e., a bag-of-words containing the expected terms in the answer. In the referred work, the authors computed a co-occurrence matrix between the terms used in the training queries and the respective answers. Then, during test time, when a new query arrives, the query is transformed into a shadow answer and the shadow answer is used as input for a retrieval-based system that ranks all template answers. As we have seen in Equation 2.4, when decoding with a Seq2Seq model, we are computing expected terms, one by one, by conditioning not only in the query tokens but also in the previously decoded tokens. In this way, we can think of the generated answer as a structured version of the shadow answer. In our hybrid approach, after having an answer provided by the previously described Seq2Seq model, we transform it into a TF-IDF feature space and we retrieve from the training data the answer with the highest cosine similarity.

### 4.5 Generative Adversarial Networks

Deep generative models based on RNNs are typically trained by maximizing log-likelihood. During training time, when generating the next word in a sentence it is common to feed the previous word from the ground truth (teacher forcing), instead of the one generated by the model (which may be incorrect). While this is helpful in guiding the training, especially in an early stage, this technique may cause exposure bias – the model does not know how to recover from a wrongly predicted word. One way to tackle this difference between training and inference is by inputting the generated word instead, with a given probability. Another way is to create a tailor-made loss for the task and use it to score the sentence (e.g. using BLEU for machine translation). However, for tasks such as answer generation, these metrics may not be informative enough to guide the training of the model.

Following the approach of Yu et al. (2017) both previous components (Sections 4.3 and 4.2.3) are reused here: the Dual Encoder working as a discriminator and the Seq2Seq model working as a generator. Both are pre-trained in their original tasks, and later trained iteratively, one epoch for the generator and another one for the discriminator. The discriminator should now distinguish between real answers and
synthetic ones, while the generator should produce samples good enough to fool the discriminator. We decided to use the Dual Encoder with Dual Attention model and not the ELMo Dual Encoder as discriminator because, the former, was much faster in terms of computations.

Also, following the approach proposed in Yu et al. (2017), the generator was trained using the REINFORCE algorithm, to overcome the impossibility of differentiating discrete outputs. At first each sequence was generated deterministically through greedy search, and one single reward was provided for the whole sequence, based on the discriminator’s score. Later a more sophisticated reward function was used: for each partial sequence, 3 possible full sequences were generated by sampling from the softmax output. The average reward of these samples is used as the current token’s reward. This makes it possible to evaluate incomplete sequences and to provide different rewards to each token (Figure 4.5).

Figure 4.5: Sampling example for partial sequences from (Yu et al., 2017). In this example 4 full sequences are generated by sampling from the softmax output and the average reward is used as reward for the third token.
Experiments

Contents

5.1 Evaluation Metrics .................................................. 53
5.2 Quantitative Analysis ................................................ 56
5.3 Qualitative Analysis .................................................. 59
5.4 Human Evaluation ..................................................... 62
5.5 Generative Adversarial Networks ................................. 63
In this Chapter, we will start with a brief explanation of the evaluation metrics that will be used to evaluate our models (Section 5.1). In Section 5.2 we describe the results achieved for the retrieval, generative and hybrid approaches. In Section 5.3 we dive deeper into our models and manually inspect interaction samples and the attention mechanisms. Section 5.4 presents our human evaluation results, and, finally, in Section 5.5 we describe our attempts to unify retrieval and generative approaches with a GAN.

5.1 Evaluation Metrics

We will start by presenting the ranking metrics that are commonly used to evaluate retrieval-based models and, then, we will explain two type of evaluation metrics commonly used in natural language generation tasks that measure how close are our models answer from the ground-truth.

5.1.1 Ranking Metrics

Retrieval-based models are commonly evaluated in an answer ranking task. From the candidate answers, only one answer is the original one and the goal of the model is to rank the correct answer as high as possible. The evaluation metrics used for this setup are recall_{N}@k and MRR_{N}.

For a set of queries \( Q \), with \( N \) candidate answers and only one correct answer \( A^* \) the recall_{N}@k is defined as:

\[
\text{recall}_{N}@k = \frac{1}{|Q|} \sum_{i=1}^{Q} I[\text{rank}(A^*) \geq k]
\]  

(5.1)

Intuitively this metric tells, on average, how many times the model ranked the ground-truth answer above a certain threshold. In-addition the MRR_{N} tells on average how high the correct answer is ranked among \( N \) candidates, and is defined as:

\[
\text{MRR}_{N} = \frac{1}{|Q|} \sum_{i=1}^{Q} \frac{1}{\text{rank}(A^*)}
\]  

(5.2)

5.1.2 Word overlap Measures

Word overlap measures evaluate the amount of words that appear simultaneous in the predicted answer and in the ground-truth. For this type of evaluation we will use METEOR (Lavie and Agarwal, 2007) and ROUGE-L (Lin, 2004).

METEOR starts by creating an alignment between the candidate and reference sentences and mapping each word in the candidate sentence with the corresponding word in the reference (if that word
exists). The corresponding word can either be a stem or a synonym and, based on this alignment, word
precision and recall are computed as follows:

\[ P = \frac{m}{|C|} \]  \hspace{1cm} (5.3)
\[ R = \frac{m}{|R|} \]  \hspace{1cm} (5.4)

Where \( m \) is the number of matching words in the candidate and reference sentences, \(|C|\) is the
length of the candidate sentence and \(|R|\) is the length of the reference. Then, precision and recall are
combined using the harmonic mean as follows:

\[ F_{mean} = \frac{10PR}{R + 9P} \]  \hspace{1cm} (5.5)

Also, a constant \( p \) is computed that penalizes non-adjacent mappings. The equation to compute \( p \) is
given by:

\[ p = 0.5 \left( \frac{c}{u_m} \right)^3 \]  \hspace{1cm} (5.6)

where \( c \) is the number of adjacent matches, and \( u_m \) is the number of matching words. Finally,
METEOR is computed as follows:

\[ \text{METEOR} = F_{mean}(1 - p) \]  \hspace{1cm} (5.7)

where \( p \) can reduce the final score by half if no \( n \)-grams with \( n > 1 \) matches.

ROUGE-L measures the longest matching sequence of words between the predicted answer and
the ground truth using the Longest Common Subsequence (LCS). The LCS is defined as the longest
set of words that occurs in both sequences in the same order, but, contrarily to an \( n \)-gram, they do not
need to be contiguous. Lin (2004) defined the ROUGE-L score between a sequence \( X \) of length \( m \) (the
ground-truth) and a candidate \( Y \) of length \( n \) (the predicted sentence) as follows:

\[ R = \frac{\text{LCS}(X,Y)}{m} \]  \hspace{1cm} (5.8)
\[ P = \frac{\text{LCS}(X,Y)}{n} \]  \hspace{1cm} (5.9)

\[ \text{ROUGE-L} = \frac{(1 + \beta^2) \times R \times P}{R + \beta^2 \times P} \]  \hspace{1cm} (5.10)
5.1.3 Semantic Evaluation Measures

Since in dialog generation, two completely different answers can share the same semantics, we decided to explore other type of metrics. A commonly used alternative to word overlap measures, is to use methods based on word embeddings, such as embedding average, greedy matching and vector extrema.

The embedding average computes the embedding of a sequence $s$ by averaging the embeddings of the words in that sequence:

$$e_s = \frac{\sum_{w \in s} e_w}{\| \sum_{w' \in s} e_{w'} \|}$$ (5.11)

After computing a sequence-level embedding for our candidate answer $e^c$ and a sequence-level embedding for our ground-truth answer $e^*$ we just need to apply the cosine similarity to obtain a score of how close these two answers are in a semantic space.

Similarly, the vector extrema (Forgues and Pineau, 2014) computes a sequence-level embedding for both sequences and compares them using a cosine similarity. The difference lies in how the embeddings are computed. In the vector extrema a sequence-level embedding $e_s$ is computed by taking the most extreme value of the word embeddings in $s$ along all dimensions:

$$e^d_s = \begin{cases} \max_{w \in s} e^d_w & \text{if } e^d_w > \| \min_{w' \in s} e^d_{w'} \| \\ \min_{w \in s} e^d_w & \text{otherwise} \end{cases}$$ (5.12)

The intuition behind this metric is that more informative words will be highlighted in comparison to common words. This is due to a property of the embedding space in which common words tend to lie closer to the origin of the semantic space.

In contrast to the previously described semantic metrics, the greedy matching (Rus and Lintean, 2012) does not compute sequence-level embeddings. This metric computes directly a similarity score between a model answer $A$ and the ground-truth $A^*$ as follows:

$$G(A, A^*) = \frac{\sum_{w \in A} \max_{w' \in A^*} \cosine_sim(e_w, e_{w'})}{|A|}$$ (5.13)

$$GM(A, A^*) = \frac{G(A, A^*) + G(A^*, A)}{2}$$ (5.14)

The greedy matching score matches each word in the model answer with a word in the ground-truth answer that maximizes the cosine similarity. The score is the average of the computed similarities over all the words in the sequence. Since the sequences $A$ and $A^*$ might have different lengths, this score is computed reversing the roles of the sequences and the average is taken as a final similarity score.
5.2 Quantitative Analysis

Taking into consideration the evaluation metrics described in the previous section, we start by evaluating our retrieval-based models in an answer ranking task. Then, after selecting our best retrieval-model for the private email corpus and Twitter Customer Support corpus, we compare the retrieval, generative and hybrid approaches in an answer generation task using Word Overlapping Metrics and Embedding-based Metrics. We decided to exclude the Ubuntu Dialog corpus from some experiments due to the fact that this corpus is a community support corpus, rather than a customer support one and, for that reason, we had no template answers to use.

With exception of the ranking metrics, all other metrics have shown similar correlation with human judgement for Task-Oriented Dialogue systems (Sharma et al., 2017) and can be easily computed with the NLG-eval python toolkit.\[1\]

5.2.1 Answer Ranking Task

Tables 5.1, 5.2 and 5.3 summarize our results for the retrieval-based models in the described ranking task across different corpora. In this tables, TF-IDF is our retrieval baseline explained in Section 4.2.1, SLDE stands for Siamese-like Dual Encoder from Section 4.2.2, DADE stands for Dual Encoder with Dual Attention from Section 4.2.3 and ELMoDE is the Dual Encoder with ELMo embeddings from Section 4.2.4.

After observing that the ELMoDE model achieved superior performance across all corpora, and since both the Twitter corpus and the private email corpus are much smaller than the Ubuntu Dialog corpus, we experimented with a transfer learning technique in which the ELMoDE is pre-trained with the Ubuntu corpus and then adapted to a smaller corpus. The adapted model is referred as “ELMo+transfer” in Table 5.2 and 5.3. This adaptation techniques yield to improvements for both corpora. In addition, we observe that starting the training with a previously trained model provides a much better initialization, that, in consequence, leads to a faster convergence and better local minimal (Figure 5.1).

Table 5.1: Ubuntu Dialog corpus ranking task results.

<table>
<thead>
<tr>
<th></th>
<th>R_{10}@1</th>
<th>R_{10}@2</th>
<th>R_{10}@3</th>
<th>R_{10}@5</th>
<th>MRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-IDF</td>
<td>0.5006</td>
<td>0.6210</td>
<td>0.6951</td>
<td>0.8127</td>
<td>0.6372</td>
</tr>
<tr>
<td>SLDE</td>
<td>0.3031</td>
<td>0.5052</td>
<td>0.6556</td>
<td>0.8479</td>
<td>0.5200</td>
</tr>
<tr>
<td>DADE</td>
<td>0.6060</td>
<td>0.7717</td>
<td>0.8595</td>
<td>0.9479</td>
<td>0.7461</td>
</tr>
<tr>
<td>ELMoDE</td>
<td>0.6468</td>
<td>0.8095</td>
<td>0.8866</td>
<td>0.9590</td>
<td>0.7766</td>
</tr>
<tr>
<td>Lowe et al. (2015)</td>
<td>0.5520</td>
<td>0.7210</td>
<td>-</td>
<td>0.9240</td>
<td>-</td>
</tr>
</tbody>
</table>

---

1https://github.com/Maluuba/nlg-eval
Table 5.2: Twitter Customer Support corpus ranking task results.

<table>
<thead>
<tr>
<th></th>
<th>$R_{10}@1$</th>
<th>$R_{10}@2$</th>
<th>$R_{10}@3$</th>
<th>$R_{10}@5$</th>
<th>MRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-IDF</td>
<td>0.4035</td>
<td>0.5215</td>
<td>0.5993</td>
<td>0.7081</td>
<td>0.5478</td>
</tr>
<tr>
<td>SLDE</td>
<td>0.3438</td>
<td>0.5307</td>
<td>0.6768</td>
<td>0.8564</td>
<td>0.5476</td>
</tr>
<tr>
<td>DADE</td>
<td>0.4437</td>
<td>0.6278</td>
<td>0.7454</td>
<td>0.8738</td>
<td>0.6224</td>
</tr>
<tr>
<td>ELMoDE</td>
<td>0.5082</td>
<td>0.7090</td>
<td>0.8201</td>
<td>0.9235</td>
<td>0.6804</td>
</tr>
<tr>
<td>ELMoDE+transfer</td>
<td><strong>0.5708</strong></td>
<td><strong>0.7489</strong></td>
<td><strong>0.8441</strong></td>
<td><strong>0.9361</strong></td>
<td><strong>0.7220</strong></td>
</tr>
</tbody>
</table>

Table 5.3: Private Email corpus ranking task results.

<table>
<thead>
<tr>
<th></th>
<th>$R_{10}@1$</th>
<th>$R_{10}@2$</th>
<th>$R_{10}@3$</th>
<th>$R_{10}@5$</th>
<th>MRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-IDF</td>
<td>0.3730</td>
<td>0.5118</td>
<td>0.5949</td>
<td>0.7060</td>
<td>0.5283</td>
</tr>
<tr>
<td>SLDE</td>
<td>0.3037</td>
<td>0.5281</td>
<td>0.6892</td>
<td>0.8823</td>
<td>0.5307</td>
</tr>
<tr>
<td>DADE</td>
<td>0.4616</td>
<td>0.6927</td>
<td>0.8211</td>
<td>0.9325</td>
<td>0.6555</td>
</tr>
<tr>
<td>ELMoDE</td>
<td>0.5417</td>
<td>0.7656</td>
<td>0.8736</td>
<td>0.9592</td>
<td>0.7152</td>
</tr>
<tr>
<td>ELMoDE+transfer</td>
<td><strong>0.5669</strong></td>
<td><strong>0.7860</strong></td>
<td><strong>0.8831</strong></td>
<td><strong>0.9620</strong></td>
<td><strong>0.7327</strong></td>
</tr>
</tbody>
</table>

Figure 5.1: Loss curve comparison over epoch for the ELMo Dual Encoder model trained from scratch with Twitter data (left side) and adapted from Ubuntu Dialog corpus to the Twitter data (right side). In both figures the green line represents the training loss while the blue line represents the validation loss.

### 5.2.2 Answer Generation Task

Tables 5.4, 5.5 and 5.6 present the obtained results, for the different corpora, in an answer generation task in which the model answer $\hat{a}$ is compared against the ground truth answer $a$ in terms of word overlapping and word embeddings based metrics. In these tables, the referred generative model is the Seq2Seq model explained in Section 4.3 and the hybrid model is the Seq2Seq model plus answer retrieval explained in Section 4.4. Also, in Table 5.5 and Table 5.6, the presented retrieval model is the ELMo Dual Encoder model explained in Section 4.2.4 and adapted from the Ubuntu corpus to these two corpora. For the Twitter Customer Support corpus and for the private email corpus, due to a clear language standardization and template usage, we were able to extract 1000 answer templates and use...
those as candidate answers for the retrieval and for the hybrid model. In contrast, for the Ubuntu Dialog corpus, due to a significantly different linguistic style between users, we could not find a set of candidate answers that guaranteed a good answer coverage and fast computations. For that reason, we were not able to evaluate our retrieval-based model in such corpus. However, since it is relatively fast to compute distances inside a TF-IDF space, we were able to experiment with our hybrid model assuming that, in the retrieval phase, all the training answers are possible candidates.

<table>
<thead>
<tr>
<th>Model</th>
<th>METEOR</th>
<th>ROUGE-L</th>
<th>Embeddings Average</th>
<th>Vector Extrema</th>
<th>Greedy Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>generative (beam-1)</td>
<td>0.1835</td>
<td>0.2579</td>
<td>0.5191</td>
<td>0.4045</td>
<td>0.7655</td>
</tr>
<tr>
<td>generative (beam-3)</td>
<td>0.2478</td>
<td>0.3290</td>
<td>0.4104</td>
<td>0.3361</td>
<td>0.7376</td>
</tr>
<tr>
<td>generative (beam-5)</td>
<td>0.2615</td>
<td>0.3382</td>
<td>0.3692</td>
<td>0.3131</td>
<td>0.7271</td>
</tr>
<tr>
<td>hybrid (beam-1)</td>
<td>0.2740</td>
<td>0.2988</td>
<td>0.4252</td>
<td>0.3529</td>
<td>0.7259</td>
</tr>
<tr>
<td>hybrid (beam-3)</td>
<td>0.2715</td>
<td>0.3341</td>
<td>0.3663</td>
<td>0.3274</td>
<td>0.7280</td>
</tr>
<tr>
<td>hybrid (beam-5)</td>
<td>0.2697</td>
<td>0.3337</td>
<td>0.3589</td>
<td>0.3197</td>
<td>0.7247</td>
</tr>
</tbody>
</table>

Table 5.4: Answer generation results for the Ubuntu Dialog corpus v2.0.

<table>
<thead>
<tr>
<th>Model</th>
<th>METEOR</th>
<th>ROUGE-L</th>
<th>Embeddings Average</th>
<th>Vector Extrema</th>
<th>Greedy Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>generative (beam-1)</td>
<td>0.1678</td>
<td>0.2368</td>
<td>0.8903</td>
<td>0.4185</td>
<td>0.7508</td>
</tr>
<tr>
<td>generative (beam-3)</td>
<td>0.2232</td>
<td>0.2675</td>
<td>0.8823</td>
<td>0.4693</td>
<td>0.7911</td>
</tr>
<tr>
<td>generative (beam-5)</td>
<td>0.2217</td>
<td>0.2641</td>
<td>0.8790</td>
<td>0.4639</td>
<td>0.7878</td>
</tr>
<tr>
<td>hybrid (beam-1)</td>
<td>0.1104</td>
<td>0.1809</td>
<td>0.8733</td>
<td>0.4786</td>
<td>0.7886</td>
</tr>
<tr>
<td>hybrid (beam-3)</td>
<td>0.1291</td>
<td>0.1941</td>
<td>0.8812</td>
<td>0.4635</td>
<td>0.7809</td>
</tr>
<tr>
<td>hybrid (beam-5)</td>
<td>0.1297</td>
<td>0.1936</td>
<td>0.8831</td>
<td>0.4627</td>
<td>0.7793</td>
</tr>
<tr>
<td>retrieval (best)</td>
<td>0.1298</td>
<td>0.1723</td>
<td>0.8600</td>
<td>0.4421</td>
<td>0.7505</td>
</tr>
</tbody>
</table>

Table 5.5: Answer generation results for the Twitter Customer Support corpus.

<table>
<thead>
<tr>
<th>Model</th>
<th>METEOR</th>
<th>ROUGE-L</th>
<th>Embeddings Average</th>
<th>Vector Extrema</th>
<th>Greedy Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>generative (beam-1)</td>
<td>0.2565</td>
<td>0.3614</td>
<td>0.9073</td>
<td>0.5034</td>
<td>0.7906</td>
</tr>
<tr>
<td>generative (beam-3)</td>
<td>0.2241</td>
<td>0.2916</td>
<td>0.9382</td>
<td>0.4596</td>
<td>0.7809</td>
</tr>
<tr>
<td>generative (beam-5)</td>
<td>0.2149</td>
<td>0.2890</td>
<td>0.9401</td>
<td>0.4492</td>
<td>0.7811</td>
</tr>
<tr>
<td>hybrid (beam-1)</td>
<td>0.2083</td>
<td>0.3127</td>
<td>0.9262</td>
<td>0.4849</td>
<td>0.7856</td>
</tr>
<tr>
<td>hybrid (beam-3)</td>
<td>0.1681</td>
<td>0.2407</td>
<td>0.9381</td>
<td>0.4569</td>
<td>0.7711</td>
</tr>
<tr>
<td>hybrid (beam-5)</td>
<td>0.1566</td>
<td>0.2404</td>
<td>0.9387</td>
<td>0.4458</td>
<td>0.7807</td>
</tr>
<tr>
<td>retrieval (best)</td>
<td>0.1710</td>
<td>0.2589</td>
<td>0.8532</td>
<td>0.4385</td>
<td>0.7371</td>
</tr>
</tbody>
</table>

Table 5.6: Answer generation results for the private email corpus.

Analysing the previous tables we can observe that our generative model seems to outperform all others. Also, for the Twitter Customer Support corpus and for the private email corpus, we can observe that our retrieval-based model is the one with the lowest scores both in terms of word overlapping metrics and embeddings distance. This result indicates that the ranking task commonly used to test Dual Encoder models is not realistic. Despite that, although these metrics have demonstrated a similar correlation with human judgement (Sharma et al., 2017), this correlation is weak and a more reliable evaluation is required in order to decide which model works best in a customer support scenario.
5.3 Qualitative Analysis

In order to assess the answer quality of the models evaluated in the previous section we selected some question samples from the Twitter Customer Support corpus and manually inspect the produced answers. We decided to perform this analysis only with the Twitter Customer Support corpus due to the fact that the email corpus is private and the Ubuntu Dialog corpus is not an example of a real customer support scenario.

In Table 5.7 we can observe the different answers provided by the trained models in comparison to the ground truth.

<table>
<thead>
<tr>
<th>Context</th>
<th>ELMo Dual Encoder</th>
<th>Seq2seq</th>
<th>Seq2seq + shadow</th>
<th>Ground Truth</th>
</tr>
</thead>
<tbody>
<tr>
<td>@AppleSupport unable to add my debit card details for iTunes... Plz help. Was this order placed through our online store?</td>
<td>Let's get you in touch with our iTunes Store support here: <em>URL</em>.</td>
<td>You can do that here: <em>URL</em>. Or we can get you in touch with our iTunes Store support here: <em>URL</em>.</td>
<td>It's important to be able to get those apps in the iTunes and App Store when you need them. Our iTunes Store Team can help look into this with you. You can reach them here: <em>URL</em> to get started.</td>
<td></td>
</tr>
<tr>
<td>Dear @AppleSupport, I'm in China and GPS is very inaccurate. Am I doing something wrong?</td>
<td>Let's get you pointed in the right direction. Our team will take it from here.</td>
<td>Let's get you pointed in the right direction. Our team will take it from here.</td>
<td>Hey there! Check out this article, which has troubleshooting steps for Maps: <em>URL</em>. Let us know if you are still having issues after trying these steps.</td>
<td></td>
</tr>
<tr>
<td>@AppleSupport what kind of card do I need to get into the app store, not for music. Just for in app purchases?</td>
<td>Let's see if we can get those playlists going for you again. Which device are you using Apple Music with where you see this?</td>
<td>Thanks for reaching out to us. We would love to help you. Please follow the steps in this article: <em>URL</em>.</td>
<td>That's a great question. Although there isn't a payment method specific to in-app purchases, you can use any method mentioned here: <em>URL</em>.</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.7: Customer question plus the answers given by our models plus the ground truth answer.

Also, for the ELMo Dual Encoder model, we decided to inspect if the attention mechanism learned to highlight important words inside a context. For that we feed the model with two different context/answer pairs; one pair containing only a single sentence of context and, the other, containing a long context sequence with multiple turns.

In Figure 5.8 we can observe the attention distribution after passing to the model the context sentence “@AppleSupport I've downloaded the new iPhone update and my battery lasts half as long. What's going on?!?” and the following customer support agent answer: “We completely understand wanting to have good battery life. How long is the battery lasting on a full charge after the update?” From this figure, we can observe that the attention weights are correctly placed over the words “update”, “my”, “battery” which indeed are three important words in the given context. Nonetheless, since the provided answer also contain those words, from this example, it looks like that the attention mechanism is creating a simple terms matching and highlighting some common words between the two sentences.
Table 5.8: ELMo Dual Encoder Attention weights over a short context sentence.

In Figure 5.9 we show the last reply message of the user ("Worked" followed by 2 Unicode signs), and the last agent message ("Please keep us posted afterwards"). However, this context was long, containing 75 tokens and describing a problem with downloading apps. In the middle of the conversation, the customer support agent suggests updating the iPhone version and, in the end, after following the agent instructions, the user solved the problem. Given the described context, and the following customer support agent answer: “Glad to hear and just reach out again if you need help. Have a nice day!" the attention mechanism in our model is placed over the first\(^2\) three tokens, which, indicates that the model is able to perform some reasoning beyond simple terms matching.

Table 5.9: ELMo Dual Encoder Attention weights over a long context with multiple turns.

Also, for our Seq2Seq model, we decided to inspect the attention alignments created during decoding. Interestingly, from Figure 5.10, we can observe that the attention mechanism is consistently highlighting the last tokens in the context. Our explanation for such phenomenon is that, since no ex-

\(^2\)Since the context is appended in reversed order, the last message is in truth the first message that the model sees.
licit alignment could be found, the encoder “encodes” all the context information into the last hidden states of the context. Then, during decoding, the attention mechanism, will frequently revisit those states in order to “retrieve” a summary of the encoded context.

Table 5.10: Seq2Seq context/answer attention alignments during decoding. In this Figure, dark blue squares represent strong attention values over the column token while decoding the row token.
5.4 Human Evaluation

Considering that the usual metrics can lead to inaccurate results, as answers tend to share some phrases or even sentences, which are positively scored by these metrics, but are semantically different (e.g. “We’d love to help out. Tell us more details about what you’re experiencing, and we’ll check out some options for you.” vs “We’d love to help out. Tell us more details of what you’re experiencing. Which device are you trying to update?”, we decided to perform an human evaluation. For the same reasons explained in Section 5.3, we had to use only the Twitter Customer Support corpus.

This evaluation was performed by 30 annotators, organized into groups of 6. Each group received a batch of 10 different contexts and 4 possible answers for each context: the ground-truth, the answer from the Seq2Seq model, the answer from the ELMoDE model and the answer from the hybrid model. The annotator is then asked to evaluate the quality of the different answers by providing a score from 1 (worse) to 5 (best) to each answer. The annotation criteria is:

1. Completely irrelevant answer.
2. Irrelevant answer.
3. Related but generic answer.
4. Relevant answer.
5. Perfect answer.

The great majority of annotators were Apple device users recruited from a MSc. in Information Management taking a text analytics course. Results are shown in Table 5.11.

<table>
<thead>
<tr>
<th>Model</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Average Score</th>
<th>Fleiss Kappa</th>
<th>95 % CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid</td>
<td>28.67</td>
<td>17.0</td>
<td>21.33</td>
<td>21.67</td>
<td>11.33</td>
<td>2.7</td>
<td>0.14</td>
<td>[0.10, 0.19]</td>
</tr>
<tr>
<td>Generative</td>
<td>29.67</td>
<td>20.33</td>
<td>20.00</td>
<td>21.00</td>
<td>9.00</td>
<td>2.593</td>
<td>0.13</td>
<td>[0.09, 0.17]</td>
</tr>
<tr>
<td>Retrieval</td>
<td>30.33</td>
<td>22.00</td>
<td>23.67</td>
<td>17.00</td>
<td>7.00</td>
<td>2.483</td>
<td>0.08</td>
<td>[0.04, 0.13]</td>
</tr>
<tr>
<td>Ground Truth</td>
<td>7.33</td>
<td>10.67</td>
<td>20.67</td>
<td>30.00</td>
<td>31.33</td>
<td>3.273</td>
<td>0.08</td>
<td>[0.03, 0.13]</td>
</tr>
</tbody>
</table>

Table 5.11: Scores given by human annotators on the different models. CI stands for confidence interval.

In contrast to the obtained automatic results shown in Table 5.5, our hybrid model obtains a higher average score than our generative model. Although with a slight Fleiss agreement (Landis and Koch, 1977), these results, yet again, suggests that automatic evaluation for task-oriented dialog systems correlates weakly with human judgement (Sharma et al., 2017).
5.5 Generative Adversarial Networks

In an attempt to combine both previous approaches (discriminative and generative) a GAN was used to generate answers for the private email corpus. Both the generator and the discriminator were pretrained separately, and then iteratively trained in an adversarial setup in which the discriminator’s guess (of whether a sample was real or synthetic) is used as a signal to update the generator’s parameters.

In our first experiments, a simple approach was used to design the generator’s reward function: one single reward was given to a whole answer after it was generated. This has the major drawback of not distinguishing which words were responsible for poor/good results. Although this approach may provide reasonable results in different contexts such as videogames (Karpathy, 2016), in this case a bizarre behaviour became dominant. After a few training iterations, the pre-trained generator started to output a single symbol repeatedly. In different runs it would get stuck on different symbols. This issue could be caused by exaggerated learning rates (although the validation loss never decreased even with much lower learning rates). Alternatively, the simplistic reward adopted could be the reason behind this.

To investigate this issue further a more elaborate reward was designed, following Yu et al. (2017). At each step of the sequence, a unique reward was provided to the partial sequence generated so far. To do this, 3 full answers were generated from each partial sentence, and the average reward provided by the discriminator would become the partial sentence’s reward. To escape the deterministic behaviour of our model, each token was sampled instead of always selecting the most likely one. This allowed to provide different reward values to each word of the sentence. The behaviour of picking one token repeatedly disappeared, but the validation loss increased even further than before. We hypothesize that this may be due to improper pre-training of the discriminator (either too much or too little could cause it to provide bad reward values). Poor hyper-parameter choice could also be the cause, since balancing the amount of training of the discriminator and generator may impact greatly results.

An interesting aspect of this model, which could not be explored due to this initial issue, is that the discriminator is not necessarily learning the same task during pre-training and training. In pre-training, the discriminator must distinguish between correct and incorrect question/answer pairs. During training time, it must distinguish between "real" answers and a generated one. If the discriminator is not pre-trained, it may have no incentive to look at the question when classifying the answer. It may also lose this behaviour during training, if the generated answers are not fluent enough.
Conclusion

Contents

6.1 Contributions ................................................................. 67
6.2 Future Work ................................................................. 68
The goal of this dissertation was to go beyond previous studies in the customer support email domain and apply recent NLP findings to create deep learning models capable of assisting customer support human agents dealing with the vast amounts of emails received by companies every day. This chapter concludes this dissertation by presenting our contributions and findings in Section 6.1 and future directions in Section 6.2.

6.1 Contributions

The main contribution of our work is a series of deep learning models that are capable of assisting customer support human agents in their craft by suggesting email answers that can be used as templates. These models include state-of-the-art NLP techniques such as the incorporation of attention mechanisms and transfer learning via pre-trained word embeddings and/or domain adaptation.

Particularly, in an answer ranking task we were able to show that using an attention mechanism to highlight important words in long contexts and/or answers can greatly improve upon a simpler model without attention. Also, we showed that training a Dual Encoder with the Ubuntu Dialog corpus and then adapting the same model to a different domain with much less data can significantly improve the models performance, even when the target domain is clearly different (e.g: pre-training the model with the Ubuntu corpus, characterized by long conversations with short messages and non-uniform language, and then adapting it to our private email corpus characterized by short conversations with long messages and a uniform language).

In addition, during this work, we highlight differences between community support and traditional customer support and, we compare two commonly used approaches, namely, retrieval-based and generative-based. Our results show that for a customer support scenario, a generative approach, based on a Seq2Seq model with attention obtains better results than a retrieval-based approach, based on a Dual Encoder model. In addition, by adding a simple retrieval step after the Seq2Seq answer generation, we were able to achieve a better human score, which suggests that combining both approaches is advantageous.

Finally, during this work, we have also tried to unite these two approaches with a GAN. Using a GAN did not work as expected. Still, we believe that the problems we have encountered could be helpful in future attempts to jointly train these models.
6.2 Future Work

For future work, it would be interesting to explore the data from other companies, rather than Apple, in the Twitter Customer Support corpus. We believe that this corpus is highly relevant for the area but it has not been properly explored. Also, during this thesis, we have explored retrieval-based models more deeply, but, in the end, a simpler hybrid model worked better. In our future work we would like to create a more complex hybrid model that uses retrieved answers as start-point for generation. More precisely, we would like to test the prototype-then-edit model proposed by Wu et al. (2018), that, until here, proved to be effective for open-domain dialogues.

Also, one interesting idea would be to explore pre-trained transformer models such as BERT (Devlin et al., 2018), in order to create a more powerful generative-model.
Bibliography


73

