RADBOUD UNIVERSITY

MASTER'S THESIS

Natural language generation for commercial applications

Author:
Arianne van de GRIEND

Supervisors:
Wouter OOSTERHEERT
Tom HESKES

A thesis carried out at

Machine2Learn

in Amsterdam

for the Computing Science Master

December 23, 2018

RADBOUD UNIVERSITY

Abstract

Machine2Learn

Computing Science Master

Natural language generation for commercial applications

by Arianne van de GRIEND

This master thesis gives an overview on natural language generation with the focus of dialogue systems for commercial use.

We give a description of the general approach to natural language generation and their neural architectures first.

Then three application domains are discussed in more detail: language style transfer, dialogue response generation and controlling dialogue response generation.

For each domain, a use case was implemented and the results are discussed. We investigated automatic customer support, an empathetic automatic customer support and sentiment adjustment of reviews. We show promising results for the first two use cases, but the last use case was inconclusive due to difficulties with implementation.

We finish with a short discussion of the use of natural language generation in commercial applications and what can be improved in our current model architectures.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

From recent news coverage, one might conclude that the current techniques of *natural language generation* (NLG) are very powerful. There have been reports of an AI that coauthored a book and entered a literary contest (Shoemaker, 2016). Even the recent controversy surrounding Microsoft's chatbot *Tay* (Vincent, 2016) makes it seem like NLG is not some concept from science fiction anymore. However, the actual functioning of these applications is not publicly known. Which begs the question: What are the current limitations of natural language generation for real world scenarios?

In this master thesis, we will describe our investigation of the limitations of NLG with respect to commercial applications. As such, we focus on *chatbot* applications, since automated customer support applications already exist. But do the existing customer support systems actually use NLG? We have investigated a selection of these applications (see section 6.1) and realised that, in terms of NLG, not much has changed since the first chatbot, *Eliza* (Weizenbaum, 1966). But can these applications be improved with NLG?

Thus a more exploratory approach to NLG for commercial applications is taken. We have investigated the existing state-of-the-art algorithms and applications using NLG. We selected three application domains for NLG: *dialogue response generation*, controlling generation and *language style transfer*. Each domain was investigated in more detail and a specific use case was implemented for that domain.

The focus of the use cases was on end-to-end applications, since this required less task-specific data that was not available. As such, we focused on the following three use cases. We investigated NLG for an automated customer support (section 6.7). We attempted to control the sentiment of the generated responses to create an empathetic customer support (section 7.2). And we tried to adjust the sentiment of Yelp reviews using *language style transfer* (section 8.3).

The thesis itself follows a similar structure that contains three parts: the general approach to NLG (Part I), the three application domains (Part II), and the conclusions and discussions (Part III).

In Part I, we describe the current approach to any NLG application. We first describe how text can be represented numerically as a sequence of symbols

or words (*tokens*) in Chapter 2. Then, we describe how the meaning of the text can be captured in a single vector using a *language model* (Chapter 3). Combining a language model with a *generation model* creates the *NLG pipeline* that can be used to generate text token-by-token (Chapter 4).

In Part II, each chapter describes one of the three application domains in more detail. The last section of each chapter describes the implemented use case belonging to its application domain.

The first application domain, *dialogue response generation*, is discussed in Chapter 6. An overview of existing applications is given first (section 6.1). Followed by an overview of training methods for these dialogue systems (section 6.2) and evaluation methods (section 6.3). Dialogue response generation is a challenging task. The three most occurring challenges are generating meaningful, relevant and consistent responses, as discussed in section 6.4. These challenges may be overcome by adding context (section 6.5) or by adding knowledge (section 6.6) to the dialogue system. Lastly, we describe the automated customer support use case (section 6.7).

The second application domain tries to add external control to NLG (see Chapter 7). Current research on this topic is mainly focused on politeness. An overview of this is given in section 7.1. However, these techniques can also be adjusted to control other aspects of the generated sentence, like sentiment. This is shown in section 7.2, where we discuss the empathetic customer support use case.

The last application domain is *language style transfer* (LST) (Chapter 8). This is a technique of adjusting a text such that its information is the same, but the style of the text is different. There are two possible approaches to LST, with parallel data (section 8.1) or without parallel data (section 8.2). The LST use case tries to replicate the sentiment adjustment of Yelp reviews by Shen et al. (2017) such that it may be used to improve responses from *retrieval-based chatbots* (section 8.3).

Lastly, in Part III, we will discuss the results with respect to three topics. First, the current challenges with NLG (Chapter 9), aside from those discussed in section 6.4. Then, the effects of the chosen dataset on the performance of the dialogue system (Chapter 10). And lastly, the applicability of NLG for commercial applications (Chapter 11).

Part I

Natural Language Generation

Chapter 2

Sentence representation

The first step in language generation is defining a general representation of a text. This is done by splitting the text into smaller components, so called *tokens*. These tokens need to be described in a numerical way, in order for a computer to calculate with them. This chapter will discuss the different choices that can be made when splitting a piece of text into tokens, a process called *tokenisation* (section 2.1), and how these tokens can be transformed into (a sequence of) numbers, an *embedding* (section 2.2).

2.1 Tokenisation

Tokenisation is the process of splitting a text into smaller components, *tokens*. When tokenising, the text is first preprocessed and it can then be split into different kinds of components: symbols, words, and sentences.

Preprocessing a text, before tokenising it, allows the tokenisation algorithm to produce the same tokens regardless of the format the text is in. Usually, all letters in the text are changed into lower case and any format annotation (e.g. HTML tags) is removed. Sometimes it is necessary or useful to replace specific information by a general tag. For example, by replacing a username with the string _name_. For some applications it is useful to indicate the start or end of the text (or both) with a specific, reserved token (e.g. "<start>" and "<eos>", respectively).

After preprocessing, the text can be tokenised. This is relatively straight forward when using *symbol tokens*. The text itself is already represented as a sequence of symbols. The main strengths of this type of tokens is that there are only a limited amount of symbols (the alphabet and some punctuation) and that small set of tokens can create any sentence. Unfortunately, a drawback is that symbols on their own do contain much information about the text. This can be a problem for *natural language generation* (NLG) because most sequences of symbols do not create a sentence. The number of possible sequences generated is of a different order of magnitude than the number of possible correctly generated sequences. This makes it difficult for an algorithm to learn how to generate natural language.

Original	"<p> Hi @Guy98! How are you? <\p>"
Preprocessed	"<start> hi _name_! how are you? <eos>"
Symbol token	["<start>", "h", "i", " ", "_", "n", "a", "m", "e", "_", "!", " ", "h", "o", "w", " ", "a", "r", "e", " ", "y", "o", "u", "?", "<eos>"]
Word token	["<start>", "hi", "_name_", "!", "how", "are", "you", "?", "<eos>"]
Sentence token	["<start>", "hi _name_!", "how are you?", "<eos>"]

TABLE 2.1: An example text that is preprocessed and tokenised using different token types.

Tokens that better represent the meaning of a text are *word tokens*. For generation, these tokens have the added benefit that using words as smallest sentence component causes that less strings can be generated. It also has the effect that poorly generated sentences can still be received as fine because it contains sensible components. A random sequence of symbol tokens will seem as if someone sat on a keyboard, where a random sequence of word tokens seems to contain information because the word tokens have meaning. However, there are a lot more possible word tokens than symbol tokens and tokenising a sentence into word tokens is not trivial. In English, most words are separated with spaces, but this is not always the case. You may want to tokenise the word "cannot" into "can" and "not", such that it is analogous to "do not". Similarly, punctuation should be represented as different tokens, even though they may not be separated with spaces from the other tokens. Words can also be abbreviated and combined, which needs to be taken into account when tokenising as well. Since word tokenisation is quite involved, we used a predefined implementation (NLTK[1]).

Lastly, a text can also be tokenised on a sentence level. Splitting a text into *sentence tokens* can be done using simpler rules than with word tokens. This token type is less expressive than the others, but that can be useful in some cases. This particularly holds for texts that have little variability (e.g. reports), such that the amount of unique sentence tokens stays small. When replacing specific information with placeholder tags, the generated text can be considered a template to be filled in, with the placeholders marking where to insert the new information. However, this token type is not suitable for most tasks.

The effect of preprocessing and tokenisation on an example sentence is visualised in Table 2.1.

2.2 Token representation

Once a text is tokenised, the tokens need to be transformed into numbers that the computer can calculate with. This is called *encoding*. The mapping from

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

token to numbers, and vice versa, is called an *embedding*. Transforming the numbers back into a token is called *decoding*. Each embedding is defined on a *vocabulary*, the set of tokens for which the mapping is defined. Sometimes, a text contains tokens that are not in the vocabulary, *out-of-vocabulary words*, and cannot be encoded. For these cases, the vocabulary contains a special "<unk>" token, representing all unknown words.

The simplest type of embedding is a *one-hot encoding*. This is an embedding where each unique token is simply numbered. The tokens are encoded by simply replacing them with their corresponding number and decoded by rounding the number and selecting it's corresponding token. A special case of this embedding is *sparse one-hot encoding*. Instead of encoding to a single number, this embedding encodes to a sequence of zeroes with a 1 on the location corresponding to the number of the token. The token represented by a sequence of numbers is determined by finding the location in the sequence where the value is largest.

Although one-hot encoding can be used for any token type, it may not be suitable for most tasks. An encoded token does not contain any meaningful information about the token itself. Values that are close to each other do not necessarily belong to tokens that are similar in some way. Similarly, sparse one-hot encoded tokens have the same distance to every other token. Since symbols do not contain any meaning without context, this embedding is very suitable for symbol tokens.

Word tokens do benefit from an embedding that represents the meaning of the words. This meaning representation is defined by a vector. Vectors that are close to each other represent words that are close to each other. This type of embedding needs a *semantic model* that needs to be trained. Most semantic models are based on word co-occurrences. This is based on the idea that words that occur in similar sentences (with the same words) must have a similar meaning. Depending on the semantic model, words are encoded into different vectors. Two commonly used word embeddings are *Word2Vec* (Mikolov et al., 2013b) and *Global Vectors* (GloVe)[2] (Pennington et al., 2014).

Since these *semantic model-based embeddings* need to be trained, it is needed to investigate if the obtained vectors actually represent the meaning of the tokens. This is done by evaluating the quality of the vectors. A straightforward approach is to randomly select some tokens from the vocabulary and retrieving which words in the vocabulary a represented by a vector close to the encoded selected token. The selected tokens and their closest tokens can be inspected manually to determine if the similar vectors represent similar words. However, manual inspecting is never ideal and this method will not determine whether the most similar vectors belong to the most similar tokens.

[2] https://github.com/stanfordnlp/GloVe

To solve this, Mikolov et al. (2013a) proposed a *word analogy task* for evaluating the quality of a trained embedding. This task uses the idea that the difference between vectors of two tokens should also be similar to the difference between vectors of two analogous tokens. This is done with a predefined list of 4-tuples; two tokens that have some semantic or syntactic relation and two analogous tokens (for example "usa", "dollar", "europe" and "euro"). For each tuple, the numerical distance between the two first tokens is added to the third token's encoding. The result is decoded and checked if it is indeed the fourth token ("europe"+("dollar"-"usa")="euro"). The more accurate this prediction is, the better the encoding.

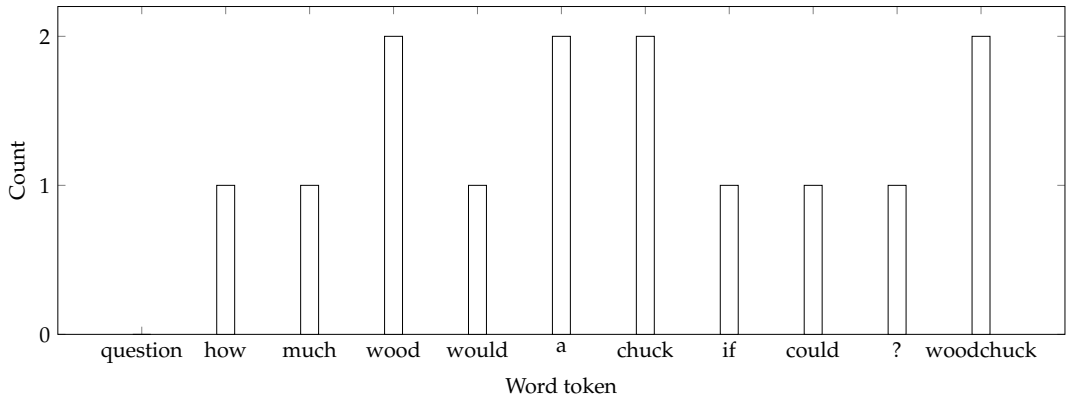
A major drawback of this word analogy evaluation is that the vocabulary needs to contain all tokens in the predefined list of 4-tuples. In addition the texts that were used to train the embedding need to use these semantic or syntactic relations in the sentences. A corpus of tweets may contain the words "usa", "europe", "dollar" and "euro", but unless the corpus contains sentences indicating the currency relation, the embedding may never learn it. Thus, from the training sentences like "usa here I come!", "europe, here I come!", "it was 8 dollar", "it was 15 euro...", the embedding cannot learn to answer the question "if usa is to dollar, then europe is to ...?".

This means that the quality of an embedding depends on its training data. This entails that the performance of embeddings trained on different datasets may not be comparable. Another effect is that embeddings need to be trained on a dataset that has a similar language style to the language used in the model that the embedding is used in. This can be quite tricky when training a model on a dataset that is too small to create a good embedding. In that case, a *pre-trained embedding* can be used. This is an embedding that is already trained on a different dataset.

Pre-trained embeddings can be very useful for many applications. However, the embedding determines the vocabulary. This may not be a problem since pre-trained embeddings usually have a large vocabulary, but it can be a problem when the vocabulary needs to contain important tokens (e.g. "<start>", "<eos>" or "_name_"). It is not possible to just define a vector for each out-of-vocabulary token, because this violates the assumption that similar tokens have similar representations. A possible workaround for this problem is to add a new dimension to the vectors in the embedding for each new token and setting the values in the new dimensions to 0[3]. The new tokens are added using a one-hot encoding with the ones only in the new dimensions. This trick maintains the trained information space of the original tokens and defines an embedding for the new tokens. However, the distance between the new tokens and all other tokens does not represent meaning.

Sentence embeddings are usually build from the word tokens in the sentence. The simplest representation is a sequence of embedded word tokens. Sentences can be one-hot encoded as well, of course, but those encodings do not

[3] https://stackoverflow.com/questions/45113130/how-to-add-new-embeddings-for-unknown-words-in-tensorflow-training-pre-set-fo



This results in the vector representation: [0,1,1,2,1,2,2,1,1,1,2].

FIGURE 2.1: *Bag of Words* (BoW) representation of the sentence "How much wood would a woodchuck chuck if a woodchuck could chuck wood?".

represent the meaning of the sentence itself. Instead, sentences may be represented using a histogram of word token occurrences in the sentence called a *Bag of Words* (BoW), see Figure 2.1. However, a BoW does not contain any information on the word order in the sentence. Word order can be added by using *n-grams* instead of word tokens. N-grams are n-tuples of n tokens occurring in sequence (e.g "how much", "much wood", "wood would" etc.). A histogram of occurrences of sequences of words, a *Bag of n-grams*, does contain some word-order information. However, the larger the n, the more unique n-grams, the wider the histogram (more bins) and thus the sparser the representation (more zeroes).

In order to better represent the meaning of a sentence, sentences can be embedded using a *language model*. Like a word embedding, a language model maps a sentence to vector. The difference lies in that a word embedding is trained from the tokens themselves and a language model is trained from a sequence of (word) tokens that represent the sentence token. Thus the resulting embedding maps the sentence token as sequence of (sub)tokens to a vector, instead of the sentence token itself. These language models will be discussed in more detail in Chapter 3.

Chapter 3

Neural Language models

A *language model* defines a mapping from a text or sentence to a number or vector. This means that the sentence *embeddings* described in the previous chapter can be used as a language model and vice versa. This chapter will focus on language models that are based on *neural networks*. These neural language models are more suitable for language understanding than simpler word frequency based models, like the *Bag of Words* (BoW) model (Kannan et al., 2016). Analogous to the *semantic model-based embeddings* described in Chapter 2, the *neural language models* define a representation that takes the meaning of the input text into account.

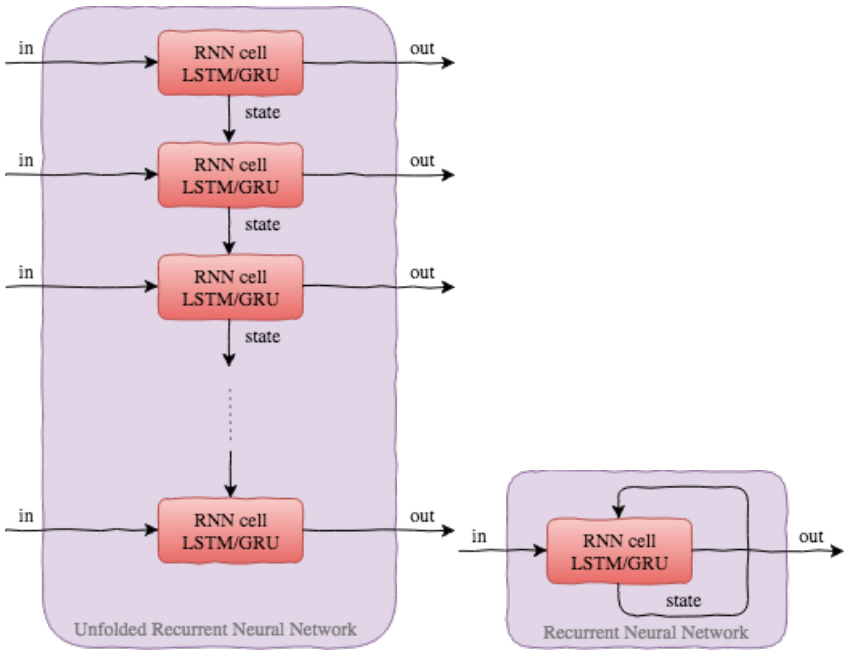
Language models are often used for *text analysis*. The vector that the text is mapped to represents a property of the text that needs to be investigated. These models are often trained using *supervised learning* (see section 6.2). The texts and the (numerical) properties of the texts are both fed into the model for training. An example of this is *sentiment analysis*, where the texts are split into positive and negative texts before training. The trained model can indicate whether a new text has positive or negative sentiment.

Labelling texts is more difficult if the property is more abstract or less well-defined. When the language model should represent the text's meaning in a fine-grained way, labelling is impossible. In that case, the language model can be trained using *unsupervised learning* (see section 6.2). It is also possible to train a language model *indirectly*. This is done by using the model as part of a larger model that can be trained using supervised learning.

In this chapter, two styles of neural language models will be discussed. Both styles use a different neural network architecture: a *recurrent neural network* (RNN) (section 3.1) and a *Convolutional Neural Network* (CNN) (section 3.2).

3.1 Recurrent language models

The use of a *recurrent neural network* (RNN) as language model was first proposed by Mikolov et al. (2010). An RNN is a neural network with a linear recursive structure that is very suitable for sequences. At each step, the RNN takes an element of the input sequence as input and produces an output and



(A) unfolded (B) folded

FIGURE 3.1: The basic structure of a *recurrent neural network* (RNN) that takes a sequence of input data (a). This can also be described per element in the input sequence (b).

a state. The state is used in the next step to condition the next output. The state represents the memory of the previous in- and outputs. This type of sequential structure is very suitable for processing sequential data such as language.

The way in which the input and the previous state of an RNN determines its output and its next state is determined by the *RNN cell*. This is the basic unit of the RNN that is called recursively. In theory, this cell or *RNN unit* can be any type of operation that takes an input and state as input and outputs an output and new state. In practice, two types of cells are used: the *Long Short Term Memory* (LSTM) (Hochreiter and Schmidhuber, 1997) and the *Gated Recurrent Unit* (GRU) (Cho et al., 2014). The basic RNN architecture is visualised in Figure 3.1.

In language, the meaning of a word in a sentence is not only determined by the previous words, but also by the future words. This is taken into account with a *Bidirectional Recurrent Neural Network* (BiRNN) (Schuster and Paliwal, 1997). This is an RNN with two *RNN cells*, one for the original sequence order, resulting in *forward states*, and one for the reverse sequence order, resulting in the *backward states*. This architecture is shown in Figure 3.2.

To summarise, an RNN can process a sequence of tokens in a text and it keeps track of a state that represents the memory of the previous tokens. This makes RNNs very suitable as language model. After processing a text, the resulting state can be used as a feature vector to classify certain properties of

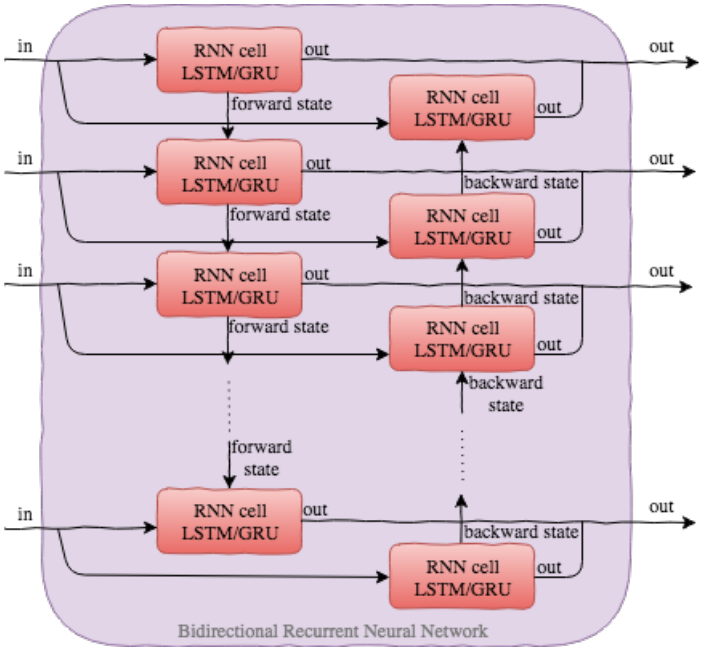


FIGURE 3.2: A schematic representation of the *Bidirectional Recurrent Neural Network* (BiRNN) architecture.

the text. The next chapter, Chapter 4, shows how this vector can also be used to condition a text generation model.

Due to the forward sequential nature of RNNs, they are biased towards the last item in the sequence. This could be the reason why they perform better when the input is reversed (Sutskever et al., 2014). *Convolutional Neural Network* (CNN) models do not have this bias (Kalchbrenner et al., 2014). They will be described in the next section.

3.2 Convolutional language models

A *Convolutional Neural Network* (CNN) (Zhang et al., 1990) is a neural network with one or more *convolutional layers*. These layers calculate the convolution (a mathematical operation) of sections of the input. The sections are defined by its size; the *window*, and its *stride*; how to move the window over the input. To train the relevance of the convolution results, the convolutions are weighted and a bias is added. The combination of the convolution, the weights, and the bias is called a *filter*. Applying a trained filter results in a *feature map*; a vector that represents some abstract feature in the input. A filter with the same convolution, but different sets of weight-bias pairs result in different feature maps. Thus, a *convolutional layer* is described by the window and stride for the convolution operation, and the number of resulting filters.

Given a fixed-length sequence of tokens, the filters of a CNN can describe more abstract features of the *n-grams* in the sequence. The size of the n-grams

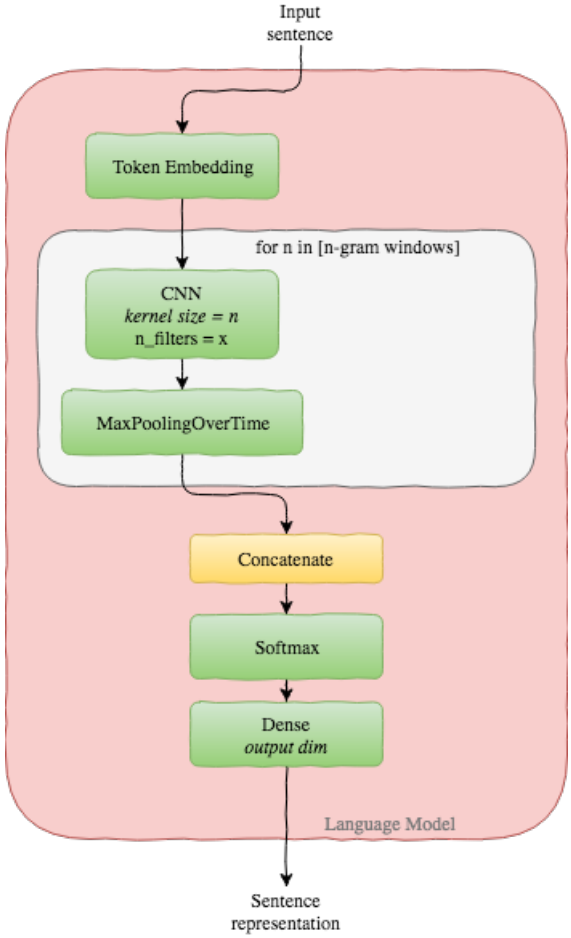


FIGURE 3.3: The CNN language model architecture as proposed by Kim (2014).

is determined by the window size of the filter. This makes CNNs suitable for language models as well.

There are two main approaches to using multiple convolutional layers in a language model: deep and broad. In a deep CNN, the layers are stacked; the output of one layer is fed to the next. This approach is usually used in image recognition, but it can also be used in *Natural Language Processing* (NLP). An example of this is the *Dynamic Convolutional Neural Network* (DCNN) proposed by Kalchbrenner et al. (2014) for *language understanding*.

The other approach combines the convolutional layers side-by-side, making a wide network. Each layer gets the same input sequence of tokens, but their filters have different window sizes. Thus the resulting feature maps represent features of n-grams of different sizes. This allows the language model to distinguish between "terrible book" and "not terrible book" by architecture design. An example of this type of language model is proposed by Kim (2014). They added a *max pooling over time layer* to determine which feature maps are most prominent in the input sentence for each n-gram (window size). Their architecture can be seen in Figure 3.3. It is a simple architecture that can learn high quality sentence representations (Zhang et al., 2016).

The size of the filters of a CNN depends on the length of the input sentence. This means that a CNN requires a fixed input length. Input sentences can be padded to adjust its length, but these padding tokens will be used in the convolutions and thus skew the results. Conversely, the weights of an RNN do not depend on the sentence length due to its recursive definition. In practice, the input for RNNs is padded for technical reasons[1], but these padding tokens do not need to be taken into account when training. This makes CNNs less suitable for applications with input text of varying length.

[1]GPUs run in lockstep, performing the same calculations over a number of different inputs at the same time, thus preferring fixed length input. Similarly, the Keras library requires a numpy array as input which also is also of fixed length. And the Tensorflow library does not free GPU memory during training, resulting in more memory usage when training on batches of different sizes.

Chapter 4

Neural generation models

The task of *natural language generation* (NLG) is considered to be a sequence-to-sequence problem where both the input and output is a sequence of variable length. This group of problems can be solved by first encoding the input to a fixed length representation and then decoding that representation back to a sequence. This latent representation can be created using a *language model* (Chapter 3). This chapter will discuss the *generation models* that can generate the sequence from the sentence encoding.

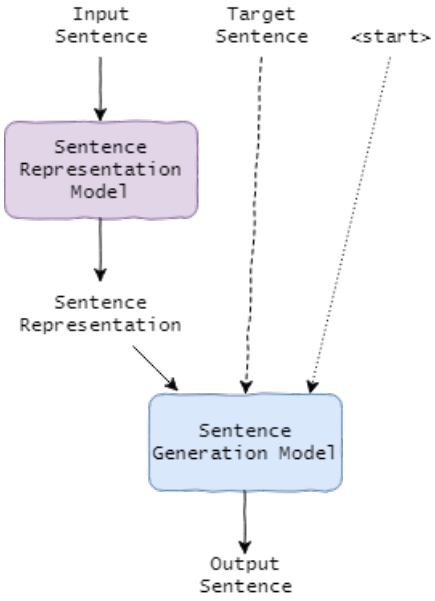
In practice, the two models are combined and trained together (see Figure 4.1), we call this the *NLG pipeline*. This way, the language model does not need a ground truth for the sentence representation, instead, the model can learn what representation is suitable. In fact, this explicit separation of sentence representation model and generation model is usually not made. Instead, the models are presented as a whole, not as a combination of two separate submodels that can be replaced with different models or different model architectures, independently.

4.1 The sequence-to-sequence model

A commonly used algorithm for NLG is the *Sequence-to-Sequence* (Seq2Seq) model (Sutskever et al., 2014). It was originally created for *Neural Machine Translation* (NMT) and later adapted for other domains, like *dialogue response generation* (Vinyals and Le, 2015).

The Seq2Seq model consists of two RNNs; one as sentence representation model and one as generation model. The state of the first RNN is used to initialise the state of the second one. The second RNN generates a sequence of token representations that are then decoded to form the generated text. This structure is visualised in Figure 4.2.

The Seq2Seq model has two main drawbacks. First, the generation model "forgets" the input sentence because the output of the sentence representation model is only used to initialise the state of the generation model. Since this state is adjusted each time step to store the previously generated output,



The dashed and dotted lines indicate training and testing steps, respectively.

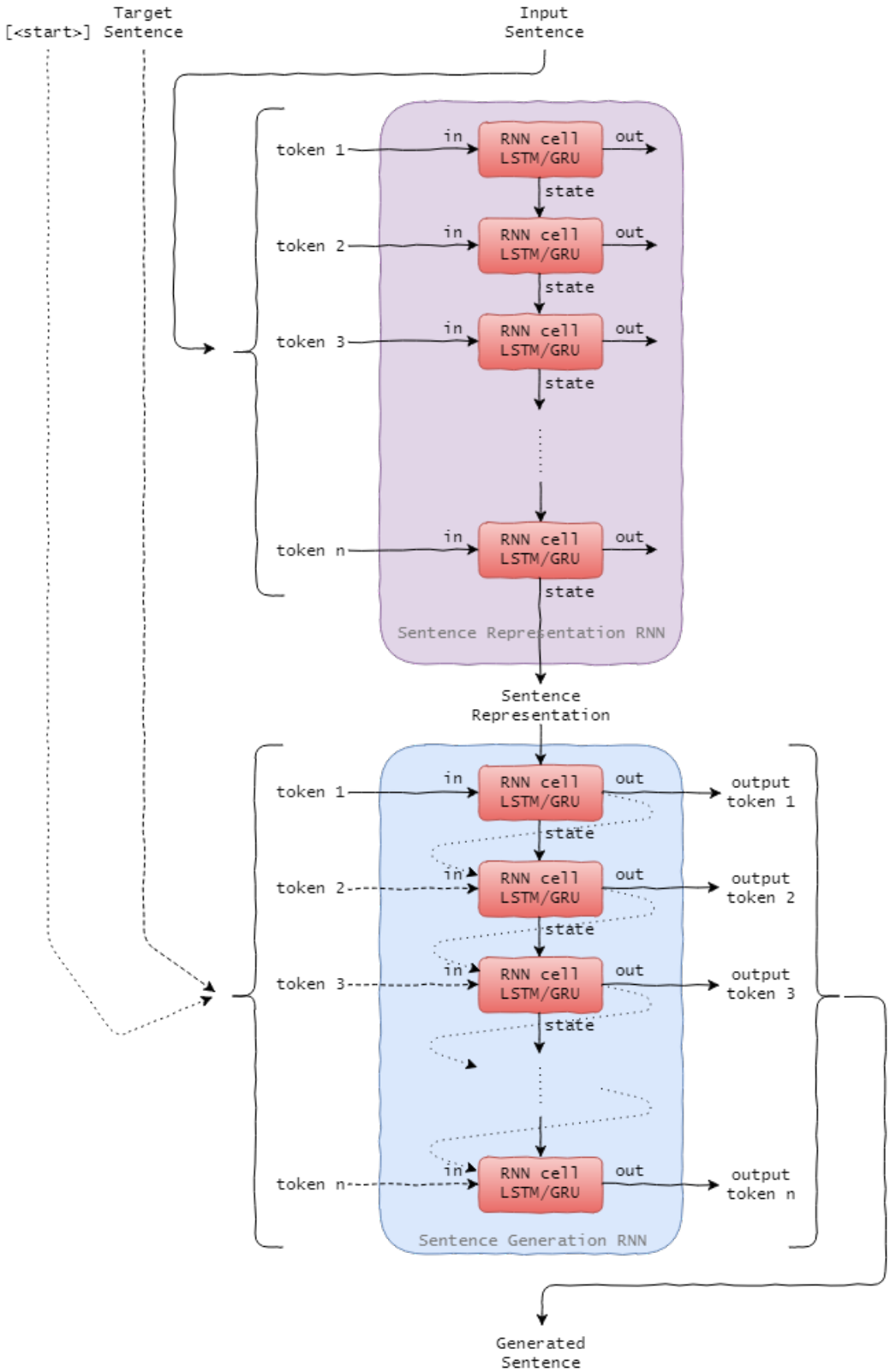
FIGURE 4.1: A schematic representation of how a *language model* and a *generation model* can be combined to be jointly trained for *natural language generation* (NLG).

the information from the sentence representation model will be lost after sufficient steps. This can be solved using the *attention mechanism* (Bahdanau et al., 2014) discussed in section 4.1.1 or the *NewGCA* architecture (Ludwig, 2017) described in section 4.2.

The second drawback is due to the difference between the generation model input when training and when using the model. During training, the generation model receives the full target sequence, this is called *teacher forcing* (Williams and Zipser, 1989) (see also section 6.2). However, when using the model, the full generated sequence is not yet known and the output of each time step is looped back as input for the *RNN cell*, as is visible in Figure 4.2. This means that the created error when using the model is propagated over each following step, resulting in worse performance than expected. A solution to this is training the model using *professor forcing* (Lamb et al., 2016). This is described in section 4.1.2.

4.1.1 The attention mechanism

The *attention mechanism* is an adaptation to the Seq2Seq model proposed by Bahdanau et al. (2014) that is three-fold. First, the sentence representation model is replaced with a *Bidirectional Recurrent Neural Network* (BiRNN). Secondly, the representation is not created from only the last state in the model, but it is a *linear combination* of all forward and backward states in the BiRNN.



The dashed and dotted lines indicate training and testing steps, respectively.

FIGURE 4.2: A schematic representation of the basic *Sequence-to-Sequence* (Seq2Seq) model introduced by Sutskever et al. (2014).

Lastly, the resulting representation is used for the states of the generation model at each time step. This architecture is visualised in Figure 4.3.

The *attention mechanism* has multiple benefits for training a Seq2Seq model. Using the sentence representation in the generation model at each time step, allows the model to keep paying attention to the input sentence. The use of the BiRNN allows the sentence representation to also reflect the relation of previous tokens, as well as future tokens, to each other (Bahdanau et al., 2014).

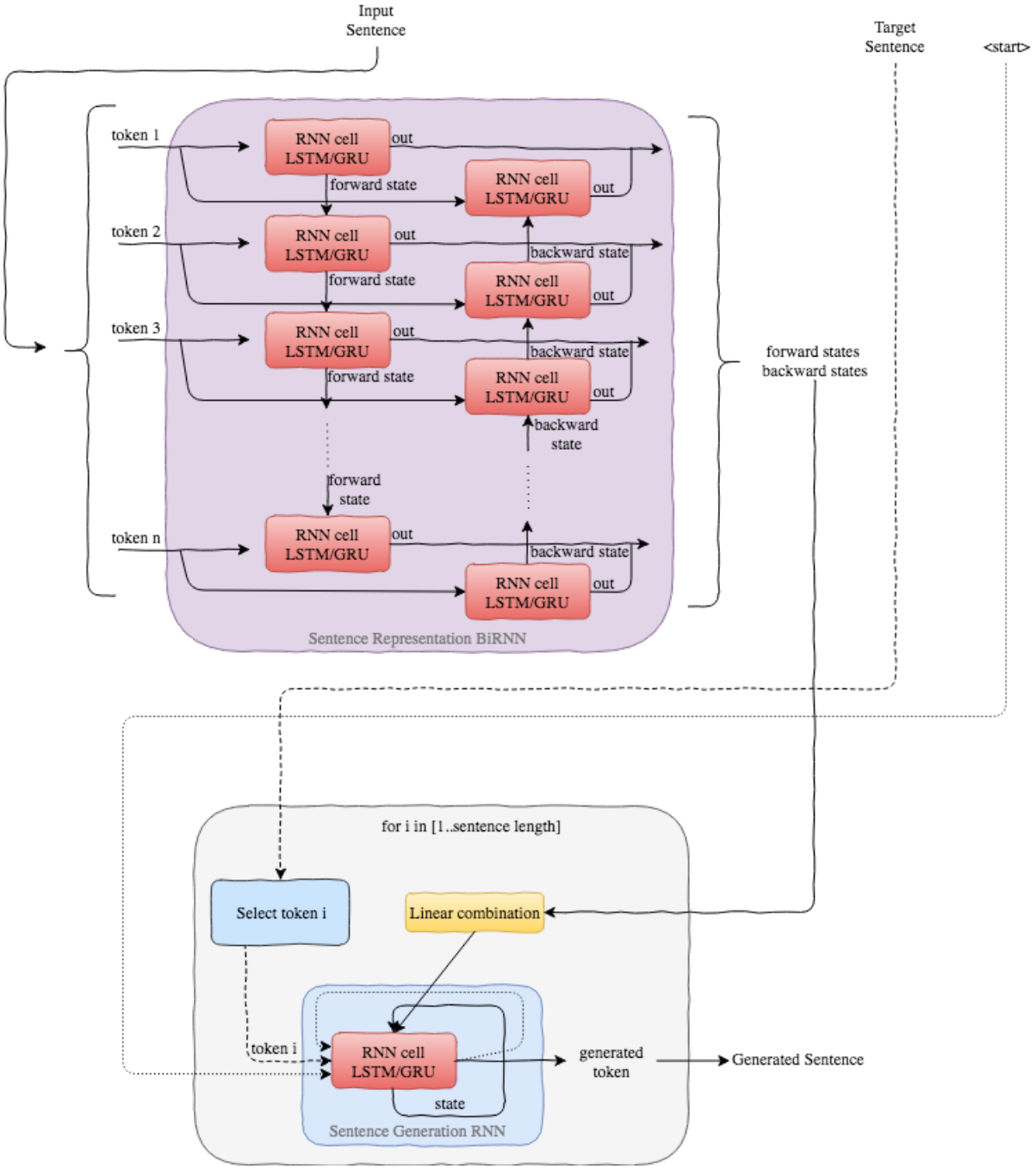
The linear combination of the forward and backward states in the BiRNN is determined by a set of weights that is obtained through training. This set of weights is different for each time step of the *generation model*. This allows the generation model to learn which states of the sentence representation model are important at each time step (Bahdanau et al., 2014). Thus, the Seq2Seq model does not only learn which input tokens the generation model should pay attention to, but also when it should not.

However, learning which states of the sentence generation model are important adds more computational complexity. The *attention mechanism* has to compute the attention weight for each token in the input for each token in the output sentence. This can make it intractable for some applications (Bahdanau et al., 2014). For instance, when automatically generating replies to long emails.

4.1.2 Professor forcing

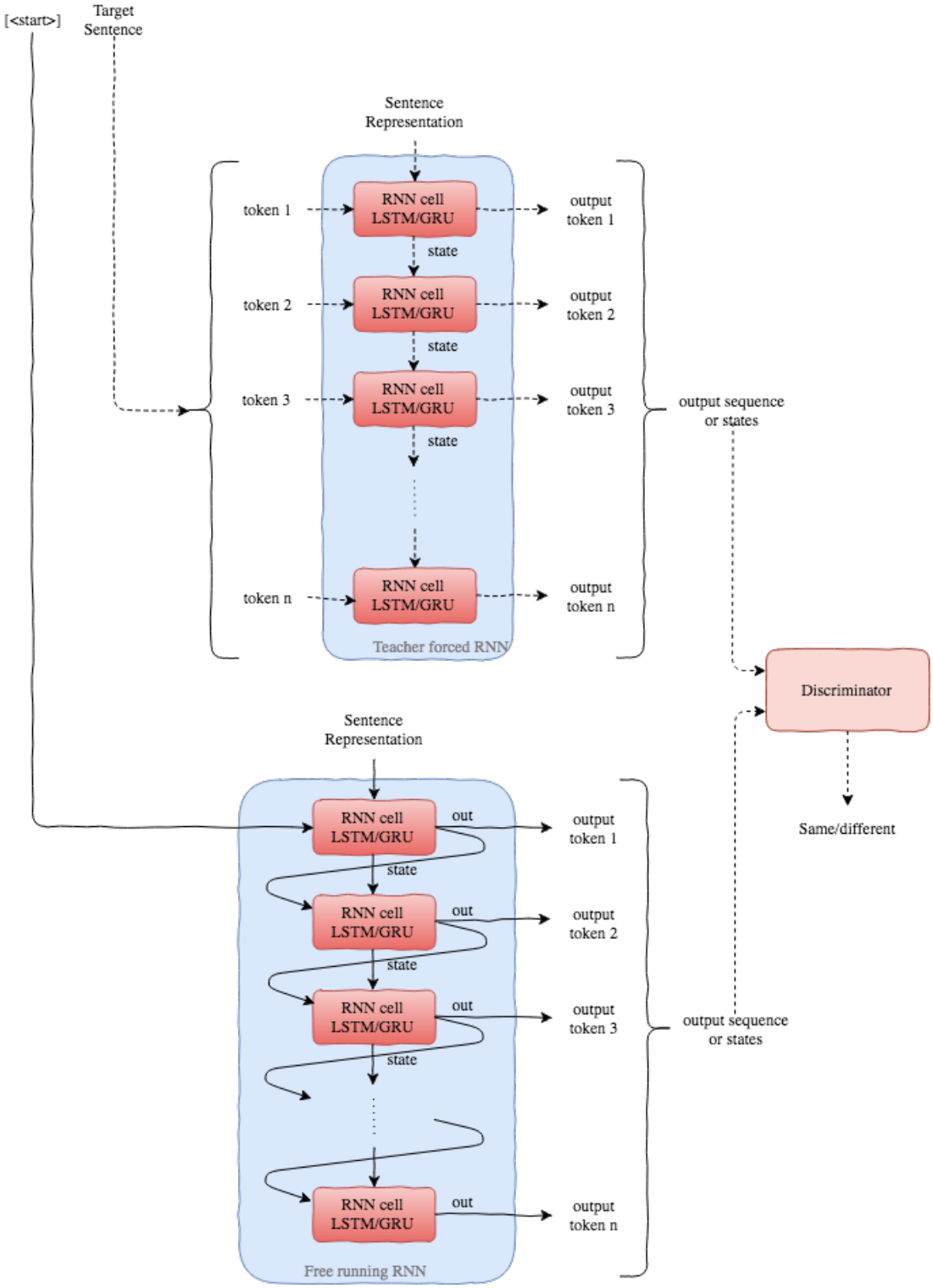
Professor forcing (Lamb et al., 2016) is a training technique for RNNs. The idea behind professor forcing is that an RNN should give the same results when a ground truth is given as input (when training, *teacher forcing*) as when the output is looped back into the next step (when testing, *free running*). This can be forced by training a *discriminator* that classifies if the output is created with a teacher forced model or a free running model. Training a model using a discriminator that needs to be fooled is called *adversarial learning* (see also section 6.2.3). The professor forcing architecture for a single RNN is shown in Figure 4.4 and it can easily be adjusted for training a Seq2Seq model.

The *discriminator* is a separate model that can be trained using either the generated sequences or the sequence of corresponding states from the generation model created in both modes; teacher forced and free running. The output of the discriminator is used in the loss of the RNN, together with the original generation loss. A good RNN should be able to give the same output in free running as it did when teacher forcing. If it doesn't that means that it is *overfit* for knowing the truth. Applying the discriminator output to the loss of the RNN forces the RNN to produce sentences or states that are indistinguishable with respect to teacher forcing and free running. Thus the difference between the teacher forced performance during training and the free running performance during inference becomes smaller. This causes that



The dashed and dotted lines indicate training and testing steps, respectively.

FIGURE 4.3: A schematic representation of the *attention mechanism* proposed by Bahdanau et al. (2014).



The dashed lines indicate training steps.

FIGURE 4.4: A schematic representation of the *professor forcing* technique (Lamb et al., 2016).

the generation loss of the model better represents the actual performance of the model when in use.

4.2 New generative conversational agent model

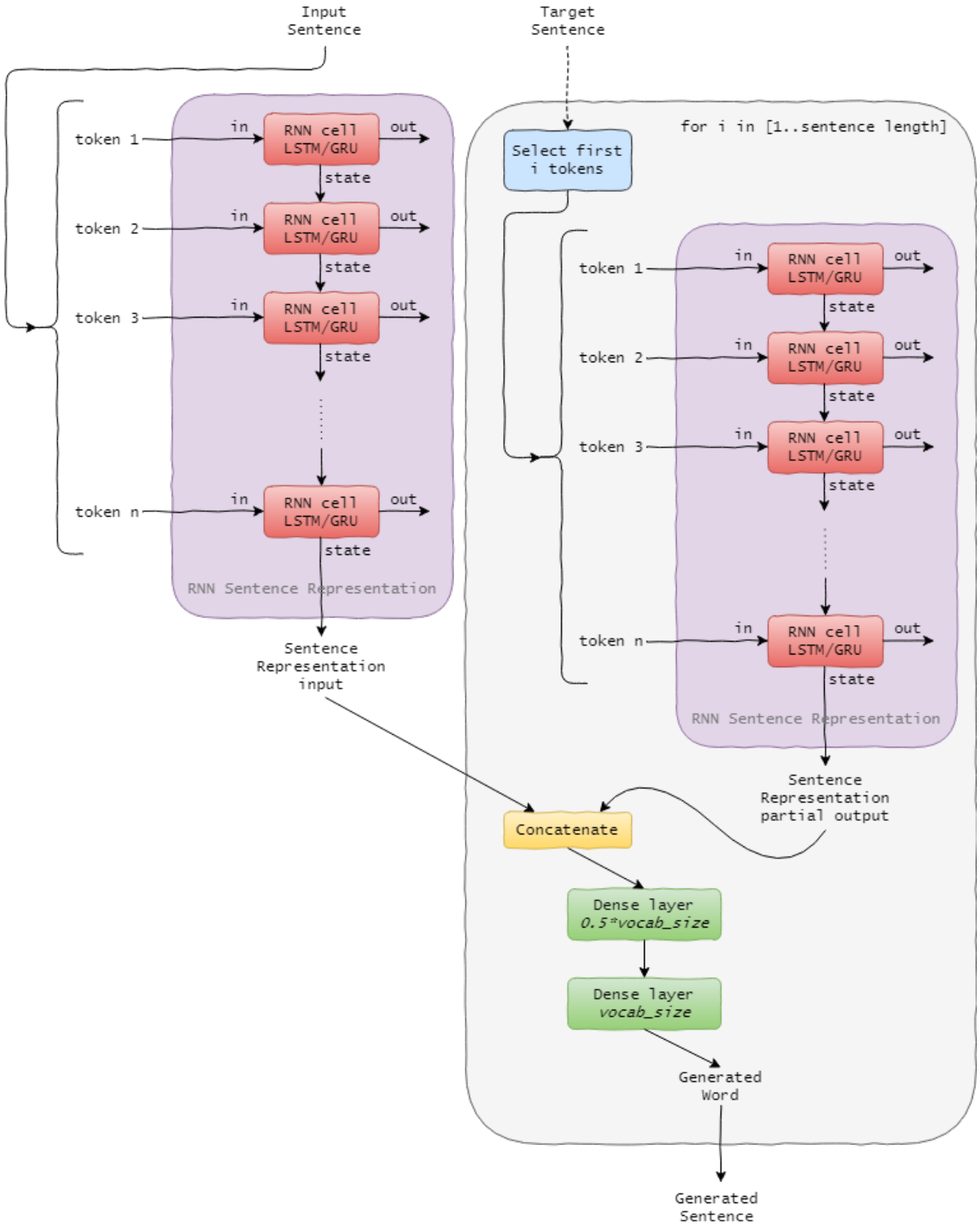
A recently proposed architecture (Ludwig, 2017) for word generation takes a slightly different approach to *natural language generation* (NLG). Instead of considering NLG as a problem of transforming a sequence of tokens into another sequence of tokens, it is considered as a problem of next token prediction given the context and previously generated tokens. This architecture was not given a name by the authors, but since they named it the "new approach to generative conversational agents", we have dubbed this model *NewGCA*.

The NewGCA architecture uses two RNNs, just like the Seq2Seq model. However, these RNNs are both sentence representation models, one for the input sequence (the context) and one for the generated sequence until now (the partial output). Thus, the NewGCA architecture creates two separate representations of the context and the partial output for each word to be generated. These two representations are then used to determine the token to be generated next. In statistical terms, it calculates the conditional probability of the next token to be generated given the context and the previous tokens.

The generation model component of NewGCA consists of two neural *dense layers*. The first has $0.5 * vocabulary\_size$ units and a *ReLU* activation function. The second layer has $vocabulary\_size$ units and a *softmax* activation function. The output word is represented using a *one-hot encoding*, allowing *categorical cross entropy* as loss function. A visualisation of the NewGCA architecture is shown in Figure 4.5.

The main advantage of the NewGCA model is that it needs significantly less training data and training time (Ludwig, 2017). This makes it extremely suitable for applications where training data is sparse or training time is expensive.

It has also been shown that the quality of the generated sentences is better than those generated with the normal Seq2Seq model (Ludwig, 2017). The sentences are longer and less generic. This can be attributed to the paradigm shift to next token prediction and the consequential separation of representation spaces for the context and partial output.



The dashed lines indicate training steps. During testing, the generated word is looped back into the partial output representation by concatenating it to the previous partial output.

FIGURE 4.5: A schematic representation of the *NewGCA* architecture proposed by Ludwig (2017).

Chapter 5

Datasets

This chapter gives a short overview of the different datasets that were used for training. We will first describe the *Kaggle Twitter customer support dataset*[1], followed by the *Yelp dataset* for style transfer[2].

5.1 Kaggle Twitter customer support

For training our dialogue system, we used the *Kaggle Twitter customer support dataset* available on Kaggle[1]. This dataset contains 3.002.524 tweets and replies in the customer support domain. Each tweet is annotated with which tweet it replies to and whether it is send from a company's customer support or not, among other annotation that we did not use.

This dataset is suitable for training a dialogue system, because the tweets are very goal oriented. The initial tweet is from someone who has a problem and all replies have the purpose of solving that problem. Due to the character limit on twitter, the tweets are relatively short and to the point, allowing a dialogue system to be trained with a smaller sentence length. The conversations are relatively natural, because they are real tweets. However, many customers are redirected to a private chat to actually solve the problem, which may skew the behaviour of the dialogue system to do the same.

We processed the dataset to be suitable for training a dialogue system. The dialogues were reconstructed using the reply annotation. We used a history of 2 turns as input and a single tweet as response. The responses are filtered to only be sent by a company. We replaced all newline characters in the tweets with a special _NEWLINE_ token. Similarly, we added a special _TURN_ token to differentiate between the different tweets in the input. Lastly, all non-ascii symbols were removed from the tweets and the resulting empty inputs or responses were then removed. This resulted in a dialogue dataset with 1.077.074 input-response pairs. These pairs were then split into a train, development and evaluation set (80%-10%-10% respectively).

[1] https://www.kaggle.com/thoughtvector/customer-support-on-twitter

5.2 Yelp style transfer dataset

For language style transfer, we used the *Yelp dataset* from Shen et al. (2017) that is available on GitHub[2]. The dataset contains 250,000 negative and 350,000 positive reviews from Yelp. All reviews have less than 11 sentences and each sentence has less than 16 words. On average, the sentences in a positive review are 1 word shorter than those in a negative review. More information about this dataset is not available, but it is relatively small.

[2] https://github.com/shentianxiao/language-style-transfer/tree/master/data/yelp

Part II

Applications of NLG

Chapter 6

Dialogue response generation

A possible commercial application of *natural language generation* (NLG) are *chatbots*, also called bots. There are two main approaches to chatbot technology: *retrieval-based chatbots* and *generation-based chatbots* (Young et al., 2017). As the name suggest, a retrieval-based bot chooses (retrieves) its responses from a (predefined) list of possible answers. If this choice is made using a set of (man-made) rules, it is a *rule-based chatbot*. To the contrary, a generation-based bot, or *generative chatbot*, actually generates its responses from scratch using NLG. Usually, a chatbot uses only one of these technologies, but they can also be combined (Song et al., 2016).

This chapter will focus on an application of NLG called *dialogue response generation* (DRG). As it is an application of NLG, the architecture, or structure, of a generative chatbot is based on the *NLG pipeline* described in Part I. Thus, a given dialogue or question is split into *tokens* using *tokenisation* (section 2.1). Then, the tokens are transformed into a numerical representation using an *embedding* (section 2.2). The resulting sequence (of variable length) of token representations can then be converted to an abstract, fixed-length representation of the entire input using a *language model* (Chapter 3). Lastly, the input representation is used to condition the *generation model* on, during generation (Chapter 4). The *attention mechanism* has been shown to be effective in dialogue systems (Yao et al., 2015; Eric and Manning, 2017), but it may not be suitable for long input sequences, like the full dialogue history.

In DRG, it is common practice to use *word tokens* (e.g. Serban et al., 2016b), but *symbol tokens* are also used (e.g. Sordoni et al., 2015b). Most model architectures are based on the Seq2Seq model (Sutskever et al., 2014; Vinyals and Le, 2015). The models can be trained in different ways, an overview is given in section 6.2. After training, a chatbot needs to be evaluated in order to judge its quality. Evaluation methods for chatbots are described in section 6.3. An overview of the challenges in dialogue response generation can be found in section 6.4. These challenges may be overcome by incorporating context (section 6.5) and knowledge (section 6.6). Lastly, the results are discussed of a DRG use case that was worked out: generative customer support (section 6.7). But first, an overview of existing chatbots is given in section 6.1.

6.1 Existing applications

Currently, *chatbots* are integrated in a number of customer support systems, e.g. for KLM [1], bol.com [2]. Although the techniques behind these bots remain a mystery in general, their functioning can usually be extrapolated from their behaviour. To the best of our knowledge, a commercial chatbot that truly generates its responses is not available yet. Most chatbots seem to be based on either a rule-based or retrieval-based approach. This is very suitable for FAQ type question answering. These *dialogue systems* can be created using *dialogue management systems* in which the rules can be easily specified. An example of such a system is *Google's DialogFlow* [3]. More complicated dialogue systems can be created with predefined frameworks, like *Amazon Lex* (when combined with other *Amazon Web Services* (AWS) components).

These rule-based systems can be very powerful nonetheless. The conversations of the open-domain chatbot *Cleverbot* [4] has been tested against those of humans in 2010 (as part of the BCS-SGAI Machine intelligence competition) (Cleverbot inc., 2010) and in 2011 (as part of the Techniche Conference in India). During the latter test, the bot scored similar to humans (Cleverbot inc., 2011), even though Cleverbot is written using Cleverscript [5] and is entirely rule-based.

In addition, retrieval-based systems can also be surprisingly smart. A well known example is Microsoft's controversial Twitter bot *Tay*. It made racist tweets and had to be taken down within the first day (Vincent, 2016). It seemed as if Tay generated its own tweets, but it turns out that it searched for similar Twitter conversations and responded with replies on similar tweets from that conversation (Neff and Nagy, 2016). Thus, Tay is retrieval based, just like the other Microsoft chatbots that use similar technologies; *XiaoIce* (China, see Neff and Nagy, 2016), *Rinna* [6] (Japan), *Ruuh* [7] (India) and *Zo* [8] (USA, successor of Tay).

Similarly, *Google's Smart Reply* (Kannan et al., 2016) that suggests email responses is also retrieval based. It uses techniques that can be implemented in a generative dialogue system as well and will be discussed in a bit more detail in section 6.4.2.

[1] https://bb.klm.com/
[2] https://www.bol.com/nl/klantenservice/index.html
[3] https://dialogflow.com
[4] http://www.cleverbot.com
[5] http://www.cleverscript.com
[6] http://www.rinna.jp
[7] https://www.facebook.com/Ruuh/
[8] http://www.zo.ai

6.2 Training generative chatbots

Once the model architecture has been created (as described in Part I), it needs to be trained. There are several techniques for training. They can be categorised into five main approaches: *supervised learning*, *unsupervised learning*, *reinforcement learning*, *adversarial learning* and *transfer learning*. Some training techniques combine these approaches. For example, *professor forcing* (as previously described in section 4.1.2) is a combination of supervised learning and adversarial learning. We will discuss the five main approaches in more detail.

6.2.1 Supervised and unsupervised learning

The general approach to training a model for dialogue response generation is *supervised learning*. This method updates the neural network based on a given input and target sequence. The input sequence is given to the model and its output is compared to the target sequence using an *objective function*. This function describes the *loss* to be minimised or the *reward* to be maximised. The loss or reward of the model is then used as feedback to optimise the model's performance.

A supervised learning technique often used when training RNNs is called *Teacher Forcing* (Williams and Zipser, 1989). Teacher forcing is when, at each time step, the RNN is fed the actual, expected output from the previous steps to predict the next step, instead of the previously predicted steps. The latter is still used when the true output is not available. Teacher forcing is so common that is the default way of training an RNN to predict sequences in a supervised manner. Conversely, training an RNN by feeding the previously predicted steps, without teacher forcing, is called *free running*. When the RNN output consists of probabilities, the previous tokens are approximated using the *softmax* function[9].

It is important to note that a suitable generated response can be very different from the target sequence. This makes training difficult as good responses may be penalised (see also section 6.4) when using a loss function that describes the difference between the target and output sequences (like *mean squared error*). Instead, a model can be trained with the *log-likelihood* that the target and output sequences are from the same distribution as objective function (Zhang et al., 2016). This is a probability to be maximised. Other common objective functions are *maximum cross entropy* (Vinyals and Le, 2015; Song et al., 2016) and *maximum mutual information* (MMI) (Li et al., 2015).

Adjusting the parameters, *weights*, of a model to obtain the smallest loss (or biggest reward) is called *optimisation*. How to adjust the weights is defined by

[9]The *argmax* function cannot be used when training an RNN, because it is not *differentiable*.

an algorithm called an *optimiser*. There are several optimisation algorithms. The most commonly used optimisers in DRG are *Stochastic Gradient Descent* (SGD) (Robbins and Monro, 1985) and *Adam* (Kingma and Ba, 2014), but other techniques can be used as well, e.g. *MERT* (Li et al., 2015, 2016a) or *AdaDelta* (Song et al., 2016).

In contrast, *unsupervised learning* is a training technique where the desired answer, the *ground truth*, is not available. The model needs to find patterns in the data without feedback about the relevance of those patterns. As such, unsupervised learning, on its own, is not suitable for dialogue response generation. However, a sub-model of a larger model, such as the *NLG pipeline* described in Chapter 4, can be trained *indirectly* by training it as part of the whole model. That way, the sub-model can learn what patterns are relevant (they are relevant for the larger model) without the need to explicitly define them.

6.2.2 Reinforcement learning

Another approach to training is *reinforcement learning*. This is used for a set of problems that can be described as a task where the system, called *agent*, needs to pick an *action* from a list, given its current state of knowledge (*environment*). Similar to supervised learning, the agent is trained by associating a *reward* with every action picked and optimising its decisions to maximise the reward. However, in reinforcement learning, the best action in a given situation may not be defined. Thus, the reward cannot be calculated based on some target action. Instead, the reward is calculated based on how the action changes the environment and whether this is desired. The latter may not be clear yet when the action is taken, but it becomes clear after several actions. As such, the accountability of a single action is not directly reflected in the reward.

This approach to training is has advantages and disadvantages. The creator of the system has a lot of agency regarding how the system is defined, but this definition needs to be specified really well. The reward, the actions and the environment need to be balanced in a way that the model will learn to pick the right actions in each situation. If this balance is slightly off, the model could learn the wrong things.

When training dialogue systems using reinforcement learning, the task of the agent is to pick a *token* given the input and the previously chosen tokens. The reward can be calculated once the generated sequence of tokens is complete. Reinforcement learning is used in dialogue generation by Li et al. (2017, 2016b) and Niu and Bansal (2018), among others.

6.2.3 Adversarial learning

Dialogue systems can also be trained using using *adversarial learning*. This is done by creating a *generator* that creates the responses and a *discriminator* that classifies the difference between real sentences and generated ones (or some other desired property). The two algorithms are trained in parallel. The generator learns to generate sentences that the discriminator classifies incorrectly and the latter learns from its mistakes. It is crucial that these two components learn at a similar speed for adversarial learning to succeed. If the discriminator is too good, the generator cannot learn, and vice versa.

6.2.4 Transfer learning

Once a response generation model is trained, it can still be adjusted or fine-tuned using *transfer learning*. This is a technique where a trained model is continued to be trained on a different dataset to learn new properties. Training a model with the purpose of using it in transfer learning as trained model is called *pre-training*.

The advantage of transfer learning is that one can first train a generic model and then use a small corpus to adjust the responses to a target domain. This makes it easier to use the model for different applications. Akama et al. (2017) showed that transfer learning can be used to build a Seq2Seq model that is stylistically consistent. They first trained the model on a large generic corpus en then continued training it with a small corpus that was stylistically consistent (the script of a single character in a TV series). Training the entire model from scratch would have needed a large stylistically consistent corpus, which is not readily available.

Transfer learning can also be used to bootstrap the learning process of a network using a network trained by someone else or for a different task. It can improve a model's performance if it is pre-trained on appropriate data (Serban et al., 2016b). Zhang et al. (2016) did this in their adversarial learning approach, described in section 6.2.3. Transfer learning is especially useful when a good model is publicly available, but its behaviour is not suitable for a specific use case.

Transfer learning can also speed up the training time of a model. This is extremely valuable when the time, finances or computation power is not available to train the model from scratch.

Adversarial learning has been shown effective for dialogue generation by Li et al. (2017). They used it in a reinforcement learning setting with an optimiser called *REINFORCE*. They showed that the quality of the output can be improved by removing short training examples, using a weighted learning rate, penalising similar phrases, and penalising word types that have already been generated.

However, one drawback of text generation using adversarial learning is that text is discrete. This makes it non-trivial to use the discriminator error to update the generator. The REINFORCE algorithm used by Li et al. (2017) can solve this problem using *Monte Carlo sampling*, but its variance could be large. Discreteness can also be simulated using a *softmax* activation function instead (Zhang et al., 2016).

A neural generation model that is trained with *adversarial learning* is called a *Generative Adversarial Network* (GAN) and they should perform better on high-entropy tasks (Li et al., 2017). Zhang et al. (2016) showed that this technique can be used for pure sentence generation. Their model is called *TextGAN*. It uses a Seq2Seq model with LSTM as generator and a CNN language model (by Kim, 2014) as discriminator. The discriminator was pre-trained to distinguish real sentences and sentences with two words swapped. This allowed the discriminator to judge word order. The encoder part of the Seq2Seq model was pre-trained as well. *Pre-training* is necessary to start with a balanced performance of the two networks. To remain balanced, the discriminator was not updated as often as the generator, because the CNN learned quicker than the LSTM-RNN. The objective function for optimising the LSTM was not to generate an exact match to the real response. Instead, it was optimised to generate output with a similar word distribution. This better represents the nature of natural language; the generator tries to pick similar words and the discriminator judges the word order.

An adversarial learning approach to training RNNs specifically is *professor forcing*. It is similar to training GANs and has been described in section 4.1.2. Professor forcing has been shown effective for DRG (Olabiyi et al., 2018).

6.3 Evaluating chatbots

Quantifying the performance of a trained model is not a straightforward task. There are two main strategies for evaluating chatbots: *automatic evaluation* and *human evaluation*.

Automatic evaluation uses predefined metrics that calculate a score based on some features of the generated sentences. It is cheaper, faster and the evaluation criteria are well-defined, making the comparison of the scores for different bots possible. However, the commonly used metrics for automatic evaluation base their score on linguistical features (e.g. *ROUGE*, *BLEU*), thus preferring grammatical sentences over relevant ones. It was even shown that these metrics have a weak correlation with human opinions at best (Liu et al., 2016).

This means that the current best method for evaluation is *human evaluation* using human judges. This can be done by asking the judges which response is best or by scoring the responses on some trait, e.g. *naturalness*, *relevance*, etc. Evaluation can be based on only a single input-response pair, *single-turn experiment*, or given a portion of the dialogue history, *multi-turn experiment*.

However, human evaluation can be costly and time consuming. Li et al. (2017) proposed a new automatic evaluation technique called *AdverSuc* which uses an adversarial classifier. The metric uses an evaluator model that is trained to distinguish real utterances from generated ones. AdverSuc (short for adversarial success) is defined by the fraction of cases when the generator is able to fool the evaluator. Since the AdverSuc value is meaningless when the evaluator classifies poorly, the evaluator needs to be evaluated as well. To this end, Li et al. (2017) defined a metric called *Evaluator Reliability Error* (ERE) that is the average performance on manually-invented situations where the gold standard response is known. Note that the evaluator is a completely separate model and is not a part of the generation model for dialogue generation. Nevertheless, the AdverSuc model can use techniques that are also used in dialogue generation, but it is trained separately. It is yet to be investigated how AdverSuc correlates with human evaluation.

6.4 Challenges with response generation

Although promising techniques exist for generating dialogue responses, there are still recurring problems with these methods. In general, the main purpose of dialogue is to convey information (Vinyals and Le, 2015). Information is an abstract concept that seems to be difficult for a dialogue system to grasp. As such, there are three main challenges for dialogue systems to overcome. The model may not learn to convey information at all, this will result in *generic responses*. A model, that can convey information, may give the wrong information, resulting in *irrelevant responses*. And lastly, a model may give relevant information, but the information changes in the course of the conversation, giving *inconsistent responses*.

It is noteworthy that these types of responses are not a problem when using NLG for *automatic translation systems* (Wei et al., 2017). Even though translation and dialogue response generation are considered similar tasks. Wei et al. (2017) hypothesised that this is due to that, in translation tasks, the input strongly defines the output, i.e. the input and target information is *aligned*. In translation, there can only be a few good translations for a given input sequence. However, in open-domain dialogue, there are almost endless responses to a given sentence, making it difficult for the model to learn what response is best. Wei et al. (2017) also concluded that this could be the reason why task-oriented dialogue systems perform better when conditioned on more information. The conditioning reduces the number of possible answers, increasing the likelihood of generating a good output. Thus, improving the overall quality of the system. Similarly, the performance gain of the *attention mechanism* may also be caused by alignment.

This section will briefly discuss these challenges and give a general overview of possible solutions.

6.4.1 Meaningful responses

The generation of *meaningful responses* is a difficult challenge in dialogue response generation. Seq2Seq models tend to converge towards generic responses. This is understandable, it is the safest strategy to respond with sentences that are always applicable, especially when input and target sequences do not align well. Thus, generating suitable responses with little information, i.e. generic responses, is a local optimum. This problem is reduced by improving the *alignment* between input and target sequence. Usually, this is done by penalising generic responses and promoting diverse responses during training, but other solutions have been proposed.

One way of improving the alignment is by simulating conversations using *reinforcement learning*. Li et al. (2016b) did this using two pre-trained agents. The agents were rewarded based on *ease of answering*, *information flow* and *semantic coherence*. This forces the input and target sentences to be more restricted. A generic response will not be rewarded because the other agent cannot respond to it and the lack of information reduces semantic coherence. The information flow reward promotes the introduction of new information, with respect to the previous turn of the same agent, by penalising the reverse. This improves the diversity of the dialogue.

The GAN proposed by (Li et al., 2017) (described in section 6.2.3) used some strategies that also improve the meaningfulness of the responses. Li et al. (2017) removed short replies from the training set which reduces the chance of learning (from) dull responses, because generic responses tend to be short. They also adjusted the *learning rate* (how severely the model parameters should be changed each step) based on the amount of meaningful tokens. The meaningfulness of the tokens is estimated using the *tf-idf* score that is based on the actual token frequency with respect to the expected token frequency. This weighted learning rate increases the amount of meaningful words in the output.

The generated output will also be less generic when it is more relevant. This can be achieved with the techniques discussed next.

6.4.2 Relevant responses

A dialogue system whose output contains information also needs to generate *relevant responses*. A meaningful response can still contain the wrong information. This can make the conversation feel unnatural and it can be especially problematic in task-oriented systems where the right task needs to be completed.

Irrelevant responses show that the generation model in the dialogue system cannot learn what is important. A straightforward approach to dealing with this is by adding context or knowledge to the dialogue system architecture. This will bias the output towards the given information and should result

in more relevant responses. Specific architectures for achieving this will be discussed in section 6.5 and section 6.6, respectively.

However, the system may not learn what is important because the input representation, created by the language model, is not appropriate. Thus, performance may be increased by adding a more suitable representation directly into the model as input. Luan et al. (2016) added context by concatenating topic information directly to the input of their model. The topic was approximated using a technique called *Latent Dirichlet Allocation* (LDA) (Blei et al., 2003) that uses word co-occurrences. They also added dialogue role information by learning a different weight set for each given role.

The input representation can also be improved by adding different abstraction layers as shown by Sordoni et al. (2015b). They used a *Bag of Words* (BoW) representation as the language model and trained two dialogue systems. The first used the BoW representation of the entire dialogue history as input to the generation model. The other used the BoW representation of both the last dialogue turn and the previous dialogue history. The two representations where concatenated and fed to the generation model. The latter model outperformed the former, because an abstraction layer was created between the last turn and the turns before that. The reason for this is that the information in the dialogue history stays the same, but which information is relevant changes over time. By splitting the two representations explicitly, the dialogue system does not need to learn when which input information is relevant, only what information.

An example of an application that does suggest relevant responses is *Google's Smart Reply* (Kannan et al., 2016), an automatic email response suggestion system used in Gmail. The system uses a complex hierarchical structure for generating a set of possible responses and selecting a set of suggestions using a Seq2Seq model. The responses were improved by penalising redundancy and enforcing both positive and negative responses. The entire systems was trained with a huge corpus and the responses are not generated, but selected, simplifying the challenge.

6.4.3 Consistent responses

A dialogue system should also generate *consistent responses*. Responses can be contradictory on their own, which can be improved by penalising the repetition of word types (see e.g. Li et al., 2017). But the responses can also be inconsistent with previous utterances in terms of content or style.

Li et al. (2016a) tried to improve the consistency of content over time by incorporating a speaker model into the hidden layer of the encoder. This was done by clustering different speakers based on features in their responses and using that as representation of the speaker. This allows the model to respond with sentences that are similar to those given by similar users. Since the users in the train data should respond consistently, so should the model.

They used a similar approach to model speaker-addressee behaviour in their dialogue system. That allowed it to adjust its speaking pattern depending on the conversational partner, e.g. addressing him/her by name.

Stylistical consistency is necessary in most applications of dialogue systems. It can feel uncomfortable if a conversation partner seems to change or addresses you differently. This may be more subtle in languages, like English, that do not have a stylistic distinction between, for example, different levels of politeness or courtesy. But it is very important in languages that do make such distinction, like the different honorifics in Japanese or the Dutch words for *you*: "*u*" and "*je*"[10]. In section 7.1, we will discuss several architectures for generating consistent politeness that can be used for other linguistical features as well.

6.5 Incorporating context

A key strategy to tackle the challenges in response generation is incorporating context, as explained in section 6.4.

This section will give an overview of techniques that have been used to condition a dialogue system on context. There are three main approaches to adding context: using the dialogue history, using dialogue acts and using language style. The research on the latter is mainly focused on politeness, but we describe how those techniques can be adapted to learn different linguistical traits like sentiment or formality.

6.5.1 Dialogue history

A popular approach to adding contextual information is by using the dialogue history. A straightforward approach is to feed all previous utterances directly into the encoder (see e.g. Vinyals and Le, 2015). However, this requires the encoder to make a dense representation that comprises all that information.

Instead, relevance can be improved by optimising the model accordingly. Li et al. (2015) proposed to use an objective function called *maximum mutual information* (MMI) that updates the model based on the similarity of the input and response word distributions. The MMI objective is linearly mixed with the standard *log-likelihood* function where the mixing variable determines the relevance of the response. However, it cannot be adjusted without retraining from scratch. In order to adjust the response relevance, Li et al. (2015) proposed to train the model using the standard objective function and use the MMI formula to rerank the top n generated tokens. We suspect that transfer

[10] It can still become relevant for English when the dialogues contain frequent uses of the royal we in combination with the normal we.

learning may also be suitable for quickly training a set of models, each with a different level of relevance.

MMI has several disadvantages. Directly decoding from the MMI function is intractable and needs to be approximated with an N-best list. This is computationally expensive. The pure MMI function also penalises grammatical responses because function words do not contain much information, but they are necessary for grammar. We refer the reader to the original paper for a detailed discussion (Li et al., 2015).

Another solution is to use an algorithm that adds context from the dialogue history by design. An example of this is the *Hierarchical Recurrent Encoder-Decoder* (HRED) model. It was proposed by Sordoni et al. (2015a) and it is an extension of the basic Seq2Seq model that was later adjusted for use in dialogue systems by Serban et al. (2016b). Like the Seq2Seq model, the HRED model transforms each dialogue turn into an abstract numerical representation using an RNN language model. This utterance representation is used in an RNN generation model to generate a response. In addition, the model uses a third RNN to create a context representation that is also used in the generation model, separating the current input from the conversational context. The third RNN takes the sequence of utterance representations of the previous utterances as input and creates a dense representation of the dialogue history. The performance of HRED can be substantially improved by using a *Bidirectional Recurrent Neural Network* (BiRNN) to create the utterance representation. This allows the model to capture a token in relation to the previous as well as the next token, improving the utterance representation (Serban et al., 2016b).

Extensions to the HRED model have been proposed, such as the *Latent Variable HRED* (VHRED) model (Serban et al., 2017b) and the *Multi-resolution RNN* (MrRNN) (Serban et al., 2017a). A comparative study showed that each architecture is able to incorporate long-term context (Serban et al., 2016a). The VHRED and MrRNN can also handle uncertainty and ambiguity. Both models generated more meaningful and more relevant responses than HRED and Seq2Seq. In addition, the MrRNN can generalise well and creates utterances with a high-level structure. We refer the reader to the original papers for a detailed description of these architectures.

6.5.2 Intent and dialogue acts

Traditional rule-based dialogue systems use a hand-coded dialogue act representation. This represents the intent of the utterance, e.g. if it is a question, the person wants an answer. In most generative dialogue system architectures, dialogue acts are not explicitly modelled, but they are part of the hidden state of the encoder. By extending the model with components that represent intent, the encoder can better encode the information in the utterance (Wen et al., 2016). This approach was also used for context in the previous section, resulting in the HRED model.

As such, it may be possible to use a dialogue act classifier to incorporate intent into a dialogue system by concatenating the classification result to the input representation. Alternatively, Wen et al. (2016) proposed using an intent network as part of a larger architecture. The intent network can be modelled using either an RNN or a CNN. The full input utterance is fed to the intent network to obtain an encoded vector that represents intent. The intent network can be trained in two ways. It can be pre-trained on a dialogue act classification task and the hidden state or last layer can be used as intent representation. Or the intent network can be trained jointly with the other components. However, training the intent network indirectly would raise the question if the network indeed represents intent as opposed to something else.

6.6 Incorporating knowledge

For practical applications, a good dialogue system may also need to use information. This could be knowledge that was obtained in the dialogue or that can be retrieved from an external source. Neither the Seq2Seq model nor the *NewGCA* (see Chapter 4) can use this information by default.

This section will describe a few methods to incorporate knowledge into a dialogue system. We make a distinction between structured and unstructured knowledge.

6.6.1 Structured knowledge

Structured knowledge is information that can be obtained in a fixed format. If the format is fairly short (e.g. the day in the week), it can be added to the dialogue system in a straightforward way by appending this information directly to the input or one of the hidden layers (see Li et al., 2016a). Otherwise this is non-optimal or even impossible.

If the information can be represented in a vector, a *Recall gate* could be used (Xu et al., 2016). This is an *RNN cell* based on the LSTM architecture. It allows the hidden state in the cell to represent the input under the condition of the knowledge vector. This technique can use the information more efficiently and performs better than the simple appending approach.

It is also possible to incorporate knowledge from a database. In that case, it is not desirable to feed the entire database to the Seq2Seq model. Instead, the generation model can be used to generate a *template response* that can be filled with information from the database. This is done by building a database query based on the input, generating the template based on the input and the query result and then substituting the template slots with the query results. This approach has been successfully used by Nayak et al. (2017); Wen et al. (2016). However, it does require a sizeable annotated dataset to train on. This

technique itself may require additional extension since it was used as part of a more complicated model in the known successful cases.

6.6.2 Unstructured knowledge

Unstructured knowledge is information that is available, but it does not have a strict format. This includes specific information that is given during the dialogue itself, in the dialogue history. It is very important to be able to use this information since a primary goal of dialogue is information exchange (Vinyals and Le, 2015). Note that section 6.5 used the dialogue history to indicate context not to obtain specific information (e.g. "the conversation was about age" versus "this person is 25 years old").

Factual information from the dialogue history can be represented in a *knowledge base*. Xu et al. (2016) proposed an easy way to create a *loose-structured knowledge base*. They identified a number of domains and for each domain they extracted *entities* from a domain specific corpus. The extraction was based on statistical metrics, identifying words that occur frequently in the domain. They used *Kullback-Leibler divergence* between words from a domain specific corpus and a general corpus to identify which words were actually specific for that domain. Those domain-specific words, the entities, were added to a domain vocabulary. An entity can have a certain value or trait which can be found as words occurring close to an entity. These words are called *attributes* and they give information about an entity. Entity-attribute pairs are identified using a sliding window. The most frequent pairs were added to the knowledge base. Given an input, the attributes of the most frequent entities can be identified. These entities and attributes are used to create a representation of each (most frequent) entity in the input. An entity representation is the sum of the embeddings of the attributes of the entity that was found in the input. The obtained vector was added to the dialogue system using the *Recall Gate* described in section 6.6.1.

Another way of using information from previous dialogues is by training on domain specific data. Choudhary et al. (2017) used a domain classifier to classify the domain of the conversation. For each domain, a different Seq2Seq model with *attention mechanism* was trained on conversations in that domain. The models generated multiple responses given a single input. The responses were then re-ranked to pick the best output.

6.7 Use case: Generative customer support

We implemented two simple models for dialogue generation: the Seq2Seq model by Sutskever et al. (2014) and the *NewGCA* model proposed by Ludwig (2017). Both models were described in Chapter 4.

It was shown that the NewGCA model performs very well on little training data (Ludwig, 2017), but the model has not directly been compared to the Seq2Seq model. Thus, we trained both models on the same dataset and compare their outputs.

6.7.1 Model descriptions

We implemented both the Seq2Seq model and the NewGCA model using two RNNs with a single LSTM cell each. Both LSTMs had a memory of 256 units. The input and output sentences were split into *word tokens* and embedded with the same GloVe embedding for each model.

The Seq2Seq model used the two RNNs as encoder and decoder, as described in section 4.1. The output of the decoder is fed to a dense layer with *softmax* activation. The dense layer has the same number of units as the dimension size of the GloVe embedding, such that the output can be mapped back to a word. This implementation is very similar to an example on GitHub[11].

The *NewGCA* model used both RNNs as two parallel encoders, one for the input and one for the partial output (see section 4.2). The encoded input and partial response were concatenated and fed to two sequential dense layers. The first layer has units equal to half the vocabulary size and a *ReLU* activation. The second layer has a unit for each word in the vocabulary and the output is mapped one-to-one, resulting in a *one-hot encoding* as output. Its activation is calculated with a *softmax*.

Both models were trained on the *Kaggle Twitter customer support dataset* (see section 5.1). The data was filtered on sentences with a minimal length of two word tokens and a maximal length of 50 *word tokens* (*punctuation* is counted as well). The words were represented with a 300 dimensional GloVe embedding (see section 2.2). The embeddings were compared by normalising them and taking the cosine distance (for the loss of the Se2Seq model).

The NewGCA model was setup to explicitly generate word by word, whereas the Seq2Seq model generates sequences directly with the RNN. This means that the target output of the NewGCA model is a single word, not a sequence. Thus, the format of the data had to be adjusted to suit each model. The Seq2Seq model was trained on batches of 64 input-response sentence pairs and the NewGCA model was trained on batches of 64 "input sentence"-"partial output sentence"-"next word token" triplets.

Both models were trained for maximally 100 epochs with *early stopping*; if the model did not improve its validation loss after 5 (*patience*) epochs, the training was interrupted. We used an *Adam* optimizer with a learning rate of 0.001 (default settings[12]).

[11] https://github.com/keras-team/keras/blob/master/examples/lstm_seq2seq.py
[12] https://keras.io/optimizers/#adam

6.7.2 Results

The basic Seq2Seq model was not able to learn to make sentences. The model learned to only respond with the repetition of a single word, regardless of input. We later realised that this may have been due to the generation of GloVe embeddings. The embedding has the property that words with little meaning have a vector representation that is close to every other vector representation. This means that a local optimum exists that only generates a single word. Unfortunately, we have not been able to repeat our experiments with the Seq2Seq model that generates *one-hot encoding*

The *NewGCA* model was able to learn to generate coherent responses word by word. Some example responses from the model are shown in Table 6.1. The responses show clearly that the model was trained on tweets, because every response starts with "@ <unk>". It is also noticeable that the training data contains many responses that refer the customer to some kind of private chat (*direct messages* (dm)) or another website with more information. It is interesting to see that the model finishes its responses with a closure that refers to different employees and that some generated responses contain URLs.

TABLE 6.1: Example responses for 10 tweets given by the trained NewGCA model.

Input	**somebody from @VerizonSupport please help meeeeee I'm having the worst luck with your customer service**
Original	@115719 Help has arrived! We are sorry to see that you are having trouble. How can we help? _NEWLINE_ ^HSB
NewGCA	@ <unk>we do n't want you to feel that way , what is going on ? _newline_ ^acm
Input	**Happy Halloween! Since I'm too old to trick or treat I now look forward to $3 booritos at @ChipotleTweets I got mine earlier, did you?**
Original	@115734 I mean, Boorito is basically the adult version of Halloween... -Becky
NewGCA	@ <unk>we 're happy to see you 're a fan . -tay
Input	**@AskPlayStation I bought Lego Star Wars in the PlayStation store but it won't let me download it because the data is corrupted..**
Original	@115745 Sorry for the inconvenience. Do you see the data corrupted message when you try to open the game or download it?
NewGCA	@ <unk>hi there ! please check the following link for more information on parental control : https : <unk>

TABLE 6.1: **Continued from previous page**

Input	**@ATVIAssist can you still pre oder ww2 on the playstation store**
Original	@115755 Yes you can pre order the game on the console store. For more information please contact your console store. ^RK
NewGCA	@ <unk>hey there , please send us a dm for further assistance . <unk>https : <unk>

Input	**Please tell me why my @115769 has expired, but Im still being charged????**
Original	@115773 Hi there, please DM us you Adobe id so that we may have a quick look at your account details. ^Tanuj https://t.co/iuwZCjz4Or
NewGCA	@ <unk>hi <unk>, please check the following article : https : <unk>, you may also find the details for the apps . ^tanuj

Input	**amazon**
Original	@115792 >https://t.co/NtNAX2Qh2u ET
NewGCA	@ <unk>https : <unk>et

Input	**@115812 I cant Connect my Interactive Broker (Canada) with yahoo finance App in ios 11, Any suggestions.. ??**
Original	@115811 Hi - are you receiving an error message when trying to link the account?
NewGCA	@ <unk>hi there , we 're here to help . can you please dm us with more details of your issue ? we 'll look into it . thanks ! https : //t.co/gdrqu22ypt

Input	**Thanks for the style advice, @115833 look ...I think? #Halloween2017 #flamingo https://t.co/XvI54La043**
Original	@115832 Alexa says both styles are working for you! My vote goes to the Flamingo look! ^SE
NewGCA	@ <unk>we 're glad to hear you 're enjoying the echo ! let us know if you have any other questions . ^ep

Input	**How @115821 packages china https://t.co/fO9vbus18E**
Original	@115848 We're always looking for ways to improve and we'd like to hear your feedback on this packaging: https://t.co/vjMFYkFFd1 ^KN
NewGCA	@ <unk>hi there ! please check the following link for more information : https : <unk>

Input	**Thank you @AppleSupport I updated my phone and now it is even slower and barely works. Thank you for ruining my phone.**
Original	@115864 We'd like to help, but we'll need more details. What's happening on your device and which model is it? Do you have iOS 11.1?
NewGCA	@ <unk>we 'd like to help . let 's take this to dm and we 'll explore ways to provide you assistance . https : //t.co/gdrqu22ypt

TABLE 6.1: **Continued from previous page**

6.7.3 Conclusion and discussion

It is clear from the previously discussed results that the *NewGCA* outperforms the simple Seq2Seq model. The former can generate sensible responses and the latter repeats a single word.

Although we suspect that the poor performance is caused by our decision to generate GloVe embeddings, there are numerous other reasons why the Seq2Seq model performed poorly. It could need more data, a better word embedding (for encoding) or different parameters. However, it is intriguing that the Seq2Seq model seems to be more sensitive to these factors than the NewGCA model, which did perform well with the same data and similar settings.

The NewGCA results show some interesting behaviours that may need to be addressed for commercial use. We will discuss four prominent response behaviours.

First of all, the model is biased to respond like a tweet, starting with "@ <unk>". This is great for personalising tweets, but the responses will need to be post-processed to add the correct Twitter handle. In the current example responses, this can be done with a simple rule based adjustment, but more complex responses may become more involved. Instead, the training data could also be augmented with some automatic labelling that finds the Twitter handles and replaces them with a special token before training. That special token can be filled in after it is generated with the appropriate handle, essentially generating a template.

Similarly, the model tends to close the responses with a proper closure including the employee handle, name or initials. Again, this is great behaviour for formal responses, but the model should be consistent with this. It can be adjusted in a similar way as the Twitter handles.

Another model behaviour is that it tends to redirect the customer to a private conversation. This is understandable because this is a prominent response in the training data as well. It is challenging to adjust this behaviour because it is caused by an artefact in the training data. It is possible to remove all or some dialogue pairs where the customer is asked to continue the conversation elsewhere, but this could leave too little data to train on and it cause introduce a selection bias to the dataset. Another option is to train another model that would be able to send appropriate direct messages. This second model can then be used whenever the first dialogue system redirects the customer. However, the second model requires a different train set, one with the actual private conversations.

The last model behaviour is a connected to this. Some model responses provide the customer with hyperlinks. Unfortunately, these URLs may not be relevant or even accessible. If the model responds with a link, it should be the correct one. Adjusting this behaviour is in theory similar to adjusting the Twitter handle and closure, but it is a lot more involved in practice. Depending on the desired responses, it could mean that a classifier is needed to retrieve the relevant URL given the input and response. This classifier will be very similar to a retrieval-based question-answering bot. The resulting architecture would be similar to the entity-attribute pair approach to incorporate knowledge from section 6.6.1.

In summary, the NewGCA model seems a lot more robust than the Seq2Seq model, but no hard claims can be made. Nevertheless, the generated responses are highly dependent on the example responses in the train data. Thus, when using *natural language generation* (NLG) for commercial settings, it is crucial that the train data is both extensive and balanced. The model needs many different examples for the same situation, but the situations should be evenly balanced. A dialogue system can also be improved by annotating the type of information that is given in the sentences in the train data, such that the right information can be filled in during post-processing. This holds in particular for personal and dialogue-specific information.

Chapter 7

Controlling generation

The advantage of *generation-based chatbots* over *retrieval-based chatbots* is that its responses are not predefined. The bot can learn to convey the same information in different ways, making it seem more human. But, the generation process is not controlled and the bot may respond with undesired sentences (even if they are meaningful, relevant, consistent and correct; section 6.4). Therefore, it is commercially relevant to investigate the possibility of controlling the generation process of a dialogue system.

This chapter will discuss the existing techniques for controlling generation. This is a rather small research area and most techniques are focused on generating responses with an appropriate level of *politeness* in Japanese. An overview is given in section 7.1. We adjusted one technique and used it to create an empathetic customer support chatbot. This use case is described in section 7.2.

7.1 Enforcing politeness

Niu and Bansal (2018) proposed three different approaches to enforcing the politeness of a dialogue system: the *Fusion model*, the *Label Fine Tuning* (LFT) model and the *Polite-RL model*. It was shown that these approaches are able to account for politeness without the loss of sentence quality.

Each architecture is based on a Seq2Seq model with *attention mechanism*. The encoder is a two-layer BiRNN with an LSTM as *RNN cell*. The decoder is a four-layer LSTM-RNN. This Seq2Seq architecture was adjusted by adding either a pre-trained two-layer LSTM-RNN language model or a pre-trained BiRNN-CNN politeness classifier. The language model is trained to be able to generate polite utterances only.

The *fusion model* combines the Seq2Seq model with the language model. It uses two generation models to generate a response. At each step in the generation process, both models generate a token given the previously generated tokens. The Seq2Seq model selects the token that would result in a response to the conversational input. Whereas the language model selects the token that would create a polite utterance. The two tokens are combined using

a *linear combination* with a specified weight. The weight determines which model's token is more important.

The *polite-RL model* describes *dialogue response generation* (DRG) as a *reinforcement learning* (RL) problem. Instead of judging the model on how well it can match a target politeness score (like the language model) or how well it can reply on the input (like the original Seq2Seq model), the model can learn how to compromise. Thus, generating a response that is as polite and natural as possible. The model uses the politeness classifier to add politeness. During training, a response is sampled (*free running*) and its politeness is classified. This politeness score is then combined with the *maximum likelihood* loss of the Seq2Seq model itself (*teacher forcing*) to create the objective function that the model weights are updated with.

However, neither the fusion model nor the polite-RL model allow the generation of responses with different target politeness levels. A different model needs to be trained for each level of politeness. The *Label Fine Tuning* (LFT) model does allow the specification of the politeness level. This target label is used as initial state of the encoder in the Seq2Seq model. During training, the label is automatically generated using the politeness classifier. Since the Seq2Seq model is trained to generate the target response, the target label should be the politeness of this target response. A similar architecture is shown in Figure 7.1.

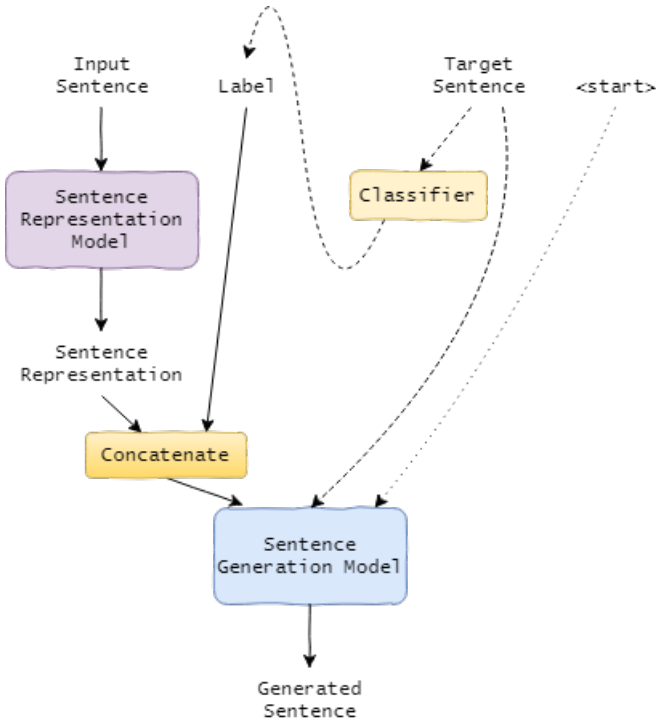
Although these three models are created for enforcing politeness, they could be used for enforcing other features as well. This is done by training the language model to generate sentences with that other feature and training the classifier to distinguish sentences with and without this feature. An example of this is the sentiment of a response. This is investigated in section 7.2.

7.2 Use case: Empathetic customer support

For some applications, it may be beneficial for a dialogue system to incorporate context in terms of language usage or style. We have discussed in section 6.4.3 that inconsistent language style can be problematic. However, other inconsistencies can cause similar problems, depending on the situation. In the customer support domain, it is very important to empathise with the customer and make them feel at ease. The basic seq2seq model does not take this into account.

This section discusses how an empathetic customer support can be created. We imagine the creation of such a system by defining some rules that determine with which sentiment the dialogue system should respond and a generation model that adheres to this target label. Since the focus of our research is on NLG, we will only describe the creation of the generation model.

Thus, for this use case, we need to control the sentiment of the generated sentence. Sentiment is a complex sentence characteristic that can be assigned



The dashed and dotted lines indicate training and testing steps, respectively.

FIGURE 7.1: A schematic representation of the adjusted *Label Fine Tuning* (LFT) model (Niu and Bansal, 2018).

automatically with an existing classifier. This means that we may not need a specific dataset, but we can use the *Kaggle twitter customer support dataset* (see section 5.1) and a sentiment classifier [1] to train the model.

We approach sentiment control by implementing the LFT model (Niu and Bansal, 2018) described in section 7.1. The LFT model was adjusted to use the *NewGCA* model (Ludwig, 2017), because it outperformed the Seq2Seq model (see section 6.7). We added the sentiment label as an extra input to the NewGCA model. The label is concatenated to the sentence representation, instead of using it as bias for the sentence representation. This results in an architecture where the NewGCA model generates the next word given the sentence representation, the target label and the previously generated words. This adjustment to the *NLG pipeline* is shown in Figure 7.1.

To force the model to generate sentences with the desired sentiment, we adjusted the *loss function* to take the sentiment label into account. This was done by estimating the sentiment of the partially generated sentence, including the new word, and comparing it to the actual sentiment of the given partial answer, including the true expected word. The annotation of the response sentiment is described in section 7.2.1 and the loss function is described in section 7.2.2.

[1] https://www.clips.uantwerpen.be/pages/pattern-en

7.2.1 Sentiment annotation

We annotated the true sentiment of a sentence using the Pattern library[1]. This library uses a database of manually annotated words to calculate the sentiment. The sentiment of a sentence is given by two decimals: the polarity and the subjectivity. The *polarity* determines whether the sentence has positive or negative sentiment and is a value between 1.0 (positive) and -1.0 (negative). The *subjectivity* of a sentence is indicated with a value between 0.0 (objective) and 1.0 (subjective). For the investigation of this use case, we only use the polarity of the sentence.

The basic principle behind the sentiment score of Pattern, is that the sentiment of a sentence is determined by the sentiment of the words. It is based on the average sentiment of the words with sentiment in the sentence. Since words can be combined to create nuance, the Pattern library uses some extra rules to capture this nuance better. Thus, the actual sentiment, calculated by Pattern, is slightly different from the average sentiment of the words with sentiment. However, this average can be a good approximation for the sentiment score that Pattern would give. We use this approximation to calculate the sentiment loss when training.

7.2.2 Loss function with sentiment

In order to force the model to adjust its generation behaviour based on the provided sentiment label, we defined a specific loss function consisting of two parts: the *sentiment loss* and the *generation loss*.

The *sentiment loss* of a partially generated sentence can be calculated by exploiting the fact that the NewGCA model generates words that are one-hot-encoded. Thus, it generates a vector with for each word in the vocabulary, the probability that the word should be next. We created a sentiment weight matrix with the sentiment score for each word in the *vocabulary*. This allows us to calculate the expected sentiment score for each generated word with a simple matrix multiplication. Then we counted the amount of non-zero sentiment scores in the partially generated sentence and used it to approximate the total sentiment score. This approximation is then compared to the sentiment score of the true partial answer using the *mean absolute error*. The loss can be summarised as:

$$sentiment(\mathbf{y}_i, j) = \frac{\sum_{k=1}^{j} \mathbf{y}_{ik} \cdot \mathcal{S}}{max(1, |\{\mathbf{y}_{ik} \cdot \mathcal{S}) > 0 \mid k \in \{1, 2, \ldots, j\}\}|)} \tag{7.1}$$

$$\mathcal{L}_{sentiment} = \frac{1}{\sum_{i=1}^{S} W_i} \sum_{i=1}^{S} \sum_{j=1}^{W_i} |sentiment(\mathbf{y}_i, j) - sentiment(\hat{\mathbf{y}}_i, j)| \tag{7.2}$$

where S is the number of sentences, W_i the number of words in sentence i, $sentiment(\mathbf{y}_i, j)$ the partial sentiment of sentence $\mathbf{y}_i$ until word j (included),

$\hat{\mathbf{y}}_{ik}$ the generated word vector for sentence i at word k and $\mathcal{S}$ is the sentiment matrix.

The *generation loss* is determined by whether the right word is generated. This is calculated using the *categorical cross entropy*:

$$\mathcal{L}_{generation} = \frac{1}{\sum_{i=1}^{S} W_i} \sum_{i=1}^{S} \sum_{j=1}^{W_i} \sum_{v=1}^{\mathcal{V}} \mathbf{y}_{ijv} \cdot \log(\hat{\mathbf{y}}_{ijv}) \tag{7.3}$$

where $\mathcal{V}$ is the vocabulary size and $\mathbf{y}_{ijv}$ is the vth dimension of the word vector $\mathbf{y}_{ij}$. The other parameters are as in equation 7.2.

The total loss for each word is a *linear combination* of the generation loss (equation 7.2) and the sentiment loss (equation 7.3):

$$\mathcal{L}_{total} = \mathcal{L}_{generation} + \lambda \mathcal{L}_{sentiment} \tag{7.4}$$

where λ is the weighting scalar. It is possible to tune the sentiment control by adjusting the λ parameter. A high value will cause the model to generate words with a specific sentiment, disregarding the quality of the sentence structure and content. A low value for λ makes the model less sensitive for the sentiment control. Note that λ is used for training the model and cannot be adjusted without training the model anew.

7.2.3 Training specification

We trained the adjusted LFT model on the same *Kaggle twitter customer support dataset* (see section 5.1) as the NewGCA and Seq2Seq models from section 6.7.

The label given to the model was the sentiment of the partial answer combined with the target word. The sentiment is the score determined by the Pattern library[2]. We trained a model using only the *polarity*.

The model was trained with the default Adam optimiser[3] and a batch size of 64 quadruples (the NewGCA triples and the sentiment score). We used a λ of 1.0 for the loss function and the words were embedded using same GloVe embedding as the models in section 6.7.

7.2.4 Results

A sample of sentiment controlled responses, generated by the LFT model, are shown in Table 7.1. The LFT model learned to generate different sentences from the NewGCA model to the same tweets (see Table 6.1). The LFT

[2] https://www.clips.uantwerpen.be/pages/pattern-en
[3] https://keras.io/optimizers/#adam

model was able to learn to respond with a given polarity sentiment, at the cost of relevancy of the reply when compared to the responses from the pure NewGCA model. It is noticeable that the model is slightly overfit towards the polarity label. When someone complains in a tweet, the model generates a response that is very positive about it. This shows that it is controlled by the polarity label.

TABLE 7.1: Example responses with different polarity for 10 tweets. Polarity is a scale between 1.0 (positive) and −1.0 (negative). The bold sentence is the input tweet that the model responds to.

Polarity	Sentence
	somebody from @VerizonSupport please help meeeeee I'm having the worst luck with your customer service
1.0	@ <unk>awesome ! we 're glad to hear that you are able to ! please let us know if you have any questions . ^ddd
0.5	@ <unk>ok , we can help you with this . please follow and dm us so we can assist . _newline_ ^hsb
0.0	@ <unk>we can help you with your internet . please follow and dm us . _newline_ ^jay
-0.5	@ <unk>sorry to hear that . we can help you out with your internet service . _newline_ ^hsb
-1.0	@ widespread horrible ! we 'd like to help . please dm us your account email @ 115990 and we will assist . ^ddd
	Happy Halloween! Since I'm too old to trick or treat I now look forward to $3 booritos at @ChipotleTweets I got mine earlier, did you?
1.0	@ <unk>awesome ! thanks for the kind words ! -james
0.5	@ <unk>love it . -james
0.0	@ <unk>we 're so <unk>. -james
-0.5	@ <unk>sorry to hear that . please write us at https : <unk>. -gabe
-1.0	@ 81 horrible ! -james
	@AskPlayStation I bought Lego Star Wars in the PlayStation store but it won't let me download it because the data is corrupted..
1.0	@ <unk>excellent question . please check your direct messages .
0.5	@ <unk>glad to help ! please follow us via twitter and let us know so we can send you a dm with further instructions .
0.0	@ <unk>hi there . please check your dm 's for further instructions .
-0.5	@ <unk>sorry to hear that . please check your dm 's for further instructions .
-1.0	@ talked awful sorry to hear that . please check your dm 's for the instructions .

TABLE 7.1: **Continued from previous page**

Polarity	Sentence
	@ATVIAssist can you still pre oder ww2 on the playstation store
1.0	@ <unk>excellent , please let us know if you have any questions . thank you . ^rn
0.5	@ <unk>glad to hear that you were now now now now . stay tuned for updates . ^jp
0.0	@ <unk>hey there , please send us a dm so we can investigate this further . ^jw https : <unk>
-0.5	@ <unk>sorry to hear that , things are still being looked into . please stay tuned for updates . <unk>
-1.0	@ widespread afraid ! ! ! please report this to our <unk>team here : https : <unk>. ^fb
	Please tell me why my @115769 has expired, but Im still being charged????
1.0	@ <unk>awesome ! let us know if you need any help in the future . ^raj
0.5	@ <unk>glad to hear it 's resolved . please let us know if you need any help ! thanks ! ^aj
0.0	@ <unk>hi , please let us know if you need any help ! ^rb
-0.5	@ <unk>sorry for the <unk>issue . please let us know if you need any help . ^raj
-1.0	@ widespread horrible sorry to hear that ! can you please dm us your adobe id email address ? we 'd like to help . ^sv
	amazon
1.0	@ <unk>awesome ! thanks for keeping us updated ! we 're always here to help ! <unk>
0.5	@ <unk>glad you 're enjoying the <unk>! <unk>
0.0	@ <unk>(<unk>) _newline_ mh
-0.5	@ <unk>sorry to hear that , <unk>. we 'd like to help . please reach us by phone or chat here : https : //t.co/jzp7hla23b ^dw
-1.0	@ talked awful !
	@115812 I cant Connect my Interactive Broker (Canada) with yahoo finance App in ios 11, Any suggestions.. ??
1.0	@ <unk>excellent ! thanks for the kind words ! i 'm glad to hear ! please let us know if you need anything else ! !
0.5	@ <unk>sure thing , <unk>. you can find more information here : https : <unk>. hope this helps ! <unk>
0.0	@ <unk>hi <unk>, we 're here to help . please reach out to us here : https : //t.co/jzp7hla23b so we can look into this further . ^pk
-0.5	@ <unk>sorry to hear that , <unk>. you can find that information here : https : <unk>. if you have any questions , please let us know .

TABLE 7.1: **Continued from previous page**

Polarity	Sentence
-1.0	@ talked awful ! i 'll be here to help . ^pk
	Thanks for the style advice, @115833 look ...I think? #Halloween2017 #flamingo https://t.co/XvI54La043
1.0	@ <unk>awesome ! thanks for the kind words ! we 're always here to help ! <unk>
0.5	@ <unk>glad to hear that you 're enjoying the <unk>! we 're always here to help ! ^ab
0.0	@ <unk>we 're always looking for ways to improve ! please reach out to us here : https : //t.co/haplpmlfhn so we can look into this further . ^pk
-0.5	@ <unk>sorry to hear that , <unk>. we 'd like to help . please reach out to us here : https : //t.co/jzp7hla23b so we can look into this for you . ^pk
-1.0	@ talked awful ! we 're glad to hear that you 're referring to . we 'd like to help . ^pk
	How @115821 packages china https://t.co/fO9vbus18E
1.0	@ <unk>excellent question . please check your direct messages .
0.5	@ <unk>glad to help . please follow us via twitter and let us know so we can send you a dm .
0.0	@ <unk>hi there . please check your dm 's for further instructions .
-0.5	@ <unk>sorry to hear that . please check your dm 's for further instructions .
-1.0	@ talked awful sorry to hear that . please check the next article to troubleshoot the ps4
	Thank you @AppleSupport I updated my phone and now it is even slower and barely works. Thank you for ruining my phone.
1.0	@ <unk>excellent . we 're glad to hear that you were able to ! happy to help ! please reach out if you need help in the future .
0.5	@ <unk>glad to hear it 's working again . we 'll be happy to help . please dm us the answer . https : //t.co/gdrqu22ypt
0.0	@ <unk>we 'd like to help . send us a dm and we 'll go from there . https : //t.co/gdrqu22ypt
-0.5	@ <unk>sorry to hear this . we 'd like to help . send us a dm with your current ios version and we 'll go from there . https : //t.co/gdrqu22ypt
-1.0	@ talked awful ! we 'd like to help .

7.2.5 Conclusion and discussion

We have shown that natural language generation can be controlled using a set of parameter inputs that the model is trained with.

Although the results are promising, they are not yet at a level that is suitable for commercial use. We will discuss five ways to adjust the model in a way that could improve performance.

First of all, we trained the full LFT model from scratch. It is also possible to start with a pre-trained NewGCA model and continue training it with *transfer learning*. That way, the model only needs to learn to adjust the sentences that it can already generate to incorporate the sentiment label. The model would not need to learn how to construct sentences anymore.

Secondly, the LFT model can also be adjusted architecturally. A dense layer with linear activation function can be used to make a multi-dimensional embedding of the label. This would allow the model to incorporate the sentiment in a more flexible way.

Another way to improve the performance is by adjusting the loss function. Training with different values of λ will tweak the behaviour of the model to be less sensitive to the sentiment loss ($\mathcal{L}_{sentiment}$, see equation 7.4). It is also possible to change the way the sentiment loss is calculated in the first place (equation 7.2) or to compare the sentiment of the generated word with the sentiment of the true next word, instead of using the partial responses:

$$\mathcal{L}_{alt\_sentiment} = \frac{1}{\sum_{i=1}^{S} W_i} \sum_{i=1}^{S} \sum_{j=1}^{W_i} |\mathbf{y}_{ij} \cdot \mathcal{S} - \hat{\mathbf{y}}_{ij} \cdot \mathcal{S}| \tag{7.5}$$

We used the partial responses because this is also the value that the model receives as label, but changing this could result in a model that can better incorporate the sentiment because it better represents how the original sentiment label is constructed.

The model may also be improved by using the sentiment loss over the entire generated response. During training time, the model receives the partial sentiment of the sentence generated until now. However, when using the model, the control label is used to indicate the sentiment of the entire sentence. This means that when generating the sentences, the model will start with a word of the right sentiment and then continue the sentence as if it was not controlled. Instead, the model can be trained using the label of the full target reply. It is also possible to adjust the generation procedure to change the given label to be spread over the generated words, mimicking the distribution of the label over the words in the train sentences.

Lastly, the model could be improved by using another sentiment annotation. The Pattern library was chosen for its simplicity, but other methods for sentiment analysis exist. Since other approaches were not investigated, it

is possible that some sentiment models have a more suitable sentiment representation. Readers interested in deep learning applications of sentiment analysis are referred to the survey paper by Zhang et al. (2018).

Chapter 8

Language style transfer

Another use of natural language generation is *language style transfer* (LST). This is when a text, given in some style, is adjusted to fit some other style. The style of a sentence can be described in several ways. For example, modern English versus Shakespearean English or positive-negative sentiment.

There are two approaches to language style transfer: *parallel style transfer* and *non-parallel style transfer*. The term parallel refers to the nature of the data that is available for training. The train data is called parallel if each sentence in the training set has a corresponding sentence in another style. This type of data is very hard to obtain. Non-parallel data is a dataset that is split into two (or more) subsets, one for each style, but there is no direct one-to-one correspondence between the sentences in one subset to the other.

8.1 Parallel style transfer

Most LST models assume non-parallel data, but they are also suitable for parallel data. Style transfer for parallel data is in essence a regular sequence to sequence problem and can be approached with (a variation of) the basic *NLG pipeline* (Chapter 4). An example of this is *Neural Machine Translation* (NMT). Even dialogue response generation can be considered a style transfer problem where a sentence in "question" style is transferred to the "answer" style, but this perspective is a bit far fetched.

8.2 Non-parallel style transfer

Current *language style transfer* (LST) algorithms use a variation of *adversarial learning* to train (see e.g Hu et al., 2017). The *generator* encodes a sentence to a generic (style independent) representation and decodes it to a specified style. The *discriminator* needs to determine whether a given sentence is original or generated. The generator tries to fool the discriminator. This approach is very similar to *professor forcing* (section 4.1.2), since the generic representation is reconstructed in *teacher forcing* mode and it is transferred in *free running* mode. The resulting model is essentially a GAN.

Fu et al. (2017) showed that language style transfer can be approached from two perspectives: with multiple decoders or with a style embedding. *Multi-decoder LST* uses multiple generation models in the generator, each trained to generate a specific style for the generic representation. *Style embedding LST* has a single decoder (like *dialogue systems*) that has an extra input for the target style. The style is represented with an *embedding*. This embedding is used to condition the output of the decoder such that it will generate sentences in the given style.

Shen et al. (2017) proposed a version of the style embedding model called the *Cross-aligned auto-encoder* (CAAE). This is a *style embedding LST* that is based on the Seq2Seq model. Both the encoder and decoder use GRU cells with the same memory size. Part of the GRU memory is used for the sentence representation, the other part for the style embedding; they are concatenated. The style of the input and the target style are embedded into a vector using a dense layer. The encoder GRU-RNNs memory is initialised with the style embedding of the input text and zeroes for the sentence representation. The decoder GRU-RNNs memory is initialised with the style embedding from the target style and only the sentence representation from the encoder memory.

The encoder and decoder are trained using *adversarial learning*. Shen et al. (2017) used a discriminator that was a simple *feed-forward network* with a single layer and *sigmoid* activation function.

To give an idea of the performance of the CAAE model, some example transferred sentences are shown in Table 8.1 and Table 8.2. We created these results with the available code on GitHub[1]. We show the results for the first 5 sentences in the validation set.

TABLE 8.1: Example negative to positive style transfer on 5 test reviews by the CAAE model (from Shen et al., 2017).

Original	**Transfer**
ok never going back to this place again .	all far to come back here .
easter day nothing open , heard about this place figured it would ok .	this place , for one , it 's 's delicious , it is excellent .
the host that walked us to the table and left without a word .	the manager and the us us us a made us our order .
it just gets worse .	it 's good .
the food tasted awful .	the food tastes amazing .

[1] https://github.com/shentianxiao/language-style-transfer

TABLE 8.2: Example positive to negative style transfer on 5 test reviews by the CAAE model (from Shen et al., 2017).

Original	Transfer
staff behind the deli counter were super nice and efficient !	the staff are in _num_ hours and the staff was so slow !
love this place !	avoid this place !
the staff are always very nice and helpful .	the staff are not very nice and rude or rude .
the new yorker was amazing .	the whole room was not amazing .
very ny style italian deli .	very small italian taco style sauce .

Since the *loss function* in adversarial learning is dependent on the quality of the discriminator, the CAAE model can be improved by using a *language model* as discriminator (Yang et al., 2018). Similarly, the *TextGAN* architecture (section 6.2.3) may be adjusted for style transfer by adding a style embedding or training multiple decoders.

8.3 Use case: Sentiment adjustment in reviews

We suspect that language style transfer can be used for adjusting sentences that are already retrieved or generated. The idea behind this is that the information in the input sentence is as desired, but the sentence itself is not suitable enough. This can be valuable for improving non-specific responses from *retrieval-based chatbots* and adjusting them to better suit the addressee.

We investigated this technique by adjusting the sentiment in Yelp reviews because the *Yelp dataset* (section 5.2) was a suitable, readily available dataset for style transfer. We did this using the *Cross-aligned auto-encoder* (CAAE) model by Shen et al. (2017) (section 8.2) and some adjusted versions of the *TextGAN* architecture by Zhang et al. (2016) (section 6.2.3).

8.3.1 Model descriptions

We first attempted to replicate the results from Shen et al. (2017) with the CAAE model. We recreated the model in our own pipeline as much as possible and referenced the GitHub implementation[2] for details.

The CAAE model uses a Seq2Seq model that is trained in a way similar to *professor forcing*. The input sentence is encoded using a RNN with a single GRU cell. The class label is fed to a dense layer with a *linear activation* function and used to initialise a part of the encoder state. The other part of the encoder state is initialised with zeroes and will contain the representation of the encoded sentence. After processing the input sentence, the resulting RNN state

[2] https://github.com/shentianxiao/language-style-transfer

is split into the label representation and the sentence representation. The original label information is replaced with the target label information. The sentence representation and the target label are used to initialise the initial decoder state and used to generate the new sentence.

This Seq2Seq model is called in two different ways during training. To reconstruct the original sentence with the decoder, the model is called using *teacher forcing*. However, teacher forcing is not suitable when the target sentence is not known, as in the case of language style transfer. Thus for style transfer, the model needs to be called in *free running* mode.

The adjusted *TextGAN* model uses the *NewGCA* architecture instead. The resulting architecture is the adjusted LFT model from section 7.2 that is trained with a *professor forcing* architecture. This TextGAN-like architecture is visualised in Figure 8.1.

Both LST models are trained using *adversarial learning* with two discriminators to determine how well the model incorporates each new style. The discriminators of the CAAE model had to distinguish between original and generated sentences for each style. The discriminators of the TextGAN model had to either distinguish between original and generated sentences, regardless of style or classify the style of the generated sentences.

Each discriminator is the language model by Kim (2014) as described in section 3.2. The language model consists of a number of *1-dimensional convolutional layers* with a *leaky ReLU* activation function, followed by a *max pooling layer* over the time dimension to identify which tokens are more prominent. For each convolutional layer, the max pooled result is concatenated and a *dropout layer* is applied. Afterwards, the result is fed to a *dense layer* with a single unit with *hard sigmoid* activation in order to make the classification.

All discriminators have the same architecture, but they are trained separately, resulting in different network weights.

Note that in the original implementation of the CAAE model, the discriminators use the sequence of hidden states for the Seq2Seq decoder to classify the style (see Shen et al., 2017). However, we used the generated sequence of words instead due to technical limitations[3]. This will reduce the performance of the discriminators and thus make it more difficult for the generator to adjust the sentence style.

8.3.2 Training loss

The loss of both LST models is determined by the linear combination of the word loss and the adversarial loss.

The word loss depends on the way the word is represented numerically. For the CAAE, we used a GloVe embedding, thus the resulting word loss is the

[3]The Keras library exposes only the last state, not the entire sequence.

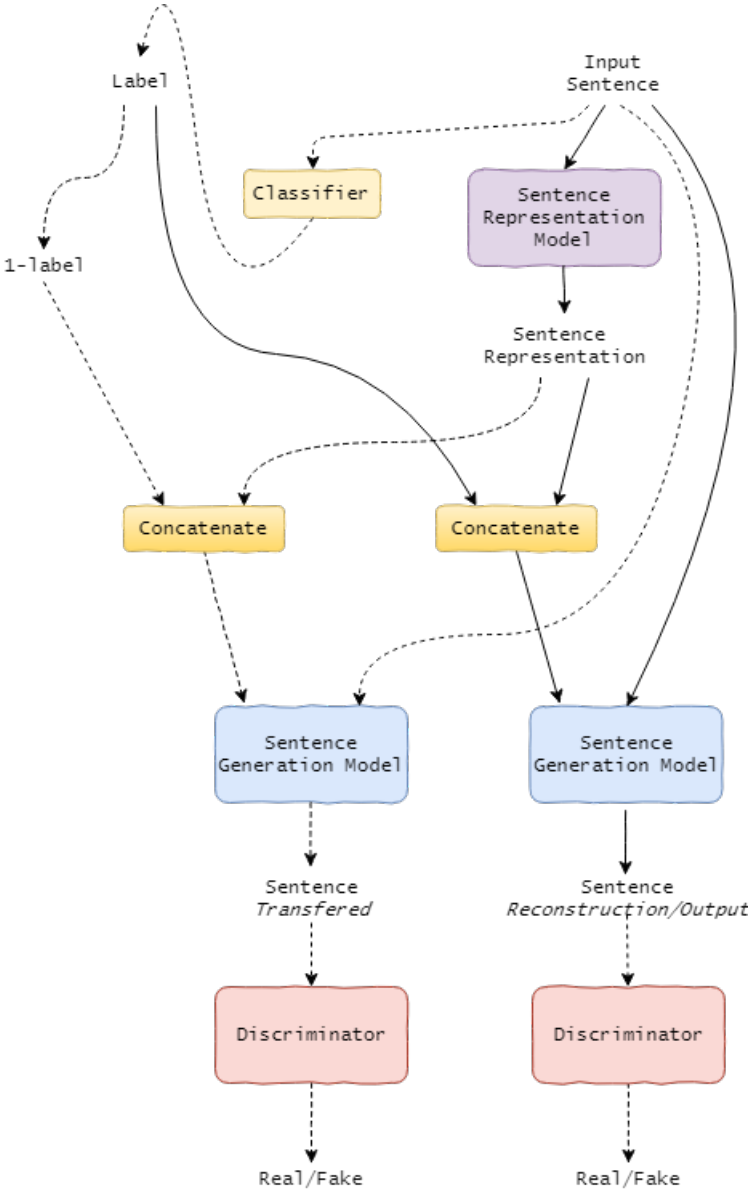


FIGURE 8.1: A schematic representation of our *TextGAN* (Zhang et al., 2016) approach to *language style transfer* (LST). The structure is similar to the adjusted *Label Fine Tuning* (LFT) model (Niu and Bansal, 2018) from section 7.2.

distance between the generated word and the target word. We calculate this using the cosine distance, after normalising the generated word:

$$\mathcal{L}_{cos} = \frac{1}{\sum_{i=1}^{S} W_i} \sum_{i=1}^{S} \sum_{j=1}^{W_i} \frac{\mathbf{y}_{ij} \cdot \hat{\mathbf{y}}_{ij}}{||\mathbf{y}_{ij}|| \cdot ||\hat{\mathbf{y}}_{ij}||} \tag{8.1}$$

where S is the number of sentences, W_i is the number of words in sentence i, $\mathbf{y}_{ij}$ is the embedding of the target word j in sentence i and $\hat{\mathbf{y}}_{ij}$ is the predicted word j in sentence i.

For the *TextGAN* model, we generated words embedded with a *one-hot encoding*. Thus we used the *categorical cross entropy* loss from section 7.2:

$$\mathcal{L}_{cce} = \frac{1}{\sum_{i=1}^{S} W_i} \sum_{i=1}^{S} \sum_{j=1}^{W_i} \sum_{v=1}^{\mathcal{V}} \mathbf{y}_{ijv} \cdot \log(\hat{\mathbf{y}}_{ijv}) \tag{7.3}$$

The *adversarial loss* can be determined by the *binary cross entropy* or the *mean squared error* between the predicted style and the actual style. The former is a special case of the *categorical cross entropy* (equation 7.3):

$$\mathcal{L}_{bce} = \frac{1}{\sum_{i=1}^{S} W_i} \sum_{i=1}^{S} \sum_{j=1}^{W_i} \mathbf{y}_{ij} \cdot \log(\hat{\mathbf{y}}_{ij}) + (1 - \mathbf{y}_{ij}) \cdot \log(1 - \hat{\mathbf{y}}_{ij}) \tag{8.2}$$

$$\mathcal{L}_{mse} = \frac{1}{\sum_{i=1}^{S} W_i} \sum_{i=1}^{S} \sum_{j=1}^{W_i} (\mathbf{y}_{ij} - \hat{\mathbf{y}}_{ij})^2 \tag{8.3}$$

where the parameter naming is the same as for equation 8.1.

For the CAAE model, we used the total loss is defined by:

$$\mathcal{L}_{total\_caae} = \mathcal{L}_{cos} + \rho \mathcal{L}_{bce} \tag{8.4}$$

where $\mathcal{L}_{cos}$ and $\mathcal{L}_{bce}$ are defined in equation 8.1 and equation 8.2. The ρ parameter determines how important the discriminator loss is.

The discriminators of the CAAE model were trained using the *binary cross entropy* loss (equation 8.2).

Note that the *adversarial loss* is only relevant for training the generator if the discriminator is accurate. Since the models are trained in parallel the discriminators are clueless at the start of training. Therefore, the adversarial loss is meaningless at first. Thus we adjust the ρ parameter during training. As long as the discriminator loss is higher than a pre-determined value κ, we train the generator using only the word loss; $\rho = 0$. Once the discriminator is good enough, we train using the total loss with the given ρ.

The adversarial loss of the TextGAN model is the same as the loss that was used to train the discriminators. The discriminators had to be trained from

scratch using *mean squared error* (equation 8.3), because the initial *binary cross entropy* loss (equation 8.2) was too high and the model could not learn how to decrease the loss. Nevertheless, the binary cross entropy better reflects the classification task of the discriminator and is the preferred loss function. Thus we experimented with pre-training the discriminators with the mean squared error and continuing training them with the generator using binary cross entropy. The total loss for training the TextGAN from scratch is defined by:

$$\mathcal{L}_{total\_textgan} = \mathcal{L}_{cce} + \rho(\mathcal{L}_{mse} + \mathcal{L}_{mse}) \tag{8.5}$$

and the total loss when training the TextGAN model with pre-trained discriminators is defined by:

$$\mathcal{L}_{total\_textgan} = \mathcal{L}_{cce} + \rho(\mathcal{L}_{bce} + \mathcal{L}_{bce}) \tag{8.6}$$

where $\mathcal{L}_{cce}$, $\mathcal{L}_{mse}$ and $\mathcal{L}_{bce}$ are described in equation 7.3, equation 8.3 and equation 8.2, respectively.

Again, the adversarial loss is only informative if the discriminators are accurate. The solution used for the CAAE model is not very suitable. Adding the adversarial term to the total loss equation after it was removed will dramatically increase the total loss, making it less representative of the actual fault in the output. Thus, we experimented with a *conditional loss* that also takes the discriminators accuracy into account. If the discriminator loss was higher than a pre-specified value, the adversarial loss of that discriminator is set to a specific large value. This value was similar to the initial loss of an untrained network. This stabilised the training loss of the generator.

8.3.3 Training specification

We trained the both models on the *Yelp dataset* (see section 5.2).

The CAAE model was trained using a GloVe embedding trained on the entire dataset. We used a batch size of 64. The discriminators were trained in separate batches for each style. We used a latent label embedding size of 200 and a latent input embedding of 500. Thus the encoder and decoder RNNs had $500 + 200 = 700$ units and the dense layer for the label embedding had 200 units. For the discriminators, we used 5 convolutional layers, each with 128 output filters, but different kernel sizes: 1,2,3,4 and 5. The dropout percentage was set to 50%. We set ρ to be equal to 1.0, which would only be used if the sum of the loss of the discriminators was lower than 1.2. To stabilise the training loss, we used an optimiser with gradient clipping. Specifically, we used the Adam optimiser with a normalised clip value of 30.0. We set the optimiser parameters to $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The learning rate for the discriminator was 0.002, while we set it to 0.0005 for training the generator. We trained for 100 epochs at maximum, using early stopping with a patience of 5.

The TextGAN model was trained in different ways. We used the GloVe vectors created from the *Yelp dataset* itself or pre-trained GloVe vectors[4] that were trained on Twitter data. The pre-trained vectors were extended with an extra dimension for the one-hot encoded special tokens that indicate the start and end of a sentence, as described in section 2.2. Due to GPU memory limitations[5], the model was trained with a batch size of 4. The model was trained with the default Adam optimiser[6]. The TextGAN model was trained from scratch, with a pre-trained style discriminator with perfect accuracy, and with both discriminators pre-trained. The TextGAN discriminators had 2, 3, 4, and 5 as kernel sizes for the four convolutional layers with each 15 output filters. The number of output filters was empirically determined by comparing the accuracy of models with a different number of output filters. The discriminators were trained with the same parameter settings as the generator.

8.3.4 Results

Similar to the results of the Seq2Seq model in section 6.7.2, the both models were unable to learn to generate sentences either. The models did perform slightly better than the Seq2Seq model, because they generated more than a single repeated word, but no grammatical sentences were generated. This holds for both the positive and the negative reviews. These results do not compare to the results in the original paper (see Shen et al., 2017). The original model is able to generate sentence and had some promising transfer results, as was shown in Table 8.1 and Table 8.2 in section 8.2.

8.3.5 Conclusion and discussion

Considering the performance of the original CAAE model (see table 8.1 and 8.2), neither LST model performs as is expected. Combined with the poor results of the Seq2Seq implementation (section 6.7.2), we cannot exclude the possibility that there is a bug in our implementation[7]. Nevertheless, there are other causes to consider. We will discuss some of these.

In section 6.7.3, we considered that the Seq2Seq model can be very sensitive to the quality of the word embedding. If this is the case, the CAAE model would be very sensitive to this as well, since it uses a Seq2Seq model as foundation. However, this does not explain the poor performance of the TextGAN model nor its similar performance when using the pre-trained Twitter embedding.

[4] https://github.com/stanfordnlp/GloVe

[5] The TextGAN model contains several models and uses a lot of GPU memory.

[6] https://keras.io/optimizers/#adam

[7] The pre-trained components of the TextGAN did not give the same results when loaded into the full TextGAN, even though they had the same architecture and weights. This behaviour was unexplained.

The both models are a type of GAN. This architecture allows the model to determine the quality of generated sentence that are not in the train data. Although the use of a GAN is needed for style transfer, it does have its caveats. In order to determine the quality of a sentence, you need a discriminator that can actually distinguish original sentences from generated ones. However, if the discriminator is too good, the generator is not able to learn what good sentences are. When training the LST models, the discriminators were never able to learn the difference between the two classes and were thus unable to provide the generator with a representative adversarial loss.

This was solved when using pre-trained discriminators in the TextGAN model, but they were not properly used when training the generator. The pre-trained models in our implementation were unable to recreate their output when pre-training, even though their weights were properly loaded in. We were unable to find the cause of this behaviour.

Nevertheless, when considering the original CAAE results (Table 8.1 and Table 8.2) and comparing them to the sentiment control results (Table 7.1), our adjusted LFT model is better at grasping the nature of sentiment. This could be due to the difference in base model (seq2seq versus NewGCA), but no conclusions about this can be drawn from the TextGAN model.

In conclusion, we suspect that we were not able to investigate the power of language style transfer due to problems with our implementation. This is unfortunate, but we like to explain our methods such that others can learn from it.

Part III

Conclusions and Discussions

Chapter 9

Challenges in NLG

This thesis provided and overview of natural language generation with a detailed elaboration of *language style transfer*, *dialogue response generation* (DRG) and controlling the generation process. The results of the investigation on an automatic (empathetic) customer support, that are shown and discussed in section 6.7 and section 7.2, are very promising. Unfortunately, we were unable to reproduce the previous results by Shen et al. (2017) on language style transfer in our own pipeline.

The overall conclusion of the use of NLG in these applications is that they are better when adding more complexity to the models. This is in line with previous research that showed that an ensemble of retrieval-based and generative solutions outperform the simpler solutions on their own (Song et al., 2016). Similarly, the NewGCA model outperforms the Seq2Seq model (section 6.7.2) by splitting the sentence representation from the previously generated representation. The solutions discussed in section 6.5 and section 6.6 also show that a more complex dialogue system will generate better responses.

This chapter will discuss the challenges with NLG in general, aside from the main challenges in DRG described in section 6.4. In particular, we will discuss the evaluation of these NLG systems and the training of *Generative Adversarial Network* (GAN)s. But first we will discuss the ability of these NLG system to actually capture the text.

9.1 Language understanding

Our approach to *dialogue response generation* (DRG) makes the assumption that conversations can be modelled as sequences of utterances that are sequences of tokens (Serban et al., 2016b). But, does this model actually capture the entire dialogue? Do textual conversations contain more information than just the text itself? Body language can also contain a lot of information in case of conversations that take place in person. Does textual dialogue contain a similar type of non-textual information? An example of this may be emojis that were removed from the data. Although we are not aware of any such filtering in our datasets, these are important questions to keep in mind when creating an NLG system.

Similarly, it is important for the *NLG pipeline* that the tokens are properly represented in the model. We used a GloVe *embedding* to represent our word tokens in a meaningful way, but other options could improve the results. The quality of the GloVe embedding is dependent on the quality of the dataset. The vector representations are based on coocurrences and each cooccurrence is weighted equally, thus the embedding becomes less accurate if the training text is noisy (Pennington et al., 2014).

The quality of the sentence representation is also crucial for creating a good NLG system. A neural language model can create a suitable generic representation that can be fine-tuned for specific tasks (Kalchbrenner et al., 2014). But, it is non-trivial to determine the quality of this representation because it is very abstract. The fact that the full NLG system performs well shows that the sentence representation is good enough, but it does not indicate whether it is the best option. It may be interesting to research the quality of other neural language models, like the CNN-based models described in section 3.2.

9.2 Evaluating generated results

Evaluation of text generated by an NLG model is a difficult problem in its own right. As is explained in section 6.3, the existing automatic metrics do not correlate with human opinion on DRG. This forces evaluation to be done manually, which is subjective.

This makes it not only difficult to evaluate a model on its own, it makes it even more difficult to compare different models with each other. To compare models, the models need to be trained on the same data. Since the datasets nor the models are usually not publicly available, the models need to be reconstructed first. Then, they need to be compared in the same way, preferably with the same human judges. Manual comparison is usually performed with just a handful of human judges, whose opinion may not correspond to another set of human judges. Thus most models are not compared to each other.

9.3 Training GANs

For *language style transfer*, we trained several models using a GAN structure without success. Shen et al. (2017) proved that if the sentence representation has a more complex distribution, such as a Gaussian mixture model, then the style transfer can be uniquely determined. Thus, language style transfer may not be possible if the sentence representation follows a simple normal distribution (Shen et al., 2017). Our approaches to language style transfer assume that the distribution of the information in the two parallel datasets is the same for both datasets (Shen et al., 2017). Since we reused the dataset from Shen et al. (2017) for our experiments and their original results with the

CAAE model were promissing, we do not expect that these two assumptions on the dataset were the reason behind the poor performance in section 8.3.

Instead, it may have something to do with training GANs. Text data is discrete, which makes it difficult for the generator to back-propagate the generation loss (Zhang et al., 2016) as well as the adversarial loss from the discriminator (Li et al., 2017). This is why our generation loss was estimated using a *softmax* activation function (similar to Zhang et al., 2016). However, we did not solve this problem for the adversarial loss. Due to technical limitations, we could not train the discriminator to discriminate based on the intermediate hidden states of the generator, which was a solution proposed by Lamb et al. (2016) for *professor forcing*, nor could we implement policy gradient reinforcement learning, as proposed by Yu et al. (2016).

Chapter 10

Dataset artefacts

The quality of the generated sentences of an NLG model is dependent on the quality of the train data. The dialogue response generation results in section 6.7.2 really show that the dialogue system learns to generate sentences that are similar to the train data. This means that when training a system for a specific task, it is crucial that the training data represents the expected input data and the desired output data.

This may be overcome when using *transfer learning*. This is when the model is first trained on a large, generic dataset and then training is continued on a more specific dataset. Akama et al. (2017) showed that the improvement of the model after transfer learning is dependent on the similarity of the generic and the task specific datasets. Thus, transfer learning improves more when the initial dataset is already close to the task specific dataset.

Similarly, if the dataset is not properly filtered, the model might learn undesired patterns in the data. An example of this is that a model by Serban et al. (2017b) accidentally learned to respond in Dutch, because the training set unintentionally had Dutch conversations in them. Although this specific example may not be problematic, other analogous situations may be undesired.

Thus it is important to investigate the possible biases that are present in the dataset when training an NLG system.

Chapter 11

NLG for commercial applications

Using *natural language generation* (NLG) for commercial applications is a double-edged sword. On the one hand, we have seen that NLG can be a very powerful tool that allows the creation of complex sentences without the need to specify a large set of hand-made rules or predefined sentences to be retrieved. On the other hand, this comes at the cost of control over what sentences the NLG system creates.

We have shown in section 6.7 that a trained model can give suitable responses. We have also shown that a model can also be controlled to generate sentences with a given sentiment in section 7.2. However, we cannot state that these models are unable to generate ungrammatical, meaningless or inappropriate sentences. We expect that this would be a deal breaker for most commercial applications, since it may harm the business.

With this in mind, we attempted to adjust sentences using *language style transfer* in section 8.3. In theory, this system could be used in combination with a *retrieval-based chatbot* to fine-tune the generic retrieved response to better fit the situation. However, we were unable to show the performance of language style transfer. We also realised that the style transfer model suffers from the same problem as the *generative chatbot*. The transferred sentence may no be grammatical, nor contain the right information.

Unfortunately, this leads us to conclude that the current state-of-the-art neural language generation techniques are not suitable for commercial applications on their own. It may be possible to use these techniques in conjunction with human checks, thus speeding up the manual customer support. A good example of this is *Google's Smart Reply* (Kannan et al., 2016) that proposed quick email replies for Gmail, while still allowing the user to adjust the proposed response or not use it at all. The current NLG technology can best be used together with human evaluation for commercial applications.

Bibliography

Akama, R., Inada, K., Inoue, N., Kobayashi, S., and Inui, K. (2017). Generating stylistically consistent dialog responses with transfer learning. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 408–412.

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

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Choudhary, S., Srivastava, P., Ungar, L., and Sedoc, J. (2017). Domain aware neural dialog system. *arXiv preprint arXiv:1708.00897*.

Cleverbot inc. (2010). Cleverbot wins machine intelligence prize. *Retrieved from* `http://www.cleverbot.com/machine` *on 2018-06-19*.

Cleverbot inc. (2011). Cleverbot comes very close to passing the Turing test. *Retrieved from* `http://www.cleverbot.com/human` *on 2018-06-19*.

Eric, M. and Manning, C. D. (2017). A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. *arXiv preprint arXiv:1701.04024*.

Fu, Z., Tan, X., Peng, N., Zhao, D., and Yan, R. (2017). Style transfer in text: Exploration and evaluation. *arXiv preprint arXiv:1711.06861*.

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

Hu, Z., Yang, Z., Liang, X., Salakhutdinov, R., and Xing, E. P. (2017). Toward controlled generation of text. *arXiv preprint arXiv:1703.00955*.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kannan, A., Kurach, K., Ravi, S., Kaufmann, T., Tomkins, A., Miklos, B., Corrado, G., Luk'acs, L., Ganea, M., Young, P., et al. (2016). Smart reply: Automated response suggestion for email. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 955–964. ACM.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lamb, A. M., GOYAL, A. G. A. P., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pages 4601–4609.

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

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

Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J., and Jurafsky, D. (2016b). Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*.

Li, J., Monroe, W., Shi, T., Jean, S., Ritter, A., and Jurafsky, D. (2017). Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*.

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

Luan, Y., Ji, Y., and Ostendorf, M. (2016). LSTM based conversation models. *arXiv preprint arXiv:1603.09457*.

Ludwig, O. (2017). End-to-end adversarial learning for generative conversational agents. *arXiv preprint arXiv:1711.10122*.

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., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Nayak, N., Hakkani-Tur, D., Walker, M., and Heck, L. (2017). To plan or not to plan? Sequence to sequence generation for language generation in dialogue systems. In *Proceedings of the 2017 Interspeech conference*.

Neff, G. and Nagy, P. (2016). Automation, algorithms, and politics| talking to bots: Symbiotic agency and the case of tay. *International Journal of Communication*, 10:17.

Niu, T. and Bansal, M. (2018). Polite dialogue generation without parallel data. *arXiv preprint arXiv:1805.03162*.

Olabiyi, O., Salimov, A., Khazane, A., and Mueller, E. (2018). Multi-turn dialogue response generation in an adversarial learning framework. *arXiv preprint arXiv:1805.11752*.

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

Robbins, H. and Monro, S. (1985). A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer.

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

Serban, I. V., Klinger, T., Tesauro, G., Talamadupula, K., Zhou, B., Bengio, Y., and Courville, A. C. (2017a). Multiresolution recurrent neural networks: An application to dialogue response generation. In *AAAI*, pages 3288–3294.

Serban, I. V., Lowe, R., Charlin, L., and Pineau, J. (2016a). Generative deep neural networks for dialogue: A short review. *arXiv preprint arXiv:1611.06216*.

Serban, I. V., Sordoni, A., Bengio, Y., Courville, A. C., and Pineau, J. (2016b). Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*, volume 16, pages 3776–3784.

Serban, I. V., Sordoni, A., Lowe, R., Charlin, L., Pineau, J., Courville, A. C., and Bengio, Y. (2017b). A hierarchical latent variable encoder-decoder model for generating dialogues. In *AAAI*, pages 3295–3301.

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.

Shoemaker, N. (2016). Japanese AI writes a novel, nearly wins literary award. *Retrieved from https://bigthink.com/natalie-shoemaker/a-japanese-ai-wrote-a-novel-almost-wins-literary-award on 2018-12-12*. [Online; posted 24-March-2016].

Song, Y., Yan, R., Li, X., Zhao, D., and Zhang, M. (2016). Two are better than one: An ensemble of retrieval-and generation-based dialog systems. *arXiv preprint arXiv:1610.07149*.

Sordoni, A., Bengio, Y., Vahabi, H., Lioma, C., Grue Simonsen, J., and Nie, J.-Y. (2015a). A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 553–562. ACM.

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

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Vincent, J. (2016). Twitter taught Microsoft's AI chatbot to be a racist asshole in less than a day. *Retrieved from* `https://www.theverge.com/2016/3/24/11297050/tay-microsoft-chatbot-racist` *on 2018-12-12*. [Online; posted 24-March-2016].

Vinyals, O. and Le, Q. (2015). A neural conversational model. *arXiv preprint arXiv:1506.05869*.

Wei, B., Lu, S., Mou, L., Zhou, H., Poupart, P., Li, G., and Jin, Z. (2017). Why do neural dialog systems generate short and meaningless replies? a comparison between dialog and translation. *arXiv preprint arXiv:1712.02250*.

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.

Wen, T.-H., Vandyke, D., Mrksic, N., Gasic, M., Rojas-Barahona, L. M., Su, P.-H., Ultes, S., and Young, S. (2016). A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

Xu, Z., Liu, B., Wang, B., Sun, C., and Wang, X. (2016). Incorporating loose-structured knowledge into LSTM with recall gate for conversation modeling. *arXiv preprint arXiv:1605.05110*.

Yang, Z., Hu, Z., Dyer, C., Xing, E. P., and Berg-Kirkpatrick, T. (2018). Unsupervised text style transfer using language models as discriminators. *arXiv preprint arXiv:1805.11749*.

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

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2017). Recent trends in deep learning based natural language processing. *arXiv preprint arXiv:1708.02709*.

Yu, Z., Xu, Z., Black, A. W., and Rudnicky, A. (2016). Strategy and policy learning for non-task-oriented conversational systems. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 404–412.

Zhang, L., Wang, S., and Liu, B. (2018). Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1253.

Zhang, W., Itoh, K., Tanida, J., and Ichioka, Y. (1990). Parallel distributed processing model with local space-invariant interconnections and its optical architecture. *Applied optics*, 29(32):4790–4797.

Zhang, Y., Gan, Z., and Carin, L. (2016). Generating text via adversarial training. In *NIPS workshop on Adversarial Training*, volume 21.

Index