Unsupervised Syntax-based Probabilistic Sentence Generation

by

Shahrzad Sayehban

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Sentence reconstruction and generation are essential applications in Natural Language Processing (NLP). Early studies were based on classic methods such as production rules and statistical models. Recently, the prevailing models typically use deep neural networks. In this study, we utilize deep neural networks to develop a model capable of generating new and unseen sentences or reconstructing the given input with minor changes. To achieve this goal, we develop an unsupervised tree-based model based on the Variational Autoencoder (VAE) framework.

Our approach utilizes the grammar rules of natural language and generates sentences based on phrases. This approach helps the generated sentence to be semantically and syntactically correct. Previous models typically considered the tokens sequentially, and the syntax was only learned implicitly. By contrast, our model learns both the sequence of the tokens and the syntax of the sentences explicitly in order to generate better samples. The variational modelling enables us to sample from the continuous latent space to generate new sentences or reconstruct the input.

We demonstrate the effectiveness of this model through experiments. The tree-based VAE model is trained on the Stanford Natural Language Inference (SNLI) dataset. First, we compute the BLEU score for the given input to evaluate the reconstruction capability of the model and how the model can preserve the information from the input. This score shows that our proposed model reconstructs the input sentence better than the baseline. Second, random sam-

pling from the latent space is used to evaluate how fluent the generation is. We observe the perplexity, UniKL, and entropy to evaluate the quality of the generated sentences. The results show that the generated sentences are less semantically meaningful. However, the sentences are correct in terms of the syntax and the order of phrases. The reason is that the rules are applied in a way that correct parse trees are generated.

To my beloved parents, sister, and brother
For their unlimited love, care, and support.

Acknowledgements

I would like to thank some people who supported me during my study.

I would take a moment to show appreciation to my supervisors, Professor Lili Mou and Professor Davood Rafiei, for their guidance, knowledge sharing, and providing invaluable advice. They have been patient and supportive throughout this research. They helped to motivate me to deal with the challenges in the research path. I would like to take the opportunity to express my gratitude to them for whatever I learned from them while working on my research.

I am highly thankful to my parents, Masoud and Maryam, my sister, Shekoofa, and my brother, Shahab, for always supporting and caring about me, even during the most challenging times.

Having friends who encouraged me was another favourable situation. I thank all my friends who made this journey of study and life more enjoyable by supporting me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The fields of Artificial Intelligence (AI) [34] and Machine Learning (ML) have been flourishing in recent years, and there are many applications where machines learn to make decisions as if a human being takes them. Broadly speaking, machine learning refers to the field that computers learn from data [17] and obtain knowledge without being programmed explicitly [37]. Robotics, computer vision, and natural language processing are some machine learning applications, and there are significant advancements in these fields.

Machine learning has been improving recently at a much faster pace, and there are two reasons for that. First, the applications of the deep neural networks have been increasing every day. The reason is that these networks are able to perform feature engineering on their own [5]. Second, the computational power that enables machines to learn complicated models is more reachable.

This study focuses on applying and developing deep neural network models for a task in Natural Language Processing (NLP). Natural language processing is concerned with enhancing how computers interact with humans in natural language. This interaction involves various components such as natural language understanding and generation. Each of these can be used in the tasks of question answering, machine translation, text summarization, etc.

Our research further focuses on natural language text generation that strives to generate sentences in a natural language as humans do. Here, a

natural language sentence is a group of words sequenced together. Two factors in this task are expected to be satisfied with a generated sentence: for a sentence to be considered correct, both its syntax and semantics should be correct. The constraint on syntax is satisfied if the structure of a sentence follows the rules of the desired language, while a semantic condition ensures it is semantically acceptable.

We aim to learn sentence generation based on the given input sequences. In addition to word orders considered in sequence-to-sequence (Seq2Seq) models [45], we also consider the tree format of the sentences. The proposed model learns the structure of the input sentences to be able to generate sentences based on those structures.

1.2 Problem Definition and Motivation

We study the problem of sentence generation in two cases: (1) an input sentence is given, and the output is the reconstruction of the input, and (2) an output sentence is generated from scratch while an input sentence is unavailable. Sentence generation by itself is an essential technique in many natural language processing tasks, such as text summarization [27], dialogue generation [38, 39], image captioning [47], and storytelling. Most of the models for these tasks generate sentences from scratch and mostly have the issue of generating generic outputs [23], while diverse sentences are desirable to be generated. We also need to regenerate sentences, given an input, to assess how the model can maintain the information from the input by having a reconstruction task that can be used in many applications such as paraphrasing [32] and code generation [50].

As mentioned before, there are various applications for generating sentences from scratch. Creating such sentences from a distribution provides a robust environment to obtain the goal of diverse sentence generation. For example, a dialogue generation system is supposed to receive a sentence and respond, but various responses may be available [38]. Therefore, it is important to consider probabilistic sentence generation to produce diverse sentences.

The principal motivation for this research is to focus on the syntax explicitly to generate sentences. As mentioned, generating sentences is a key concept in natural language processing. We would like to create sentences as either reconstruction of input sentences with some variation or to develop sentences from scratch. We want these sentences to be semantically and syntactically correct. So, in the proposed model, we consider the tree-structure of the sequence, along with a probabilistic encoder-decoder.

The baseline, which is the study of Bahuleyan et al. [2], does not consider if the reconstructed or the generated sentences are grammatically correct and the study is generally based on a sequence-to-sequence encoder-decoder model. A sequence-to-sequence model works so that an output sequence of tokens will be generated given an input sequence of tokens. For example, given an input sequence $a = (a_1, a_2, ..., a_m)$ with m tokens, the model may generate an output sequence $b = (b_1, b_2, ..., b_n)$ with n tokens. This work generally pays attention to the series of tokens within sentences, and the fact that sentences in natural language must be generated following specific structures is only implicitly satisfied.

Implicitly considering the structures of sentences has the potential of generating syntactically incorrect output sentences. The reason is that we are using a variational autoencoder model for the text generation task. The continuous latent space of this architecture merges both the syntactic and semantic information [3]. So, many structural details miss while generating sentences [41]. If the syntactic part is not modelled explicitly, it causes producing syntactically incorrect text [52]. Researchers have shown that explicitly involving syntax while generating sentences enhances the output quality for sequence-to-sequence models [4, 9, 11, 41]. Therefore, adding grammar rules to generate syntactically correct sentences is helpful because it considers the syntax directly.

Let us use an illustration to clarify the idea. For instance, consider the syntactic part of the following sentences. The sentence "I like do this book" has some flaws in the grammatical aspect of the natural language. The reason is that the verb phrase includes two verbs consecutively (i.e., "like" and "do")

and a noun phrase after that (i.e., "this book"), where there is no valid tree structure for this sentence; thus, this sentence is not correct syntactically. But generating this sentence is possible by not having a grammar-based model and not modelling the syntax explicitly. On the other hand, the sentence "I like this book" is considered a correct sentence, according to the natural language grammar, because it consists of a noun phrase and a verb phrase in the first division of the sentence and the other subphrases afterwards in the order they must be shown. By considering the syntax directly in our proposed model, we can generate syntactically correct sentences, such as the latter example, instead of the former. So, in this study, we consider the tokens and the applied grammar rules during the generation process.

Usage of grammar rules in a model can be applied for different applications in natural language generation. Language generation [3], unsupervised paraphrase generation [52], and syntax-transfer generation [3] are examples of the application of this model. Moreover, some studies on tasks such as machine translation [9, 11, 24, 54] explicitly model the syntax to improve the quality of the generated sentences. Explicit syntax modelling can be applied to the natural language tasks because this model does whatever a non-grammatical model does and also pays attention to the syntax of the generated sentences.

In this work, we study how grammar rules and their application can affect the generation of natural language sentences. The generation can happen under two cases: (1) we have an input and aim at generating sentences given the input (i.e., reconstruction), and (2) the generation happens with no input. This policy of explicitly demanding grammar rules in generating natural language sentences is applied to a variational autoencoder model to assess how different sentences can be generated through a stochastic generative model. This architecture uses various features of the sentences in the generation task. These features include the tokens, the rules, the context of the sentences, and the information from the parent of a node in the parse tree. Yin et al. [50] study the influence of these parameters in the generation task.

1.3 Contributions

This study proposes a new way of generating sentences from the latent space or regenerating them given input sentences. The suggested model works in an unsupervised manner and combines a token-based with a grammatical-based sentence generation.

The previous studies on natural language generation only use the tokens within the sentences and are mainly based on Seq2Seq models [2, 7, 16]. We do not ignore the effect of considering tokens within the sentence, but we think that is not sufficient for the task of sentence generation.

If we think of natural language generation by human beings, not only does the word choice play an important role in transferring their idea, but also following a correct structure is essential because the sentences must be grammatically correct.

Our framework is based on a variational encoder-decoder, and we consider the following parameters to learn the structure of the sentences to finally be able to generate new or reconstruct sentences that are similar to the previously seen ones:

- actions that are applying grammar rules and generating tokens,
- context of the sentence,
- previously generated tokens, and
- information of the parent nodes in the parse tree of the sentence.

We use all these factors in a variational autoencoder architecture [19]. The encoder and the decoder in our VAE model use LSTM, an RNN-based approach.

Based on a learned distribution and the encoded input, in the case of a provided input, the most probable rule is applied or a token is generated at each time step. Compared to a word-level sentence generation or reconstruction [2], our approach uses grammar and considers syntax as the centre of the attention of the model. So, we will have a tree-based VAE model.

The evaluation is done on the Stanford Natural Language Inference (SNLI) dataset [6][1] to demonstrate how our approach works. The dataset includes 570k sentences in natural language. To evaluate if the generated sentences are effective and of good quality in terms of reconstruction, the BLEU score [31] is used if the input is available. Moreover, perplexity (PPL) is utilized to measure the fluency of the generated sentences. UnigramKL and entropy are also used to evaluate the closeness and similarity of distribution between the generated sentences and the corpus.

Our quantitative evaluation demonstrates that a structure-based variational autoencoder works well in terms of reconstruction compared to the non-tree-based model. In the case of sentence generation without giving an input, although all the previously generated tokens are considered at each step, the word choice does not result in developing fluent sentences. But, in the case of evaluating the syntax of these generated samples, they are correct and produce valid parse trees.

To summarize, the main contributions of this thesis are as follows.

1. We propose a variational autoencoder (VAE) [19], in which previous tokens, context, and tree structure of a sentence consisting of actions are effective in learning a model of sentence generation. These actions include rule application and token generation in a top-down format of the tree.

2. In this process, the parse tree is considered in addition to the simple format of each sentence.

3. We study the quantitative evaluation of the model and assess its effectiveness compared to the simpler models through different experimental metrics.

Although explicitly considering the grammar rules can be beneficial and be used in many applications and tasks in the natural language generation, there are some issues. Compared to the non-grammatical models, this model

[1] https://nlp.stanford.edu/projects/snli

requires more time to be learned and is more complicated. The reason is that syntax-based models need more information than the sequence of tokens to understand the tree structure of the sentences. In learning the tokens, the model only needs to focus on the previous tokens at each step, and these tokens are sequentially joined together. On the other hand, our framework also pays attention to the structure of the sentence tree, including the parent nodes and siblings. Our approach differs from the non-grammatical approaches in the way that it is not completely sequential, and other topics also play roles.

Another challenge is having the ability to process the text with its grammar rules. It is required that all the rules be extracted beforehand based on the given dataset, while this is not needed when we are not explicitly working with the grammar rules. When we introduce a model that utilizes the grammar rules, we should provide those rules explicitly so that we can learn a proper model based on the rules.

To conclude, not considering the grammar rules explicitly in the model can achieve good results based on measurements, such as fluency. Moreover, these models do not spend time in the pre-processing steps to extract grammar rules. They also do not take time to learn the rules structure in their architecture during training, which results in performing these models in less time. So, obtaining good results while spending less time is the reason many applications get away without explicitly using the grammar rules in their models.

1.4 Thesis Outline

This chapter presents the motivation and background behind our study and an outline of our approach. The remaining parts of this thesis are structured as follows:

- Chapter 2 reviews the related works on text and sentence generation. We also review the background material referenced in the rest of the thesis, including the variational autoencoders, the parse trees, and the Context Free Grammars (CFGs).

- Chapter 3 introduces our tree-based VAE model and the use of grammar rules in natural language sentences to reconstruct input sequences and generate new sentences.
- Chapter 4 provides details of our dataset, evaluation metrics and the results of the proposed model. The model is assessed both in sentence reconstruction and generation. We compare our model with the baseline that does not benefit from the grammar rules within the language.
- Chapter 5 gives a summary of the study, followed by the conclusion and directions for future research.

Chapter 2

Related Work and Background

In this chapter, we review the relevant studies on generating sentences in the natural language. We also cover the background topics that are needed in this study, including autoencoders and tree-based models.

2.1 Natural Language Generation

In Natural Language Generation (NLG), we want to generate sentences or paragraphs in natural language, and we want those sentences and paragraphs to be similar to the human language. The goal is to create fluent, natural, correct, and grammatical sentences while satisfying the criteria and objective for a specific task. There are different well-established tasks and sub-fields in NLG, including Machine Translation (MT), summarization, and chatbots. We will go through each of these tasks briefly in the following.

2.1.1 Text Generation Applications

Given an input utterance in one language, **machine translation** is the task of creating a new sentence in a different language (for example, translating an English sentence into German) while maintaining the same semantics and meaning. Recently proposed methods have mainly relied on neural networks and are capable of generating accurate translations [1, 48].

Text summarization should produce a comprehensive and concise summary of the input while retaining all the crucial details from the original text. There are two general approaches for generating an input text summary: ex-

tractive and abstractive. The important sentences in a text are found using an extractive method. Without making any changes, each sentence that conveys vital information is added to the summary output. On the other hand, a more sophisticated abstractive method interprets the context to identify the most important sections and then reproduces them in a condensed form. The second strategy, which is more relevant to this thesis, is regarded as a generation task [30, 36].

Dialogue response generation is the process of creating a dialogue based on a source utterance. Applications developed for this task include chatbots and other similar services. More recent studies on dialogue generation use neural network encoder-decoder techniques [15, 40].

2.1.2 Text Generation Techniques

Now that we talked about some significant applications of NLG, we want to discuss different techniques developed for the task.

Rule-Based Methods

Previous techniques in NLG, which were proposed in the early stages, mainly focused on rules. The rule-based methods may be based on production rules of grammars, probabilistic or statistical techniques, finding patterns of interconnected entities, or hierarchical classes and their relationship [8].

Sequential Models

Most state-of-the-art approaches are based on deep Neural Networks (NNs). Various architectures are arising from NNs, and they are becoming popular.

This process of shifting from the early stages of NLG (i.e., rule-based methods) to the approaches centred on neural networks started with Recurrent Neural Networks (RNNs) [44]. Later, Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) architectures were used to generate sequences. The work of Graves [14] is an example that uses Long Short-term Memory recurrent neural networks for different tasks. These methods are helpful because they are applied to get a sequence in a natural language as input in order to

predict the output. For generating natural language, regardless of the task, the encoder-decoder architecture is shown to be a good option [10]. Sutskever et al. [45] apply LSTM for both the encoder and the decoder to generate sequences.

Many different studies utilize the previously mentioned sequential methods and architectures for the task of natural language generation. These models are called **text generation from scratch**. Many of these studies use VAE to generate sentences or sequences. For instance, Bowman et al. [7] use an RNN-based VAE model to create sentences. The distribution of the latent space and sampling from it allows the whole sentence generation and learning of the aggregated features. The study of Bowman et al. [7] encodes the input data to a continuous hidden space and decodes the sampled vector from that distribution to generate sentences.

A closely related study to our work is the study of Bahuleyan et al. [2], which is considered a baseline in our study. It uses the Variational Autoencoder (VAE) and Wasserstein Autoencoder (WAE) for probabilistic sentence generation. The encoder might be deterministic or stochastic in WAE. While this method works better than previous approaches, the generated results may not be grammatically correct since the training focuses only on the words and not on the syntax of the sentences explicitly. Utilizing the syntax of the sentences while using the sequence of words is what we study in this thesis.

Bao et al. [3] use disentangled syntactic and semantic spaces to model the syntactic information and language generation. There are two latent spaces needed to learn the syntax and semantics independently. Linearized tree sequences are used as the input for the syntactic part of this model. The defined loss for this work is a combination of multitask loss, adversarial loss, and adversarial reconstruction loss. Multitask loss makes sure that each space learns the respective information. Adversarial loss predicts semantic information from the syntactic space and inverse. Lastly, adversarial reconstruction loss discourages the sentence prediction by only a single subspace.

Another study that considers syntax is the study of Zhang et al. [52]. This work combines the semantics of a sentence with its syntactic tree to generate a

more grammatical sentence. Similar to the study of Bao et al. [3], Zhang et al. [52] use two separate latent spaces, one caring for developing a sentence and the other learning a syntactic tree. Therefore, there is a joint distribution. In the study of Zhang et al. [52], the two latent spaces can be treated in two ways; the latent variables can be either dependent or independent of each other. The loss function is designed accordingly.

Furthermore, Yuan et al. [51] model the syntactic structure of the data as a graph and encode it based on Graph Neural Networks (GNNs). It utilizes VAE for the generation part. The semantic aspect and the syntactic component of the sentence are learned, but there are two different versions. One of the versions merges both latent spaces into a single space. While the other considers them as two separate latent spaces that continue to the decoder separately.

The studies mentioned above for the generative tasks mostly used variational autoencoders, but there are other architectures that allow us to generate sentences, such as Generative Adversarial Networks (GANs). Zhang et al. [53] apply GANs to create texts. However, there are some challenges while using this strategy. The problem is that accurate gradient computation cannot be applied because the variables and outputs are discrete. So a gradient estimation method is obtained. Monte Carlo estimation is used for the gradient estimation of discrete variables.

Other studies accomplish the sentence generation task using **edit-based text generation**. Miao et al. [26] propose a method for dealing with the sentence generation task that includes the condition of containing predetermined words in the generated sentences. In order to implement the method, an approach based on Metropolis-Hastings sampling via unsupervised learning was employed to overcome sequential sentence generation. In this approach, one operation from the set of insertion, deletion, and replacement operations, each comes with a specified probability, is selected. At each step, a word is chosen randomly, and then the selected operation is performed to generate the sentence.

In a similar study, Song et al. [42] propose a model to generate sentences with the predetermined word at a specific position while using RNNs. The

model uses two RNNs to generate forward and backward sentences, starting from the desired word. They also add the embedding of a position into the model training.

It should be considered that sentence generation does not always start from scratch, as some studies sample prototypes from a training set and edit them to produce new ones [16]. In the study of Guu et al. [16], sentence generation is accomplished by randomly selecting a prototype sentence from a prototype distribution and a random edit vector. Then a new sentence can be generated by applying these to a neural editor. Additionally, the edit vector, which determines the type of edit to be executed, is sampled from an edit prior. The goal is to satisfy semantic smoothness and consistency. The smoothness means that each edit is minor for the sentence, while several edits will obtain a significant change. The consistency controls the change type in the sentence, meaning that an edit vector makes semantically equivalent changes, even if the prototypes are different. Guu et al. [16] claim that their proposed model has better quality than the ones without having a prototype in the first step.

Attention Mechanism

Although the sequential models can be beneficial for sentence generation, some issues might affect the quality of the long sentences due to the challenges in managing the dependencies in long sentences using the standard networks. To address this challenge, a new network model called the attention mechanism [1] is introduced. The attention mechanism is expected to search for relative words in an input utterance to help with predicting the next word in a target utterance. Since the vectors in encoder-decoder models have fixed sizes, the attention mechanism can be an improvement by working to create vectors that are not necessarily fixed in size. In the other words, the attention model uses context vectors based on positions to predict new words. Thus, this approach helps to improve performance while working with longer sequences.

The Transformer model [46] is the model enabling us to work easier and more efficiently in comparison to the recurrent models. This model is based on the encoder-decoder architecture. Additionally, the Transformer uses attention

as its main idea without recurrent structure. This model can handle long-range dependencies within the input, which is a challenge in the sequential model architectures. Moreover, parallelization of the computation is possible with this architecture. In other terms, the initial proposed Transformer [46] uses multi-head self-attention as its attention part, which means that the model attends the input to itself and has multiple attention heads that perform in parallel. Recently, Transformer-based models have been used for generative tasks, such as Generative Pre-trained Transformer (GPT) [33].

2.2 Autoencoders

2.2.1 Representation Learning

Learning the representation of the data results in understanding its properties. The posterior distribution of the learned representation focuses on the variation of the data and is helpful in tasks like generation. This distribution is learned while working with probabilistic models.

The "manifold hypothesis" is expressed based on the dimension of the data. This hypothesis states that even high-dimensional data tends to lie near a low-dimensional topological space called a manifold [12]. This hypothesis is useful for representation learning. Manifold learning is supposed to learn and represent data based on a manifold hypothesis. Principal Component Analysis (PCA), which is used for dimensionality reduction, is an example of manifold learning.

Our work in reducing dimension and manifold learning is based on a neural network architecture called autoencoder [35]. PCA finds a linear correlation between the features, while autoencoders can find nonlinear correlations.

2.2.2 Autoencoders

As mentioned above, we can use autoencoders [22] for dimensionality reduction and information compression. We want to reduce the number of data features while missing as little information as possible in an unsupervised training paradigm. This model can also be used to learn how to remove the noise of

the input data.

Autoencoders use neural networks to learn a model on how to reconstruct the input data points. This architecture is based on an encoder and a decoder [22], as in Figure 2.1.

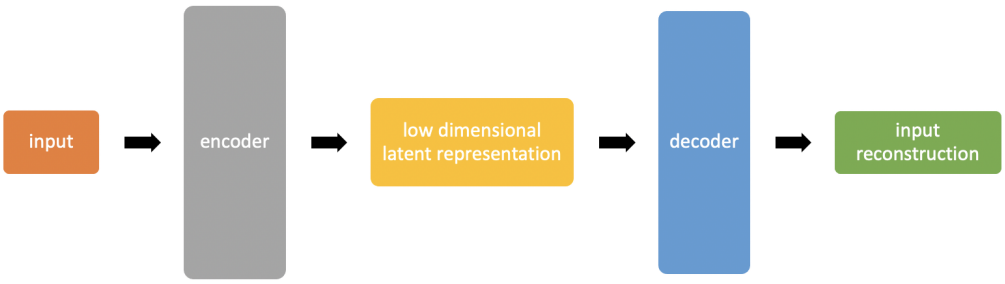

Figure 2.1: Architecture of an autoencoder

The encoder component is supposed to learn a mapping of the essential information in the input data to a lower dimensional manifold. The mapping result will be the input encoding in the Euclidean latent space of the new features representation based on the original features in the input data.

The intermediate representation, also called the latent space, is the part that has the key features and information from the input. Latent representation of the data points can be used in different cases. The dimensionality of the latent space is usually less than that of the input, and the latent space is an encoded and compacted version of the input data while maintaining the essential information within the original data point.

The last component in the autoencoders is the decoder. The decoder uses the data stored in the latent representation phase in the learned manifold to decode it and reconstruct the input.

If we represent the encoder and decoder as f and g, respectively, we aim to find these two functions such that they satisfy the following equation, for input x,

$$g(f(x)) = x. \tag{2.1}$$

If f and g are linear functions, the framework is PCA. The encoder and

decoder can be considered neural networks if a non-linear function is used instead of the linear function.

Normally, the output of the reconstruction does not satisfy the equality of Equation 2.1. The output can be considered as $g(f(x)) = \hat{x}$. Then, the objective function is

$$J_{reconstruction} = \sum_{n=1}^{N} \|\hat{x}_n - x_n\|_2^2 . \tag{2.2}$$

The loss of the autoencoder depends not only on the input data but also on its structure. In our case, we are working with text and a sequential architecture, so the training loss for predicting the x_t token is computed based on the previous tokens and the latent representation,

$$J_{reconstruction} = \sum_{n=1}^{N} \sum_{t=1}^{|x^{(n)}|} -log(p(x_t^{(n)}|h^{(n)}, x_1^{(n)}, .., x_{t-1}^{(n)})). \tag{2.3}$$

In this equation, h represents the latent space, superscript (n) indicates the data point n that is from 1 through N, the total number of data points, and t is the decoding step.

The autoencoders suffer from the lack of regularity in the latent space. The reason is that we use neural networks in autoencoders that can learn complex functions. Because of this, the model is more likely to overfit, which leads to a less regular latent space [43]. Variational autoencoders are introduced to address this problem.

2.2.3 Variational Autoencoders

The variational autoencoder [20] model forces a prior distribution $p(z)$ on the latent variable z. This distribution is commonly the normal distribution [20]. In simple autoencoder models, the latent space lacks a specific distribution while encoding. The reason is that the model is trained on encoding and decoding by minimizing the reconstruction loss, no matter how the latent space is regularized. Not regularizing the latent space causes overfitting, which makes the model not generalize well. A regularizer can be added explicitly to

increase its generalizability and solve the unstructured latent space problem [2].

The generative process is by sampling $z \sim p(Z)$ and computing $f(z, \theta)$ that generates X. During the optimization process of θ, we require maximizing the likelihood of the generated X. The optimization problem is to maximize the probability of the training data, as follows:

$$p_\theta = \int p_\theta(z) p_\theta(x|z) dz. \tag{2.4}$$

While computing this likelihood, some values will be intractable (i.e., it cannot be computed for all the values z). Being intractable also happens for computing the posterior distribution, $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$. The variational autoencoder helps us to learn an approximation of the posterior $p_\theta(z|x)$ and handle the intractable variables.

The Variational Autoencoder (VAE) [20] learns a generative latent variable model and handles intractable distributions. The key idea is to have an approximate posterior model $q_\phi(z|x)$ and a true posterior $p_\theta(z|x)$ and to learn the parameters ϕ and θ to make the approximate posterior distribution to be similar as much as possible to the true posterior. To compute the distance between these two distributions and minimize it, the KL divergence is commonly used. The formula is as follows:

$$D_{KL}(q_\phi(z|x)||p_\theta(z|x)) = E_{z\sim q_\phi}[log\ q_\phi(z|x) - log\ p_\theta(z|x)]. \tag{2.5}$$

Based on Equation 2.5 and applying the Bayes rule, we have the following equation:

$$log\ p_\theta(x) - D_{KL}(q_\phi(z|x)||p_\theta(z|x)) = E_{z\sim q_\phi}[log\ p_\theta(x|z)] - D_{KL}[q_\phi(z|x)||p_\theta(z)]. \tag{2.6}$$

The term $D_{KL}(q_\phi(z|x)||p_\theta(z|x))$ in the above equation is intractable, but since the output of the KL divergence is non-negative, we can ignore this term and obtain the following inequality:

$$log\ p_\theta(x) \geq E_{z\sim q_\phi}[log\ p_\theta(x|z)] - D_{KL}[q_\phi(z|x)||p_\theta(z)] = \text{ELBO}. \tag{2.7}$$

The right-hand side of the inequality, known as evidence lower bound (ELBO), is maximized in variational autoencoder models. And maximizing ELBO is equivalent to minimizing the negative of the above expression.

The first term on the right-hand side of Inequality 2.7 is the output of the sampling. On the other hand, the second term, which acts as the regularization of the latent space, demonstrates the KL divergence between the posterior and the prior. The regularizer controls sample generation and avoids overfitting. As can be seen, this model works with reconstructing the input data point and also generating new samples and contents. It can be interpreted because the first term in the loss function is in charge of reconstructing the input data, and the other term attempts to make the approximate posterior distribution close to the prior as much as possible that can be used for sample generation. These two terms are differentiable everywhere and will be considered as the objective function for the VAE models.

In variational autoencoder models, the encoder encodes the given input data point and produces a distribution in the latent space. $q_\phi(z|x)$ can be thought of as the encoder that encodes the input into two different latent representation variables, the mean (i.e., μ) and the variance (i.e., σ^2) of the approximate posterior. On the other hand, the decoder decodes the output data given the latent variables, where $p_\theta(x|z)$ is supposed to be the decoder. Typically, the encoder and decoder are neural networks with parameters ϕ and θ to be learned, respectively. Therefore, based on the right-hand side of Inequality 2.6, the encoder encodes the data point x to a distribution. The decoder decodes the data and generates a new data point, given a sample from the distribution. A graphical representation of this architecture is shown in Figure 2.2.

Considering the posterior and prior distributions, it is common to think of the posterior as a normal distribution (i.e., $q_\phi \sim \mathcal{N}(\mu, \sigma I)$) and the prior to be a standard normal distribution (i.e., $p_\theta(z) \sim \mathcal{N}(0, I)$) [2]. Sampling directly from the approximate posterior is not correct since it is a random variable that is not differentiable. We can use the reparameterization trick [20] to backpropagate for the gradients. Reparameterization makes the function

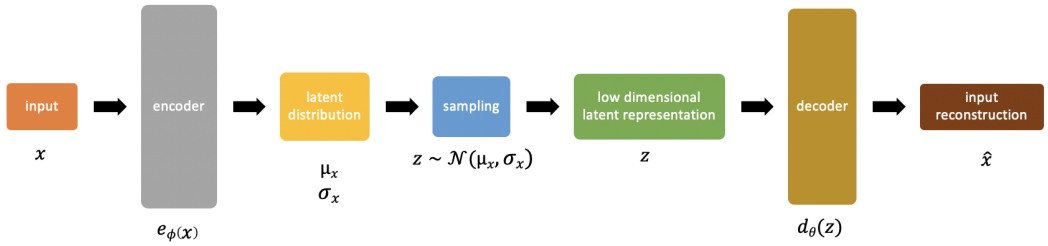

Figure 2.2: Architecture of a variational autoencoder

differentiable by removing the probabilistic variable, as shown in Figure 2.3. In this way, we can sample from the prior and use the parameters of the posterior to generate a sample in the posterior distribution. So, we can backpropagate through z and compute the gradients. The sample can be generated from the normal distribution (i.e., $\epsilon \sim \mathcal{N}(0, I)$), then the sample from the posterior distribution can be produced as follows:

$$z = \mu + \sigma\epsilon. \tag{2.8}$$

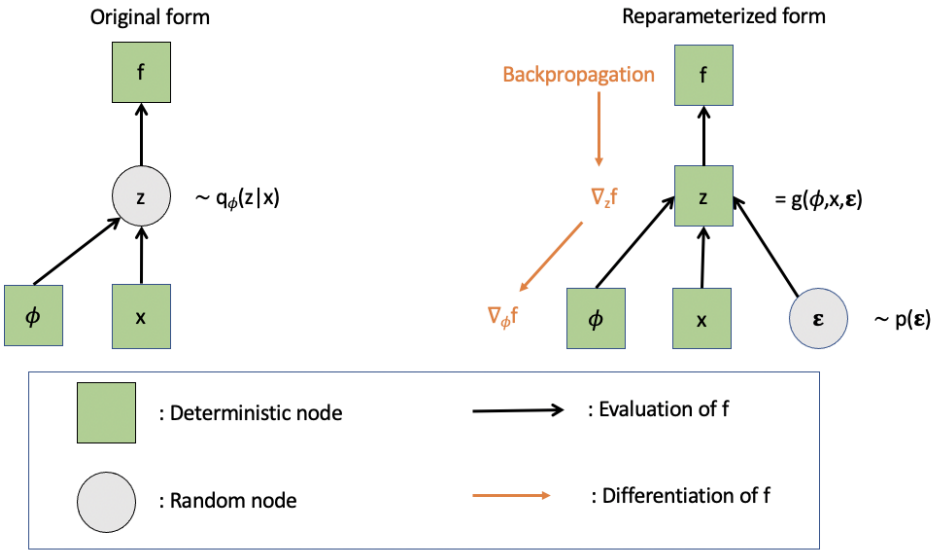

Figure 2.3: Demonstration of reparameterization trick, adapted from Kingma et al. [21]

This reparameterization trick helps us learn the mean and variance of the

posterior distribution by making the function differentiable. After training, the inference process consists of generating innovative samples and reconstructing input sentences by sampling latent vectors from the posterior distribution.

As mentioned, decoding can be used for generative purposes if the latent space is regularized. Regularization prevents overfitting and makes it possible for the latent space to produce new samples. The objective function (i.e., right-hand side of Equation 2.7) pushes the posterior distribution close to the standard normal distribution to minimize the loss function. On the other hand, it can cause the reconstruction error to increase. Therefore, there is a trade-off between the two terms in the loss function (i.e., reconstruction error versus KL divergence).

2.3 Parsing and Tree-Based Models

Context-Free Grammars

Context-free grammar (CFG) is a grammar that can be written as a 4-tuple $G =< N, \Sigma, R, S >$. In this tuple, N represents the set of non-terminal variables, Σ is the set of terminals disjoint from N (i.e., words in the natural language), and R denotes the set of grammar rules, which is a relation in $N \times (N \cup \Sigma)^*$. S is the starting symbol that represents the whole sentence and is an element of N. The rules in R can be used to generate sentences by calling them iteratively, to get a tree structure.

Context-free grammars have been used for a long time in Computational Linguistics. Figure 2.4 gives a CFG as an example, with the description of their notations shown in Table 2.1.

Probabilistic Context-Free Grammars

The key idea of probabilistic CFGs is that there will be a probability distribution over all the possible derivations and parse trees. Each group of the rules with the same left-hand side will have a different distribution from the others. When working with the grammar rules, the model tries to learn their distributions. New sentences can be generated based on the learned distributions

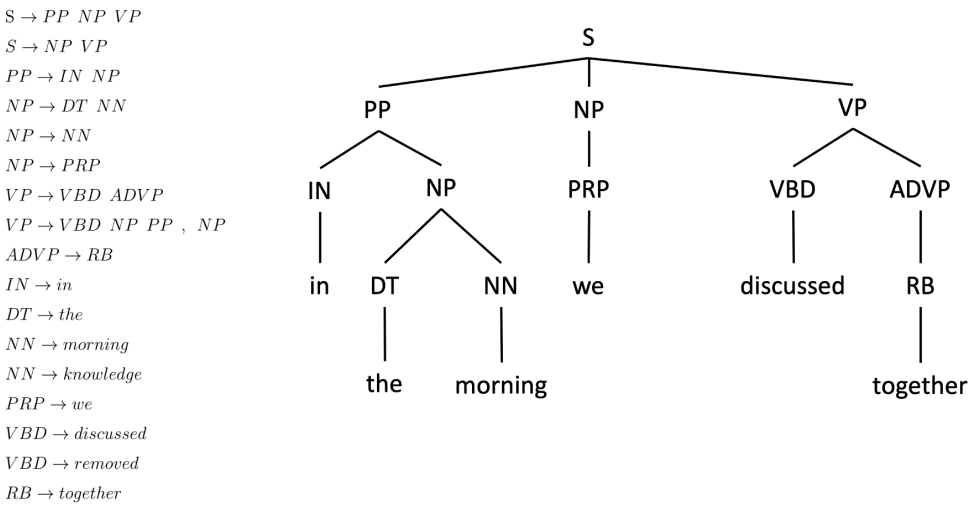

Figure 2.4: Example of context-free grammar

notation	description
S	sentence
PP	prepositional phrase
NP	noun phrase
VP	verb phrase
IN	preposition
DT	determiner
NN	noun
PRP	pronoun
VBD	verb (past tense)
ADVP	adverb phrase
RB	adverb

Table 2.1: Notation of context-free grammars for natural language

while considering the sequence of rules and tokens together.

Since each grammar rule is a part of a distribution, each rule has a specific probability. Furthermore, as already mentioned, a single distribution contains all the grammar rules with the same left-hand side. Hence, for each grammar rule $a \to b_i$ in the set of all grammar rules R, we have

$$p(a \to b_i) \geq 0. \tag{2.9}$$

We also have

$$\sum_{\forall i; a \to b_i \in R} p(a \to b_i) = 1. \tag{2.10}$$

Constituency Parse Tree

As our work is syntactical sentence generation, we must consider the parse trees. We use the Stanford Parser[1] to get the parse trees of the dataset. Then, based on those trees of the dataset samples, we use the parsed sentences to extract the natural language grammar rules.

The constituency parse trees are established based on context-free grammars. Processing the parse trees can assist us in learning a model that extracts the syntactical patterns in the natural language. Understanding these patterns enables the framework to know about generating valid phrasal constituents in natural language. By combining these phrases, valid sentences can be generated.

Our framework is mainly based on parse trees and their generation. The reason is that by using these trees, we can better understand the syntactic structure of sentences. According to section 3.2.1, switching the generation level of a sentence from words to phrases will assist us in producing grammatically correct sentences. Working with the phrases contained in the sentences is made more accessible by the constituency parse tree concept.

[1] https://nlp.stanford.edu/software/lex-parser.shtml

Chapter 3

Approach

3.1 Outline

In this thesis, our goal is natural language sentence generation. Our focus is on phrase-level and syntax-based generation instead of word-level generation [25], which is common in the literature. We hypothesize that a structured approach has the following benefit. The structural and syntax-based information helps to model the information flow in the neural network. Passing information happens from both the previous nodes and the parent node in the tree to use relevant phrases in the natural language.

This chapter is organized as follows. Section 3.2 introduces phrase-level sentence generation while it is compared to the word-level. We then present our phrase-level sentence generation approach. Section 3.3 discusses our training process, where we study the roles of grammar rules and phrases in the task of sentence generation. Finally, Section 3.4 discusses the inference after the model is trained. We talk about how the model can be applied to sampling new sentences and reconstructing a sentence from an input.

3.2 Tree-Based VAE

The primary contribution of this thesis is introducing phrase-level generation into unsupervised sentence generation. In this section, we discuss the motivation behind our contribution and the process of phrase-level sentence generation using the VAE architecture.

3.2.1 Drawback of Word-Level Generation

With a word-level sentence generation model, the continuous latent space of VAE mixes the syntax and semantic parts of a sentence [3]. Thus, the syntactic information is not explicitly modelled, which often results in generating grammatically incorrect sentences [52]. Shi et al. [41] demonstrate that not explicitly modelling the syntax makes many structural details missing while generating a sentence. Studies show that explicitly modelling the syntax enhances the generation quality in sequence-to-sequence models [9, 11, 24, 54].

In our proposed approach, the phrases are also considered during generation. So, the generated candidate is grammatically and syntactically correct. During the generation process, our approach considers the learned distribution of the natural language grammar rules and the distribution of generating tokens, not just the latter.

3.2.2 The Proposed Model

The final goal of training our model is to obtain the syntax tree of the given input sentence, while the tree has the highest score. Generating the syntax tree is accomplished by an intermediate step (i.e., transition) that obtains all the required actions to construct the sentence in a grammatical way. Those actions can be applied to build the constituency parse tree of the sentence, as described in Section 2.3. In this case, each subtree is viewed as a phrase. The proposed model tries to capture the distribution of the rules such that the final syntax tree is a regularized format of the most probable tree for the input sentence. So, we mainly focus on the part of the transition from input utterance to detailed actions, which can construct the final syntax tree. We employ a generation modelling technique, called variational autoencoder, to manage this transition part. Based on the parent and previous nodes in the tree structure, this model learns the probability of the subtrees in a parse tree.

Encoder and decoder are the two major components of VAE models, as was mentioned in Section 2.2.3. The input consists of natural language utterances, and the encoder converts each input sentence into a vectorial representation

using a bidirectional Long Short-Term Memory (LSTM) network. The probabilistic latent space uses the vectorial representation as the input, and a sample is generated from the distribution of the latent space. The decoder, a standard LSTM, regenerates the sentence by decoding it. However, this time, decoding also focuses on extracting the grammar rules. Obtaining the grammar rules can help in learning the transition phase. Section 3.3 contains more information on this.

As discussed in Section 2.2.3, VAE penalizes the divergence between posterior and prior of the latent variable z. This divergence results in posterior regularization, which enables the framework to generate sentences from the continuous latent space [2].

We require natural language grammar rules in order to achieve these actions for each input sentence. Then, the tree constructing actions are extracted, which help to generate a syntax tree. Thus, modelling syntax trees can be done using natural language grammatical formalization.

It is possible to observe the natural language grammar as a context-free grammar. The left side of each rule contains a single phrase, while the right side contains one or more phrases or tokens (i.e., vocabulary) from the language.

There are two main parts in the grammar: *type* and *constructor*. Types are either *composite* or *primitive*. If the type is composite, there will be at least one grammar rule with that type on its left so that the parse tree to generate the sentence will extend further. In this case, the constructor is on the right side of these rules. If the type is primitive, the values will be the terminal vocabularies within the dictionary in the natural language. So, whenever a primitive type appears, the tree expansion in that branch of the tree terminates, and a token is generated. Each constructor carries a list of fields as its parameters, and each field has a specific type, which may be primitive or composite. It is easy to handle a primitive type, as was previously mentioned. But, when facing a composite type, we will look for the grammar rules with the same left-hand side to expand the sentence further. The entire sentence is generated while the constructors of the sentence serve as the nodes of the syntax tree and the values of the fields serve as the edges of the tree, and our final goal is to

generate the corresponding tree of an input utterance.

For instance, a composite type S represents a *sentence*. Several rules in the natural sentence generation phase can have S on their left-hand side, such as $S \rightarrow NP\ VP$, where a sentence consists of a noun phrase followed by a verb phrase. This rule will be interpreted as follows. When the type S is observed while extracting the actions in the sentence, one of the rules that we can choose is to expand type S to NP and VP. The right-hand side, here $NP\ VP$, is called the constructor. Each of the NP and VP on the right-hand side expresses a field. They are expanded from S and have NP and VP types, respectively. As the next step, we will look at the grammar rules with NP and VP on their left, separately, to expand different tree branches. This process continues until no more composite type exists, such as the rule $NN \rightarrow$ *computer*, while *computer* is present in the dictionary and is a primitive type component.

After extracting the sequence of actions that produce each sentence, the syntax tree is created. The tree is interpreted as a set of actions in the order they are applied. The order determines the succession of constructing the nodes and edges when generating the syntax tree.

The transition phase is mostly based on the study of Yin et al. [50] for code generation. However, we must adapt this framework to natural language generation. At each step of this transition, we use one of the two following actions to extract all the actions used to form a specific syntax tree.

apply-constructor$[p]$ action takes phrase p as a parameter and applies a grammar rule with p on its left-hand side. It expands the derivation based on the frontier node in the tree having the phrase p and gets the sub-phrases based on the components on the right-hand side of the rule. Each element on the right-hand side of the grammar rule (i.e., sub-phrases) will construct a single node in the syntax tree.

generate-token$[t]$ action takes token t as a parameter that occupies the tree nodes when the frontier field has a primitive type. When this action is called, the token t is added to the tree as a leaf node.

So, the only required actions for sentence generation in the natural language are **apply-constructor**$[p]$ and **generate-token**$[t]$. The transition completes

when there is no other frontier node for the derivation to expand the tree. Stopping the expansion of the tree implies that all the tree branches have become leaves, which are no longer extensible.

Once these actions are learned to form the syntax tree, we can generate an output sentence described by the syntax tree structure. We extract the tree leaves (i.e., outputs of the generate-token actions) based on their applied order and generate a plain sequence of tokens as a sentence.

The aim of the model would be to learn the probabilities of each of these actions for different constructors and tokens, which will enable us to recreate sentences.

To do this, learning the probability of the constructor application action is not straightforward. The reason is that there may be various rules in the grammar that have the same left-hand side but different right-hand sides. So deterministic context-free grammars do not work here, and we need to utilize the probabilistic context-free grammars, as introduced in Section 2.3.

We can use the Probabilistic Context-Free Grammars (PCFGs) to generate phrases. By using PCFGs, not only are we able to generate new phrases that have not been seen before, but also we can access the tree structure of the input sentences. As a result, we can comprehend the grammatical components of sentences.

Based on what has been discussed in Section 2.3, a whole sentence will have a specific probability as we work with numerous probabilistic grammar rules for each sentence. Consider a set of rules $r_i : a_i \rightarrow b_i$, with $1 \leq i \leq n$, where n is the total number of grammar rules that have been applied to produce a sentence S. We can compute the probability of this sentence as follows:

$$Prob(S) = \prod_{i=1}^{n} Prob(a_i \rightarrow b_i). \tag{3.1}$$

In the next section, there will be more discussion on computing the action probabilities.

In our study, the decoder predicts the probability of a phrase or a token. This value is the product of the probabilities of all the grammar rules of the

subtrees that are produced as the decoder output. We combine several steps of generating phrases to develop a complete sentence.

3.3 Training Objectives

Our framework is based on the probability computation of the actions. These action probabilities are used at each step of sentence generation. In this section, we will cover training objectives for the proposed model. Different computations that will be mentioned in the following are mostly taken from the study of Yin et al. [50].

Our objective function is inspired by the loss function in the VAE (i.e., Section 2.2.3), including the reconstruction loss and the KL loss. We first focus on the reconstruction loss. As discussed next, this part of the loss handles how the input and output sentences can be more similar during training.

For a newly generated sample y, where an input x is given, we should break down the output into the different actions in the transition system process to compute the probability of producing y. Thus, the final value will be as follows,

$$Prob(y|x) = \prod_{t} Prob(a_t|a_{<t}, x), \tag{3.2}$$

where a_t is the action in time step t and y is generated by combining all the actions in its transition phase.

As mentioned earlier, our work relies on variational autoencoders that consist of encoder-decoder architecture. Hence, we compute the probability based on an encoder-decoder LSTM network.

Whenever an encoder is used in a neural network architecture, it is supposed to encode the input value into a vector representation. Since the input is a natural language sentence, we would like to encode the utterance x that contains n tokens, $\{x_i\}_{i=1}^{n}$, and get a vector of the encoded x.

As we work with the variational autoencoder architecture, the encoder network outputs a distribution (i.e., a normal distribution with its specified mean and variance). But the input to the decoder is not a distribution, and we sample from the encoded distribution in the latent space.

The decoder makes an effort to decode the sample from the encoded input. For each hidden state, s_t, the function is as follows,

$$s_t = f([a_{t-1} : \tilde{t}_{t-1} : \tilde{s}_{t-1} : p_t], s_{t-1}), \tag{3.3}$$

where f in the above equation is the LSTM function. The first parameter is a concatenation of several features, and we will discuss them in detail. a_{t-1} is the embedding of the previous action, $\tilde{t}_{t-1}$ is the embedding of the last generated token, and $\tilde{s}_{t-1}$ refers to the attention of the previous state as in the following equation

$$\tilde{s}_t = tanh(W_c[c_t : s_t]). \tag{3.4}$$

In Equation 3.4, c_t is the context vector from the hidden space using attention. The last term of the first parameter in Equation 3.3, called parent feeding, is responsible for taking care of the tree structure of the syntax trees and using the tree to generate valid sentences while considering their tree structure. Thus p_t focuses on the parent field, n_{ft}, and encodes its information. The information is made of the concatenation of embedding of the parent field and the decoder's state that generated the apply-constructor action of the parent field.

As previously seen in Section 3.2, we have to compute the probability of each apply-constructor[c] and generate-token[t] actions. Calculating the probability for these two actions is different. If the action is apply-constructor[c], the probability is as follows,

$$Prob(a_t = apply\_constructor[c] | a_{<t}, x) = softmax(a_c^T W \tilde{s}_t). \tag{3.5}$$

For the generate-token[t] action, we know that there are two possibilities: generating a new token without considering the input tokens or simply copying the same token seen in the input utterance. Therefore, the probability for generate-token[t] is in the following equation,

$$\begin{aligned} Prob(a_t = generate\_token[t] | a_{<t}, x) = {} & Prob(gen | a_t, x) Prob(v | gen, a_t, x) \\ & + Prob(copy | a_t, x) Prob(v | copy, a_t, x). \end{aligned} \tag{3.6}$$

$Prob(gen|.)$ and $Prob(copy|.)$ are computed similarly. Representing both *gen* and *copy* as *action*, we have

$$Prob(action|.) = softmax(W\tilde{s}_t). \tag{3.7}$$

The next computation term is $Prob(v|gen, .)$, which is similar to Equation 3.5, and $Prob(v|copy, .)$, which is as in Equation 3.8. Assuming the i-th token in the input utterance is supposed to be copied, we have the equation as follows,

$$Prob(x_i|copy, a_{<t}, x) = softmax(h_i^T W\tilde{s}_t), \tag{3.8}$$

where h_i is the encoded vector of x_i.

According to Equation 3.3, we consider the previous actions and the parent node in the tree at each time step during the decoding. These considerations help to model the sentences based on their hierarchies in the parse trees because the information of the parent node is important. In this case, when the frontier information in the tree structure of the sentence is considered, phrases themselves or the dependencies between different phrases and sub-phrases can be more consistent. Thus, we can generate syntactically correct sentences.

We mentioned earlier that there are two terms in the objective function of our model. The reconstruction loss (i.e., cross entropy) is what we have covered so far. This term tries to learn how to reconstruct the input sentence and make the output as close as possible to the input sample. The next term is the Kullback-Leibler loss (i.e., KL loss).

The KL loss is in charge of regularizing the objective function, as discussed in Section 2.2.3. This term computes the difference between the prior and posterior distributions of the latent variable z, using the KL divergence. The model learns a posterior distribution as a normal distribution. On the other hand, the prior distribution is a standard normal distribution. The final goal is to minimize the difference between these two.

As a recap, the training objective is to minimize the expected reconstruction loss that is regularized with the KL divergence between posterior (i.e., $q(z|x)$) and prior (i.e., $p(z)$) distributions while z is sampled from the continuous latent space.

The two terms constructing the loss function are contrary to each other. As the divergence between the prior and posterior distributions of the latent variable given any input x decreases, it becomes impossible to have a perfect reconstruction. Thus, a trade-off between these two goals should be satisfied.

3.4 Inference

According to Section 3.3, the model satisfies two factors while training: reconstruction and regularization. These two objectives do not act in the same direction. Thus, a learned model can be used for two tasks in the inference part: sentence reconstruction and sampling new sentences. Section 3.4.1 discusses the process of sentence reconstruction, while Section 3.4.2 talks about how sentences can be generated from scratch.

3.4.1 Sentence Reconstruction

Reconstructing a sentence requires a plain input sentence to be given into the model. It is then encoded, and a sample from the learned continuous latent space is generated. Then, the generated sample is fed into the decoder. Afterwards, the decoding phase begins. All of these steps are based on the model architecture that was described in Section 3.2.2.

This model consists of an encoder and is supposed to regenerate the input sentence. If we use the whole model architecture without any changes, we can reconstruct the given input sentences, including their phrases and tokens. Reconstruction enables us to assess how the autoencoder model can preserve the input information.

Moreover, since our model tends to be regularized and to be able to generalize well, having the best reconstruction of the input may not be possible. The reason is that the continuous latent space is a distribution. So, a sample from this distribution is generated before decoding starts, which results in an imperfect reconstruction. Hence, phrases and tokens within the sentences may differ from the input. The KL divergence term of the loss function is the cause of this phenomenon.

3.4.2 Sampling New Sentences

There is another application other than sentence reconstruction for our learned model. Consider a situation where there is no input sentence. We only would like to assess the ability of the model to generate unseen sentences based on grammar.

We can use our proposed model in this case, but we need some modifications to the original encoder-decoder architecture. The reason is that we want to create sentences entirely from scratch and do not have any input sentences.

The regularization of our proposed model, which stopped us from a perfect reconstruction in Section 3.4.1, can help us here to generate various novel sentences from scratch.

For the goal of generating new samples, there is no need for an input sentence. A random sample from the prior (i.e., standard normal distribution) is generated to skip the encoder since there is no input to be encoded. Then, the sample is fed into the decoder. The decoder chooses an action at each step based on the input sample. The action can both apply constructors and generate tokens. The final result is a generated sentence including all the phrases and tokens in it, based on the order they have been applied. Thus, new sentences can be produced.

Chapter 4

Experiments and Results

4.1 Overview

This chapter presents an experimental evaluation of our proposed model. We discuss the dataset used in our assessment, the evaluation metrics, the implementation details of our framework, and a performance comparison of our method with the baseline.

4.2 Dataset

For evaluating our model, we use the Stanford Natural Language Inference (SNLI) dataset [6][1] for the training and validation of the model. This dataset can be used for sentence generation and reconstruction via the tree-based variational autoencoder.

4.2.1 SNLI Dataset

This dataset consists of sentences written for an image captioning task through crowdsourcing. The SNLI dataset is massive, which is a collection of 570k sentences; however, the sentences are generally simple. This corpus is used as a resource for various NLP tasks.

We aim to generate sentences in an unsupervised task by learning the underlying patterns in the natural language sentences. Knowing the patterns helps us to learn a model for developing sentences. An unlabelled dataset

[1] https://nlp.stanford.edu/projects/snli

Sentences
the women are walking down the street
a couple are going for a long hike in the forest
a brother is taking his sibling for a stroll
a woman is climbing a rocky hill overshadowed by a partly cloudy sky
a young woman has blond hair
a man is preparing for the graduation ceremony
a dog playing with its owner

Table 4.1: Samples from SNLI dataset

consisting of simple human-written natural language sentences enables us to learn the structures and patterns of these sentences. Some sample sentences from this dataset are shown in Table 4.1. After training the model, we can generate new sentences from scratch or reconstruct given input sentences.

4.2.2 Data Pre-Processing

In most of the datasets, the text data is not entirely clean or we need to extract some information from the data. So we need to pre-process our dataset to prepare it for training the model. The following data pre-processing steps are adopted for our dataset:

- All sentences are converted to lower case.
- All the words are embedded using Word2Vec, which is required for initializing the word embeddings. The CBOW model was used with a word embedding dimension of 300d and a context window size of 5.
- The unsupervised setting of the task would be to split the dataset the same as the baseline. So, the dataset is divided into train/validation/test with 90%/5%/5% of the whole sentences.
- We detect the phrases in all train and validation sentences by parsing them using the Stanford Constituency Parser[2].
- Since we are working with the grammar rules, all the rules in the natural language must be extracted from the parsed dataset using NLTK.

[2]https://nlp.stanford.edu/software/lex-parser.shtml

4.2.3 Pre-Train Embedding

As mentioned in the previous section, we need to generate a word embedding vector for each token in the corpus. Embedding the words into numerical vectors enables the computer to process the data. We need vectors representing the words in the text corpus that encode the meaning of the words. The closer the words are in the vector space, the more they are similar in their meaning. Word2Vec [28] is a common method for word embedding because it is efficient in the size of the vector and considers the semantic information of the words, which are not taken into account in the basic methods. Moreover, it is able to compute the word vectors fast [28]. So we use Word2Vec in our work.

The structure to learn Word2Vec embeddings is a neural network with two layers. The input is the text, and the output is the set of word embedding vectors. Each output vector is the weight matrix of the network for the corresponding word in the corpus after the training is done.

Continuous Bag of Words (CBOW) and continuous skip-gram models are the two different algorithms for learning Word2Vec embeddings [28]. Their difference is that CBOW uses the surrounding words as the context to predict a word, while continuous skip-gram uses a word to predict its context. We use the CBOW model in our work because of the more computational complexity in the continuous skip-gram model [28]. So CBOW trains the model faster, which works better for massive datasets. Furthermore, it is more consistent with our baseline, the study of Bahuleyan et al. [2], so our results can be compared more accurately.

4.3 Training Setup

Our proposed model includes an autoencoding part that uses variational autoencoder architecture over the whole sentence of each input sample. This model finds the posterior $q(z|x)$ as close as possible to the prior distribution $p(z)$, where x is the sequence of words in the input and the prior is the standard normal distribution. As mentioned in Section 4.2.2, Word2Vec is used to generate the word embeddings. Training of the embedding is done on the

LSTM Hidden Dimension	100d
Latent Dimension	100d
Word Embeddings	300d
Action Embed Size	100d
Field Embed Size	20d
Type Embed Size	20d
Epochs	20
Learning Rate	0.001 (constant number)
Batch Size	128
Vocab Size	30000 (most frequent tokens)
Dropout Rate	0.2

Table 4.2: Experimental settings for the autoencoding model

whole SNLI dataset.

Because of the KL collapse issue in the VAE model, for the RNN-based text settings, there are some methods to avoid it and make the posterior more dependent on the input. One of these approaches is cyclical annealing [13], which we will discuss in Section 4.5.2. The total number of cycles is set to 10, which means the value will be set to zero 10 times during the training process, and we will start annealing from that point again. The reason for choosing this number for cycles is that the model should be able to achieve a KL loss that converges to a non-zero value. By having experiments, it is demonstrated that these conditions will be violated by choosing smaller and larger values. The value of coefficient λ is annealed with a sigmoid function.

Experimental settings are mentioned in Table 4.2 in detail. For the encoder and the decoder, LSTM architecture is used, and for the encoder, it is a bidirectional LSTM (i.e., bi-LSTM). Adam [18] is used as the optimizer of the model with the parameters β_1 and β_2 set to 0.9 and 0.999, respectively. Action, field, and type embed sizes refer to the tree structure of the model, which are covered in Section 3.2.2.

4.4 Evaluation Metrics

After designing and training the model, to figure out how the model is working, it is necessary to assess and analyze its performance. The inference phase of

the model, as mentioned in Section 3.4, consists of reconstructing the given sentences and sampling new sentences. For the reconstruction task, the BLEU score [31] measures the similarity between the input and the generated output. On the other hand, perplexity (PPL) evaluates the fluency of the generated sentences for the sentence generation task. Moreover, unigram-KL (UniKL) measures if the distribution of the generated sentences is close to the training corpus, and the entropy checks the similarity of the generated sentences to the corpus. We will discuss these measurements as follows.

The first metric for this task is the BLEU score [31], computed between a model-generated output and a ground truth. It was introduced initially for the task of Machine Translation (MT) systems, but it was also used to evaluate the reconstruction performance of autoencoders. We will use this measurement for the reconstruction task, where the output of the model is the reconstruction of the input, and the input serves as the ground truth. This measurement helps to assess the ability of the proposed model to preserve the input information.

To explain more, BLEU (Bilingual Evaluation Understudy) can be applied to the reconstruction task because it can measure the overlap between the set of reference sentences with the generated sentence. Two factors play roles in this score computation. One of them controls the brevity penalty, while the other, which is precision-based, computes the n-gram overlaps. It penalizes short generated sentences by comparing them to their proper length. By not considering this penalty, precision for the shorter sentences will be higher, which means a better score. Obtaining better scores in this way is not desired because it encourages the model to generate shorter sentences and be assured that a high score is achieved, even though that short sentence may not be informative. Additionally, Papineni et al. [31] report that the proposed approach is correlated with human judgment and will have various benefits, such as less time computation and reusability.

The other metric is perplexity (PPL), which determines the fluency of the newly generated samples without having input sentences. So, random samples from the continuous latent space (i.e., standard normal distribution) are chosen. Then, the sentences are generated by decoding those samples. It

is a way of probabilistic sampling and sentence generation. Therefore, the model can generate novel sentences that are not present in the training set. These sentences should be semantically and syntactically satisfactory. Before computing the perplexity of a sentence, a language model should be trained.

In our experiment, we learned the language model using Kneser-Ney interpolation, an interpolated version of Kneser-Ney smoothing. The model at the end is an n-gram language model. Kneser-Ney smoothing calculates the probability distribution of n-grams in the corpus, which identifies the possibility of n-grams that can appear in a language. We consider 5-grams in this study.

Moreover, unigram-KL (UniKL) evaluates the word distribution closeness of the original sentences (i.e., training corpus) with the generated sentences. The entropy of the word distribution is also reported. It measures the similarity between the generated sentences from the latent space and the corpus.

4.5 VAE Optimization Challenges

As mentioned previously, VAE consists of two terms in its loss function. The first one is the reconstruction loss, which is the likelihood of the input given the posterior distribution and can be the cross entropy loss. The other term is the Kullback–Leibler (KL) divergence loss, which is the divergence between the posterior distribution $q(z|x)$ and the prior of the latent space $p(z)$.

Bowman et al. [7] observe that implementing the vanilla VAE framework using an autoregressive model fails to learn well. The KL divergence vanishes for NLP tasks when the model is based on RNNs. Thus, VAE is challenging to train for the RNN-based models. When using VAE in RNN encoder-decoder models for text generation, in most cases, the model learns in a way that the posterior equals the prior distribution. Thus, it is inevitable to make the KL divergence term of the loss function intend to zero. KL vanishing results in a decoder that ignores the latent variable z [49]. So, there is no helpful information passing through the network from the input. But, the goal of VAE is to store useful information in the latent space, and it needs the KL divergence term to be non-zero and also to have a small value for the

reconstruction loss.

If this does not happen, the KL divergence term will be zero, and only the first term in the cost function is present, which results in a deterministic autoencoder model. This phenomenon has the outcome that each input is memorized in the model as a single point instead of having a continuous latent space distribution [49]. However, we need a latent space (i.e., posterior distribution) that is not a single point, which is the source of being variational.

Fu et al. [13] find an explanation for why this happens. As shown in Figure 4.1(a), the traditional VAE learns to construct the model easily. But the problem appears with the autoregressive encoder-decoders because each step depends on the previous step. This phenomenon causes the vanishing of the KL divergence. Let us explain the reason. Figure 4.1(b) has two parts that reconstruct the input x. One part is the same as Figure 4.1(a), and the other part obtains information from the previous steps of the sequential decoding of the input x itself. This second part generates the input again in the time step t based on the input's decoding output for the time steps smaller than t, which makes the model find an easier path to regenerate input x, tending the KL loss to zero.

But this occurrence has a reason behind it, which is related to the beginning of the training process, where decoding z in the first part (i.e., the part illustrated in Figure 4.1(a)) has low quality. Low quality adds more trouble to the reconstruction; thus, the model tends to learn a simpler path to reach the goal. That leads the model to lean more toward the other path, i.e., being dependent on the autoregressive decoder. As Mikolov et al. [29] mentioned, autoregressive decoders are able to obtain complex distributions, such as in language models.

Some methods, such as KL annealing [7], are proposed to balance these two terms of the objective function. A modified version of KL annealing is adopted in this study and will be described in the following sections.

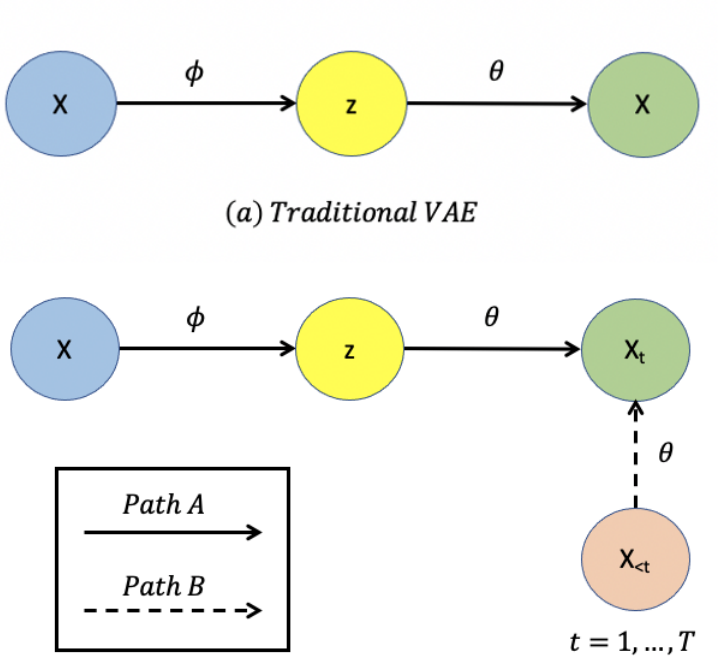

Figure 4.1: Two approaches of learning VAE, adapted from Fu et al. [13]

4.5.1 KL cost annealing

In this approach, the objective function of VAE goes through a simple change. The change is to add a coefficient to the KL divergence term. Hence, the new objective function will be as follows,

$$J = J_{reconstruct}(\phi, \theta, x) + \lambda D_{KL}[q_\phi(z|x)||p_\theta(z)]. \tag{4.1}$$

This weight (i.e., λ coefficient) is zero at first and then is gradually annealed to a threshold value, which is almost always 1 [7]. In other words, at first, it is a vanilla autoencoder and slowly transforms into a variational autoencoder. It works because, as the process starts, the model learns to reconstruct the input well and encode information into the latent space. Then, when the weight is slowly increased, the input encoding is mapped to continuous latent space, while the posterior distribution is pushed as close as possible to the prior. At the end of the annealing, the cost function is the variational lower bound, as in Equation 2.7. The value of λ is set at each step based on the iteration number

and the sigmoid function.

4.5.2 Cyclical KL Cost Annealing

KL cost annealing gives less weight to the prior regularization at first, which, in return, makes the posterior prone to collapse into a single point instead of being a distribution. By underestimating the regularization, the decoder can be weak and suboptimal [13]. Cyclical annealing [13] was proposed to improve the latent representation.

It is introduced in a way that, at first, the weight is zero, which opens path A in Figure 4.1, then it anneals fast to become one and adds path B in Figure 4.1 (i.e., such as Section 4.5.1). Then, for several iterations, the weight is constant. At this step, the model has the variational autoencoder objective function. So, the first version of the latent distribution is found. But, the weight cannot be set to one for the rest of the learning. Because by adding path B to the model, it is a simpler path for the model to learn the reconstruction rather than the global latent representation (i.e., path A). Therefore, path A will be blocked for the rest of the training. Since learning the latent representation z is required, the weight is again set to zero, and the process is repeated. By doing so, the decoder can be trained again with the determined latent space. Also, by iteratively doing the annealing, there is the opportunity to converge to a better latent space and progressively improve the latent representation [13].

The process is split into the number of cycles determined at the beginning. As each cycle starts, the weight is zero and gradually increases to become one. Then for the subsequent remaining iterations, until the starting point of the next cycle, the weight is fixed at one to make the model a VAE. This process happens for several cycles until the whole learning process is finished. The difference between the weights of a cyclical and a monotonic weight annealing can be seen in Figure 4.2.

Fu et al. [13] compare and visualize different weight schedule methods for the KL divergence term in VAE models. The constant weight case generates a latent space, where different latent variables are mixed for different samples

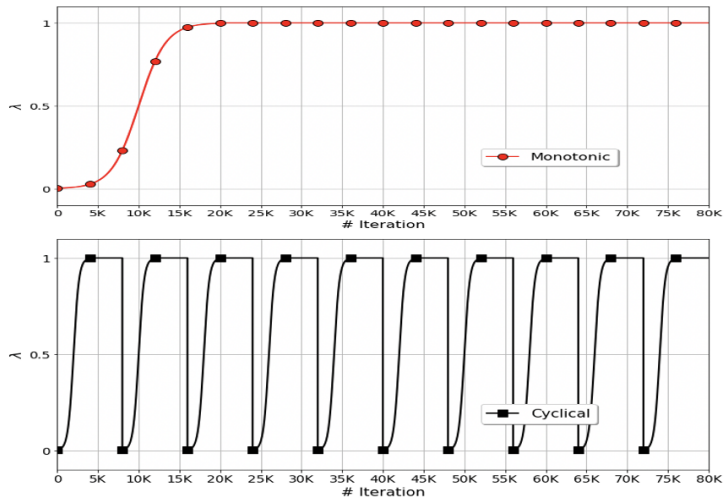

Figure 4.2: Comparison between the monotonic and cyclical weight annealing with the number of 10 cycles and λ as the weight, adapted from a similar illustration from Fu at al. [13].

in the dataset. For the monotonic annealing, the space is still mixed but more divided into clusters compared to the constant schedule. Finally, when cyclical annealing is applied, the result is similar to the monotonic case after the first cycle, but continuing the annealing several times produces divided clusters at the end, and all the variables of the latent space are separate.

The final result of our method can be seen in Figure 4.3, with different tricks for the value of λ. The figure shows that the cyclical annealing policy results in a more converged value in comparison to the others for the KL divergence in the loss term, with a non-zero value.

4.6 Evaluation and Analysis

This study evaluates the model for sentence reconstruction and sampling from the latent space tasks. The results are shown in Table 4.3.

We examine the BLEU [31] score of the reconstructed sentences to measure the goodness of the model in regenerating the sentences. Considering the mean (μ) of the latent space distribution, and ignoring its variance, results in a better reconstruction. The reason is that considering the variance makes the model inclined to generate varied sentences from the input, which is not desired for

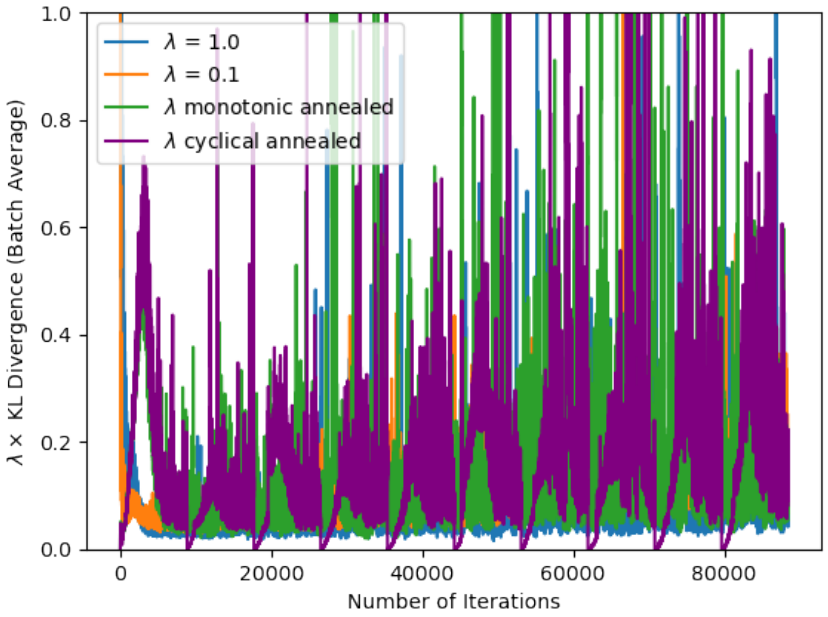

Figure 4.3: KL term of the VAE loss function (λ.KL) learning curve for different values of λ and the effect of cyclical λ annealing.

reconstruction.

On the other hand, for random sampling that is appropriate for evaluating the ability of sentence generation from scratch, we randomly sample from the prior distribution (i.e., $\mathcal{N}(0, I)$). Then we feed that sample to the decoder as z to generate new sentences. Only the decoder is observed in the inference phase, and the encoder is ignored. The generated sentence that is sampled from the latent space is expected to consider the syntax and semantics and to be close to the natural language.

	BLEU↑	**PPL↓**	**UniKL↓**	**Entropy**
Corpus†	-	-	-	→5.65
DAE†	86.35	146.2	0.178	**6.23**
VAE (kl-annealed)†	43.18	**79.4**	**0.081**	5.04
tree-DAE	**90.67**	11688.44	0.094	1.992
tree-VAE (cyclical kl-annealed)	90.54	10160.31	0.094	1.993

Table 4.3: Results of different autoencoder models for text generation. ↑ / ↓ means the larger/lower, the better. For the Entropy, the better happens when the result is close to the corpus distribution that is indicated by →. †Results quoted from the previous paper [2]; others are given by our experiments.

The results in Table 4.3 for DAE (i.e., Deterministic Autoencoder) and VAE (i.e., Variational Autoencoder) are reported from the study of Bahuleyan et al. [2]. The results of our framework are reported under tree-DAE and tree-VAE since we are using the parse tree structures of the sentences, which also help us to learn the model syntactically. To compare the models, we obtain the BLEU score, perplexity (PPL), UniKL, and the entropy of the variational and deterministic models for reconstruction and random sampling.

We will first start with sentence reconstruction. The best BLEU score is obtained using the tree-DAE model, demonstrating that learning a model that relies on both the tokens and their grammatical structures results in a much better score when reconstructing a given input. The score is significantly higher than that of a simple model that does not use the grammar rules. The score of tree-DAE is higher than that of the tree-VAE model. The reason is that the tree-VAE model aims for a distribution in the latent space, resulting in variation in the output from the input. In contrast, tree-DAE looks for a deterministic vector, not a distribution. So, tree-DAE can reconstruct better than tree-VAE.

On the other hand, in terms of the fluency of the generated sentences from the latent space, the scores of the tree-based models are higher than the simple DAE and VAE models. But a model is considered a better model in terms of fluency if its score is lower. However, when analyzing the generated sentences, it is observed that they follow grammatical rules and apply the proper rules at each step. Still, the tokens chosen as the tree leaves do not generate reasonable sequences in natural language. Some of these generated sentences can be seen in Table 4.4, where the generated samples are non-fluent, though they are generated grammatically and syntactically, based on the rules. One of the instances of the trees can be seen in Figure 4.4. In the case of the two tree-based models, the deterministic model, which does not have a probabilistic latent space, generates a higher PPL score than the variational model. So the variational model can generate more fluent sentences than the deterministic model, while it is still worse than the basic model that does not work with the grammar rules.

Type	Generated Samples
Simple Linearized Tree	hands being playing (ROOT__X (X__NP_NP_NP (NP__NNS (NNS__hands hands)) (NP__JJ (JJ__being being)) (NP__NN (NN__playing playing))))
Simple Linearized Tree	building being there clothed (ROOT__X (X__X_ADJP (X__ADVP_NP (ADVP__NN (NN__building building)) (NP__JJ (JJ__being being))) (ADJP__RB_VBN (RB__there there) (VBN__clothed clothed))))
Simple Linearized Tree	an there (ROOT__X (X__X_ADJP (X__DT (DT__an an)) (ADJP__RB (RB__there there))))
Simple Linearized Tree	produce (ROOT__X (X__NP (NP__NN (NN__produce produce))))
Simple Linearized Tree	several wearing whenever red wearing a one with be out (ROOT__ADJP (ADJP__ADJP_SBAR (ADJP__JJ_NN (JJ__several several) (NN__wearing wearing)) (SBAR__WHADVP_S (WHADVP__WRB_ADJP (WRB__whenever whenever) (ADJP__JJ_JJ (JJ__red red) (JJ__wearing wearing))) (S__NP_ADVP_VP (NP__NNP (NNP__a a)) (ADVP__NP_IN (NP__CD (CD__one one)) (IN__with with)) (VP__VB_NP (VB__be be) (NP__NN (NN__out out))))))))
Simple Linearized Tree	old tennis an her near one both down near hands visible (ROOT__ADJP (ADJP__ADJP_SBAR (ADJP__JJ_NN (JJ__old old) (NN__tennis tennis)) (SBAR__WHNP_S (WHNP__DT (DT__an an)) (S__S_CC_S (S__ADVP_VP (ADVP__NP_IN (NP__PRP (PRP__her her)) (IN__near near)) (VP__NN (NN__one one))) (CC__both both) (S__NP_ADJP (NP__RB (RB__down down)) (ADJP__NP_JJ (NP__JJ_NNS (JJ__near near) (NNS__hands hands)) (JJ__visible visible))))))))
Simple Linearized Tree	several hands wearing picture (ROOT__X (X__NP_NP_NP (NP__JJ_NNS (JJ__several several) (NNS__hands hands)) (NP__JJ (JJ__wearing wearing)) (NP__NN (NN__picture picture))))

Table 4.4: Generated sample examples with the tree-structured VAE

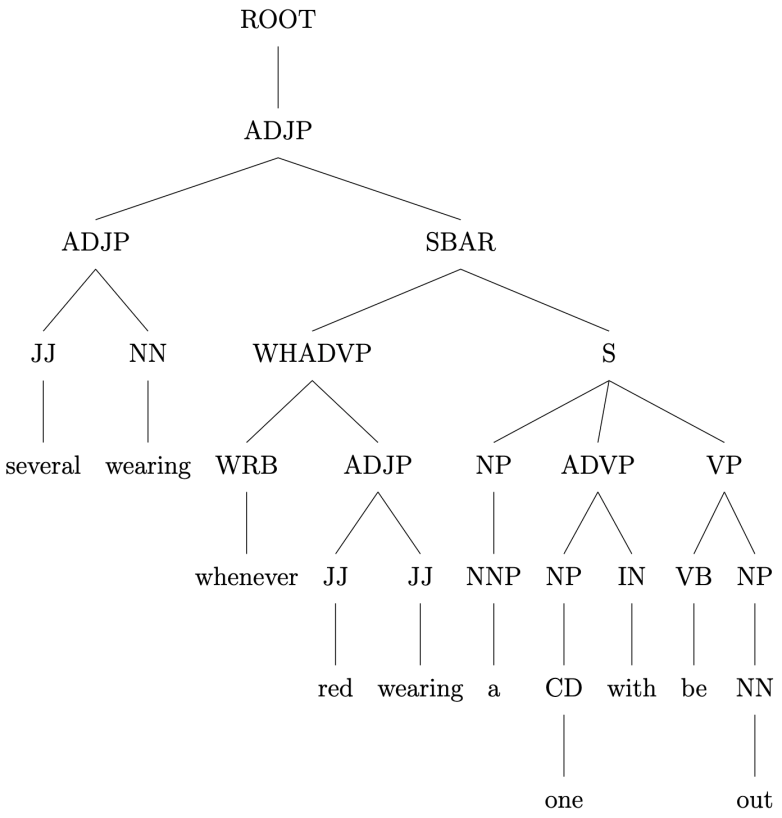

Figure 4.4: Tree of a generated sample from the latent space

Moreover, the UniKL is the lowest for the VAE model, which shows that the generated sentences for the VAE model are closer to the corpus distribution. This value for the tree-based models is slightly higher.

Again, the rules do not affect the best model in terms of entropy, which means the simple DAE model gives the best score. So the simple DAE model generates more similar sentences from the latent space to the corpus than any other model that we consider in this study.

In summary, the proposed model works significantly better in terms of reconstruction. Regarding fluency, the generated sentences follow syntactical concepts when considering the parse tree, but they are not strong when choosing the sequence of tokens.

Chapter 5

Conclusion and Future Work

5.1 Summary of Contributions

Our main goal in this research is to develop a deep learning approach for sentence reconstruction and probabilistic natural language generation. In order to achieve this goal, we implement an unsupervised variational autoencoder model. The model focuses on the words in a sentence and also the syntax between words and phrases to learn a sentence structure. This model can generate sentences at the phrase-level instead of the word-level. Generating sentences based on grammar rules improves the quality of the produced sentences grammatically.

We discussed the drawbacks of word-level sentence generation, and why our proposed model considers the phrases and the syntactic tree of the sentences in addition to the tokens.

However, there is difficulty in training the VAE models in natural language tasks when an autoregressive model is learned. The KL divergence term of the loss function collapses to zero in sequential models. We apply a cyclical annealing strategy for the KL loss term in order to avoid it tending to zero and instead converging to a non-zero value.

The proposed model can reconstruct sentences relying on the parse tree of the sentences and generating various sentences from the continuous latent space. This model is evaluated with BLEU score, perplexity, UnigramKL, and entropy. The BLEU score evaluates the n-gram similarity of the input and output. At the same time, the perplexity is responsible for assessing the

semantic part of the generated sentences and determining how the model can generate fluent novel sentences that are correct semantically. Additionally, UnigramKL assesses the distribution closeness of the training corpus with the generated sentences. Finally, entropy considers the similarity between the generated sentences and the corpus.

5.2 Limitations and Future Work

There are some challenges in this study. Compared to the earlier studies that are not grammar-based, this model requires more time to learn and is more complicated. The reason is that grammar rules and their applying order in generating sentences play a vital role in our model, and we are explicitly modelling this aspect of the sentences.

Moreover, to continue with this model, we should extract the rules in the natural language before learning, which is not needed in the models that implicitly learn the structure of the sentences.

The approach in this thesis for generating sentences is based on LSTM, which is a recurrent neural network model. But by introducing transformers [46], different models have been proposed that work effectively in natural language tasks. Since there is more training parallelization for these models compared to sequential models, they can be used as a future step to improve our model in order to reduce the learning time of the model.

References

1. Bahdanau, D., Cho, K. & Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
2. Bahuleyan, H., Mou, L., Zhou, H. & Vechtomova, O. Stochastic Wasserstein autoencoder for probabilistic sentence generation. *arXiv preprint arXiv:1806.08462* (2018).
3. Bao, Y. *et al.* Generating sentences from disentangled syntactic and semantic spaces. *arXiv preprint arXiv:1907.05789* (2019).
4. Bastings, J., Titov, I., Aziz, W., Marcheggiani, D. & Sima'an, K. Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675* (2017).
5. Bengio, Y. *Deep learning of representations for unsupervised and transfer learning* in *Proceedings of ICML workshop on unsupervised and transfer learning* (2012), 17–36.
6. Bowman, S. R., Angeli, G., Potts, C. & Manning, C. D. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326* (2015).
7. Bowman, S. R. *et al. Generating Sentences from a Continuous Space* in *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning* (Association for Computational Linguistics, Berlin, Germany, Aug. 2016), 10–21. `https://aclanthology.org/K16-1002`.
8. Cambria, E. & White, B. Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]. *IEEE Computational Intelligence Magazine* **9,** 48–57 (2014).
9. Chen, H., Huang, S., Chiang, D. & Chen, J. *Improved Neural Machine Translation with a Syntax-Aware Encoder and Decoder* in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2017), 1936–1945.
10. Cho, K. *et al.* Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

11. Eriguchi, A., Hashimoto, K. & Tsuruoka, Y. *Tree-to-Sequence Attentional Neural Machine Translation* in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2016), 823–833.

12. Fefferman, C., Mitter, S. & Narayanan, H. Testing the manifold hypothesis. *Journal of the American Mathematical Society* **29,** 983–1049 (2016).

13. Fu, H. *et al. Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing* in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (Association for Computational Linguistics, Minneapolis, Minnesota, June 2019), 240–250. `https://aclanthology.org/N19-1021`.

14. Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

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

16. Guu, K., Hashimoto, T. B., Oren, Y. & Liang, P. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics* **6,** 437–450 (2018).

17. Jordan, M. I. & Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science* **349,** 255–260 (2015).

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

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

20. Kingma, D. P. & Welling, M. Auto-Encoding Variational Bayes. *CoRR* **abs/1312.6114** (2014).

21. Kingma, D. P. Variational inference & deep learning: A new synthesis (2017).

22. Kramer, M. A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal* **37,** 233–243 (1991).

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

24. Li, J. *et al. Modeling Source Syntax for Neural Machine Translation* in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2017), 688–697.

25. Liu, X. *et al.* Unsupervised paraphrasing by simulated annealing. *arXiv preprint arXiv:1909.03588* (2019).

26. Miao, N., Zhou, H., Mou, L., Yan, R. & Li, L. *Cgmh: Constrained sentence generation by metropolis-hastings sampling* in *Proceedings of the AAAI Conference on Artificial Intelligence* **33** (2019), 6834–6842.

27. Miao, Y. & Blunsom, P. Language as a latent variable: Discrete generative models for sentence compression. *arXiv preprint arXiv:1609.07317* (2016).

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

29. Mikolov, T., Karafiát, M., Burget, L., Cernock, J. & Khudanpur, S. *Recurrent neural network based language model.* in *Interspeech* **2** (2010), 1045–1048.

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

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

32. Quirk, C., Brockett, C. & Dolan, W. B. *Monolingual machine translation for paraphrase generation* in *Proceedings of the 2004 conference on empirical methods in natural language processing* (2004), 142–149.

33. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., *et al.* Improving language understanding by generative pre-training (2018).

34. Rich, E. *Artificial intelligence* (McGraw-Hill, Inc., 1983).

35. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. *Learning internal representations by error propagation* tech. rep. (California Univ San Diego La Jolla Inst for Cognitive Science, 1985).

36. Rush, A. M., Chopra, S. & Weston, J. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685* (2015).

37. Samuel, A. L. Some studies in machine learning using the game of checkers. II—Recent progress. *IBM Journal of research and development* **11,** 601–617 (1967).

38. Serban, I. *et al. A hierarchical latent variable encoder-decoder model for generating dialogues* in *Proceedings of the AAAI Conference on Artificial Intelligence* **31** (2017).

39. Shao, L. *et al.* Generating high-quality and informative conversation responses with sequence-to-sequence models. *arXiv preprint arXiv:1701.03185* (2017).

40. Shen, X., Su, H., Niu, S. & Demberg, V. *Improving variational encoder-decoders in dialogue generation* in *Proceedings of the AAAI conference on artificial intelligence* **32** (2018).

41. Shi, X., Padhi, I. & Knight, K. *Does string-based neural MT learn source syntax?* in *Proceedings of the 2016 conference on empirical methods in natural language processing* (2016), 1526–1534.

42. Song, T., Sun, J., Zhang, Y., Song, J., *et al.* An RNN Model for Generating Sentences with a Desired Word at a Desired Position. *Tehnički vjesnik* **27,** 81–88 (2020).

43. Steck, H. Autoencoders that don't overfit towards the Identity. *Advances in Neural Information Processing Systems* **33,** 19598–19608 (2020).

44. Sutskever, I., Martens, J. & Hinton, G. E. *Generating text with recurrent neural networks* in *ICML* (2011).

45. Sutskever, I., Vinyals, O. & Le, Q. V. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* **27** (2014).

46. Vaswani, A. *et al. Attention is all you need* in *Advances in neural information processing systems* (2017), 5998–6008.

47. Vinyals, O., Toshev, A., Bengio, S. & Erhan, D. *Show and tell: A neural image caption generator* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), 3156–3164.

48. Wu, Y. *et al.* Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

49. Yang, Z., Hu, Z., Salakhutdinov, R. & Berg-Kirkpatrick, T. *Improved variational autoencoders for text modeling using dilated convolutions* in *International conference on machine learning* (2017), 3881–3890.

50. Yin, P. & Neubig, G. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. *arXiv preprint arXiv:1810.02720* (2018).

51. Yuan, W., Ding, L., Meng, K. & Liu, G. *Text Generation with Syntax-Enhanced Variational Autoencoder* in *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), 1–8.

52. Zhang, X., Yang, Y., Yuan, S., Shen, D. & Carin, L. *Syntax-Infused Variational Autoencoder for Text Generation* in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (Association for Computational Linguistics, Florence, Italy, July 2019), 2069–2078. `https://aclanthology.org/P19-1199`.

53. Zhang, Y., Gan, Z. & Carin, L. *Generating text via adversarial training* in *NIPS workshop on Adversarial Training* **21** (2016), 21–32.

54. Zhou, H. *et al. Chunk-Based Bi-Scale Decoder for Neural Machine Translation* in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (2017), 580–586.