ChatBot with GANs

Auteur : Castillo Lenz, Sergio Miguel
Promoteur(s) : Ittoo, Ashwin
Faculté : Faculté des Sciences appliquées
Diplôme : Master en sciences informatiques, à finalité spécialisée en "intelligent systems"
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/11395

University of Liège

Faculty of Applied Sciences

Chatbots with GANs

Author:
Sergio Castillo

Supervisor:
Prof. Ashwin Ittoo

Master thesis submitted for obtaining the degree of MSc in Computer Science.

Academic Year 2020 - 2021

Abstract

Since its introduction in Goodfellow et al. (2014), the architecture of Generative Adversarial Networks (GANs) have experienced various evolutions to reach its current state where it is capable to recreate realistic images of any given context. Those improvements, both in terms of complexity and stability, enabled successful applications of GANs frameworks in the field of computer vision and transfer learning. On the other hand, GANs lack of successful applications within the field of Natural Language Processing (NLP) where models based on *Transformers* architecture, such as Bidirectional Encoder Representations from Transformers (BERT) and Generative Pre-Training (GPT), remain the current state-of-the-art for various NLP tasks.

Given this current situation, this thesis investigates why GANs remain underused for NLP tasks. As such, we explore some researchers' proposals within the area of Dialog Systems by using data from the Daily Dialog dataset, a human-written and multi-turn dialog set reflecting daily human communication.

Moreover, we investigate the influence of an embedding layer of the proposed GAN models. In order to do so first, we test pre-trained "word-level" embeddings, such as Stanford's Glove and Spacy embeddings. Second, we train the model by using our own word embeddings coming from the Daily Dialog dataset. The Word2Vec algorithm is used in this case. Third, we explore the idea of using BERT as a contextualized word embeddings. From these experiments it was observed that the use of pre-trained embeddings, not only accelerates the convergence during the training but also, improves the quality of the produced samples by the model, to some extents avoiding an early arrival of mode collapse.

In conclusion, despite their limited success in the NLP area, GAN-trained models offer an interesting approach during the training phase, as the generator $\mathcal{G}$ is able to produce different but potentially correct response samples and is not penalized by not producing the most likely single correct sequence of words. This actually follows an important characteristic of the human learning process. Overall, this thesis successfully explores propositions made to tackle drawbacks of the GAN architecture within the NLP area and opens doors for critical progresses in the area.

Acknowledgements

I would like to express my sincere gratitude to all those who help me in the completion of this thesis.

- I would like to acknowledge my supervisor Prof. Ashwin Ittoo for the time that he spent with me, providing advice, suggesting articles to read and above all for giving me the opportunity to work on this interesting topic.
- I would like to thank all the members of the Dethier family, who supported me through all these years. To Fanny Dethier for her time spent proofreading parts of this document.
- Then, I would like to thank my family for the effort and dedication in supporting me unconditionally throughout my life.
- Finally, I would like to thank users Mikel Chateau and serk909, for their support and for the information provided when using the Colab platform.

Contents

Abstract **ii**

Acknowledgements **iii**

1 Introduction **1**

2 Theoretical Background **3**

2.1 Natural Language Processing 3

2.2 Recurrent Neural Networks 4

2.2.1 Gated Recurrent Unit 5

2.2.2 Regularization Techniques 6

2.3 Word Representations 7

2.3.1 Word embeddings 8

2.4 Auto-encoders 9

2.4.1 Wasserstein auto-encoders (WAE) 9

2.5 Generative Adversarial Networks 10

2.6 Transformers 10

2.7 Brief Literature overview 13

3 Experimental Setup **14**

3.1 Model Architecture 14

3.1.1 DialogWAE Model 14

3.2 Dataset 16

3.3 Evaluation metrics 16

3.4 Methodology 17

3.5 Implementation and training 18

4 Results **28**

4.1 DialogWAE Model 28

4.1.1 Base Model 28

4.1.2 Spacy word vector 31

4.1.3 Own Trained word vector 33

4.1.4 No Pre-trained word vector 35

4.1.5 Bert as embedding layer 37

4.2 Comparison of the different approaches 38
4.2.1 Quantitative analysis 39
4.2.2 Qualitative analysis 40

5 Conclusion 43

A Complementary information and results 45
A.1 List of articles with buggy BLEU scores 45
A.2 Comprehensive list of missed tokens from the word embeddings 45
A.3 Trained word embeddings visual analysis 46
A.4 BLEU score reduction in “no pre-trained” model 48
A.5 Dialogue produced by the different models 48
A.6 Bert responses 50

Bibliography 51

Chapter 1

Introduction

Conversational agents or dialogue systems are programs that communicate with users through natural language, which could be speech, text or both[1]. They are divided into two classes. First, *Task-oriented dialogue agents* are used primarily to help the user to complete a task successfully. Current examples are found in digital assistants such as Google Now, Alexa, Cortana, etc. where task-oriented dialogue agents can provide information on the weather, traffic status, etc.

Second, *Chatbots* are systems that can develop extensive conversations and are capable of mimicking unstructured conversations, also called *chit chats*, specific to human-to-human interactions. Also, chatbots can be used in the process of completing a predetermined task but with the specific purpose of making task-oriented agents more natural.

Within chatbots structure, a specific task, called dialogue response generator, is dedicated to generate a response utterance both from the dialogue history (previous utterances) and the current query utterance. This generated response utterance has the characteristic to be not only coherent and fluent, but also relevant to the given query utterance. Moreover, the response produced by the dialogue response generator needs to correspond with the information provided in the dialogue history. In conclusion, an effective chatbot system should be able to generate coherent, informative and diverse responses rather than generic, short and invariant responses, avoiding as such the *safe response* problem.

Up to now, neural response generation has benefited from a strong interest from the natural language research community. The very first approaches were built on sequence-to-sequence learning (Cho et al., 2014; Sutskever et al., 2014). However, it has been proven that these models tend to suffer from the *safe response* problem, failing to produce diverse responses. Next, conditional varitional auto-encoders (CVAE) models (Shen et al., 2018; Zhao et al., 2017) came up with results that showed an ability to deal with the problem of the *safe response*. Nonetheless, similar as the previous case, some studies have proven that VAE models suffer from the posterior collapse problem (Park et al., 2018; Shen et al., 2018), according which the decoder ignores the latent variables and therefore, it degrades to a simple recurrent neural network (RNN) model.

As such, we explore in this work (1) the multi-turn dialogue generator system called DialogWAE, a novel model trained under adversarial learning, that reported better results than the state-of-the-art approaches and, (2) the influence on performance of such dialogue system when applying different word

[1]https://web.stanford.edu/ jurafsky/slp3/24.pdf

vector representations.

As a result of this work, we manage to describe a novel approach for neural dialogue generation and to evaluate this approach on a popular dialogue data set. In addition, we compare its performance during experimentations with different word vectors that are the Spacy word vector, our own-trained word vector and a hybrid one where we tried to use the outputs of a transformer model. Through quantitative metrics and a qualitative analysis, we demonstrate that the use of a simple word vector trained from the input data enables the model not only to converge faster but also, to produce informative and coherent responses in a more diverse way.

This thesis work is organized in the following way:

- Chapter 2 reviews of the main concepts and methods used in this work.
- Chapter 3 describes the architecture of the DialogWAE model, discusses the idea behind its conception and describes the data set used and, the steps that took place during the preparation of the training process of each model.
- Chapter 4 presents the details of the results of all the experiments carried on, the model selection according to specific evaluation metrics and to quantitative and qualitative analysis of the responses generated by the different models.
- Chapter 5 concludes this thesis by summarizing the work presented in the thesis and by discussing some ideas to explore in future research.

Chapter 2

Theoretical Background

The objective of this chapter is to describe the different techniques and methods used in this work rather than offer a complete historical or theoretical background.

2.1 Natural Language Processing

NLP is a branch of the artificial intelligence field that studies the interactions between the computers and the human language. It deals with the formulation and investigation of computationally effective mechanisms for the communication between people and machines through natural language, that is, the languages of the world (english, russian, etc.).

One of the goals of the NLP field is to make computers perform useful tasks for the user through inputs and outputs in human language like the written text or the speech. Therefore, there are two large research areas in NLP: Natural Language Understanding (NLU) and Natural Language Generation (NLG). The former is involved in the identification of the intended semantic of a human language expression and in the conversion into machine readable format through the use of grammatical rules, building language metamodels and creating ontologies. NLG, on other hand, is involved in the generation of coherent and grammatically correct sequences of words from some internal structured representation into a human readable language.

The architecture of NLG systems falls into two groups: rule-based systems and corpus-based systems. The former group, conceived since the early days of the NLP field, includes chatbots like ELIZA and PARRY developed by Weizenbaum (1966) and Colby et al. (1971) respectively. Both are very important in the history of NLG systems.

The latter group does not use hand-built rules, instead it calculates and develops probabilities for each word to produce later the conversation. Hence, these systems require large numbers of words to train. Nowadays, the majority of NLG systems are based on encoder-decoder generators (Sutskever et al., 2014; Cho et al., 2014), where year after year neural architectures become more sophisticated, achieving or improving state-of-the-art results (Devlin et al., 2018; Humeau et al., 2019; Brown et al., 2020).

Moreover, NLG systems can be applied in different applications such as:

Textual summaries - Where the aim of the system consists in processing a large set of data and creating a subset that represents the relevant information from the given data (Nallapati et al., 2016; Kryscinski et al., 2019; Liu and Lapata, 2019).

Neural machine translation - Here the system requires understanding a given sequence of words in one human language and converts it to its equivalent in other language. Some known systems are Bahdanau et al. (2014); Luong et al. (2015) and Vaswani et al. (2017).

Dialogue generation - The goal of this application is to produce coherent, informative and diverse responses from a given query utterance (Zhao et al., 2017; Yu et al., 2017). This application interests us in this work.

2.2 Recurrent Neural Networks

Recurrent neural networks (RNN) are a type of artificial neural networks broadly used by the research community in different tasks. They are an evolution of feed-forward networks, developed to counterpart the problems involving variable length sequential data such as audio, text, etc. Basically, they are a network with connections between nodes that form a directed graph along a time sequence (Rumelhart et al., 1986).

RNNs are used for modeling sequential data where the data element at timestep t of the sequence is dependent from all the previous data elements until timestep $t-1$ in the given sequence. Thus, the objective of a RNN is to develop the capacity to predict a data element at given timestep t when it received the entire sequence of length $t-1$.

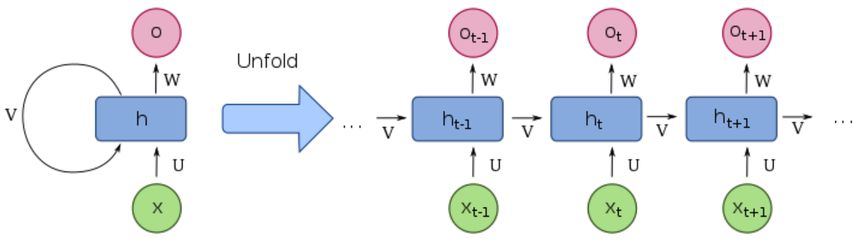


Figure 2.1: One cell of a RNN.

In Figure 2.1, it can be seen an example of one hidden cell of RNN unfolded. The graphical representation is formalized as follows: x_t is the input data at timestep t. Being o_t the output at different timesteps t, it is calculated through the hidden states h_t. U, W and V are the weights shared across the timesteps. And, the hidden states h_t are computed using the input data x_t at timestep t and the previous hidden states h_{t-1} following the equation 2.1, where typically f is a tanh activation function.

$$h_t = f(U_{x_t} + W h_{t-1}) \tag{2.1}$$

RNNs were conceived to effective model the input sequential data, typically found in natural language tasks and used in both NLG and NLU systems.

Moreover, there are two common RNN cells that are being used widely in the research field. Long short term memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated recurrent unit (GRU) (Cho et al., 2014). The former was developed with the idea to counterpart the problem of the vanishing gradients, a very typical problem in RNNs. And the latter, GRU cells are considered as a light weighted version of the LSTM cells, that despite the weak performance comparing to LSTM cells, they manage to achieved great results with less computational complexity.

2.2.1 Gated Recurrent Unit

GRU is the new generation of RNNs, very similar to an LSTM. One of its differences is that it got rid of the cell state and uses the hidden state to transfer information. Therefore its architecture has only two gates: (1) a reset gate and (2) an update gate. See Figure 2.2.

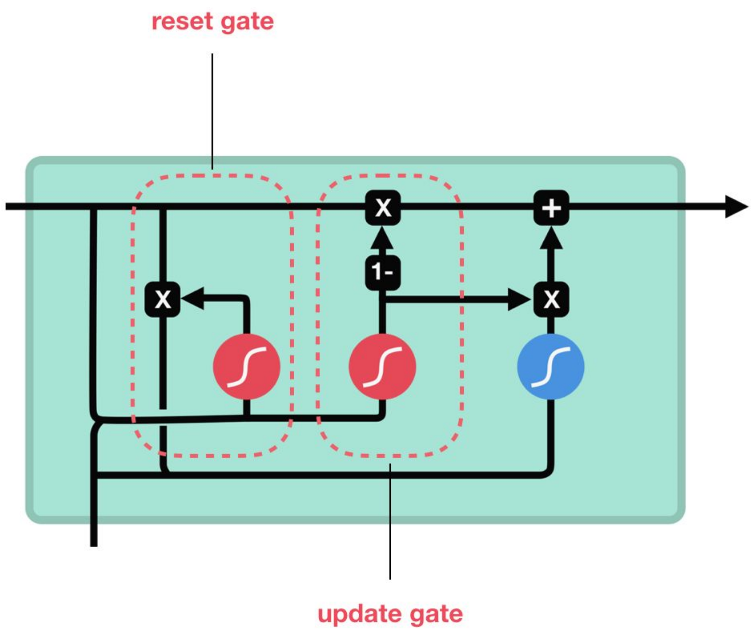


Figure 2.2: Representation of GRU unit (Cho et al., 2014).

GRU tackles the vanishing gradient problem through the reset and update gate. Those gates are basically two vectors that can decide whether the information should be passed to the output or not (Chung et al., 2014). Another characteristic is that the GRU architecture allows adaptively capture dependencies of large data sequences without discarding information from previous parts. All thanks to the gate units that are similar to the LSTM ones. These gates are responsible for regulating the information to be kept or discarded at each timestep.

Finally, while LSTMs have two different states passed between cells: the cell state and the hidden state, which carry long and short term memory. GRUs have only one hidden state transferred between timesteps. This hidden state is capable of maintaining long and short term dependencies at the same time, due to the restriction mechanisms that the hidden state and input data passes through.

2.2.2 Regularization Techniques

Overfit and underfit are common problems that a user faces when training a new model. Both problems are related to the lack of generalization of the model. The former phenomena occurs when the model begins to memorize the input data rather than learn to recognize specific trends. The latter occurs when the model cannot capture the characteristics of the input data what leads to misleading predictions. Therefore, regularization techniques are developed to help increase the model's ability to output sensible information of input data that it has never seen before. Three of these techniques are described below.

Dropout

Dropout is nowadays a common regularization technique first introduced by Srivastava et al. (2014), with the idea to prevent overfitting on the training data. Figure 2.3 shows the concept of Dropout technique, where in *(a)* we have a neural network with some hidden layers and in *(b)* it can be seen the non-output units of the hidden layers that were frozen, setting their output to 0. This process is repeated each forward pass where random units are frozen with a p probability. With this technique, the model reduces inter-dependency between units across layers so that, it becomes more robust since it is less prone to extract specific trends.

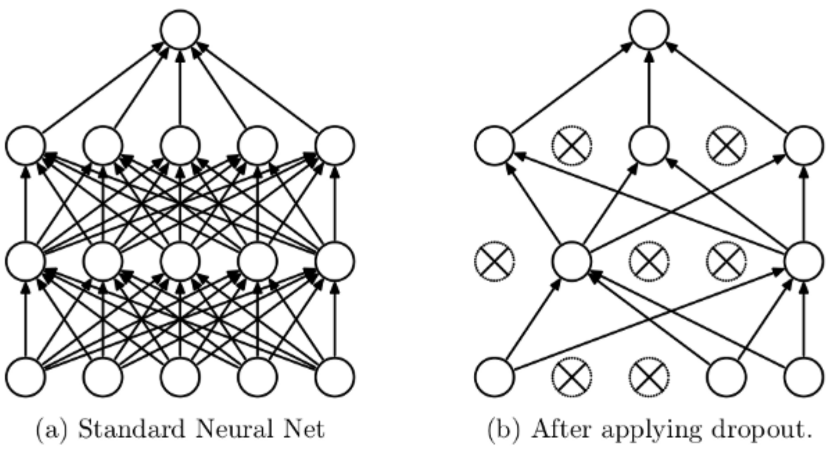


Figure 2.3: Dropout neural net representation (Srivastava et al., 2014).

Batch normalization

Batch normalization is a technique that allows neural networks to train faster and in a more stable way. Its core idea is to normalize the outputs of the input layer by re-centering and re-scaling the mean and variance computed on the batch. Batch normalization effectiveness is believed to come from the reduction of the so called *internal covariate shift* problem that is defined as the change in the distribution of network activations due to the change in network parameters during training (Ioffe and Szegedy, 2015). This reduction is calculated through the batch normalizing transformation represented as:

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \varepsilon}} * \gamma + \beta \tag{2.2}$$

where x is the layer inputs, γ and β are trainable parameter vectors of size x.

Early stopping

Early stopping is very simple technique to avoid overfitting. It consists in stopping the training process the moment values like the validation loss or the accuracy starts increasing or decreasing depending on the case, while the training loss keeps decreasing.

This algorithm requires the data set to be split into 3 sets: *training, validation* and *testing* sets. While the first set is used to adjust all the trainable parameters of the model, the loss or accuracy results of the validation set is used as reference during the training (see Figure 2.4) and finally, the testing set is left to present the results of the evaluation metrics obtained.

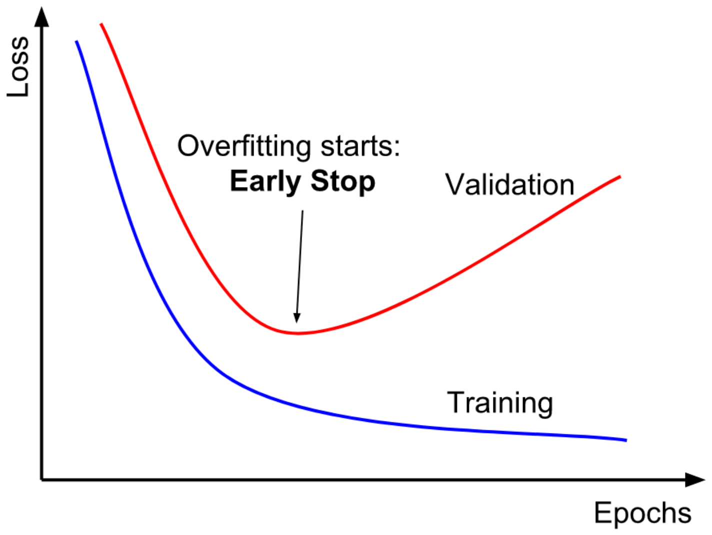


Figure 2.4: Early stopping illustration [Source].

2.3 Word Representations

For any model based on neural networks, is necessary to transform the textual information into a numerical representation so that the network can use it as input or output data, this process is called tokenization. Roughly, the tokenization process consists in separating the text into tokens by giving a defined delimiter (space, -, n-characters, etc). This process can be done at three different levels: word, character and subword (n-characters). Moreover, this is not a trivial procedure given that most common neural networks architectures (RNN, LSTM, GRU, etc.) receive and process each token from a sentence in an individual timestep.

Word level tokenization is the most used algorithm that applied to a text corpus, produces different word-level tokens, forming in turn vocabulary. Actually, word embeddings is not more than a set of language modeling where words from a vocabulary or in this case tokens, are mapped into vectors of real numbers.

2.3.1 Word embeddings

Along the years, different studies have proven the effect on performance that a words embeddings can provide to the training process (Bengio et al., 2003; Mnih and Hinton, 2009; Morin and Bengio, 2005; Mikolov et al., 2013a). Thus, word embeddings represent a key cog in the NLP research field.

A word embedding is composed of vectors that capture the semantic relationships and the different context of use that a token can have. Nowadays, most of the word embedding techniques are based on neural network architectures to help capture these semantics.

The word embeddings used in this work are:

Word2Vec It is word embedding algorithm proposed by Mikolov et al. (2013a), which makes use of unsupervised learning to learn the word associations from a text corpus. As it can be seen in Figure 2.5, Word2vec uses two model architectures to capture a distributed representation of words: (1) continuous bag-of-words (CBOW), the faster model to train, that predicts the current word from a given context window and, (2) continuous skip-gram that predicts the surrounding context window of words given the current word.

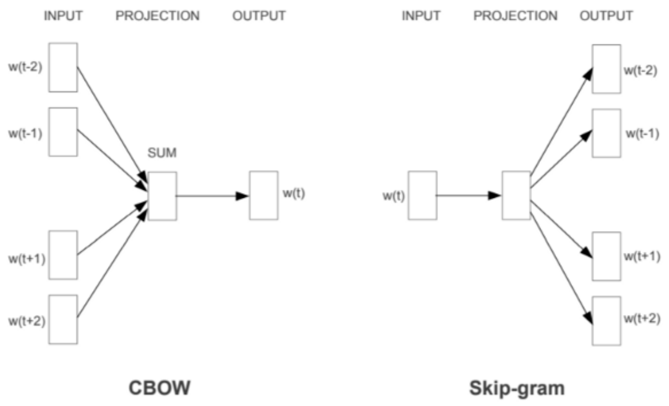


Figure 2.5: Word2Vec models (Mikolov et al., 2013b)

Global vectors Better known as GloVe (Pennington et al., 2014), it is a model that through unsupervised learning learns to capture the distributed representation of the words. The training process is performed on aggregated global word-word co-occurrence statistics from a given corpus, which results in linear substructures of the word vector representation (Pennington et al., 2014). In other words, the training aims to learn word vectors such that the logarithm of the word's probability equals their dot-product[1].

Spacy This word vector is proposed by Honnibal and Montani (2017) and developed by Spacy[2] an open-source library for NLP tasks. The data set used to train this word vector are the GloVe Common Crawl[1] and OntoNotes-5[3]

[1]https://nlp.stanford.edu/projects/glove/
[2]https://spacy.io/
[3]https://catalog.ldc.upenn.edu/LDC2013T19

Designed to be applied in named-entity recognition (NER) tasks, it makes use of convolutional layers with residual connections and maxout non-linearity to learn the word vector representations. It obtains a model with better efficiency in terms of balance, size and accuracy than the solutions provided by bidirectional LSTM models (Honnibal and Montani, 2017).

2.4 Auto-encoders

Auto-encoders are a very popular architecture for neural networks. With the sole objective to copy its input data to its output[4].

The components of its architecture are the encoder and the decoder. During the training time the encoder learns to encode the original space into a lower dimensional latent space. Then, the decoder takes this latent space as input and performs a reconstruction to the original space.

Formally, an auto-encoder can be denoted by the combination of the encoder function f and the decoder function g, where given input x, the corresponding latent space h and the reconstruction result as x'. We have $h = f(x)$ and $x' = g(h)$ that is similar to $x' = g(f(x))$

The reduced dimension of the latent space h is due to the fact that the model gives priority to learning the most salient features of input x. This ability to reduce input space makes auto-encoders suitable for tasks such as "feature learning" (Shi et al., 2015; Cao et al., 2020) or "dimension reduction" (Wang et al., 2012, 2016).

Popular extensions of this architecture are Variational auto-encoders (VAE) (Kingma and Welling, 2013) and Wasserstein auto-encoders (WAE) (Tolstikhin et al., 2017).

2.4.1 Wasserstein auto-encoders (WAE)

WAE models are proposed as an alternative to the VAE models, given that studies has proven that VAE models tend to suffer from the so-called *posterior collapse* problem (Shen et al., 2018; Zhao et al., 2017).

WAE models makes use of Wasserstein distance as regularizer which leads to encoded the training distribution to match the prior, as opposed to the KL-divergence term used in VAEs.

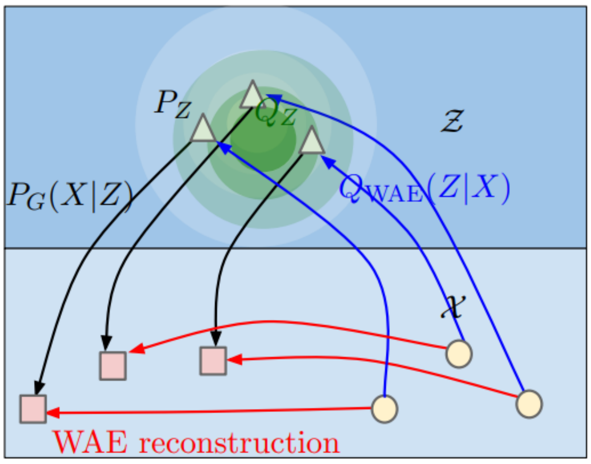


Figure 2.6: WAE reconstruction cost (Tolstikhin et al., 2017).

[4]https://www.tensorflow.org/tutorials/generative/autoencoder

WAE models need to minimize two terms: (1) the reconstruction cost and (2) the regularizer penalizing discrepancy between the prior distribution P_Z and distribution produced by the encoder Q. WAE forces the continuous mixture of the encoded latent space Q_Z to match P_Z, the green zone in Figure 2.6. This behaviour produces that latent spaces of different examples get a better chance to stay far from each other, leading to better reconstruction cost (Tolstikhin et al., 2017).

There is another alternative to minimize the reconstruction cost, that is through adversarial learning proposed by Arjovsky et al. (2017). That will be later reviewed in section 3.1.1.

2.5 Generative Adversarial Networks

Generative Adversarial Networks (GANs) is a deep learning framework based on the minimax two-player game that has been proposed by Goodfellow et al. (2014).

It is composed of two neural networks, a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$ that form of a zero-sum game. And, the training process is perform simultaneously for both networks.

Formally, GANs have been defined as:

$$\underset{G}{min}\,\underset{D}{max}\,V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log\,(1-D(G(z)))] \tag{2.3}$$

where we can denote p_g as the generator's distribution over the data $x \in X$. The prior on the input noise variables is denoted as p_z. $G(z;\theta_g)$ represents the mapping to data space, where the generator is a differentiable function represented by a multi-layer perceptron with parameters θ_g. And, the second perceptron is defined as the discriminator $D(x;\theta_d)$ that outputs a single scalar value. $D(x)$ represents the probability that x was detected as a fake or real sample. The discriminator D is trained to maximize the probability to detect the correct label to both, the real examples and the ones generated by generator G. Generator G is trained to minimize the $log(1-D(G(z)))$. Both discriminator and generator follows the two player minimax game with the given value function $V(G,D)$ (Goodfellow et al., 2014). Therefore, forming equation 2.3.

Figure 2.7 shows a graphical representation of the training process in the GANs framework.

Overall, GANs are based on training in a “indirect” way through the discriminator D, which in turn is also updated dynamically. This makes the generator G to be trained to fool the discriminator rather to minimize the distance of a regularizing function. Which enables the model to learn in an unsupervised way.

2.6 Transformers

Recurrent neural networks are very popular in the NLP field, being applied in a large number of tasks. They remain as the way to go in many state-of-the-art approaches. But just as their nature allows them to process each token in a particular timestep. This in turn, does not allow them to parallelize the training process. Therefore, they have long training times and they are not able to take advantage of all the power from a GPU.

Vaswani et al. tackle all this inconveniences with its proposed model called Transformer (Vaswani et al., 2017). A model based on an encoder-decoder architecture and on attention mechanisms. An

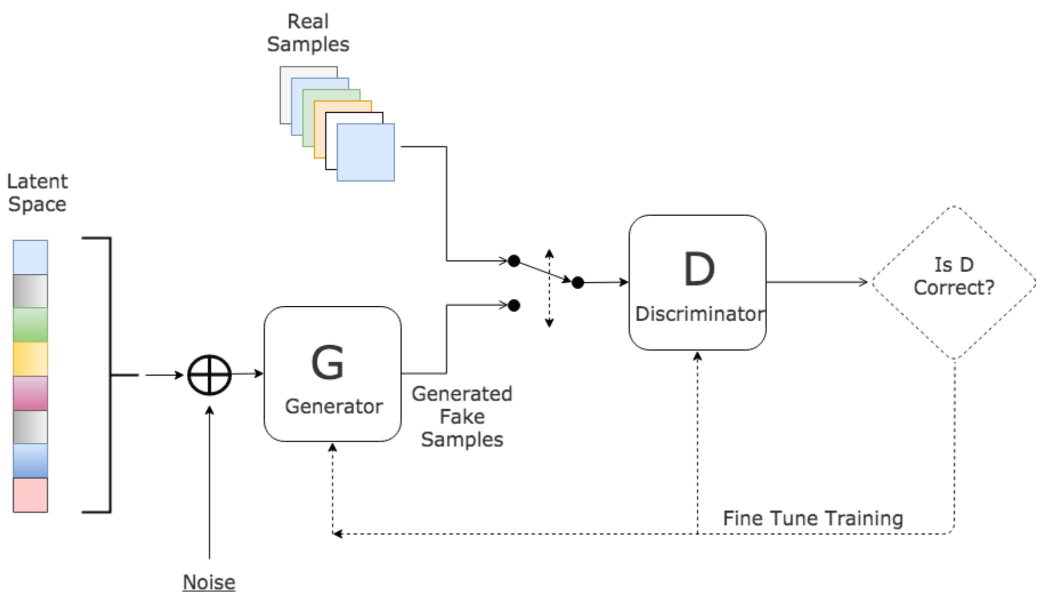


Figure 2.7: GAN architecture [Source].

important characteristic of Transformers is that input data does not longer required to be fed in sequential mode. This disables the necessity to process the beginning of a sentence before the end. Therefore, the Transformer allows for much more parallelization than RNNs what reduces the training time. Also, the model can learn long-range dependencies, a challenging task for most RNN architectures.

As mentioned, Transformer architecture does not require input data in sequential mode. Nevertheless, the model requires certain information so that it ca retain the relative position of the words. *Positional encoding* is the solution proposed in Vaswani et al. (2017), where a positional encoding vector is included to the embedding vector. Embeddings vector does not keep the positional information but it encodes the words in small clusters based on their similarity. Vaswani et al. provided the following formula for calculating the positional encoding:

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{\frac{2i}{d_{model}}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{\frac{2i}{d_{model}}}) \end{aligned} \tag{2.4}$$

In Figure 2.8 can be seen the Transformer architecture. On left is the encoder part of the architecture and on the right, the decoder part. In both parts, the initial layer is the *Multi-Head attention*. This layer contains 8 heads where each of them consist of four parts: (1) linear layers for query Q, key K and value V split into the different heads, (2) scaled dot-product attention that enables the model to attend different words at different positions, (3) the concatenation of the heads and (4) the linear output layer. The attention function perform at each dot-product attention is

$$Attention(Q, K, V) = softmax_k(\frac{QK^T}{\sqrt{d_k}})V \tag{2.5}$$

To denote that in the decoder part, the Multi-head attention layer, makes use of the *look-ahead mask* function that is used to mask future tokens in a sequence to prevent positions from attending to subsequent positions.

Also, one of the key factors of the Transformer models is the use of residual connections that are connected around each of the sub-layers. Each of the sub-layers output is added to a residual connection value, to later be normalized.

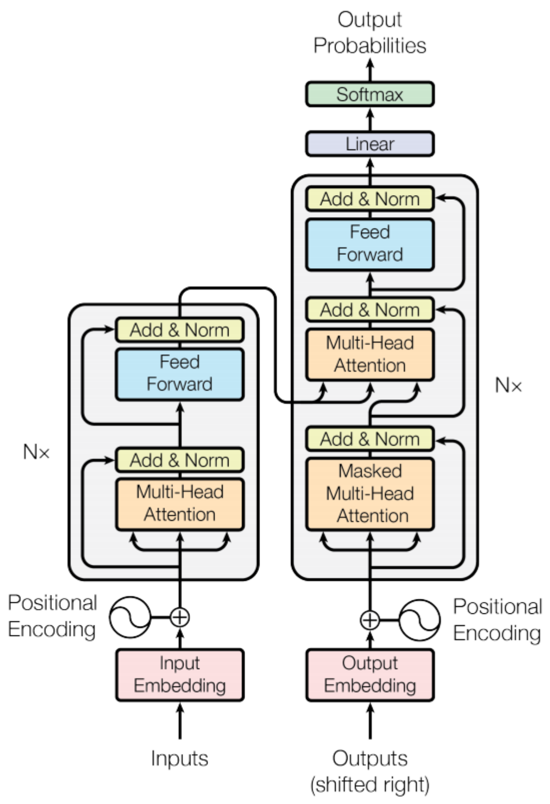


Figure 2.8: The Transformer - model architecture (Vaswani et al., 2017).

Then, the normalized outputs of the Multi-head attention layers are later used as input to the *point wise* feed-forward network, that basically consists of two linear layers with an activation function in between.

The Transformer follows the same principle as a standard sequence-to-sequence model with attention mechanism (Bahdanau et al., 2014; Luong et al., 2015). Where the input sequence passes through the encoder layers, producing an output for each token, that later is passed to the decoder part of the model to predict the next word.

Vaswani et al. evaluated the Transformer model on neural language translation tasks where surprised the research community by achieving state-of-the-art results.

BERT: It is model trained to be a language representation (Devlin et al., 2018). It is designed to pre-train bidirectional representations from unlabeled data, conditioned on the left and right context. Bert is a well-know model within the NLP community, where researchers used it to apply in various tasks such as text summarization (Liu and Lapata, 2019) or to develop more complex models like SciBert (Beltagy et al., 2019), where they fine-tune Bert for sequence tagging, sentence classification and dependency parsing tasks.

2.7 Brief Literature overview

As it was stated in section 2.1, the goal of a dialogue response generation system is to generate response utterance from a given query utterance in a coherent, informative and diverse way while considering the dialogue history.

Neural dialogue response generation is an active field of research where the NLP community explores possible solutions by applying deep learning techniques. The initial approaches adopted the auto-encoder architecture that was applied under the sequence-to-sequence learning (Sutskever et al., 2014; Cho et al., 2014). They were first introduced for machine translation tasks and soon after, they were implemented for dialogue response tasks with certain level of success (Sordoni et al., 2015).

Subsequent investigations showed that auto-encoders based models suffered from the so-called *safe response* problem, where utterances like *"i do not know"*, *"that's ok"* or *"no, thank you"* were frequently generated. To tackle this problem, Sato et al. (2017) developed the idea of incorporating various types of situations into the training data and Xing et al. (2017) proposed to include information regarding the topic within the model to generate more informative and diverse responses.

Currently, VAE (Kingma and Welling, 2014) models are one of the most popular architectures for dialogue response generation. Although, studies have shown that VAE models suffer from the so-called *posterior collapse* problem, where the decoder within the model ignores the latent variables. The latter leads to the degradation of the model to a vanilla RNN model. Nevertheless, researches presented models, like VHRED proposed by Serban et al. (2016) to tackle the problem by modeling the encoded utterances of a dialogue through the use of latent variables at multiple levels. Also, Shen et al. (2018) proposed a collaborative CVAE model which samples the latent variable with a Gaussian noise transformation. The DialogWAE model intends to overcome the VAE limitations by training the model with adversarial learning to sample the latent space.

GANs (Goodfellow et al., 2014) architectures turned out to be difficult to adapt to NLP tasks, all due to the non-differentiable nature of language tokens (Xu et al., 2017; Shen et al., 2017). Ian Goodfellow himself proposed some ideas to combine GANs with natural language[5] and he was even the co-author of Fedus et al. (2018) where it was proposed to introduce an actor-critic conditional GAN that fills in the missing token conditioned on the surrounding context. Yu et al. (2017) proposed a model that induces a sequence-to-sequence model to learn through adversarial learning, by combining GAN with reinforcement learning (RL) so that the data generator is modeled as a stochastic policy in RL, bypassing the generator differentiation problem by performing a gradient policy update. However, studies proved that training with RL can lead models to be unstable due to the high variance of the sampled gradient (Shen et al., 2017).

[5] `https://www.reddit.com/r/MachineLearning/comments/40ldq6/generative_adversarial_networks_for_text/`

Chapter 3

Experimental Setup

In this chapter, the novel approach taken by the DialogWAE model is described. This model reported results that overcomes the state-of-the-art approaches. Moreover, this chapter also includes a description of all the required components of the training steps for each of the experiments.

3.1 Model Architecture

3.1.1 DialogWAE Model

In this section, the reasoning behind DialogWAE model is disclosed. Next, each of the model's component is going to be described, to later talk about the data set with which the model is trained and the evaluation metrics used to assess model experimentations. Finally, we describe the implementation of the model and the training steps took for each experiment.

We begin by defining and formulating the problem that DialogWAE model faces. We have a dialogue context c fulfill with u_{k-1} utterances and a response utterance $x = u_k$ which represents the next utterance to predict. The goal is to calculate the conditional distribution $p_\theta(x|c)$.

Given that x and c are discrete tokens, it is challenging to find a way to pair them directly. So, the continuous latent variable z is introduced, a variable that represents the high-level representation of the response utterance. To generate a response we need first to sample a latent variable z from the distribution $p_\theta(z|c)$ on the latent space $\mathcal{Z}$ and then the response utterance x is decoded from the variable z with $p_\theta(x|z, c)$. It follows that the likelihood of a response is

$$p_\theta(x|c) = \int_z p(x|c,z)\,p(z|c)\,d_z \tag{3.1}$$

As the log probability cannot be computed given the impossibility to marginalize z, we can only try to approximate the posterior distribution of z as $q_\theta(z|x,c)$ by calculating it with a neural network called *recognition network*. Using the approximate posterior, we compute the evidence lower bound:

$$\begin{aligned} log\,p_\theta(x|c) &= log\int_z p(x|c,z)\,p(z|c)\,d_z \\ &\geq l(x,c) = E_{z\sim\, q_\theta(z|x,c)}[log\,p_\psi(x|c,z)] - KL(q_\theta(z|x,c)\,||\,p(z|c)) \end{aligned} \tag{3.2}$$

where $p(z|c)$ is the prior distribution of z given context c, that is modeled by the *prior network*.

Conditional Wasserstein auto-encoders

Thanks to the work of Makhzani et al. (2015); Tolstikhin et al. (2017) and Zhao et al. (2018), it is known a possible solution to model the distribution of z through adversarial learning from the latent space. The idea is to sample the prior and posterior over the latent variables from random noise ϵ with the help of neural networks. The prior sample $\tilde{z} \sim p_\theta(z|c)$ would be generated by generator $\mathcal{G}$ from the random noise $\tilde{\epsilon}$ and the posterior sample $z \sim q_\phi(z|c,x)$ is generated by generator $\mathcal{Q}$ from the random noise ϵ. Both random noises ϵ and $\tilde{\epsilon}$ are sampled from Gaussian distributions whose mean and covariance are calculated from c by a feed-forward neural network. The *prior network* looks like:

$$\tilde{z} = G_\theta(\tilde{\epsilon}),\ \tilde{\epsilon} \sim \mathcal{N}(\epsilon; \tilde{\mu}, \tilde{\sigma}^2 I), \begin{bmatrix} \tilde{\mu} \\ log\, \tilde{\sigma}^2 \end{bmatrix} = \tilde{W} f_\theta(c) + \tilde{b} \tag{3.3}$$

The *posterior network* looks like:

$$z = Q_\phi(\epsilon),\ \epsilon \sim \mathcal{N}(\epsilon; \mu, \sigma^2 I), \begin{bmatrix} \mu \\ log\, \sigma^2 \end{bmatrix} = W g_\phi(\begin{bmatrix} x \\ c \end{bmatrix}) + b \tag{3.4}$$

where $f_\theta(\cdot)$ and $g_\phi(\cdot)$ are feed-forward networks.

The goal is to solve the following problem:

$$\min_{\theta,\phi,\psi} - E_{q_\phi(z|x,c)}\, log\, p_\psi(x|z,c) + W(q_\phi(z|x,c) \,||\, p_\theta(z|c)) \tag{3.5}$$

where we try to minimize the divergence of $p_\theta(z|c)$ and $q_\phi(z|c,x)$ while maximizing the log-probability to reconstruct a response utterance from the latent variable z. $p_\psi(x|z,c)$ is the decoder and $W(\cdot||\cdot)$ is the Wasserstein distance between the normal distributions (Arjovsky et al., 2017).

The whole architecture of the model can be seen in Figure 3.1. The *utterance encoder*, *context encoder* and the *response decoder* are all RNN. The *utterance encoder* converts each of the dialogue utterances into a vector of real values. The *context encoder* takes as input the concatenation of a dialogue utterance with its respective floor value, value that represents 1 if the utterance comes from the speaker, otherwise is 0.

During the generation of a response, the model samples a random noise $\tilde{\epsilon}$ from the *prior network* transforming the context dialogue c through matrix multiplications into the mean and covariance. Then, the generator $\mathcal{G}$ takes as input this transformed random noise and generates the latent variable $\tilde{z}$. Finally, the *decoder* takes the generated $\tilde{z}$ and produces a response.

During the training of the auto-encoder phase, the model infers a posterior distribution from the latent variable $z \sim q_\phi(z|c,x)$ given the context dialogue c and the response utterance x. The *recognition network* transforms the concatenation of x and c through matrix multiplications into the mean and covariance. ϵ is the Gaussian noise sampled from the recognition network with the re-parametrization trick (Kingma and Welling, 2013; Doersch, 2016). Then, generator $\mathcal{Q}$ takes as input this Gaussian noise ϵ and transforms it into a sample of the latent variable z by using a feed-forward network. Finally, the response decoder calculates the reconstruction loss from:

$$\mathcal{L}_{rec} = -E_{z=Q(\epsilon),\, \epsilon \sim RecNet(x,c)}\, log\, p_{\psi}(x|c,z) \tag{3.6}$$

To complete the GAN architecture, a discriminator $\mathcal{D}$ is implemented with a feed-forward network taking as input the concatenation of z and c and outputting a real value. The discriminator tries to match the approximate posterior with the prior distributions of z by distinguishing the prior from the posterior samples. The discriminator $\mathcal{D}$ is trained by minimizing the following loss:

$$\mathcal{L}_{disc} = E_{\epsilon \sim RecNet(x,c)}[D(Q(\epsilon), c)] - E_{\tilde{\epsilon} \sim PriNet(c)}[D(G(\tilde{\epsilon}), c)] \tag{3.7}$$

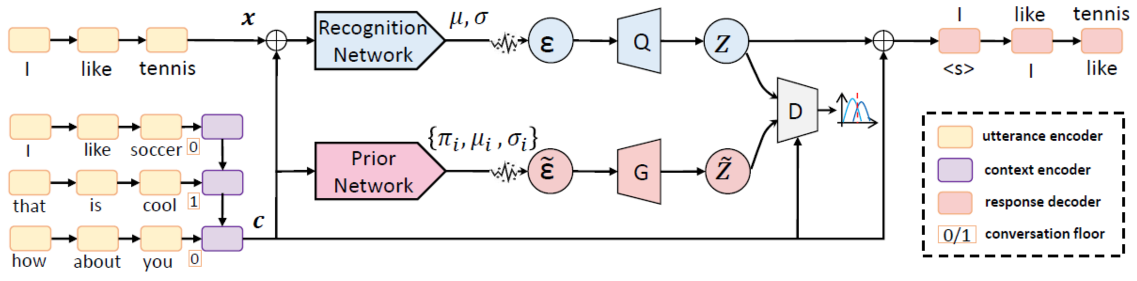


Figure 3.1: Architecture of DialogWAE model (Gu et al., 2018).

From now on, this DialogWAE model is considered in this work as the base model when comparing with the different experiments.

3.2 Dataset

The DialogWAE model and its different modifications would be trained and evaluated on the DailyDialog (Li et al., 2017) dataset. DailyDialog is a high-quality multi-turn dialog dataset based on human-written English language that reflects daily communication, covering several topics about daily life. It is manually labelled towards communications intention and emotion information, making it less noisy from irrelevant utterances.

Overall, DailyDialog contains 13 118 multi-turn dialogues with an average speaker turns of 8 per dialogue and 15 tokens on average per utterance.

3.3 Evaluation metrics

The following evaluation metrics are used to assess the performance of DialogWAE model and the different experiments with word embeddings. The metrics are computed for each query utterance given 10 responses sampled from the model outputs and then, the average result of it is reported.

BLEU score: The BLEU (BiLingual Evaluation Understudy) score is a metric widely use in the NLP field. It was first developed by Papineni et al. (2002) to evaluate neural machine translations systems but rapidly adopted in different NLP tasks. It measures the overlaps of the *n-grams* between the reference and the sampled hypothesis; the higher the value, the better the performance of the model.

For this work, we computed the 3-*gram* BLEU scores, smoothing the generated responses using smoothing technique 7 (Chen and Cherry, 2014). In addition, from the given 10 sampled responses was computed the n-gram recall (R) and n-gram precision (P) following Zhao et al. (2017). The F_1 score from n-gram R and P was computed and included in the report.

Furthermore, we must noticed that after implementing the base model and performing a few training runs, the model was performing differently from expected as we obtained higher BLEU scores that the ones reported in Gu et al. (2018). As we could not review the quality of the numerous generated responses, we started a deep investigation, checking all the details of our implementation. Eventually, we detected the source of the error, located in the version of the NLTK[1] library the authors Gu et al. used. Indeed, they used the version 3.2.5, where a bug in the smoothing function 4[2] was reported.

Later, we reported this bug to the authors and asked to release an updated version of their article. Authors answered us that they were aware of the bug but did not plan to release any other version of their model. Thus, the results reported in Gu et al. (2018) are not included in this work and we report only the results obtained during our own training passes. Through our investigation, we concluded that this is not an isolated event as there exists a small number of published articles that, aware or not of the NLTK library bug, reported BLEU scores with the problematic version 3.2.5. A small list of articles, collected by us is shown in the appendix A.1.

BOW embedding score: The Bag-of-Words (BOW) embedding metric consists of calculating the cosine similarity of the reference and the sampled hypothesis. There are three different ways to compute this metric. The first one, called *Average*, consists of computing the cosine similarity between the averaged word embeddings in the two (reference and hypothesis) utterances (Mitchell and Lapata, 2008). In the second one, *Greedy*, we greedily match the words of the two utterances based on the computed cosine similarities and report the average obtained scores (Rus and Lintean, 2012). And the third one, *Extrema*, reports the cosine similarity between the largest values from the word embeddings presented in the two utterances (Forgues et al., 2014). The maximum BOW embedding score from the 10 sampled utterances, at each evaluation step, for each of the experiments, is reported.

3.4 Methodology

Overall, the experiments that were carried out in this work are the following:

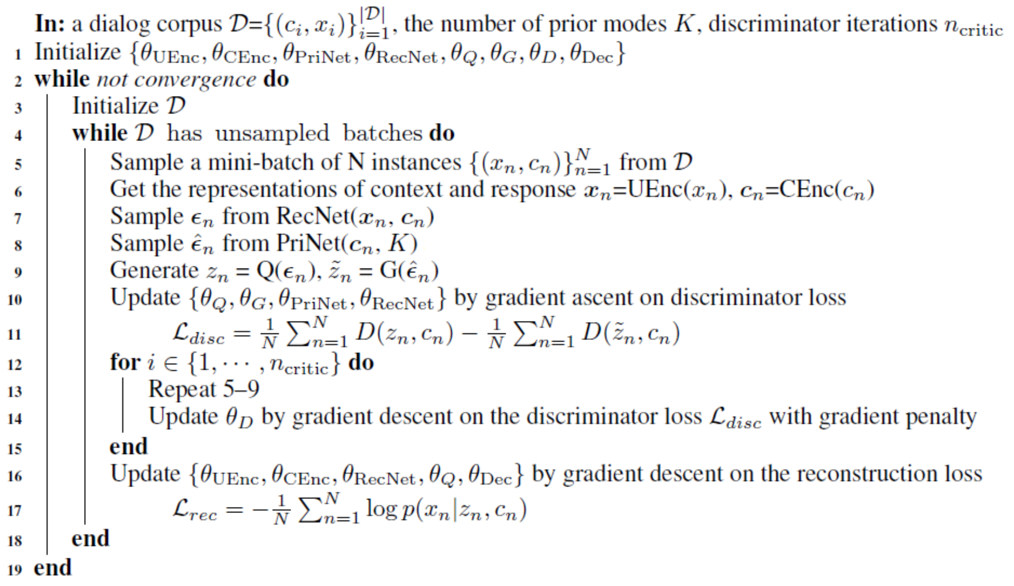
- DialogWAE model: it is considered as the base model and with which the other models are later compared. This base model already includes GloVe as word vector.
- Spacy word vector: the DialogWAE model uses Spacy as the pre-trained word representations.
- Own-trained word vector: the base model is trained using a pre-trained word vector from the input data set.
- No pre-trained word vector: the base model initializes the word embedding layer with random values.
- Bert as word embeddings: the base model is trained using the outputs of the Bert model.

[1] `https://www.nltk.org/`

[2] `https://github.com/nltk/nltk/issues/2341`

3.5 Implementation and training

In this section, the details of the training steps of the DialogWAE model are described. Also, the different adopted training steps performed for the different experiments are included in this section.



```
   In: a dialog corpus $\mathcal{D}=\{(c_i, x_i)\}_{i=1}^{|\mathcal{D}|}$, the number of prior modes $K$, discriminator iterations $n_{\text{critic}}$
1  Initialize $\{\theta_{\text{UEnc}}, \theta_{\text{CEnc}}, \theta_{\text{PriNet}}, \theta_{\text{RecNet}}, \theta_Q, \theta_G, \theta_D, \theta_{\text{Dec}}\}$
2  while not convergence do
3      Initialize $\mathcal{D}$
4      while $\mathcal{D}$ has unsampled batches do
5          Sample a mini-batch of N instances $\{(x_n, c_n)\}_{n=1}^{N}$ from $\mathcal{D}$
6          Get the representations of context and response $x_n$=UEnc($x_n$), $c_n$=CEnc($c_n$)
7          Sample $\epsilon_n$ from RecNet($x_n$, $c_n$)
8          Sample $\hat{\epsilon}_n$ from PriNet($c_n$, $K$)
9          Generate $z_n$ = Q($\epsilon_n$), $\tilde{z}_n$ = G($\hat{\epsilon}_n$)
10         Update $\{\theta_Q, \theta_G, \theta_{\text{PriNet}}, \theta_{\text{RecNet}}\}$ by gradient ascent on discriminator loss
11             $\mathcal{L}_{disc} = \frac{1}{N}\sum_{n=1}^{N} D(z_n, c_n) - \frac{1}{N}\sum_{n=1}^{N} D(\tilde{z}_n, c_n)$
12         for $i \in \{1, \cdots, n_{\text{critic}}\}$ do
13             Repeat 5–9
14             Update $\theta_D$ by gradient descent on the discriminator loss $\mathcal{L}_{disc}$ with gradient penalty
15         end
16         Update $\{\theta_{\text{UEnc}}, \theta_{\text{CEnc}}, \theta_{\text{RecNet}}, \theta_Q, \theta_{\text{Dec}}\}$ by gradient descent on the reconstruction loss
17             $\mathcal{L}_{rec} = -\frac{1}{N}\sum_{n=1}^{N} \log p(x_n | z_n, c_n)$
18     end
19 end
```

Figure 3.2: Training process (UEnc: utterance encoder; CEnc: context encoder; RecNet: recognition network; PriNet: prior network; Dec: decoder) K=1, ncritic=5 in all experiments (Gu et al., 2018).

The learning process of the different models was conducted in epochwise form, training the models until convergence was reached. The training is described in Figure 3.2. The whole process was performed in two phases: (1) the auto-encoder phase where the reconstructed loss from the decoded responses is minimized and (2) the GAN phase which focuses on minimizing the Wasserstein distance between the prior and the posterior distributions over the latent variables.

For the RNN encoders and decoders, the GRU unit (Cho et al., 2014), described in section 2.2.1, was used. The utterance encoder is a single bidirectional GRU layer (Schuster and Paliwal, 1997) while the context encoder and decoder layers are unidirectional GRU layers. All the RNN layers have 300 hidden units in each direction. The prior and the recognition networks are typical feed-forward networks with 200 hidden units.

Generators $\mathcal{Q}$ and $\mathcal{G}$, together with the discriminator $\mathcal{D}$, are 3 layers of feed-forward networks with ReLU as activation function (Nair and Hinton, 2010) and respectively 200, 200 and 400 hidden units. All initial weights for the feed-forward networks were initialized from a uniform distribution of ± 0.02. Also, we performed a gradient penalty when training the discriminator $\mathcal{D}$ as described in Gulrajani et al. (2017) with the hyper-parameter $\lambda = 10$. The maximum number of utterances per dialogue is set to 10 with a maximum utterance length of 40.

All the models were implemented with Pytorch 1.7.1[3]

[3]https://pytorch.org/docs/stable/index.html

Training details of each experiment

All the experimented models were trained in mini-batches of 32 dialogue examples following an end-to-end learning process. During the auto-encoder phase, the models were trained using SGD activation function with an initial learning rate of 1.0. This learning rate has a decay of 40% every 10 epoch. On the other hand, during the GAN phase, the models are updated using RMSprop (Tieleman and Hinton, 2012) with fixed learning rate of 0.0005 for the generator and 0.0001 for the discriminator.

Base model: This model is the version described in section 3.1.1 (Gu et al., 2018). Therefore, this model is considered as the control model and serves as a starting point to describe most of the training processes.

Data pre-processing - When authors of the Dailydialog dataset released the data, they also released a script that builds and splits the data information into train, valid and test sets with a ratio of 10:1:1 respectively. The full dataset of 13 118 conversations was split into 10 930 conversations for the training set, and 1 093 for each valid and test sets.

For any NLP tasks the creation of a vocabulary is a compulsory task. To achieve this first, we applied some pre-processing techniques to our data. As each conversation in the dataset is merged into a single line using as separator the word "*__eou__*" (end-of-utterance), we split the conversation at each appearance of this separator. Afterwards, the whole utterance was transformed to the lower case so that, words like "church" and "Church" received the same token, as they represent the same domain. Then, the lower case sentence was tokenized by using the NLTK Word-Punctuation Tokenizer[4], turning each utterance in a set of tokens. Also, as part of the pre-processing, the tokens <*s*> and </*s*> were added to each utterance, indicating the start-of-utterance and end-of-utterance respectively. These tokens are important given that they allow the model to recognize whether an utterance is complete or not.

Finally, given that we are going to feed several conversations to the model when training, it is important for the model that we point out when a new conversation begins. This enables the learning process for each conversation to re-establish. The token that indicates the beginning of a new conversation is <*d*>, thus forming the utterance <*s*> <*d*> </*s*>. The latter is added at the top of each conversation.

The resulting conversation looks like the following:

Utterance 1: '<s>' '<d>' '</s>'

Utterance 2: '<s>' 'overseas' 'operator' '.' '</s>'

Utterance 3: '<s>' 'i' 'would' 'like' 'to' 'make' 'a' 'collect' 'call' 'to' 'taipei' ',' 'taiwan' ',' 'please' '.' '</s>'

Utterance 4: '<s>' 'your' 'name' ',' 'please' '.' '</s>'

Utterance 5: '<s>' 'tim' 'chen' '.' '</s>'

Utterance 6: '<s>' 'what' ''' 's' 'the' 'number' ',' 'please' '.' '</s>'

Utterance 7: '<s>' 'the' 'area' 'code' 'is' '2' ',' 'and' 'the' 'number' 'is' '2367' '-' '9960' '.' '</s>'

Then, we used the training set to build actually the vocabulary, as it is the most extensive set regarding to the number of tokens. The raw number of words is 1 469 757 while the number of unique tokens is 17 716. Therefore, to create the vocabulary, we sort first the tokens according to their frequency and then, set a limit to the first 10 000 tokens. Despite this cut-off eliminated 7 716 unique tokens, this

[4]https://www.nltk.org/api/nltk.tokenize.html

limitation actually removed only 9 931 tokens occurrences given that the deleted tokens had a frequency of only 1 or 2 words. Therefore, the Out-of-Vocabulary (OOV) rate is 0.006757 indicating that we eliminated rare tokens that did not bring any useful information to the model learning process.

The resulting vocabulary includes 10 002 tokens given that the auxiliary tokens <*pad*> and <*unk*> were added to the vocabulary. The whole set of auxiliary tokens are the following:

- <*s*> - represented by 3, means the start-of-utterance.
- </*s*> - represented by 4, means the end-of-utterance.
- <*d*> - represented by 21, means the beginning of a new conversation.
- <*pad*> - represented by 0, used to accommodate the utterances of different length sizes.
- <*unk*> - represented by 1, means the unknown token that might appear in some utterances due to its rareness and was eliminated during the cut-off.

Once the vocabulary is built, we proceeded to the encoding of all the utterance's tokens with their respective value. Taking the previous conversation example, the encoded conversation would look as follow:

Utterance 1: [3 21 4]

Utterance 2: [3 1671 3385 2 4]

Utterance 3: [3 6 60 36 11 113 12 1675 182 11 3434 5 1156 5 75 2 4]

Utterance 4: [3 30 188 5 75 2 4]

Utterance 5: [3 2489 1792 2 4]

Utterance 6: [3 26 10 14 9 248 5 75 2 4]

Utterance 7: [3 9 794 1654 16 419 5 15 9 248 16 1 61 1 2 4]

Let us highlight that the <*unk*> token replaced the sequences of numbers (namely '2367' and '9960') found in the utterance 7. This is a clear example of tokens that were suppressed during the cut-off.

Continuing with data pre-processing, each utterance in a conversation must be assigned with its respective *floor* value. As mentioned in section 3.1.1, the *context encoder* layer takes as input the concatenation of an encoded utterance and its floor value. This value enables the model to learn to differentiate the type of utterance provided, either a question or a response, to generate later an output in concordance with the flow of the conversation.

An example of the conversation resulting from all the pre-processing steps is shown below. The floor value is now added to the encoded utterance array, forming a *Tuple*[5].

Utterance 1: ([3 21 4], 0)

Utterance 2: ([3 1671 3385 2 4], 1)

Utterance 3: ([3 6 60 36 11 113 12 1675 182 11 3434 5 1156 5 75 2 4], 0)

Utterance 4: ([3 30 188 5 75 2 4], 1)

Utterance 5: ([3 2489 1792 2 4], 0)

[5]`https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences`

Utterance 6: ([3 26 10 14 9 248 5 75 2 4], 1)

Utterance 7: ([3 9 794 1654 16 419 5 15 9 248 16 1 61 1 2 4], 0)

Data loader - The way how data is fed to the model is a crucial task by itself. Indeed, to determine the most optimal way, with minimum padding requirements, comes very useful during the training of any NLP task, especially models that takes as input variables text data sequences.

Nowadays, most of the trending models like xlnet, bert, roBerta, gpt2, etc., achieve their outstanding results by feeding their models with a type of *sequence bucketing* (Khomenko et al., 2016), which is an algorithm implementation that tries to find the most optimal way to sort the data sequences according to their length. A visual representation of the sequence bucketing can be seen in Figure 3.3.

DialogWAE model has a similar implementation: it reshuffles the data and filters the sequences to keep only the ones with length equal or less than the maximum utterance length value established for the model. With this sequence size threshold the data has now the following length sequences:

	Max length	**Average length**
train set	36	8.84
valid set	32	9.06
test set	27	8.74

Table 3.1: Sequence length for the different sets after filtering out utterances greater than maximum utterance size.

The *batch_size* was set to 32. Therefore, the training set generated 2374 batches and the validation set 225 batches to be used in the training process. Each batch was also padded to the longest sequence, using the <*pad*> token.

Load word embeddings - The authors of DialogWAE model used a pre-trained word vector representation provided by the Stanford NLP Group[6], that is a word vector trained with GloVe algorithm, see section 2.3.1. The data fed to this algorithm was taken from a Twitter corpus with 2 billions of tweets with a total of 27 billions of tokens[1]. The GloVe word vector has a dimension size of 200.

Having the vocabulary already created, the next step consists in looking after the vector representing the word from the GloVe vector. If the word is not present in the pre-trained vector then a vector with normal distribution of size 200 is created. Overall, the word ratio that the pre-trained vector could not cover is 0.032194, for a total of 322 missed words. The complete list of missed words is available in appendix A.2. Finally, the obtained word vector contains the weights from our vocabulary ready to be fed into the embedding layer.

Spacy word vector: This model explores the idea of using another, larger, pre-trained word vector representation to be fed into the embedding layer. The chosen word vector used for this model was built and trained by Spacy[7], a library dedicated to apply NLP techniques at industrial levels. It has a dimension of 300 and it is described in section 2.3.1.

This word vector representation was trained combining the GloVe Common Crawl[1] and OntoNotes-5 datasets to be optimized later for tasks like Part-of-Speech Tags, parser, Named-entity recognition

[6]`https://nlp.stanford.edu/`

[7]`https://spacy.io/models/en`

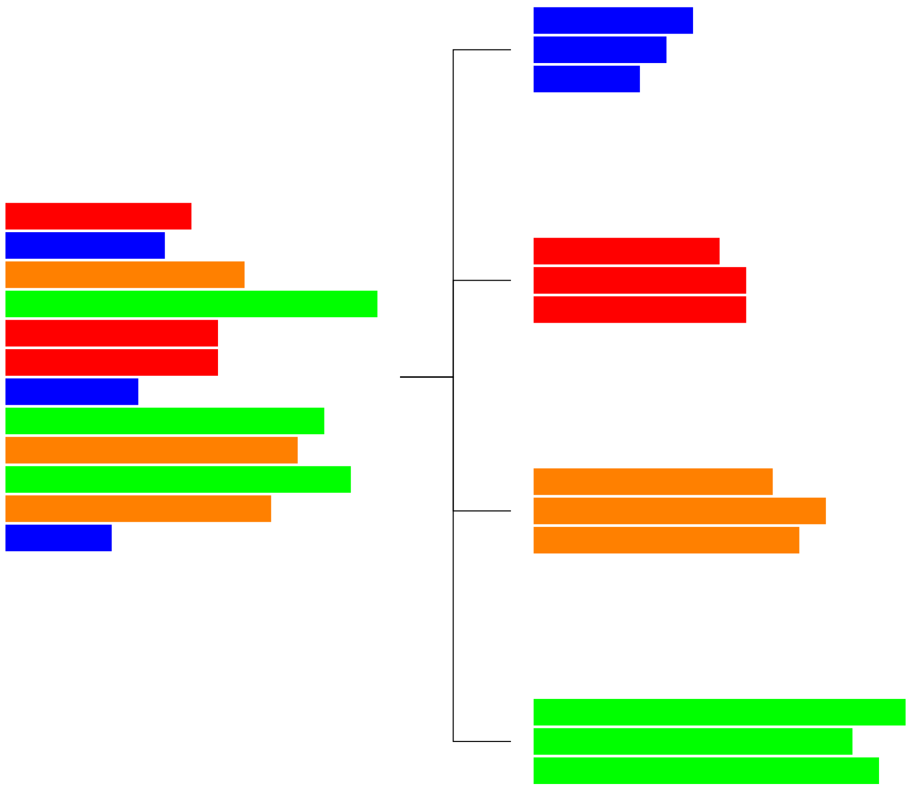


Figure 3.3: An example of sequence bucketing, where each sequence is arranged according to its length.

(NER).

The resulting word vector representation is a very light file of 48mb, of only 684 830 tokens but highly optimized for specific NLP tasks.

Other steps regarding the vocabulary creation, like the pre-processing of the data, data loader or any other step are similar to the base model. Hence, once the vocabulary is created we continue to build the embedding layer's weights by looking for the word representation in the Spacy vector. If the word from the vocabulary cannot be found, then a vector with normal distribution of size 300 is created to fill the missed word. Overall, 51 words were not found in the Spacy vector representing a word ratio of 0.005099, most of the missing words were related to punctuation tokens (e.g. '..', '::', '>'). The complete list can be found in appendix A.2.

Own-trained word vector: In this model the objective is to investigate if a simply word embeddings trained from scratch with our input data is strong enough to achieve results as the base model.

As it was mentioned in Section 2.3.1, the algorithm used to train our own word embedding is Word2Vec. Thanks to the help of the Gensim library Řehůřek and Sojka (2010), this procedure can be easily implemented.

Given that the other word embeddings were trained from millions of different documents or text files. It seemed fair to us to feed the "train, valid and test" datasets to the Word2Vec model, so that it can capture the context of a word in an optimal way.

The data pre-processing procedure for the Word2Vec model, consists on applying the NLTK Word-Punctuation Tokenizer[4] where each word and punctuation symbol is converted to tokens and converting all the sentences into lower case. After tokenizing all the dataset, we reached a total of 1 491 622 tokens, where 18 725 are unique ones.

To train the Word2Vec model the following parameters were used:

`num_features=300` - there is no a firm consensus on the dimension of the word embedding should have, longer word embeddings do not add enough information and smaller ones are not capable to represent the semantics well enough as stated in Mikolov et al. (2013a) hence, a range between 100 or 300 should fit our dataset.

`train_algorithm=‘CBOW’` - “Continuous Bag-of-Words” as we want our word embedding to learn the focus of a word given the surrounding words.

`window_context=7` - the authors of Word2Vec suggested a window size of 5 for CBOW[8] algorithm, but after several runs the value that fits better our dataset is 7.

`min_count_word=2` - the model was setup to ignore words with a total frequency lower than 2. So that, extremely rare words are not taken into account hence, now the vocabulary has 13 147 words. The rest of the hyperparameters of the Word2Vec algorithm remained with their respective default values.

The amount training time of our word embedding was about 0h2m:14s spread over 35 epochs, a relative short time if we compare with the *GoogleNews-vectors-negative300.bin.gz*[8] word embeddings that took days to train in the period it was released. Something to take into account as it will be demonstrated later.

In addition, given that there is not defined way how to measure the quality of a word embeddings for a small vocabulary, this case less than 15 000 words and given that the most cited article Schnabel et al. (2015) only works under assumptions that the word embeddings contains millions of tokens. We decided not to pursue any study about the quality of our trained word embeddings. Nevertheless, we moved forward to do a visual analysis with the help of *t-distributed Stochastic Neighbor Embedding* (T-SNE) algorithm.

The T-SNE algorithm is a nonlinear dimensionality reduction technique that help us visualize high-dimensional data, Maaten and Hinton (2008). For this purpose we made use of the library Scikit-learn[9], where the function is implemented. The new distribution of our trained embedding is now represented in 2 dimensions, see Figure 3.4. This new dimension allow us to observe how the word embeddings formed small clusters with similar context words.

In Table 3.2 and Figure 3.5 can be observed the Top-n most similar words for the weekday *Monday* and in the figure, the zoomed area where it is located the cluster of the weekdays. This analysis was performed several times, computing the cosine similarity score for different tokens and zooming to the area to perform a visual examination as can be observed in Figures 3.6 and 3.7, more results can be found in the Appendix A.3.

[8]https://code.google.com/archive/p/word2vec/
[9]https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

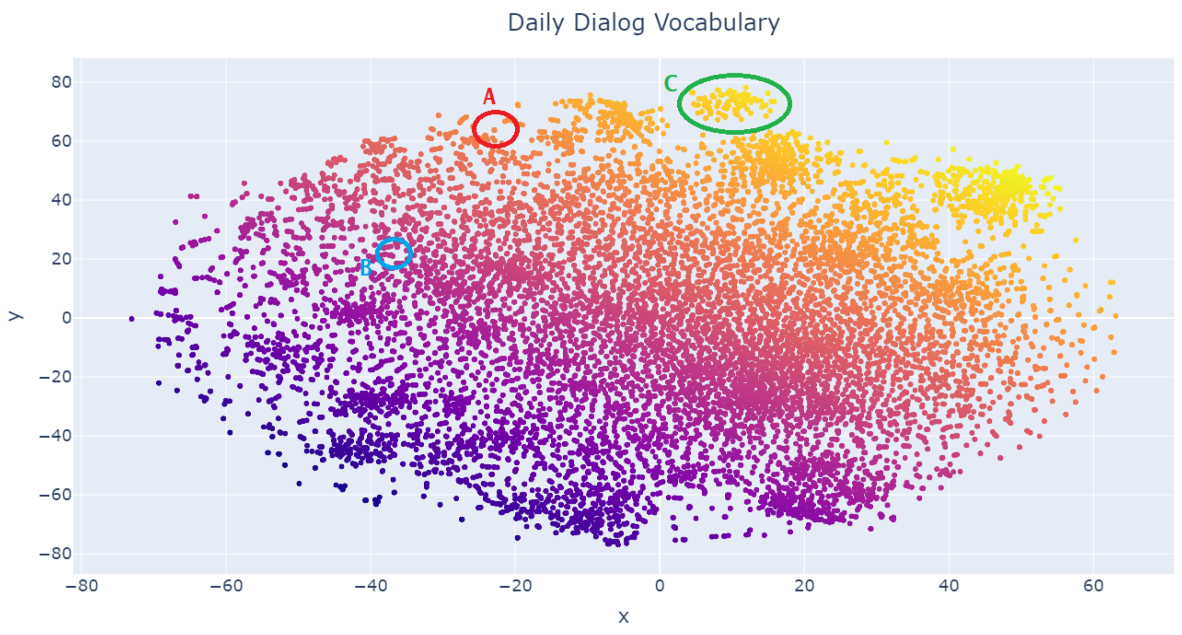


Figure 3.4: T-SNE representation from the trained word embedding, each dot represents a word token. The selected area in circle represent the zoomed area, see Table 3.2.

Weekdays	***Cosine Score***
wednesday	0.7909
tuesday	0.7161
thursday	0.6577
sunday	0.6496
friday	0.6314
saturday	0.5741
october	0.5453
september	0.5451
2nd	0.5005

Table 3.2: Top-9 most similar words of 'monday' weekday word.

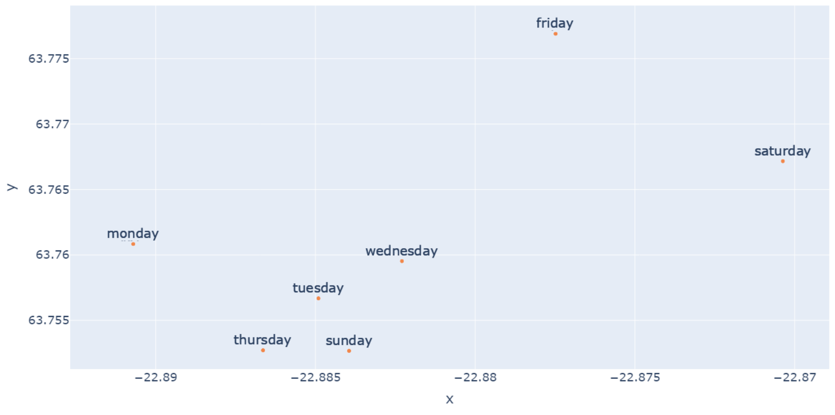


Figure 3.5: T-SNE visualization from the zoomed area 'A', representing the weekdays.

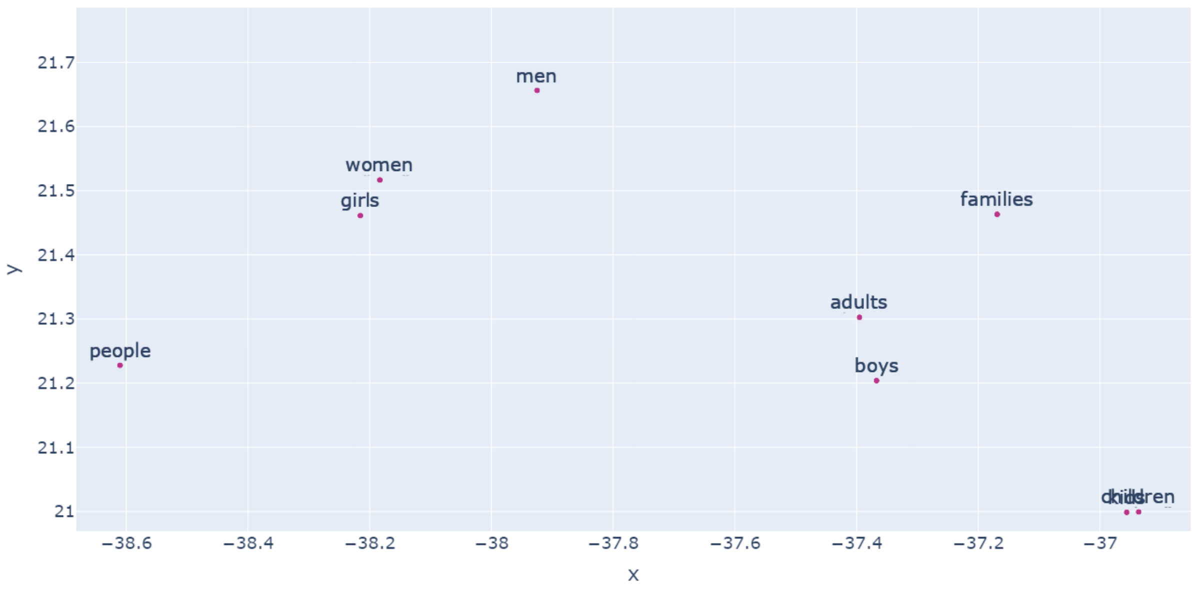


Figure 3.6: T-SNE visualization from the zoomed area 'B', where plural words related to people are represented.

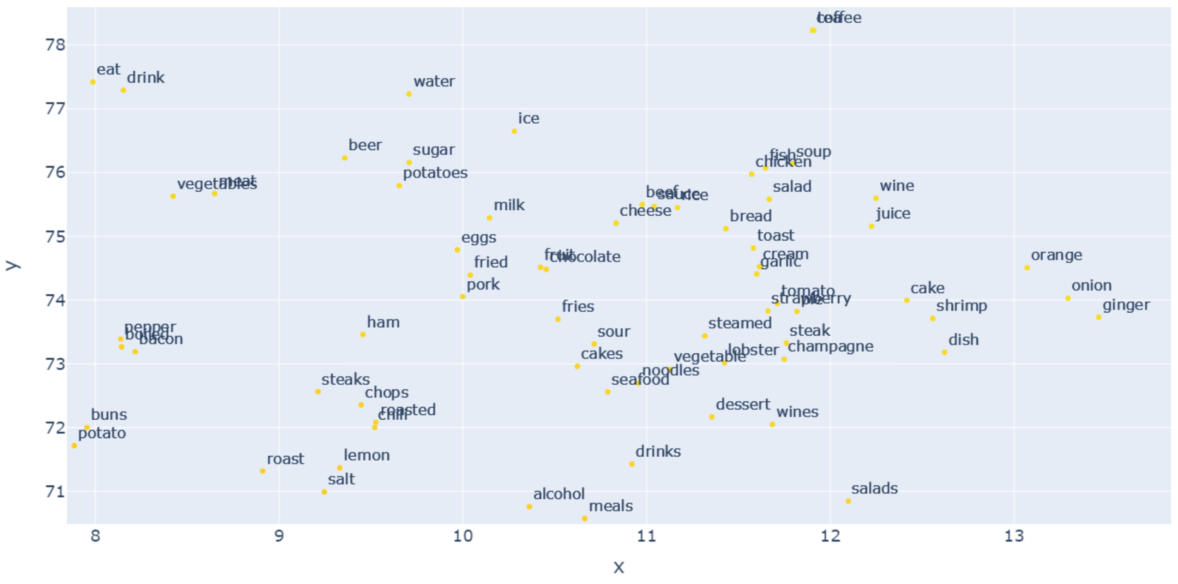


Figure 3.7: T-SNE visualization from the zoomed area 'C', where different products related to a cuisine are represented.

Something to remark is that contrary to the previous word embeddings analysed cases, where in the situation that a token was not found in the corpus vocabulary it was filled with random values, here as we used the "train, valid and test" datasets to feed the *Word2Vec* algorithm, all tokens have their respective value information. The only ones remaining with random values are the auxiliary tokens, like: $<pad>$, $<unk>$, $<s>$, $</s>$ and $<d>$, as they do not appear in the dataset they must be manually added to the embedding layer.

As result, the trained word embeddings has 10 002 tokens, given a shape for the layer of (10002, 300).

Once we have a trained word embeddings, the procedure to plug in the vector into the embedding layer is similar to the previous models. Steps like the vocabulary creation, data pre-processing and the data loader are similar to base model, too.

No pre-trained word vector: The idea behind this model is to train a DialogWAE model without any pre-trained word embeddings vector. The purpose is to visualize and analyze whether the base model has the capabilities to perform as well as the models with pre-trained embeddings.

No special procedure was added to the model. Therefore, all the steps followed in the creation of the base model are repeated here. As this model is a direct replication of the base model, the dimension for the word embeddings vector was left to 200. Hence, this model has an embedding layer that initializes with a normal distribution between 0 and 1.

Bert as embedding layer: This model experiments with the idea to use Bert as sort of word embeddings vector. Given that Bert uses contextualized embeddings generating different outputs for a word under different contexts. We believe it is interesting to investigate whether the model can feed on pre-trained Bert's hidden states.

We were aware that Bert cannot be used for language modeling tasks, given that during the generation of a new utterance, we require to sample the probability distribution of the next token given the previous contexts. This is a task that Bert cannot perform due to its bidirectional nature.

Nevertheless, there exists a model called *BertForMaskedLM*[10]. This is actually a Bert model with a language modeling head on top that, according to Wang and Cho (2019) can be tweaked to generate one word at a time, in left-to-right order.

Some modifications needed to be set up before we could train the DialogWAE model using Bert as a pre-trained embeddings. First, the modifications are related to the data where the tokenization must be in concordance with the Bert model. Therefore, tokens like *<pad>*, *<unk>*, *<s>*, *</s>* were modified to [PAD], [UNK], [CLS], [SEP] respectively, to agree the Bert tokenization system.

Second, the data loader was modified so that it can output the attention mask generated by the Bert Tokenizer[11]. This is an array filled with 0s or 1s very important to denote the relevant tokens in a input sequence passed over the Bert model.

[10]`https://huggingface.co/transformers/model_doc/bert.html?highlight=mask%20bert#bertformaskedlm`

[11]`https://huggingface.co/transformers/main_classes/tokenizer.html`

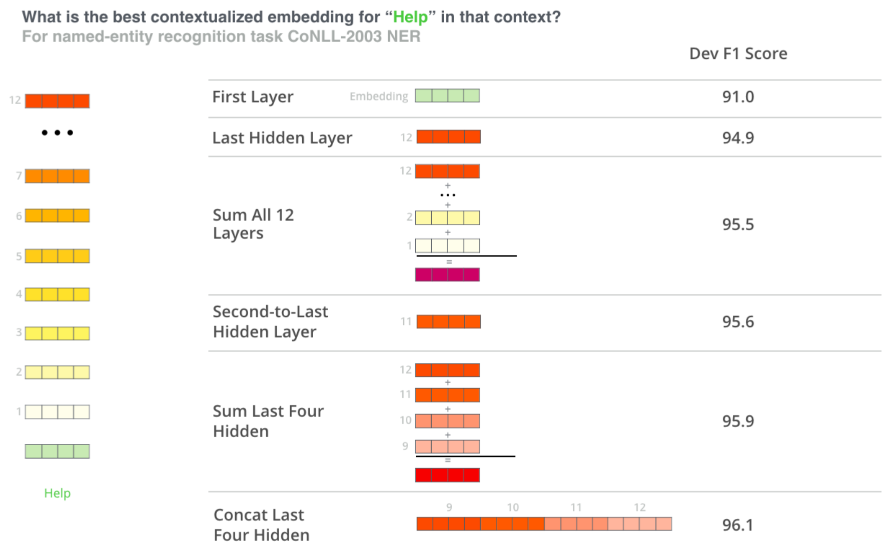

What is the best contextualized embedding for "Help" in that context?
For named-entity recognition task CoNLL-2003 NER
Dev F1 Score
First Layer
Embedding
91.0
Last Hidden Layer
94.9
Sum All 12 Layers
95.5
Second-to-Last Hidden Layer
95.6
Sum Last Four Hidden
95.9
Concat Last Four Hidden
96.1
Help

Figure 3.8: Results of the different ways of grouping the hidden states[12].

The authors of the Bert model also investigated the use of different hidden states in order to create a contextualized embedding working as a feature-based approach that can later be used in different models of diverse NLP tasks. In Figure 3.8 it can be seen the different proposed combinations of the hidden states presented in Devlin et al. (2018).

We decided to move forward with the combination of the sum of the last four hidden states to create our contextualized embedding.

[12]`http://jalammar.github.io/illustrated-bert/`

Chapter 4

Results

In this chapter, the results from the several experiments performed are presented and then, compared. As it has been mentioned in section 3.2, all the experiments were carried out with the Daily Dialog dataset. Also, a foreword dialogue was created so that we can let the different models run it in free-mode. Those results can be seen in appendix A.5.

The first section of this chapter is divided into five subsections where each embedding layer is analyzed within the base model in order to assess the impact the layer has on the model performance.

First, the results from the base-model are exposed, as they are used as base control. Second, the description of the results of each embedding is presented. In the second section, all the evaluation metrics are summarized to have an overview of the results and the impact of the different embedding layers.

GANs architectures tend to be hardware demanding and adding a NLP task only increases the demand. We came with the solution that we needed to work with Google Colab platform[1]. Hence the entire code was written in Colab notebooks and all the models were trained with the GPU randomly assigned by this free service.

The notebooks can be found in the following link: `https://drive.google.com/drive/folders/1lzG7IZnnFP7mSpkkOw6ri992PLrvpyjf`.

Although we used GRU cells with the default parameters to enable the use of CuDNN backend[2] for a faster and optimized training, the times to train a RNN remain extensive. Thus, each model was run from 3 to 6 times before selecting the one to be described in this chapter.

4.1 DialogWAE Model

In this section, we include the training results as well as the results of the evaluation metrics.

4.1.1 Base Model

Model training and evaluation process

During the training process, the validation loss and the BLEU score are the metrics used to monitor by the *Early Stopping* algorithm.

[1]`https://research.google.com/colaboratory/faq.html`
[2]`https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html#features-of-rnn-functions`

In most general GAN models, the loss value is not a reliable metric when assessing whether training has converged. Moreover, there is no general consensus regarding the metric to use in order assess when to stop training a GAN model (Borji, 2019). Nevertheless, Borji (2019) indicate that models trying to minimize the Wasserstein distance between the distribution of real and generated sequences might use the loss value, since a reduction in distance value can be interpreted as an improvement towards convergence. Therefore, the `patience` parameter of the Early Stopping algorithm was set to 20, enough epochs to observe whether exists any improvement in either the BLEU score at each evaluation step or the validation loss.

The whole training process lasted 4h:16m:1s and the model trained for 86 epochs, performing an evaluation step on the validation set every 4 epochs. In general, each epoch took ∼1m:41s to complete and the epochs with an evaluation took ∼6m:48s. This long evaluation time is due to the fact that the BLEU score is computed using the NLTK library. Hence, it does not use the high computing capabilities of a GPU, making it an expensive procedure in terms of learning time.

Training and validation loss

The training curves can be seen in Figure 4.1. This base model reaches its lowest validation loss at epoch 65 with a value of 2.6226. Around epoch 30 the training loss reaches the level of the validation loss. Several explanations can justify the initial lower validation loss but overall, the size of the validation set seems the most reasonable one. Indeed, the validation set is 10 times smaller than the training set, where one assumes that hard/unique sequences are less present.

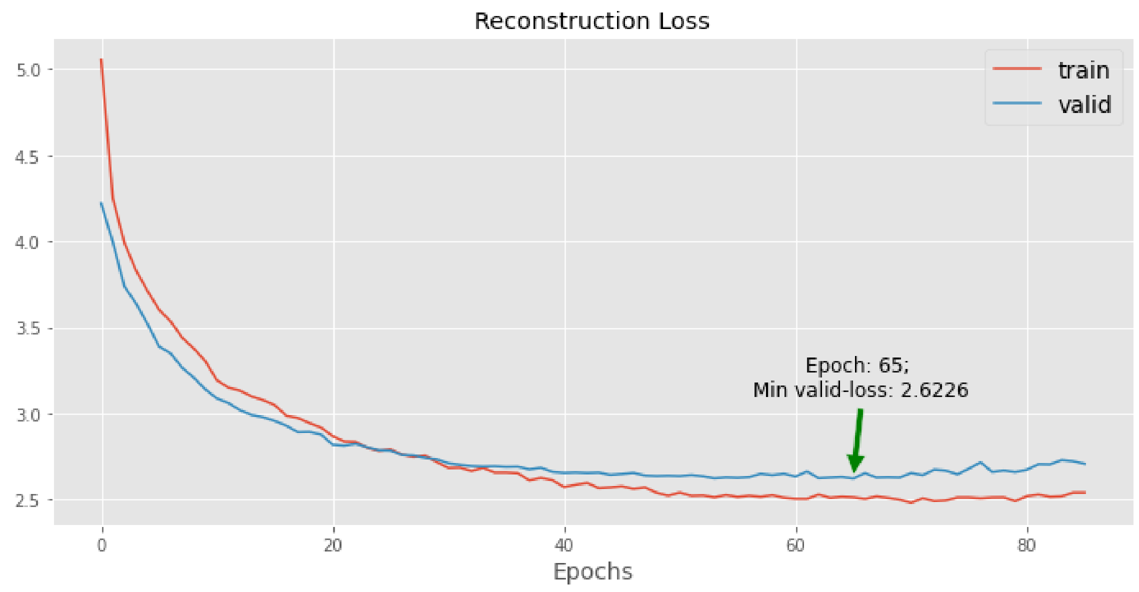


Figure 4.1: Training and validation loss evolution from the base-model.

The loss progression from each training step can be seen in Figure 4.2. Both the generator $\mathcal{G}$ and discriminator $\mathcal{D}$ begin with a rapid loss increase, going in opposite direction, as expected due to the adversarial nature of the GAN models. Figure 4.2 shows that after a certain number of steps, the model stabilizes, letting the generator and discriminator converge to a certain level of equilibrium. Convergence seems to dissipate slightly in the later epochs where the model starts to overfit.

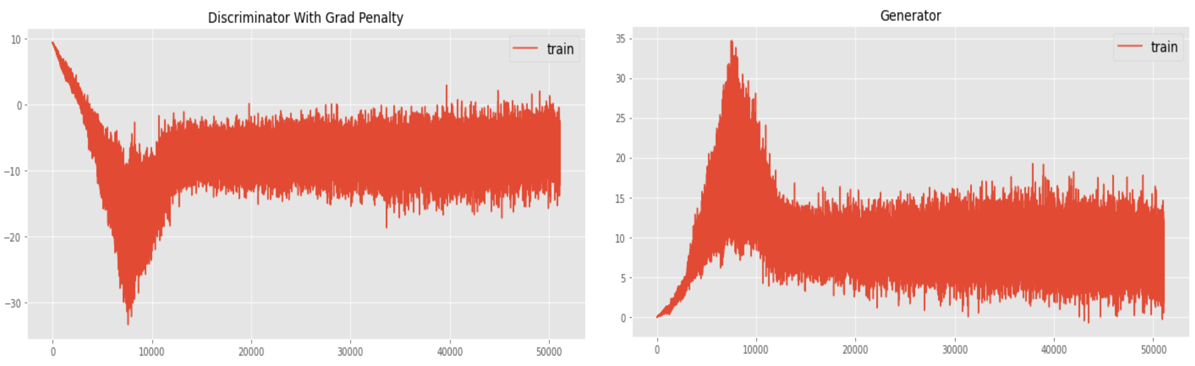


Figure 4.2: Loss progression for the discriminator and generator throughout the training steps. On the left the discriminator's loss and on the right the generator's loss.

Evaluation results

The evaluation metrics discussed here represent the quantitative analysis performed during the learning process.

At each evaluation step, the results for the BLEU and BOW embedding metrics were calculated. BLEU F_1 score was the one used to be monitored by the Early Stopping algorithm as it represents the harmonic mean between the recall and precision, previously detailed in section 3.3.

The Figure 4.3 shows the evolution of both scores during the training time. If we consider the BLEU F_1 score, the maximum value was obtained at epoch 60, while the minimum validation loss was reached at epoch 65. This demonstrates the difficulty of choosing which model would be the most optimal.

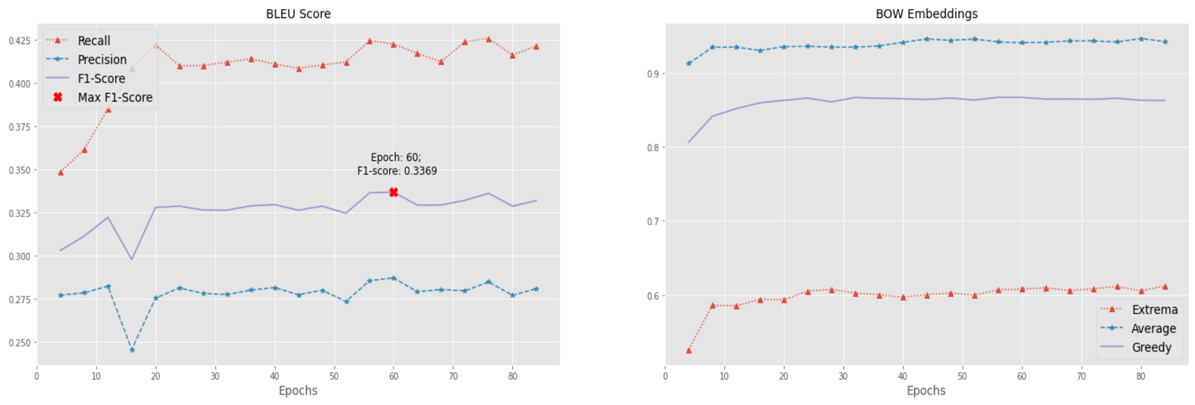


Figure 4.3: Evaluation metrics for the base model, trained with GloVe word embeddings. On the left, the BLEU score and on the right, the BOW embeddings score.

Regarding the BOW score, as it was used the GloVe word representation the three different metrics (extrema, average and greedy) achieved great results, improving slightly thanks to the fact that it continued training with the model.

Later in this chapter, we compare all the evaluation results of the different models thanks to quantitative and qualitative analysis.

4.1.2 Spacy word vector

Model training and evaluation process

The whole training process took 3h:50m:20s and complete 74 epochs, with similar times to the base model for each epoch and evaluation step.

Training and validation loss

The model reaches its lowest validation loss at epoch 54 with a value of 2.5880 (see Figure 4.4). With this pre-trained word embeddings this model reached a decent loss with less epochs than the base model. This improved performance could be attributed to the fact that the Spacy word embeddings contain more information –a larger dimension– and to its high optimization for different NLP tasks.

Regarding the backpropagated loss through the generator $\mathcal{G}$ and discriminator $\mathcal{D}$ the loss progresses with some instability for a short number of training steps. Similarly to the base model, the loss stabilizes as training increases reaching certain level of equilibrium between the generator and discriminator(see Figure 4.5).

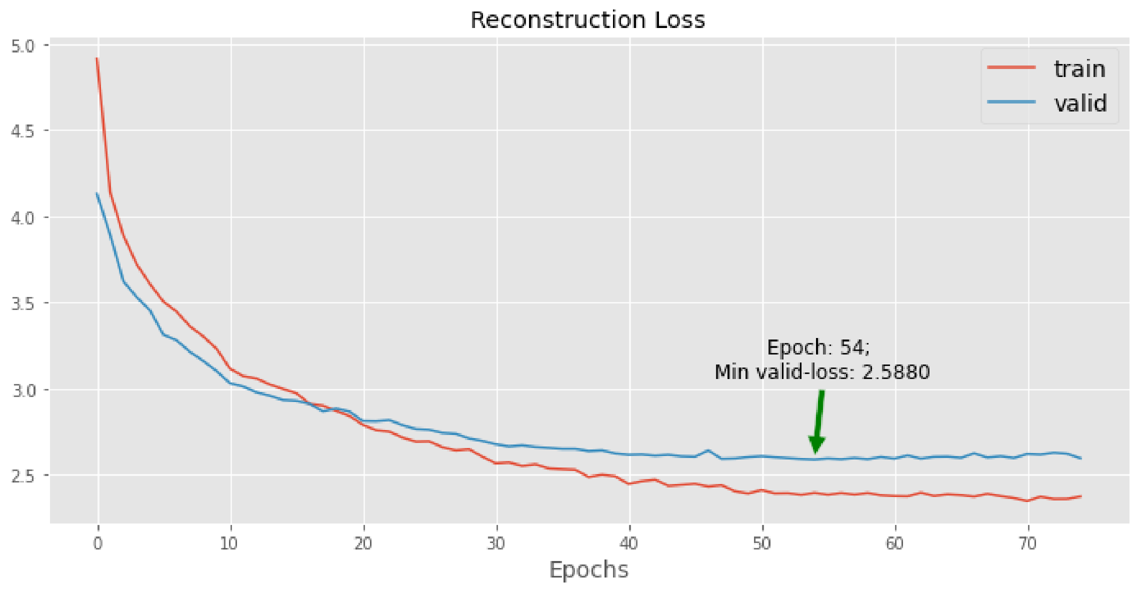


Figure 4.4: Training and validation loss evolution from the model trained with Spacy word representation.

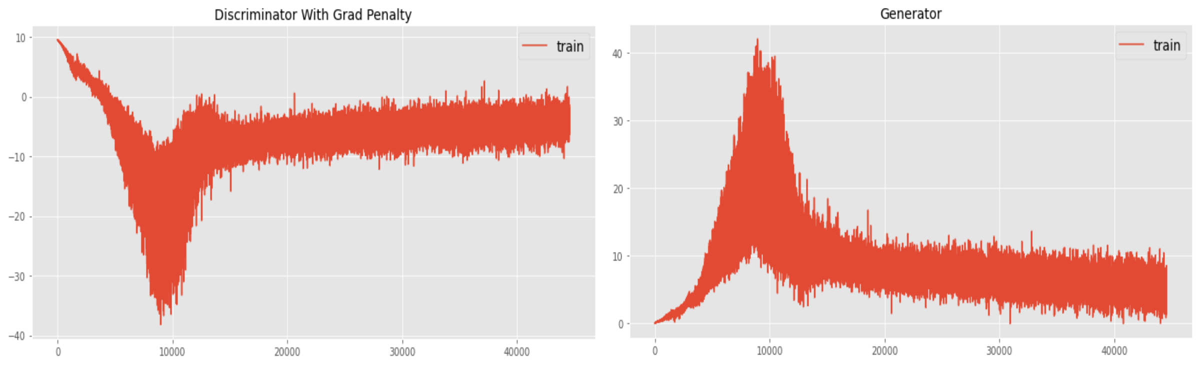


Figure 4.5: Loss progression for the discriminator and generator throughout the training steps. On the left the discriminator's loss and on the right the generator's loss.

Evaluation results

The evaluation metrics are shown in Figure 4.6. With respect to the BLEU F_1 score we can mention that it reached the maximum possible value at epoch 28. Comparing with the base model, it achieved that result relatively faster. When considering the validation loss, the model converge to its minimum result much before that the base model. However, if we look in more details, at epoch 60, there is another evaluation step where a relatively similar BLEU F_1 score was achieved.

This result arrives a few epochs later than the minimum validation loss is reached. This indicates that there is an epoch where the model could be optimal too, hence further investigation is needed. A more detailed review is discussed further in this chapter.

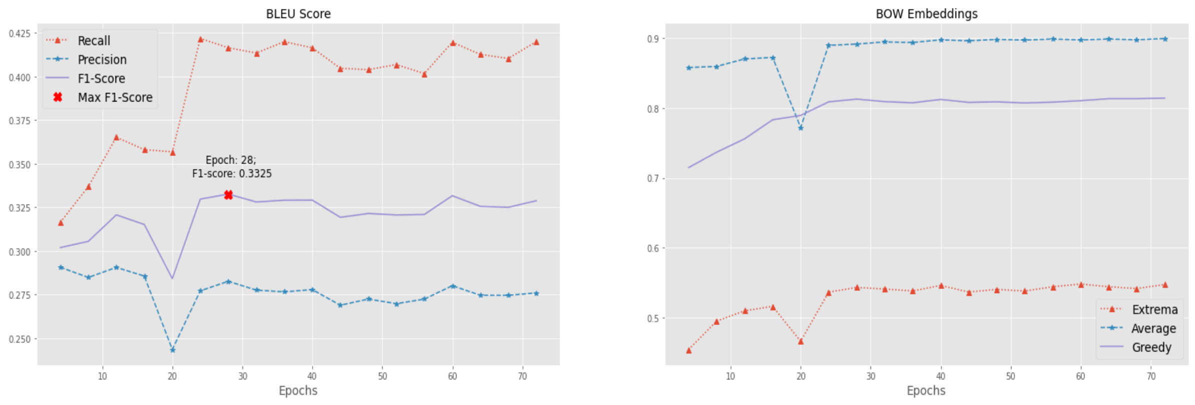


Figure 4.6: Evaluation metrics for the model, trained with Spacy word representations. On the left, we have the BLEU score and on the right, the BOW embeddings score.

Similarly to the base model, the results for the three metrics (average, extrema and greedy) in the BOW embedding score, show gradual growth as training increases. Nevertheless, if we compare these results with the base model, a slightly decrease in performance can be observed, especially on extrema and greedy metrics.

4.1.3 Own Trained word vector

Model training and evaluation process

The whole training process took 4h:27m:29s and the model completed 89 epochs.

Training and validation loss

In Figure 4.7, can be appreciated the learning loss curve over the training and validation sets. The model achieves its lowest value of 2.5640 for the validation loss at epoch 69. The training loss reaches the level of the validation loss around the epoch 22, which indicates that by using a larger dimension embedding layer, the model gets to benefit from it. A tendency similar to the word embedding from *Spacy*,

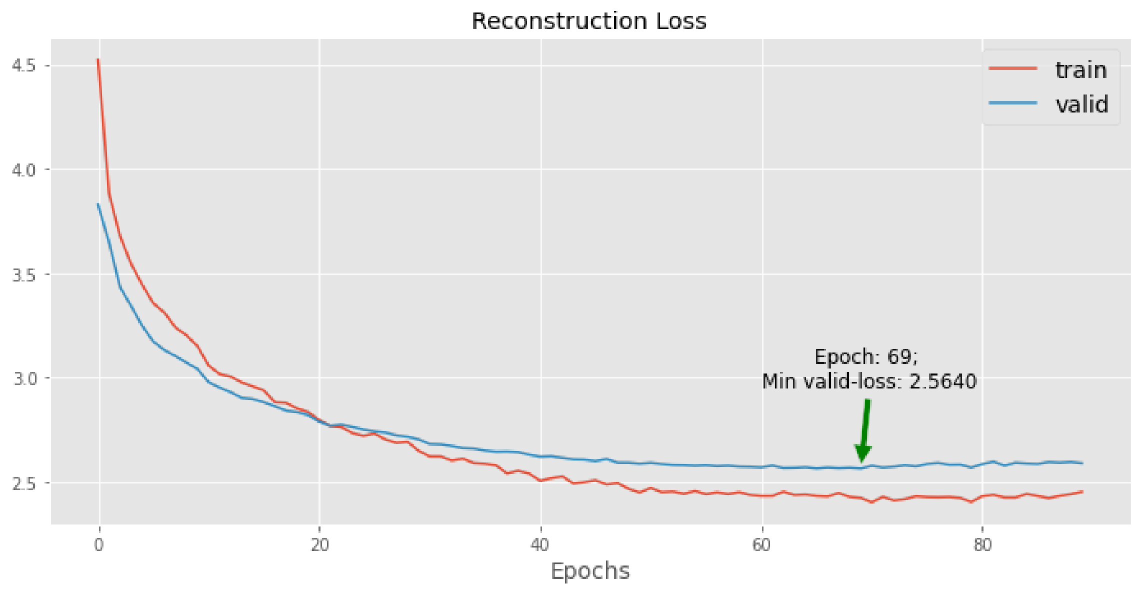


Figure 4.7: Training and validation loss evolution from the model with the own-trained word embeddings.

On Figure 4.8 is appreciated the progression of the loss backpropagated for both generator $\mathcal{G}$ and discriminator $\mathcal{D}$. Similar to previous models both losses had their period of instability during the first training steps, to later approximate to certain equilibrium between the generator and discriminator.

Also, can be observed that having a our own-trained word embedding produces a more stable backpropagated loss as the graph shows when comparing to the previous models where the convergence to an equilibrium is not as stable as in this model.

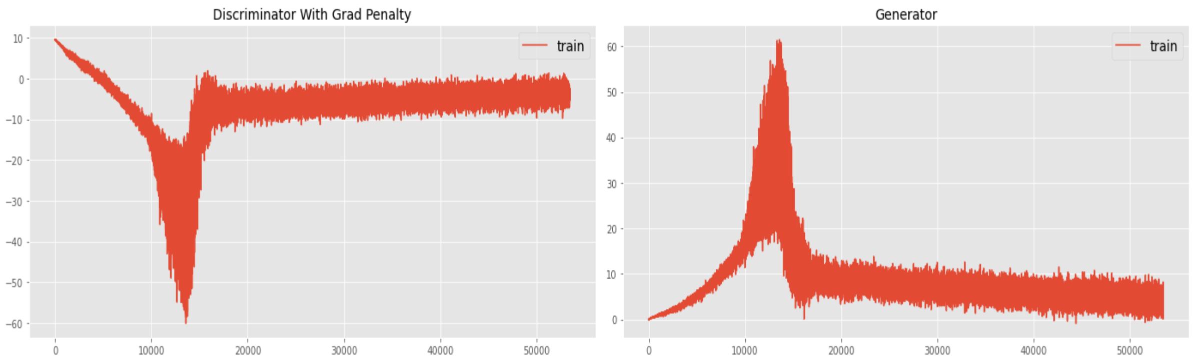


Figure 4.8: Loss progression for the discriminator and generator throughout the training steps. On the left the discriminator's loss and on the right the generator's loss.

Evaluation results

As the metrics in Figure 4.9 reflects, the BLEU F_1-score reaches its maximum value at epoch 28, far from the epoch of the minimum validation loss that is 69. Being BLEU score an important metric for language modeling, both epochs (28 and 69) were analyzed over the testing set and its results discussed later on the chapter. This situation demonstrates once again the unstable nature of the training process of GAN networks.

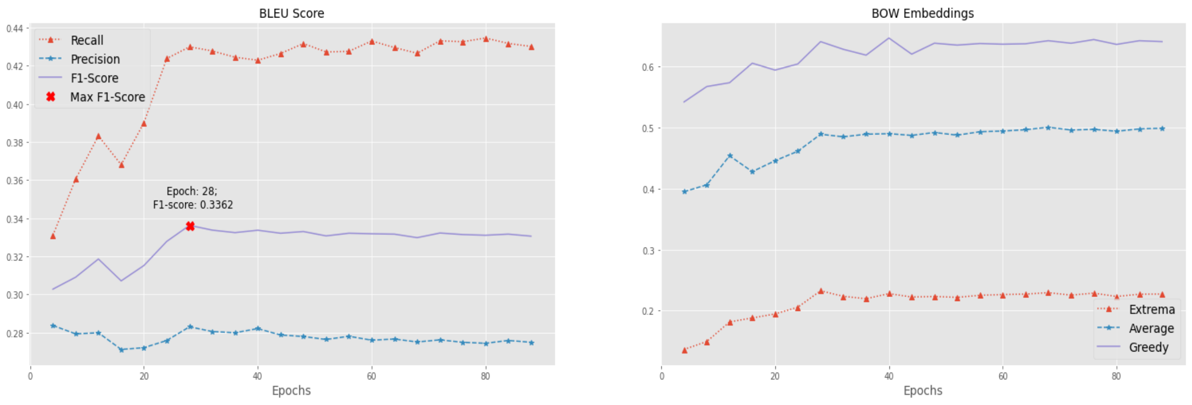


Figure 4.9: Evaluation metrics for the model trained with our own-trained word embeddings. On the left, the BLEU score and on the right, the BOW embeddings score.

Regarding the BOW embedding metric, the three results (extrema, average and greedy) show a gradual growth along with the increasing of the training epochs.

At first glance, when we compare these values with the pre-trained word embeddings models they do not stand out much. This could be a consequence of the size of the corpus in which the own-word embeddings was trained or the simple fact that the word2vec algorithm is not as strong as the results in Pennington et al. (2014) suggest.

4.1.4 No Pre-trained word vector

Model training and evaluation process

The model completed 85 epochs among 4h:4m:55s. Despite the model trained for the same number of epochs as the base model, the difference in time can be explained due to the randomness from Colab at the moment of setting the CPU for the virtual machine.

Training and validation loss

The training and validation loss can be observed in Figure 4.10. This model reaches is minimum validation loss at epoch 65 with value 2.8065.

Unlike previous models that use pre-trained word embeddings, the moment when the training loss reaches similar levels as the validation loss occurs at epoch 47, relatively late.

Regarding the loss progression, this model reflects the greater instability towards the equilibrium when comparing to other models (see Figure 4.11).

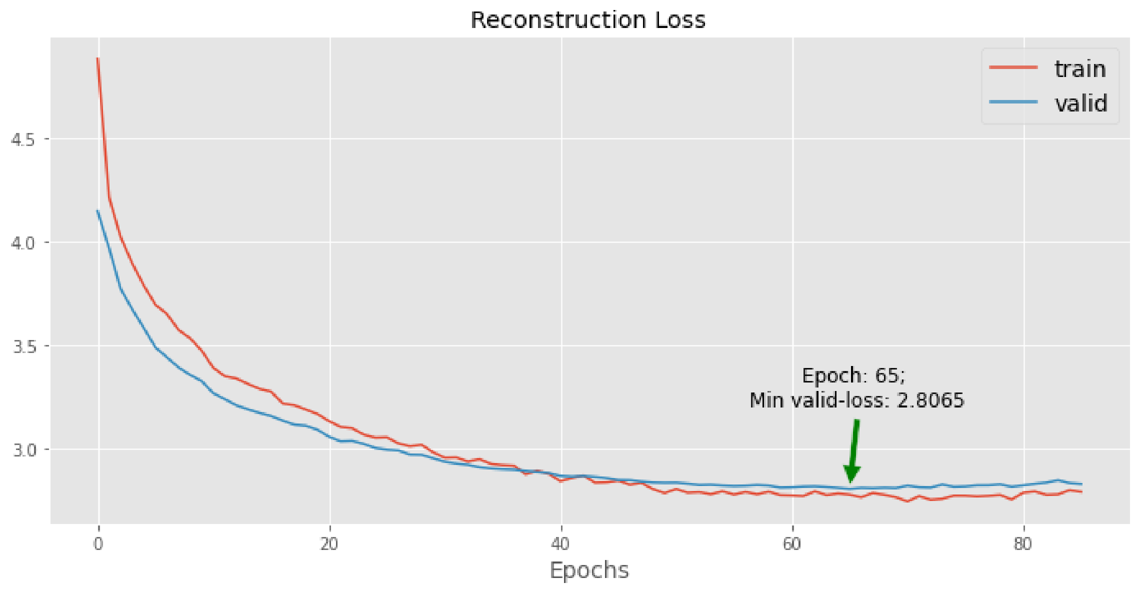


Figure 4.10: Training and validation loss evolution from the model without any pre-trained word embeddings.

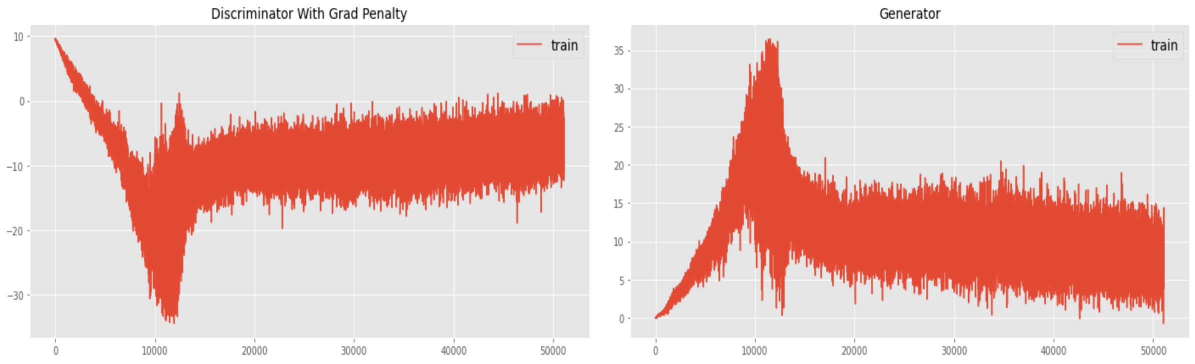


Figure 4.11: Loss progression for the discriminator and generator throughout the training steps. On the left, the discriminator's loss and on the right, the generator's loss.

Evaluation results

As the results of the metrics shown in Figure 4.12, the maximum BLEU F_1 score was reached at epoch 32 with value 0.3298. However, this value is the lowest of all the models presented previously, a result that is explained by the lack of a pre-trained word vector. Similar to previous models (Spacy and own-trained words vector), the long distance that separates the highest BLEU score with the lowest validation loss requires further investigation, detailed later in the chapter.

Also, Figure 4.12 shows that, as the training progresses, the BLEU score decreases in all the computed measures (recall, precision and F_1). This behaviour is not reflected in the BOW embedding metric as it only starts to decrease when the model begins to overfit. This continuous reduction on BLEU score can be explained by the fact that as the training progresses, the model learns to generate longer responses. This could reduce the match number of n-grams overlaps from the hypothesis, as the results presented in appendix A.4 suggest.

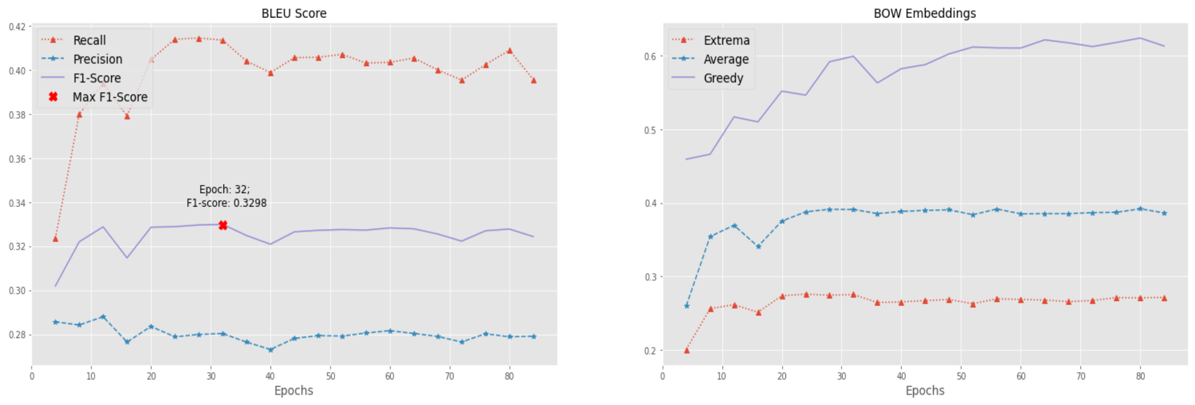


Figure 4.12: Evaluation metrics for the model trained without any pre-trained word embeddings. On the left, the BLEU score and on the right, the BOW embeddings score.

4.1.5 Bert as embedding layer

Model training and evaluation process

It turns out that the training process for this model is quite extensive, taking approximately ∼40m to complete one epoch.

Training and validation loss

Since we used Bert as a feature extractor and we plugged it in directly to the model, it can still benefit from the training process where the DialogWAE model keeps track of the gradients that would be updated later during the backpropagation of the loss.

Therefore, we decided to take two different approaches on the training process. In the first approach, the Bert model is not frozen, letting it take advantage of the learning process and the computed loss. In the second approach, the trainable layers of the Bert model are frozen which indicates that Bert only will produce the hidden states from its pre-trained weights, forcing the main model to learn how to handle the outputs from the Bert model.

The training and validation of the first approach can be seen in Figure 4.13. This model trained for 7h:59m:53s completing 9 epochs. This extensive training time is due to the 143 184 295 trainable parameters that the model contains due to its combination with the Bert model.

Regardless of the few epochs that this model trained, it was able to reach the validation loss of 0.3278, that is the lowest value from all the models described in this chapter. This great improvement in loss terms is mainly due to Bert model being able to update its weights at each forward pass, showing thus its great capacity for adaptation.

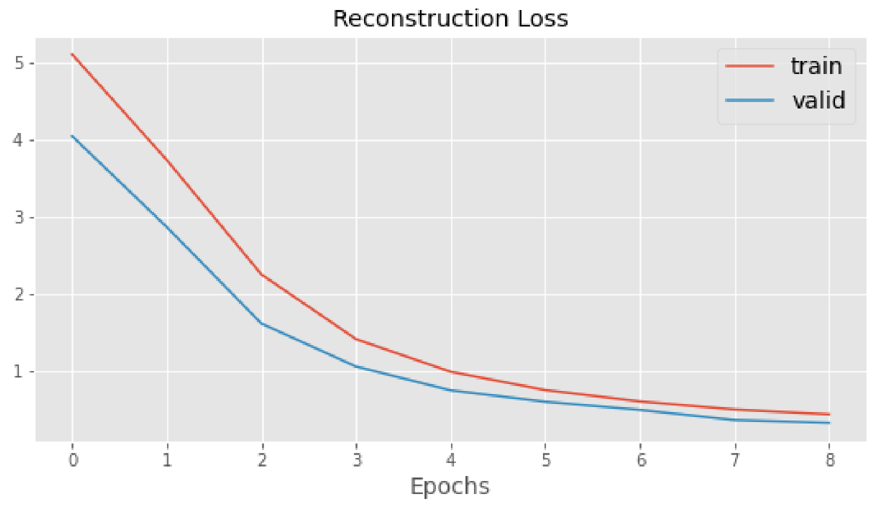


Figure 4.13: Training and validation loss evolution from the model using Bert under the first approach.

The losses for the second approach are found in Figure 4.14. In this approach the Bert model was frozen which left us with 33 669 997 trainable parameters. This reduction enabled us to have shorter training times for each epoch with ∼23m10s each. Thus, this model completed 18 epochs in 7h:46m:55s.

The model achieved its lowest validation loss at epoch 16 with value 6.2299. This value is greater than we expected, being the highest one comparing with rest of the trained models. This clearly shows us that the DailogWAE model needs to modify its architecture in order to follow this approach. However, due to the long training times we decided to stop investigating this approach, since it seemed obvious that the model struggled to handle the contextualized embeddings produced by the Bert model.

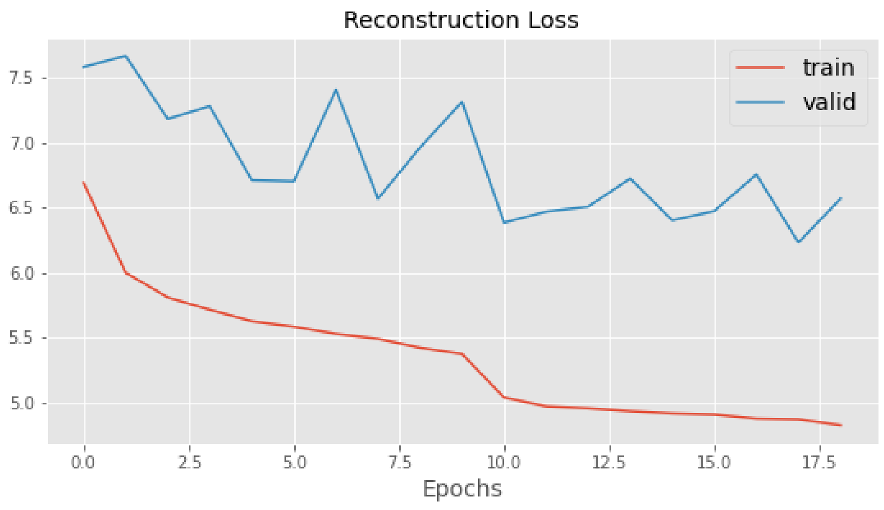


Figure 4.14: Training and validation loss evolution from the model using Bert under the second approach.

Evaluation results

Despite the effort in trying to implement the proposed approaches (parallel sequential generation, sequential generation and parallel generation) expressed in Wang and Cho (2019), the model trained on the first approach failed to generate relevant sequences of utterances. Indeed, in the majority of the cases the generated sequences did not follow the context previously given to the model. Thus, the results of this model are not going to be taken into account when we analyze and compare the results of all the trained models.
Some examples of the generated responses from this model can be found in appendix A.6.

One can attribute the flaw of the generated responses to the nature of the masked language modeling that predicts a word given its left and right context. Hence we suggest that in this auto-encoder context where decode/generate a response from Bert model is still an open question that requires further exploration which is left for future investigation.

4.2 Comparison of the different approaches

In this section we summarize the results obtained from the different models described previously and evaluate their performance on the testing set.

4.2.1 Quantitative analysis

Similarly with the learning process, we sampled 10 responses with greedy decoding so that the randomness comes from the latent variables.

We present the results of the evaluation step performed over each epoch from each model, see Tables 4.1 to 4.4. Given that BLEU score is the most relevant metric, we used it to justify our model selection. Finally, Table 4.5 contains the grouped evaluation results of all selected models.

Base model

This model achieved the highest BLEU F_1 score and minimum validation loss, in relatively near epochs. As results indicate in Table 4.1, the model at epoch 60 obtained the highest BLEU score, thus it was selected as the optimal one.

Epoch	**BLEU**			**BOW embedding**		
	R	**P**	**F1**	**A**	**E**	**G**
60	0.4121	0.2833	**0.3310**	0.9416	0.6077	0.8664
65	0.4082	0.2781	0.3261	0.9421	0.6090	0.8643

Table 4.1: Evaluation results over epochs 60 and 65 (R: Recall, P: Precision, A: Average, E: Extrema, G: Greedy).

Spacy word vector

For this model three different epochs were evaluated, where two of them achieved similar results for the BLEU score and the third was the one with the lowest validation loss.

Epoch	**BLEU**			**BOW embedding**		
	R	**P**	**F1**	**A**	**E**	**G**
28	0.4115	0.2790	**0.3283**	0.8935	0.5466	0.8124
54	0.4080	0.2745	0.3240	0.8989	0.5474	0.8110
60	0.4014	0.2742	0.3225	0.9014	0.5504	0.8113

Table 4.2: Evaluation results over epochs 28, 54 and 60 (R: Recall, P: Precision, A: Average, E: Extrema, G: Greedy).

Following the results presented in Table 4.2, model trained at epoch 28 is selected as the optimal one.

Own-trained word vector

This model has the characteristic of having a long distance between the minimum validation loss and the maximum BLEU F_1 score. Results in Table 4.3 shows that the model at epoch 28 is the optimal one.

Epoch	**BLEU**			**BOW embedding**		
	R	**P**	**F1**	**A**	**E**	**G**
28	0.4211	0.2794	**0.3312**	0.4926	0.2331	0.6289
69	0.4254	0.2719	0.3268	0.4972	0.2280	0.6346

Table 4.3: Evaluation results over epochs 28 and 69 (R: Recall, P: Precision, A: Average, E: Extrema, G: Greedy).

No pre-trained word vector

Similar as previous model, this one also has a significant distance between validation loss and BLEU score. Taking into consideration the results on Table 4.4, we selected model at epoch 32 as the optimal one.

Epoch	BLEU			BOW embedding		
	R	P	F1	A	E	G
32	0.4087	0.2787	**0.3274**	0.3917	0.2738	0.5924
65	0.4031	0.2772	0.3246	0.3829	0.2646	0.6093

Table 4.4: Evaluation results over epochs 32 and 66 (R: Recall, P: Precision, A: Average, E: Extrema, G: Greedy).

Table 4.5 shows the performance of the different models on the testing set, where it also was added the average length of the response generated and the epoch which the model reached those results.

Model	BLEU			BOW embedding			L	Epoch
	R	P	F1	A	E	G		
Base model	0.4121	**0.2833**	0.3310	**0.9416**	**0.6077**	**0.8664**	9.1	60
Spacy	0.4115	0.2790	0.3283	0.8934	0.5455	0.8117	10.9	28
Own-trained	**0.4211**	0.2794	**0.3312**	0.4926	0.2331	0.6289	10.7	28
No pre-trained	0.4087	0.2787	0.3274	0.3917	0.2738	0.5924	**11.8**	32

Table 4.5: Performance comparison from the different models (R: Recall, P: Precision, A: Average, E: Extrema, G: Greedy, L: Average length).

In terms of BLEU score, all models achieved more or less similar values. We denote that the "own-trained" word vector model where the recall result is slightly higher than the rest of the models. Both Spacy and our own-trained word vector favored to the model with its large dimension, where not only it was possible to obtain optimal results within fewer training steps but it also favored the generation of more extensive responses.

Regarding BOW embeddings metric, the base model achieved the best results. Results can be explained by the use of the GloVe vector that was pre-trained in a much more extensive corpus.

The model with no pre-trained word vector obtained the worst results, as expected. Although, its BLEU scores are not too far from the other models. This tells us that the DialogWAE model has a good architecture capable of taking advantage of adversarial learning and that the boosting in performance a pre-trained vector can offer does not represent a great part of its achieved results.

4.2.2 Qualitative analysis

In each table, we present three generated responses sampled from the models. In each of them the informative content of the dialogue context was gradually increased, leaving Table 4.9 with a dialogue context that is not found in any set (training, validation or testing). As such, we can appreciate the behaviour of the models reacting to an unknown sequence.

Context	thank you for calling world airline, what can i do for you?			
Responses	**Base model**	**Spacy**	**Own-trained**	**No pre-trained**
	Ex1: yes. sure. do you have the reservation available?	Ex1: what would you like to have a look?	Ex1: oh, no. i'm sorry to hear that. what's the matter with you?	Ex1: hello, please. may i help you?
	Ex2: yes please go to the airport.	Ex2: in a single room.	Ex2: yes, please fill out the application form.	Ex2: yes, please. may i have your name, please?
	Ex3: do i do?	Ex3: no, thanks.	Ex3: look, it's just that i've got to go.	Ex3: let me have a look at the menu.

Table 4.6: Examples of the responses generated by the models.

Context	i would like to invite you to dinner tonight, do you have time?			
Responses	**Base model**	**Spacy**	**Own-trained**	**No pre-trained**
	Ex1: yes. do you want to go to the concert?	Ex1: what' would you like for a drink?	Ex1: oh, yes. i'm sorry to hear that. but i'm afraid i can't find anything to eat.	Ex1: go ahead. what time do you want to go?
	Ex2: do please take a seat.	Ex2: of a year, please.	Ex2: yes, come in.	Ex2: yes. may i have your name, please?
	Ex3: do you want to go?	Ex3: no, thanks.	Ex3: haven't seen you for ages?	Ex3: just two weeks ago.

Table 4.7: Examples of the responses generated by the models.

Context	I heard you have found a new job. _eou_ Yes, I am now working in a company . _eou_ So you're quite happy with your new job?			
Responses	**Base model**	**Spacy**	**Own-trained**	**No pre-trained**
	Ex1: i like it very much.	Ex1: that's the most important thing to me.	Ex1: i'm not interested in working with my company. i'm not sure i'll be able to take care of it.	Ex1: not at all?
	Ex2: how do you plan to work?	Ex2: i do, but i have to work overtime.	Ex2: what a problem.	Ex2: i don't know. i am going to be late for a long time.
	Ex3: yes, i know, i am.	Ex3: yes, i'm going to have a try. i need to get a new job.	Ex3: she had a very good reputation.	Ex3: not at all of our company. we've got to go to the beach.

Table 4.8: Examples of the responses generated by the models. _eou_ indicates a change of turn for the speaker.

Context	how much for the extra luggage? _eou_ the price is $100 per suitcase.			
	Base model	**Spacy**	**Own-trained**	**No pre-trained**
Responses	Ex1: ok? would you like the money in the room?	Ex1: what is the price rate for this position?	Ex1: 's good.	Ex1: do you want to go with me?
	Ex2: please drive a taxi station.	Ex2: in a single room, sir?	Ex2: any fees?	Ex2: do i have to pay extra charge?
	Ex3: would you like to pay?	Ex3: well, a single room, please.	Ex3: look at it.	Ex3: do you have a good time?

Table 4.9: Examples of the responses generated by the models. _eou_ indicates a change of turn for the speaker.

Most of the responses generated by the models show a coherence that covers multiple probable aspects. We noticed that responses generated by the "own-trained" model produces longer responses that exhibit informative content. Also, the samples from the base model show a tendency to produce positive and interrogative questions. Moreover, it is observed that the longer the context of the dialogue, the more interesting responses the models can produce.

In addition, it is surprising to observe how the "no pre-trained" model produces responses that in some cases, would be chosen as the most accurate by a human evaluator.

Finally, regarding to the sampled responses for the unknown dialogue, most of the models were successful to create sequences that show some level of coherence related to the dialogue context. Although the luggage context was lost, the issue of payment is current in the generated responses.

Chapter 5

Conclusion

In this work, we studied and reviewed a novel model for neural response generation that outperformed previous state-of-the-art approaches. DialogWAE is a conditional Wasserstein auto-encoder that models the distribution of latent variables by training a GAN. This language model learns to encode the dialogue history and thanks to the GAN discriminator, a generator fed with this encoded query is trained to produce responses conditioned by the dialogue history. We detailed the reasoning behind this model and provided a description of the different phases of the training process. We evaluated this model on DailyDialog data set (Li et al., 2017).

Furthermore, we experimented the model with different word vectors representations, in order to assess the impact a word vector has on performance of the model. Through the quantitative analysis, we observed that word vector mainly influenced the training times and to a lesser extent, the evaluation metrics. However, from the qualitative analysis was possible to observed that even the simplest word vector trained from the input data, was producing longer response utterances, richer in information and in content diversity, than the base model.

Moreover, we unsuccessfully explored the idea of combining a transformer model within a GAN architecture, either by making use of its outputs or by fine-tuning the transformer through adversarial learning. This idea was motivated by the publication of Wang and Cho (2019) and the released of the *BertForMaskedLM*[10] model.

Overall, despite GANs frameworks are successfully applied to Image generation tasks, Chatbots with GAN architectures remain a very limited field of research, mostly due to their challenging nature at the implementation and training time. Nevertheless, we believe that models like DialogWAE shows a promising future for GAN architectures since it demonstrates that adversarial learning characteristics can be applied to the field of neural response generation. This is important given that GAN architectures does not sample the closest approximation to the real target but rather, they minimize the overall distance between the generated and real sample, generating thus less constraint responses.

Limitations and future work

This work experiences some limitations that give avenue for future research. For instance, during the experimentation stage, different word vectors were tested within the model. We focused on evaluating the performance of embeddings pre-trained at a word level. Future research can extend this experiments and use character level like ELMo(Peters et al., 2018) and Flair(Akbik et al., 2018) or subword level like fastText(Bojanowski et al., 2017).

Also, recent publication of articles, such as Croce et al. (2020); Shin et al. (2020) open doors for scholars to keep exploring the idea to combine a transformer model with GAN architecture.

Finally, regarding to the DialogWAE model, some scholars can perform a deep investigation related to the model selection given that it was computationally expensive to perform an evaluation step at every epoch. In this work, the evaluation step was performed every 4 epochs but it could be possible that some epochs may have obtained better values between that range.

Appendix A

Complementary information and results

A.1 List of articles with buggy BLEU scores

- *Improving neural conversational models with entropy-based data filtering.* Csaky, R., Purgai, P., & Recski, G. (2019)
- *Variational hierarchical user-based conversation model.* Bak, J., & Oh, A. (2019, November)
- *Conditional Response Generation Using Variational Alignment.* Khan, K., Sahu, G., Balasubramanian, V., Mou, L., & Vechtomova, O. (2019)
- *Plato: Pre-trained dialogue generation model with discrete latent variable.* Bao, S., He, H., Wang, F., Wu, H., & Wang, H. (2019)

A.2 Comprehensive list of missed tokens from the word embeddings

Base model

The GloVe vector used in the base model did not contain the following 322 tokens:

<pad>, <unk>, </s>, <d>, ..., 10, 2, 3, 5, 30, 000, 7, 6, 1, 20, 00, 15, 9, 8, 100, 4, 50, 12, 200, 500, 25, 11, 40, .., 300, 24, 80, 150, 16, 60, 90, 13, 120, 18, 45, 14, 0, 400, 17, 800, 250, 15th, webtracker, 75, 36, 35, 1st, 70, 32, 95, 2nd, 19, 22, nonsmoking, 3rd, 1000, 5th, 18th, 42, ::, 21st, 555, 600, 25th, 21, mp3, 55, 20th, 2000, 900, acknowledgments, 14th, 16th, 7th, 27, wangfujing, 12th, 65, 180, 23, 2002, 26, 29, 6th, 59, mp4, 260, 507, 850, 2008, 123, 350, 201, 110, xxxxxxxxxx, pilferage, 17th, 101, 27th, 8th, 82nd, periodicals, 5000, 1234, 160, 401k, 05, 38, 4000, 1050, xiangqi, 78, 41, 34, 10th, noirin, 2004, 3000, palmistry, 1500, 66, hebes, 98, 1886, 31, 85, telegraphic, m25, 56, 48, 130, 700, nanchang, 64, 39, 86, 20s, 125, 261, 4th, 9th, 37, 19th, 325, 911, 11th, 43, 750, 28th, 360, 486, thirtieth, 24th, chequing, 308, 626, 1739, zhongshan, 92, 88, discomgoogolation, 84, 456, 5558929, airsick, 1996, 2001, 1980, 99, sandstorms,

moblogging, westernized, pilgrimages, 550, 1997, 650, 23rd, rikknen, airsickness, 320, 0n, 2006, 235, telephoned, 28, grangerfield, 345, richton, 513, 215, 57, iccc, 1218, anmen, jingshan, bankbook, 256, 87, 268, 1021, teahouses, 789, 6000, 232, balista, idiomatic, 1995, 58, 1960, 1985, 3g, magadize, 1978, 2005, 1963, 80s, literatures, 287, abacuses, 1999, 306, 29th, micropower, 51, 102, carlsborg, 8p, picnicking, onxiu, jinyuan, 103, 01, 68, 210, 402, 818, refitted, 267, stenography, 304, 1106, reconfirmed, consignee, guiling, 138,);, 508, huangshan, 31st, minored, 212, g3, 401, 502, 1019, 83, xeroxing, jd185649000023, 713, 455623, varietal, 725, 105, zhilian, zhaopin, typewriting, leisured, 1920s, 880, huanghe, 225, 560, 6pm, 209, 275, wudaokou, cloisonn, 207, danshui, 505, outstealing, noncommercial, 1808, lvan, desertification, vernassa, 282, decrescendo, pianissimo, accidentals, 1y, weightlessness, 1994, poesy, baymler, tailband, txyb, neuroanatomy, bankrupts, overcoats, discombobulate, 9p, 2003, turnbow, lavigen, 46, floatier, marquet, premedical, hemline, dormitories, spanishpod, chettri, 2010, 40c, 1980s and longly.

Spacy word vector

The following 51 tokens were not found in the Spacy word vector representation.

<pad>, <unk>, <s>, </s>, <d>, .., webtracker, mustn, ::, noirin, discomgoogolation, 5558929, moblogging, rikknen, grangerfield, richton, iccc, anmen, jingshan, bankbook, yarning, magadize, trusten, micropower, carlsborg, onxiu, jinyuan, guiling,);, jd185649000023, 455623, beancurd, zhilian, zhaopin, huanghe, wudaokou, cloisonn, danshui, outstealing, lvan, vernassa, baymler, tiao, tailband, txyb, turnbow, lavigen, floatier, marquet, spanishpod and chettri.

A.3 Trained word embeddings visual analysis

Set of images taken from the plot generated by the T-SNE algorithm from our self-trained word embedding, discussed in section 4.1.3. The images helped us to make a visual assessment of the quality of our word embeddings.

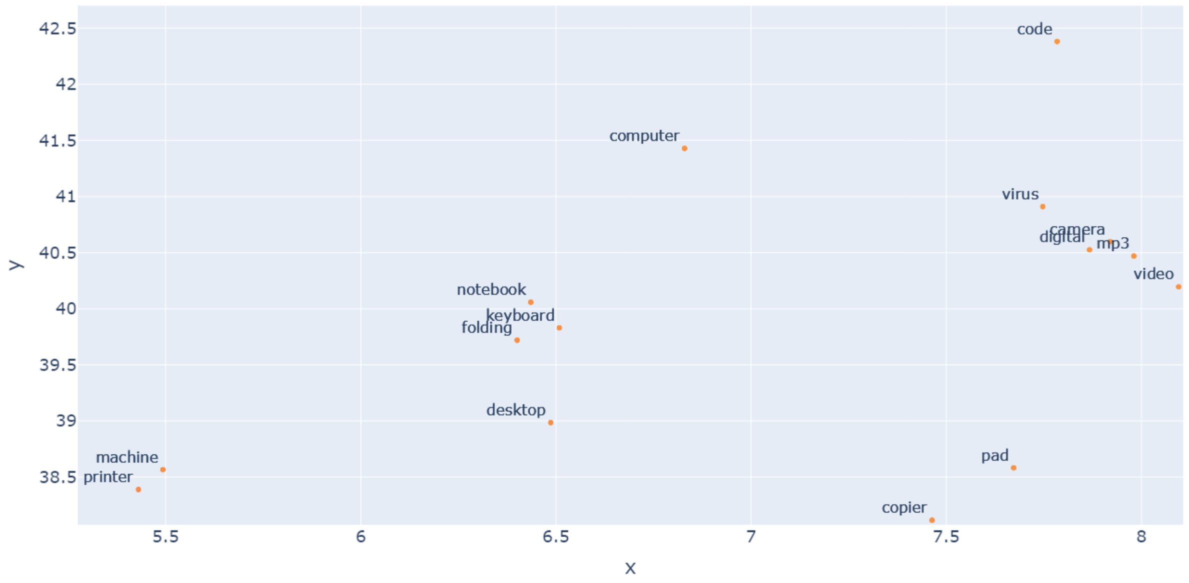


Figure A.1: Zoomed area where words related to ‘computer’ are represented.

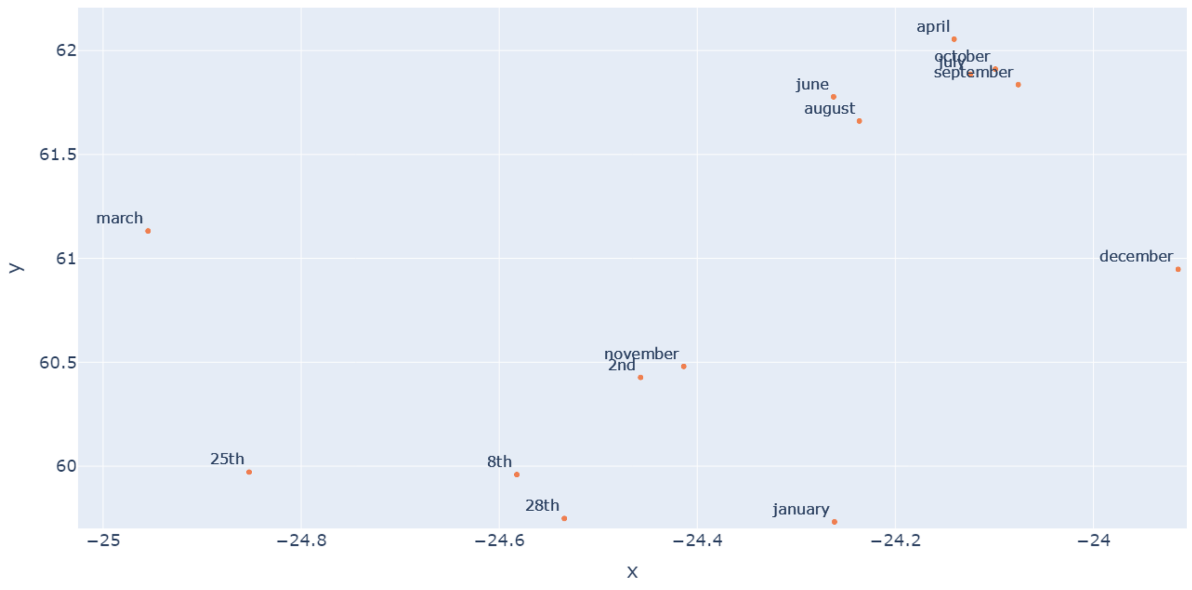


Figure A.2: Zoomed area where the months of the year and some dates are represented.

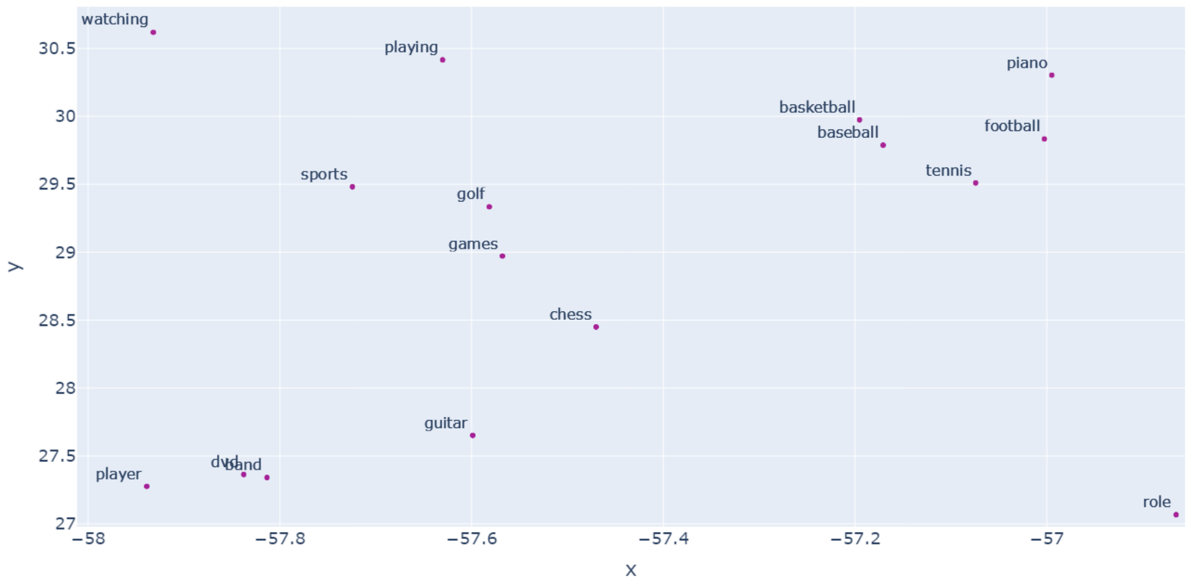


Figure A.3: Zoomed area where different words related with the verb ‘playing’ are represented.

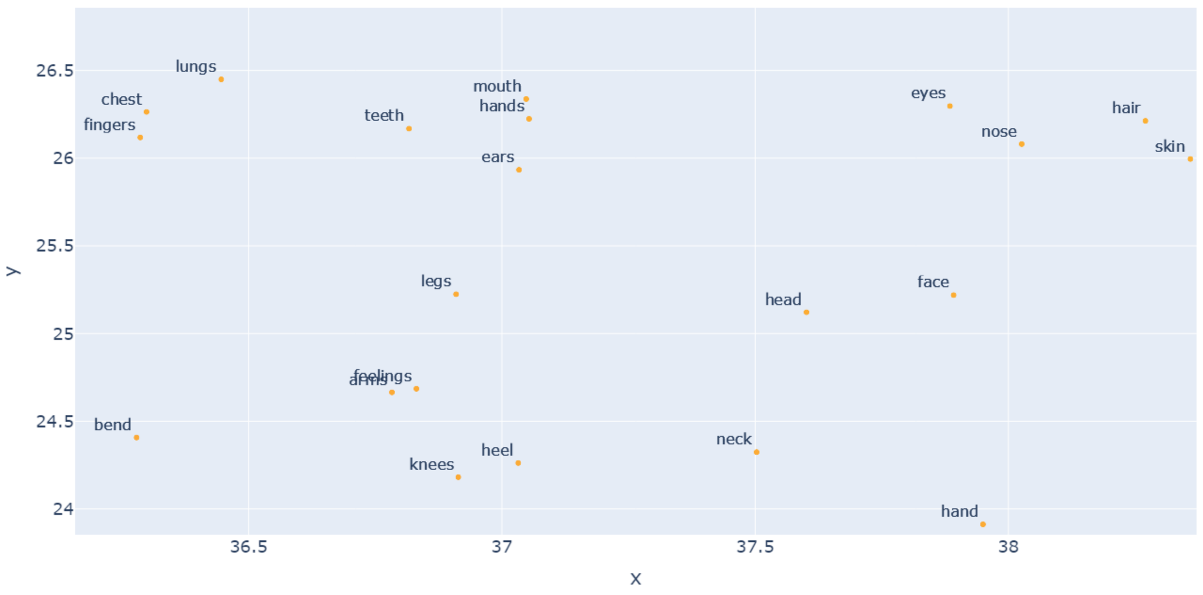


Figure A.4: Zoomed area where different body parts are represented.

A.4 BLEU score reduction in "no pre-trained" model

This section shows the results of the investigation related to the reduction of the BLEU score presented at the training time of the model without pre-trained word vector.

Epoch	Average Length	BLEU F1 score
20	8.6	0.3234
32	11.8	0.3274
46	13.7	0.3218
66	13.8	0.3235
86	15.6	0.3193

Table A.1: Average length of the generated responses at different epochs.

As the results shown in Table A.1, we conclude that the cause of the decrease of the BLEU score is due to the model generates more extensive sequences length as long as the training steps keep increasing.

A.5 Dialogue produced by the different models

The dialogues presented here were generated by applying the foreword dialogue "this saturday i'm thinking to go to the beach __eou__ that's wonderful, are you going with your family?", extracting the next token through greedy decoding and run the model in free mode.

Base model

FLOOR 0 : <d>

FLOOR 1 : this saturday i'm thinking to go to the beach

FLOOR 0 : that's wonderful, are you going with your family?

FLOOR 1 : yes. let me go to the beach and the beach.

FLOOR 0 : do you have any other?

FLOOR 1 : what's up ??

FLOOR 0 : yes, go straight and see a movie.

FLOOR 1 : would you like to have a drink?

FLOOR 0 : no, don't you? the food is so fresh.

Spacy embeddings

FLOOR 0 : <d>

FLOOR 1 : this saturday i'm thinking to go to the beach

FLOOR 0 : that's wonderful, are you going with your family?

FLOOR 1 : that's the best i want to do.

FLOOR 0 : for a while, i'm sure i'll be able to get you a little longer.

FLOOR 1 : oh! oh, tell me about that. i think i'll have to go to the gym with you.

FLOOR 0 : well, what if there's a party on the first floor of the road?

FLOOR 1 : no, right. let me see. oh, no, it's a bit too long.

FLOOR 0 : ok, ok, but...

Own-trained embeddings

FLOOR 0 : <d>

FLOOR 1 : this saturday i'm thinking to go to the beach

FLOOR 0 : that's wonderful, are you going with your family?

FLOOR 1 : i'm in a hurry.

FLOOR 0 : the new york?

FLOOR 1 : many people are in the same city.

FLOOR 0 : ok.

FLOOR 1 : haven't been able to get a new one?

FLOOR 0 : have anything else?

Without pre-trained embeddings

FLOOR 0 : <d>

FLOOR 1 : this saturday i'm thinking to go to the beach

FLOOR 0 : that's wonderful, are you going with your family?

FLOOR 1 : not much than mine.

FLOOR 0 : anything else?

FLOOR 1 : not a long time. we'll be able to get a better's next week.

FLOOR 0 : i am not sure.

FLOOR 1 : just take your left. the only thing is to be the best of the city.

FLOOR 0 : but there's nothing wrong with it.

A.6 Bert responses

Context	do you know the circulation of this magazine? _eou_ you mean this month?
Responses	**Bert as embeddings**
	Ex1: ##s ugly robert ya surgery begun ya ya statue ##gon ##ish sucks abby foster elect ##bby ##ve chatting kobe pajamas pajamas peas bloom gp ##bly commitment athletes ##lf
	Ex2: tones worms lifelong worms worms worms worms lifelong worms lifelong worms worms lifelong worms lifelong worms worms worms worms worms lifelong worms worms worms worms worms worms worms striped worms worms striped

Table A.2: Examples of generated responses using Bert model. _eou_ indicates a change of turn for the speaker.

Context	i wonder if you could help me, i'm looking for a room. _eou_ well, i have got a vacancy.
Responses	**Bert as embeddings**
	Ex1: abby m robert ya moon ya scottish ya statue ##gon ##ish critical bloom foster pajamas pajamas ##igen foster pajamas ##ad billy necessarily organizing target ##ty horn ya
	Ex2: based ugly cancelled ya surgery begun ya ya statue ##gon titanic 'critical abby foster

Table A.3: Examples of generated responses using Bert model. _eou_ indicates a change of turn for the speaker.

Bibliography

Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

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

Beltagy, I., Lo, K., and Cohan, A. (2019). Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Borji, A. (2019). Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

Cao, Z., Li, X., and Zhao, L. (2020). Unsupervised feature learning by autoencoder and prototypical contrastive learning for hyperspectral classification.

Chen, B. and Cherry, C. (2014). A systematic comparison of smoothing techniques for sentence-level bleu. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 362–367.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Colby, K. M., Weber, S., and Hilf, F. D. (1971). Artificial paranoia. *Artificial Intelligence*, 2(1):1–25.

Croce, D., Castellucci, G., and Basili, R. (2020). GAN-BERT: Generative adversarial learning for robust text classification with a bunch of labeled examples. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2114–2119, Online. Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.

Fedus, W., Goodfellow, I., and Dai, A. M. (2018). Maskgan: Better text generation via filling in the_. *arXiv preprint arXiv:1801.07736*.

Forgues, G., Pineau, J., Larchevêque, J.-M., and Tremblay, R. (2014). Bootstrapping dialog systems with word embeddings. In *Nips, modern machine learning and natural language processing workshop*, volume 2.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Gu, X., Cho, K., Ha, J.-W., and Kim, S. (2018). Dialogwae: Multimodal response generation with conditional wasserstein auto-encoder. *arXiv preprint arXiv:1805.12352*.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Honnibal, M. and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Humeau, S., Shuster, K., Lachaux, M.-A., and Weston, J. (2019). Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Khomenko, V., Shyshkov, O., Radyvonenko, O., and Bokhan, K. (2016). Accelerating recurrent neural network training using sequence bucketing and multi-gpu data parallelization. *2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP)*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kryscinski, W., Keskar, N. S., McCann, B., Xiong, C., and Socher, R. (2019). Neural text summarization: A critical evaluation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 540–551, Hong Kong, China. Association for Computational Linguistics.

Li, Y., Su, H., Shen, X., Li, W., Cao, Z., and Niu, S. (2017). DailyDialog: A manually labelled multi-turn dialogue dataset. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 986–995, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Liu, Y. and Lapata, M. (2019). Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*.

Luong, M., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. (2015). Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Le, Q. V., and Sutskever, I. (2013b). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

Mitchell, J. and Lapata, M. (2008). Vector-based models of semantic composition. In *proceedings of ACL-08: HLT*, pages 236–244.

Mnih, A. and Hinton, G. E. (2009). A scalable hierarchical distributed language model. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems*, volume 21, pages 1081–1088. Curran Associates, Inc.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML*.

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

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Park, Y., Cho, J., and Kim, G. (2018). A hierarchical latent structure for variational conversation modeling. *arXiv preprint arXiv:1804.03424*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proc. of NAACL*.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Rus, V. and Lintean, M. (2012). An optimal assessment of natural language student input using word-to-word similarity metrics. In *International Conference on Intelligent Tutoring Systems*, pages 675–676. Springer.

Sato, S., Yoshinaga, N., Toyoda, M., and Kitsuregawa, M. (2017). Modeling situations in neural chat bots. In *Proceedings of ACL 2017, Student Research Workshop*, pages 120–127.

Schnabel, T., Labutov, I., Mimno, D., and Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 298–307.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Serban, I. V., Sordoni, A., Lowe, R., Charlin, L., Pineau, J., Courville, A. C., and Bengio, Y. (2016). A hierarchical latent variable encoder-decoder model for generating dialogues. *CoRR*, abs/1605.06069.

Shen, T., Lei, T., Barzilay, R., and Jaakkola, T. (2017). Style transfer from non-parallel text by cross-alignment. In *Advances in neural information processing systems*, pages 6830–6841.

Shen, X., Su, H., Niu, S., and Demberg, V. (2018). Improving variational encoder-decoders in dialogue generation. *CoRR*, abs/1802.02032.

Shi, B., Bai, X., and Yao, C. (2015). An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *CoRR*, abs/1507.05717.

Shin, H.-C., Ihsani, A., Mandava, S., Sreenivas, S. T., Forster, C., Cha, J., and Initiative, A. D. N. (2020). Ganbert: Generative adversarial networks with bidirectional encoder representations from transformers for mri to pet synthesis.

Sordoni, A., Galley, M., Auli, M., Brockett, C., Ji, Y., Mitchell, M., Nie, J.-Y., Gao, J., and Dolan, B. (2015). A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Tolstikhin, I., Bousquet, O., Gelly, S., and Schoelkopf, B. (2017). Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Wang, A. and Cho, K. (2019). BERT has a mouth, and it must speak: BERT as a Markov random field language model. *Proceedings of the Workshop on Methods for Optimizing and Evaluating Neural Language Generation*.

Wang, J., He, H., and Prokhorov, D. V. (2012). A folded neural network autoencoder for dimensionality reduction. *Procedia Computer Science*, 13:120–127.

Wang, Y., Yao, H., and Zhao, S. (2016). Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242.

Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

Xing, C., Wu, W., Wu, Y., Liu, J., Huang, Y., Zhou, M., and Ma, W.-Y. (2017). Topic aware neural response generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Xu, Z., Liu, B., Wang, B., Sun, C.-J., Wang, X., Wang, Z., and Qi, C. (2017). Neural response generation via gan with an approximate embedding layer. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 617–626.

Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017). Seqgan: Sequence generative adversarial nets with policy gradient. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1).

Zhao, J., Kim, Y., Zhang, K., Rush, A., and LeCun, Y. (2018). Adversarially regularized autoencoders. In *International conference on machine learning*, pages 5902–5911. PMLR.

Zhao, T., Zhao, R., and Eskenazi, M. (2017). Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. *arXiv preprint arXiv:1703.10960*.