Natural Language Generation with Neural Variational Models

by

Hareesh Pallikara Bahuleyan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Management Science

Waterloo, Ontario, Canada, 2018

Author Declaration

This thesis consists of material all of which I authored or co-authored: see *Statement of Contributions* included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Chapters 4, 5 and 6 of this thesis are based on the following papers:

1. **Hareesh Bahuleyan***, Lili Mou*, Olga Vechtomova, and Pascal Poupart. Variational Attention for Sequence-to-Sequence Models. In *27th International Conference on Computational Linguistics (COLING)*, 2018.

2. **Hareesh Bahuleyan**, Lili Mou, Kartik Vamaraju, Hao Zhou, and Olga Vechtomova. Probabilistic Natural Language Generation with Wasserstein Autoencoders. *arXiv preprint arXiv:1806.08462*, 2018

I have contributed to implementation, experimentation, and preparation of the manuscript of the above mentioned papers.

Abstract

Automatic generation of text is an important topic in natural language processing with applications in tasks such as machine translation and text summarization. In this thesis, we explore the use of deep neural networks for generation of natural language. Specifically, we implement two sequence-to-sequence neural variational models - variational autoencoders (VAE) and variational encoder-decoders (VED).

VAEs for text generation are difficult to train due to issues associated with the Kullback-Leibler (KL) divergence term of the loss function vanishing to zero. We successfully train VAEs by implementing optimization heuristics such as KL weight annealing and word dropout. In addition, this work also proposes new and improved annealing schedules that facilitates the learning of a meaningful latent space. We also demonstrate the effectiveness of this continuous latent space through experiments such as random sampling, linear interpolation and sampling from the neighborhood of the input. We argue that if VAEs are not designed appropriately, it may lead to bypassing connections which results in the latent space being ignored during training. We show experimentally with the example of decoder hidden state initialization that such bypassing connections degrade the VAE into a deterministic model, thereby reducing the diversity of generated sentences.

We discover that the traditional attention mechanism used in sequence-to-sequence VED models serves as a bypassing connection, thereby deteriorating the model's latent space. In order to circumvent this issue, we propose the variational attention mechanism where the attention context vector is modeled as a random variable that can be sampled from a distribution. We show empirically using automatic evaluation metrics, namely entropy and distinct measures, that our variational attention model generates more diverse output sentences than the deterministic attention model. A qualitative analysis with human evaluation study proves that our model simultaneously produces sentences that are of high quality and equally fluent as the ones generated by the deterministic attention counterpart.

Acknowledgements

This thesis would not have been possible without the constant support that I received from a number of people.

First and foremost, I take this opportunity to express my heartfelt gratitude to my supervisor Professor Olga Vechtomova for her guidance throughout this research. I am thankful to her for believing in me and being patient with me since the start of my program. The flexibility that she provided in research has allowed me to learn new topics and explore new ideas. I am deeply indebted to her for having taken the time and effort for reading, reviewing and providing valuable inputs for the reports that I had made.

I am grateful to Dr. Lili Mou for being a mentor, sharing knowledge and providing valuable technical advice. His subject expertise and guidance has played a crucial role in structuring this project. Interactions with him have helped me develop my skills in this field and motivated me to deal with research challenges.

I would like to acknowledge Professor Pascal Poupart for sharing his ideas and providing valuable suggestions.

I thank Professor Jesse Hoey and Professor Stan Dimitrov for taking the time to review my thesis and provide feedback.

I express my gratitude to Vineet John and Ankit Vadehra for the informative discussions and enlightening me on their respective research topics.

Life in Canada would not have have been such an enjoyable journey without the friends that I made here. I thank all my friends for making my stay here at Waterloo, a memorable one.

I am extremely grateful to my parents and my sister for supporting me and being with me even during the toughest times.

Dedication

I dedicate this thesis to my beloved parents and sister for their unconditional love, support, and care.

Table of Contents

List of Tables

List of Figures

Abbreviations

AI Artificial Intelligence 1, 2

ANN Artificial Neural Networks 8, 10

CNN Convolutional Neural Network 8

DAE Deterministic Autoencoders 31, 40, 46

DED Deterministic Encoder-Decoder 57, 64, 66

ELBO Evidence Lower Bound 28, 29, 60

GRU Gated Recurrent Unit 13, 15, 19

KL Kullback-Leibler 4, 27, 31, 43, 52, 62, 67

LSTM Long Short Term Memory 13, 15, 21, 57, 60, 62, 64

ML Machine Learning 6

NLP Natural Language Processing 2, 15, 17, 21

ReLU Rectified Linear Unit 9, 12

RNN Recurrent Neural Network 8, 11–13, 15–17, 57, 64

Seq2Seq sequence-to-sequence 2, 3, 5, 15–17, 21, 23, 50, 51, 55, 57, 58, 60, 62, 64, 68

SGD stochastic gradient descent 32, 39

VAE Variational Autoencoders 2, 18, 29, 32, 40, 43, 45, 50–52, 60, 62, 64

VED Variational Encoder-Decoder 2, 55, 60, 62–64, 66, 68

Chapter 1

Introduction

1.1 Background

The term Artificial Intelligence (AI) was introduced by Professor John McCarthy in 1956, who defined it as "science and engineering of making intelligent machines". Although the field of AI covers a broad range of topics, it is generally perceived as the task of making machines achieve *human-level* intelligence (McCarthy, 1989).

Fast-forward a few decades, we have achieved great advancements in the field of AI. For example, the neural network model developed by He et al. (2016) was able to classify images with an accuracy of 96.5%, surpassing human-level performance. A new milestone was achieved in the area of speech recognition when the system developed by Xiong et al. (2017) was able to carry out the task with a record minimum word error rate of 5.1%. There are numerous other commendable accomplishments made in the last decade such as AI beating the human grandmaster in the game of Go (Gibney, 2016) and the transformations in the transportation industry with the introduction of self-driving vehicles (Bojarski et al., 2016).

This recent success can be attributed to the research developments made in a multitude of fields such as computer science, mathematics, neuroscience, psychology, linguistics and so on. This work focuses on the field of computer science, specifically the areas of machine learning and natural language processing.

Machine learning is a sub-field of artificial intelligence in which computers are *taught* to acquire knowledge or *learn* from data, without being explicitly programmed. In the past, machine learning algorithms have been used to recognize patterns in the data and make

informed decisions. End-to-end automation with machine learning has helped improve the efficiencies of processes and workflows in sectors such as manufacturing. The AI explosion in the recent years is primarily due to the success of a sub-class of machine learning models known as deep neural networks. This field, known as deep learning, along with the availability of massive amounts of data and powerful hardware for computation has made possible the latest advancements in AI.

The focus of this work lies in the development and application of deep learning models to Natural Language Processing (NLP). The study of NLP is concerned with how computers can effectively interact with humans using natural language. Broadly speaking, it deals with manipulation, understanding, interpretation and generation of textual and speech data. A few examples of NLP tasks include question answering, sentiment analysis, named-entity recognition and machine translation.

Natural language generation is an NLP task that deals with generation of text in human language. This is challenging because the text generated by a good system has to be *syntactically* (follow the rules of the language) and *semantically* (meaningful) correct. In this work, sequence-to-sequence (Seq2Seq) models for natural language generation are explored. In Seq2Seq models (Sutskever et al., 2014), a sequence (of words) is given as input in order to generate another sequence as output. This has applications in tasks such as machine translation where a sentence in English can be fed to the model which generates its corresponding translated sentence in French. Another application is text summarization, where we may attempt to generate a shorter version of a larger body of text, such as a paragraph.

The aim of this work is to integrate attention mechanism into the Variational Encoder-Decoder (VED) framework. Attention mechanism has been shown to be particularly useful in improving the performance of sequence-to-sequence tasks such as machine translation (Bahdanau et al., 2014; Luong et al., 2015) and dialog generation (Yao et al., 2016; Mei et al., 2017). A class of models that combine deep learning and variational inference, namely Variational Autoencoders (VAE) (Kingma and Welling, 2013) have been successfully applied to the task of text generation (Bowman et al., 2015b). Attention mechanism enables the model to generate fluent sentences, relevant to the input. Simultaneously, we would be able to generate diverse outputs (for the given input) by sampling from a latent space. This way, the proposed model combines the strengths of the attention mechanism and variational models.

Input: What are you doing?	
I don't know.	*Get out of here.*
I don't know!	*I'm going home.*
Nothing.	*Oh, my god!*
Get out of the way.	*I'm talking to you.*
Input: What is your name?	
I don't know.	*My name is Robert.*
I don't know!	*My name is John.*
I don't know, sir.	*My name's John.*
Oh, my god!	*My name is Alice.*
Input: How old are you?	
I don't know.	*Twenty-five.*
I'm fine.	*Five.*
I'm all right.	*Eight.*
I'm not sure.	*Ten years old.*

Table 1.1: Output dialog generated by a conversational system as shown in Li et al. (2015a). For a given input, the first column shows the list of top-ranked (most probable) responses, most of which tend to be generic. The second column shows the lower-ranked but less generic/more engaging responses.

1.2 Motivation and Problem Definition

In this research, we work with deep neural network models known as sequence-to-sequence (Seq2Seq) models that take a sentence as input and generate another sentence as output. Let $\boldsymbol{x} = (x_1, x_2, \cdots, x_{|\boldsymbol{x}|})$ be the input sequence of words and $\boldsymbol{y} = (y_1, y_2, \cdots, y_{|\boldsymbol{y}|})$ be the output sequence generated by the model, where $|\boldsymbol{x}|$ and $|\boldsymbol{y}|$ correspond to the number of tokens (words) in the input and output sequences respectively.

Consider a conversational system such as a chatbot, where $\boldsymbol{x}$ is the line input by the user and $\boldsymbol{y}$ is the line generated by the machine. Seq2Seq neural network models can be designed to function as such conversational agents. Traditional conversational systems tend to output safe and commonplace responses such as *"I dont know"* (Li et al., 2015a). This is because the line *"I dont know"* tends to appear in the training dataset with a high frequency. One cannot label such a generic response as incorrect, since it tends to be a valid response (refer Table 1.1). However, such responses make the conversational agent uninteresting and less engaging.

The motivation for this work is derived from the above example. We would like to be

able to generate a diverse set of responses ($\boldsymbol{y}$) for a given input line ($\boldsymbol{x}$). Neural variational models can be used to encode input data into latent variables. It is further possible to sample multiple points from the latent space in order to generate diverse outputs. Attention mechanisms such as those proposed in (Bahdanau et al., 2014; Luong et al., 2015) have significantly improved the performance of sequence-to-sequence text generation tasks. Attention mechanisms help in dynamically aligning the source $\boldsymbol{x}$ and target $\boldsymbol{y}$ during generation.

In variational Seq2Seq, however, the attention mechanism unfortunately serves as a "bypassing" mechanism. In other words, the variational latent space does not need to learn much, as long as the attention mechanism itself is powerful enough to capture source information.

In this work, we study how attention mechanisms can be integrated into variational neural models, while avoiding the issue of "bypassing". In this work, we propose a variational attention mechanism to address this problem. This is done by modeling the attention context vector as random variables by imposing a probabilistic distribution. By doing this we would be able to combine the stochasticity introduced by variational models with the alignment capabilities achieved through attention mechanism.

1.3 Contributions

The contributions of this thesis are multi-fold and listed below:

1. A variational auto-encoder (VAE) is first designed following the work of Bowman et al. (2015b). We overcome the difficulties associated with training VAE models for natural language generation by employing strategies such as (1) annealing coefficient of the Kullback-Leibler (KL) loss term and (2) word-dropout. We also propose new and improved annealing schedules. The effectiveness of the model is demonstrated by random sampling and linear interpolation of sentences in the latent space.

2. We discover a "bypassing" phenomenon in VAEs that causes the latent or variational space to be ignored during training. This results in the model becoming more *deterministic* in nature; the evidence for which is revealed through the lower diversity of generated sentences.

3. We realize that traditional attention mechanism in variational encoder-decoder (VED) models serves as a "bypassing" connection. To this end and in contrast to previous

models that utilize attention in a deterministic manner, we propose a variational attention mechanism that can be applied in the context of VED models. In the proposed framework, the attention context vector is modeled as a random variable that can be sampled from a distribution.

4. We propose two plausible priors for modeling the prior distribution of the attention context vector in the variational attention VED framework. Both the priors work equally well in alleviating the problem of bypassing, which is observed in the VED baselines with deterministic attention.

5. Experiments are carried out on two tasks – question generation and conversational systems. Quantitative evaluation metrics show that the proposed variational attention yields a higher diversity than variational Seq2Seq with deterministic attention, while retaining high quality of generated sentences. A qualitative analysis with human evaluation study also supports our claim regarding the fluency of sentences generated by the proposed model.

1.4 Chapter Outline

The rest of this thesis report is organized as follows:

- Chapter 2 provides a background and brief overview of the deep learning techniques used in this work, along with related work in the area of Seq2Seq models.
- Chapter 3 describes Seq2Seq neural network architecture in depth.
- Chapter 4 introduces variational inference and a detailed description of variational autoencoders. The heuristics involved in training VAEs for natural language generation are presented followed by the results.
- Chapter 5 outlines the bypassing phenomenon that we discover, if the VAE architecture is not designed properly and its implications.
- Chapter 6 provides details about the proposed variational attention model, which is compared to the traditional deterministic attention. We demonstrate the benefits of our model through qualitative and quantitative evalutation.
- Chapter 7 gives a summary of the work, followed by conclusions and scope for future work.

Chapter 2

Background and Related Work

In this chapter, different classes of machine learning models are first outlined. Then, we introduce deep learning and the structure of artificial neural networks. Following this, we describe recurrent neural networks, specifically long short term memory networks, which are widely used in the NLP literature. Since this work deals with text generation, we review the recent advances in this area, specifically sequence-to sequence models, attention mechanism and audoencoding. The chapter concludes with the related work in the tasks of question generation and dialog systems, which are the two experiments conducted in this study, to evaluate the proposed model.

2.1 Machine Learning

The science of Machine Learning (ML) involves enabling computers to *learn* from data, without being explicitly programmed. Data is used to train the system to perform a specific task. The model, which uses some form of mathematical optimization and statistical methods, recognizes the patterns and intricacies within the data. This can be then used to automate tasks or guide decision making, simply based on data and the mathematical model.

Machine learning is being increasingly used in our day-to-day lives. For example, all email service providers today use ML to filter out spam emails. Similarly, the online shopping recommendations provided to us by ecommerce websites is based on ML. The field of machine learning is developing at a fast pace. Researchers have been developing algorithms and new methodologies and also simultaneously applying these techniques to new

application areas (such as medical diagnosis (Kourou et al., 2015; Foster et al., 2014) and climate change (Lakshmanan et al., 2015)). The evolution of intelligent systems are definitely beneficial because it makes processes more efficient, and at the same time, requiring minimal human intervention.

Broadly speaking, machine learning methodologies can be classified into two categories: supervised learning and unsupervised learning. At a high level, this categorization is based on how the learning process is carried out. The following subsections describe each one with examples.

2.1.1 Supervised Learning

In supervised machine learning, we provide the model with sample inputs and their corresponding ground truth labels. Here, the task of the machine learning algorithm is usually to modify the model parameters, such that it obtains the desired output for the given input. The important point to note here is that we have labelled outputs corresponding to each input used to *train* the model. After sufficient training, if the model is provided with a new unseen input data point, it should be able to predict the target, based on what it has previously *learnt*.

To understand supervised learning with the help of an example consider the task of image classification. We feed the machine learning model with images of huskies, retrievers, dachshunds, etc. and label them as *dogs*. Similarly, we can provide images of pigeons, eagles, sparrows, etc., all labelled as *birds*. The model tries to *learn* from the image pixel values and their corresponding class labels, as to what would be the characteristics that differentiate *dogs* from *birds*, which is then used to classify new images.

2.1.2 Unsupervised Learning

In contrast to the approach discussed in Section 2.1.1, the data provided to an unsupervised machine learning model will not contain labels or corresponding target values. The task of such models would be to identify patterns of similarity or differences within the input data points. It can also be used for detecting anomalies, wherein some parts of the data may not fit well with the rest of the data.

An example of unsupervised learning task would be clustering of documents by topic. Assume that we provide an unsupervised machine learning model with *unlabelled* documents pertaining to different topics such as sports, politics and entertainment. A good

model should be able to automatically cluster *similar* documents (belonging to the same topic) together, using information such as the word usage and writing style.

Other areas in machine learning include topics such as semi-supervised learning and reinforcement learning, which are not covered in this text.

2.2 Deep Learning

Artificial Neural Networks (ANN) are an important class of machine learning models, used for both supervised and unsupervised tasks. The structure and functioning of ANNs are loosely inspired by biological neural networks. The brain consists of a large number of interconnected neurons, which ANNs try to mimic. ANNs consist of multiple layers of simple processing units known as *nodes*, which are connected by *edges* with *weights* (Gurney, 2014) (refer to Section 2.2.1 for details).

Over the last decade, there has been an increasing interest in neural network architectures consisting of many layers. Along with the availability of massive amounts of data and powerful hardware for computation, such model architectures were able to outperform humans in a number of cognitive tasks (Schmidhuber, 2015; Najafabadi et al., 2015). This led to the creation of a sub-field of machine learning known as deep learning (referring to deep neural networks) (LeCun et al., 2015).

The most basic version of an ANN model is a feed-forward neural network. However, there exist other architectures such as Recurrent Neural Network (RNN) (Williams and Zipser, 1989; Elman, 1990) and Convolutional Neural Network (CNN) (LeCun et al., 1995). RNNs perform particularly well on sequential data such as in natural language processing (where sentences are considered as sequences of words). Hence, the focus of this work will be on RNNs, which are described in detail in Section 2.2.2

2.2.1 Introduction to Neural Networks

In order to understand the computational model of artificial neural networks, one needs to begin from its building block, known as the *perceptron* (Rosenblatt, 1958). Inspired from the brain's neurons, a perceptron is a simple computational model that takes in one or more inputs and provides a single value as output. This is illustrated in Figure 2.1. Based on this output and a pre-defined threshold, the perceptron acts as a binary classifier, i.e., if the output value is greater than the threshold, the input is assigned to class 1, else it is assigned to class 0.

Let x_1, x_2, x_3 be the inputs to the perceptron model. w_1, w_2, w_3 are the series of model weights corresponding to each input variable. This simple model consists of two operations:

- The first step is to multiply each input with its weight, followed by a summation. To this result, we also add the bias term b so that the model has a flexibility for location shift.
- Next, we assign a class label (either 0 or 1), based on a binary activation function which requires a pre-defined threshold (refer Figure 2.1).

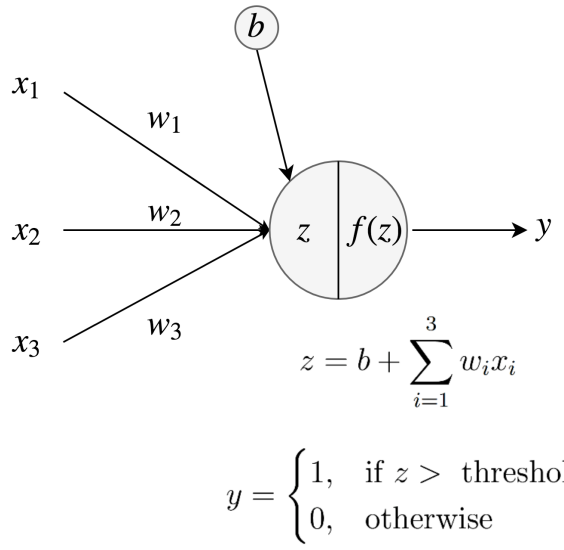

Figure 2.1: Perceptron Model

The result is the predicted value of output y corresponding to the given set of inputs. In order for the predicted output to be close to the desired output (ground truth), we would need to make adjustments to the weights w_1, w_2, w_3 and bias term b.

However, modern neural networks do not use the simple perceptron anymore. Instead, they consist of computational units known as *neurons* (or nodes), which replace the simple binary activation function with non-linear functions such as sigmoid, tanh or Rectified Linear Unit (ReLU). It is possible to combine multiple layers of neurons to form a more powerful model known as the feed-forward neural network. Each neuron is connected to every other neuron in the previous and next layer. However, there are no connections between neurons within the same layer. As illustrated in Figure 2.2, there can be multiple

inputs and multiple outputs which are connected via a number of *hidden* layers. The input at each neuron gets transformed by weighted summation followed by non-linear activation. The computation happens starting from the input layer, all the way till the output layer and is known as **forward propagation**. Feed-forward neural network are capable of learning non-linear representations of the data and have been successfully applied to many classification and regression tasks.

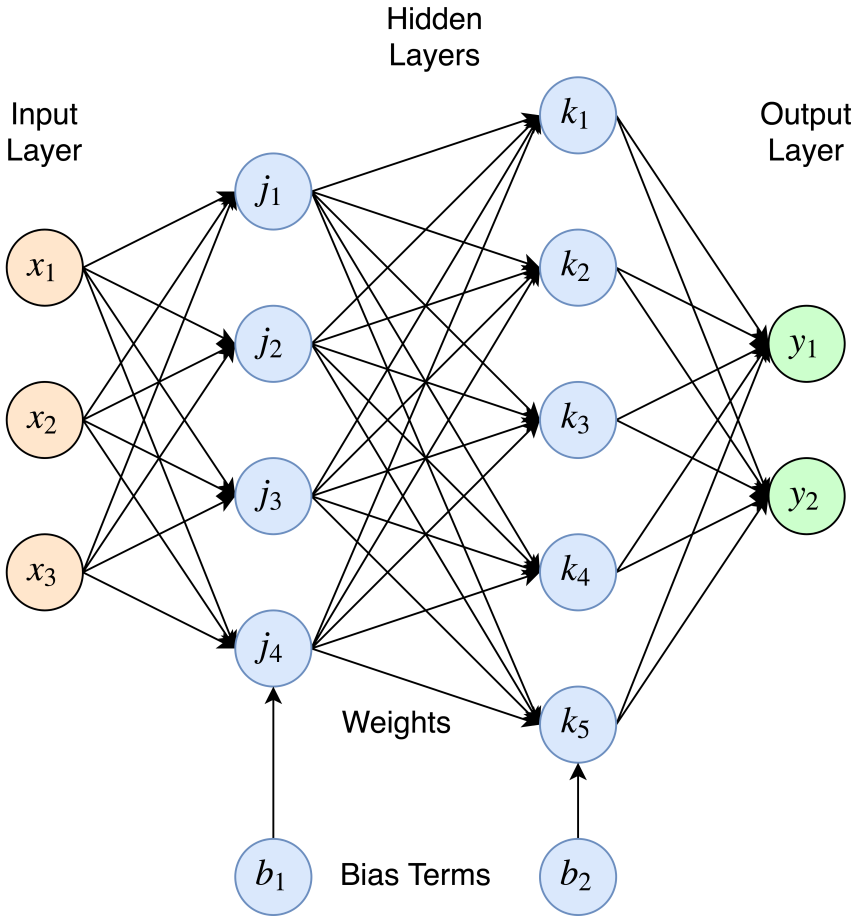

Figure 2.2: Feed-Forward Neural Network

Although artificial neural networks have been around since the 1960s, it was not until the late 1980s that an efficient training procedure for ANNs was discovered. There existed no structured methodology to adjust the model weights, other than by trial and error. Researchers like Rumelhart et al. (1986) and Werbos (1990) contributed to the development of the method known as **backpropagation of errors**, which made it possible to

estimate the weights in an ANN model. Backpropagation makes use of the *chain rule of differentiation*, and computes the gradients in an iterative manner.

In order to develop an intuition of backpropagation, it is necessary to understand how an optimization method known as **Gradient Descent** works. In neural networks, we compare the predicted output to the actual output based on a pre-defined loss function. Common examples of loss functions are mean squared error (MSE) and negative log-likelihood (NLL). Our objective is to adjust the model weights in a way that minimizes the loss. It is mathematically guaranteed that moving in the direction of the *gradient* of the loss function (derivative with respect to the model weights), results in loss minimization.

Assume $\mathcal{L}(\boldsymbol{w})$ to be the loss function, with $\boldsymbol{w}$ being the model weights. We start with a random initialization of the weights, followed by an iterative update rule as shown in Equation 2.1,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \cdot \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) \tag{2.1}$$

where, η is a hyperparameter (set by the user) known as the *learning rate*, which corresponds to the step size towards the local minima in each iteration and ∇ refers to the gradient operator. While a low learning rate results in the training process to progress slowly, a high learning rate may cause the training to diverge from the minima. Because of this trade-off, the learning rate needs to be set carefully. We stop the iteration process either when we reach the pre-defined maximum number of iterations (known as epochs) or when the change in model weights between iterations is smaller than a specified threshold ϵ. Readers are referred to Bishop (2006) for further details on backpropagation and Gradient Descent.

2.2.2 Recurrent Neural Networks

One of shortcomings of feed forward neural networks such as the one illustrated in Figure 2.2 is that it assumes that all input data are independent of each other. As a result, it fails to capture the notion of sequential order which is present in some types of data. Consider the task of predicting the next character in a word. If we are given an incomplete word such as '*neura*', one can guess that the next character in the sequence would be '*l*' and the word is '*neural*'. However, if the order of the previous characters was jumbled (such as '*renau*') and provided independently, it would be very difficult to identify the final character. This is where RNNs are found to be extremely useful. One of the earliest

versions of the recurrent neural network was proposed by Elman (1990). The input to an RNN is provided in a sequential manner, and the network makes use of the inputs in the previous timesteps in order to make a decision at the current timestep.

A recurrent neural network can be depicted as a network with loops (see Figure 2.3), through which information is transferred between timesteps of the network. By unrolling the network, we realize that the information at each timestep passes through multiple copies of the same network (Olah, 2015).

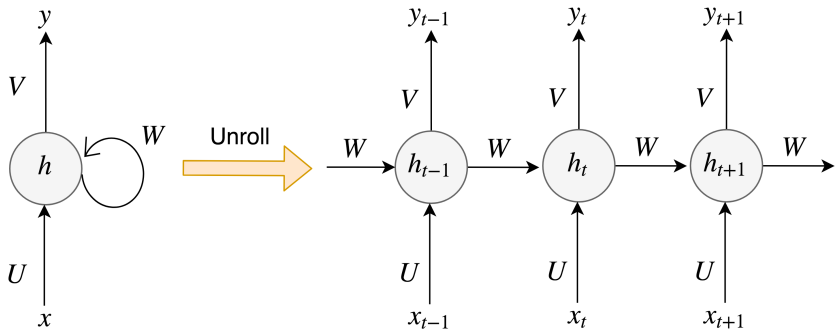

Figure 2.3: Unrolling of a recurrent neural network (RNN) (Britz, 2015)

The notation in Figure 2.3, adapted from (Britz, 2015) is described below:

- x_t corresponds to the input at each timestep t
- y_t refers to the output at each timestep t
- h_t is called the hidden state at each timestep t, and is calculated using the input at the current timestep x_t and the hidden state from the previous timestep h_{t-1}, i.e.,

$$h_t = f(Ux_t + Wh_{t-1}) \tag{2.2}$$

  where f corresponds to some non-linear activation function such as tanh or ReLU. In RNN literature, h_t is also referred to as the *memory* because in theory, it is assumed to capture information from all previous timesteps. However, this does not hold true in practice since the RNN memory fails to *remember* information beyond few previous timesteps.

- U, V and W are weight matrices. From the unrolled RNN figure, one can note that these weights are shared across all timesteps of the RNN. Doing this reduces the

model complexity by reducing the number of parameters that need to be optimized. Moreover, we aim to perform the same operation across timesteps, just with different inputs.

Training of RNNs is done via an extension of the backpropagation algorithm, known as *backpropagation through time* (BPTT). As discussed earlier, RNNs perform well on sequential data and have been extensively used for tasks such as language modelling (Mikolov et al., 2010), text generation (Graves, 2013) and speech recognition (Graves et al., 2013).

2.2.3 Long Short Term Memory

In practice, vanilla RNNs suffer from the inability to capture long term dependencies. In other words, when the length of input sequence becomes large, RNNs are unable to remember the dependencies between inputs which are far apart in the sequence. The reason for this is attributed to the vanishing/exploding gradient problem (Pascanu et al., 2013). This happens due to numeric underflow or overflow, i.e., when the multiplication of derivative terms during backpropagration become extremely small or very large. Exploding gradients can be an easier problem to solve - by truncating gradients when their absolute value crosses a pre-specified threshold (Pascanu et al., 2012).

In order to circumvent the issue of vanishing gradients, extensions to the vanilla RNN architecture, Long Short Term Memory (LSTM) Units (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Chung et al., 2014) were proposed. This thesis work makes use of RNNs with LSTM units, which will be described in this section.

In LSTMs, we replace the simple activation function f of Equation 2.2 with an entire module, also known as *cell*. The input to each repeating module consists of x_t and h_{t-1} along with a new term c_{t-1}, known as the cell state. The output at timestep t now includes both h_t and c_t. This is depicted in Figure 2.4.

The LSTM unit consists of gated operations and element-wise multiplications. Gates, represented by *sigmoid* activations (which output values between 0 and 1) essentially decide how much of the incoming information should flow through. The equations of the three adaptive gates namely, the input gate ($\boldsymbol{i}_t$), the forget gate ($\boldsymbol{f}_t$) and the output gate ($\boldsymbol{o}_t$) are given below:

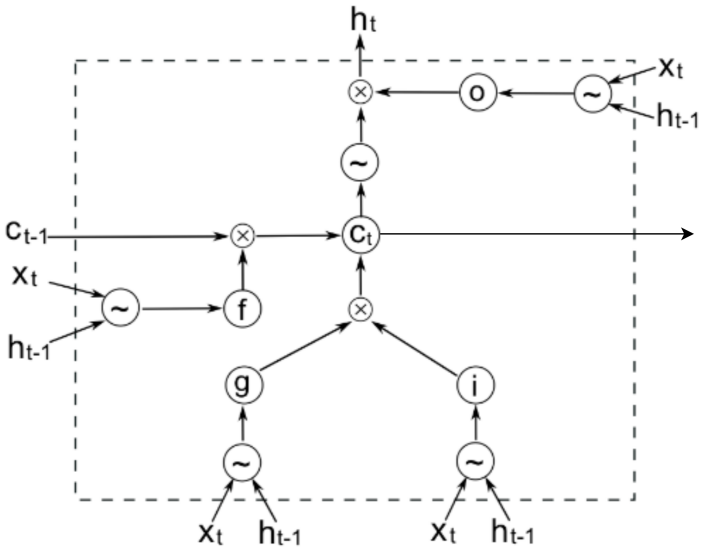

Figure 2.4: LSTM Unit Representation (Xu et al., 2015b)

$$\boldsymbol{i}_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i) \tag{2.3}$$
$$\boldsymbol{f}_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f) \tag{2.4}$$
$$\boldsymbol{o}_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \tag{2.5}$$

where σ denotes the sigmoid function. Each of the gates has its own weights (U, W and b) and require as input the previous hidden state h_{t-1} and the current input x_t. In addition, we also compute a candidate cell state vector ($\boldsymbol{g}_t$) as follows:

$$\boldsymbol{g}_t = \tanh(W_g \cdot x_t + U_g \cdot h_{t-1} + b_g) \tag{2.6}$$

We now have all the vectors required to determine the cell state (c_t) and hidden state (h_t) of the current timestep:

$$c_t = \boldsymbol{i}_t \otimes \boldsymbol{g}_t + \boldsymbol{f}_t \otimes c_{t-1} \tag{2.7}$$
$$h_t = \boldsymbol{o}_t \otimes \tanh(c_t) \tag{2.8}$$

where $\otimes$ refers to the element-wise multiplication operator.

To summarize the procedure, we start by setting the cell state and hidden state of the initial timestep, namely c_1 and h_1 as zero vectors. Then, the values of c_t and h_t are computed sequentially for each timestep using Equations 2.3 to 2.8, till the end of the sequence is reached.

LSTMs excel at capturing long term dependencies and have been applied to a number of NLP tasks: sequence tagging (Huang et al., 2015), relationship classification (Xu et al., 2015b), textual entailment (Rocktäschel et al., 2015), machine comprehension (Cheng et al., 2016), sentiment analysis (Tai et al., 2015) and many more.

Bi-directional LSTMs

LSTMs work based on the principle that the output at the current timestep is dependent on the inputs (and outputs) at previous timesteps. However, in some tasks such as part-of-speech (POS) tagging (Huang et al., 2015) and text-to-speech (TTS) synthesis (Fan et al., 2014), it is useful to be aware of the elements in the future timesteps. Bidirectional LSTMs are known to perform better than regular LSTMs in such scenarios (Schuster and Paliwal, 1997). Essentially, a bidirectional LSTM consists of two LSTMs stacked one on top of the other - the first LSTM reads the sequence in the forward manner (e.g., a regular sentence), while the sequence is fed in the backward direction to the second LSTM (e.g., the same sentence with its word order reversed). The output vectors of the two LSTMs are then concatenated and fed to the next layer in the neural network.

2.2.4 Sequence-to-Sequence Models

In natural language processing, sequence-to-sequence tasks usually refer to the ones in which the model takes as input one sequence and generates another sequence as output (instead of a single value). The sequence can range from whole documents to individual words. In the case of documents or sentences, the individual *tokens* that form the sequence are words. In contrast, when words themselves are treated as sequences, their characters become the individual *tokens*.

Seq2Seq models are typically implemented with the help of two recurrent neural networks (usually an LSTM or GRU). In the most basic version, the input/source sequence is fed token-by-token to the first RNN (encoder), which computes a vector representation for the whole sequence. This vector representation becomes the starting point for the second

RNN (decoder), which generates the output/target sequence, again in a token-by-token manner.

Sutskever et al. (2014) first introduced Seq2Seq models for the task of machine translation. Their LSTM based approach was able to achieve a translation performance close to the state-of-the-art. The advantage of their method is that it could be trained completely end to end (assuming the availability of a parallel corpora), without the need for any manual feature engineering. In order to generate vector representations for sentences, similar to the idea of `word2vec` (Mikolov et al., 2013) for words, Kiros et al. (2015) proposed a Seq2Seq learning approach. Essentially the network would be trained to generate a given sentence, based on two of its neighbouring sentences (previous and next).

In NLP, Seq2Seq models are extensively used for the text generation. Yin et al. (2015) developed a Seq2Seq model trained on question-answer pairs and knowledge-base triples for the task of answering short factoid questions. Seq2Seq models have opened up the possibility to train dialog sytems in an end-to-end manner, without the need for any handcrafted features (Vinyals and Le, 2015). Modifications to the original negative log likelihood objective function (Li et al., 2015a) and beam search optimization (Wiseman and Rush, 2016) have made dialog systems to be more *human-like*.

Seq2Seq models have also been successfully applied to multimodal data. For instance, Venugopalan et al. (2015) demonstrate how Seq2Seq LSTMs could be used to generate textual descriptions when they are trained with video clips as input. In (Yao and Zweig, 2015), the authors develop a model to synthesize speech from textual data.

Since generation of text is the focus of this thesis work, LSTM based sequence-to-sequence models are used following previous research in this area.

2.2.5 Auto-encoding

Autoencoding is an unsupervised learning technique in which we provide an input (such as image or text) to a model, learn an intermediate representation and then try to reconstruct the original input from this representation. The model is usually an artificial neural network, and the intermediate representation typically has lower dimensionality than the original input (Hinton and Salakhutdinov, 2006). Hence, the goal becomes to learn an efficient encoding that stores just the necessary information required for reconstruction.

The intermediate representation can later be used for other supervised tasks in the machine learning pipeline. For example, consider the task of recognizing hand written digits. We train an autoencoder with images of digits from 0 to 9. Next, instead of using

the original image, we could use their intermediate representations to train a feed forward neural network to classify these digits. Autoencoders have been successfully applied to solve problems such as image super-resolution (Zeng et al., 2017) and speech enhancement (Lu et al., 2013). Since the lower dimension representations generated by autoencoders are useful in identifying patterns pertaining to the original data, they have also been applied to anomaly detection (Malhotra et al., 2016).

In the domain of NLP, autoencoders are usually Seq2Seq models. RNNs encode the input sentence into its latent representation. The decoder then uses this representation to reconstruct and generate the original sentence. In (Dai and Le, 2015), the authors demonstrate how a sequence autoencoder can serve as a 'pretraining' method for enhancing the performance of downstream supervised tasks. Autoencoders can be used not just to represent sentences, but also paragraphs and documents (Li et al., 2015b).

2.2.6 Attention Mechanism

Broadly speaking, attention mechanism in neural networks is a way to guide the training process, by informing the model as to what parts of inputs or features it needs to focus on, in order to accomplish the task at hand. In this section, we will review the attention mechanisms used in NLP. This is different from visual attention in computer vision tasks such as image captioning (Xu et al., 2015a) and object detection (Borji et al., 2014).

The two popular attention mechanisms used in Seq2Seq models are Luong Attention (Luong et al., 2015) and Bahdanau Attention (Bahdanau et al., 2014). Both of these models were introduced for machine translation, where they were shown to perform better than vanilla Seq2Seq models. Attention mechanisms achieve this by aligning tokens on the target side to the tokens on the source side. While the core idea behind both attention mechanisms remain the same, the method in which the attention context vector is computed is different - it takes a multiplicative form in Luong Attention whereas in Bahdanau Attention it has an additive form.

To explain this Seq2Seq attention mechansim intuitively, consider the task of translating the following sentence from French to English: *c'est un chien* $\longrightarrow$ *that is a dog*. During the decoding phase, when we arrive at the timestep that decodes the word *dog*, the model looks at each word on the source side and has to identify that the word *chien*, is where it has to focus on, thereby giving that particular source token the highest weight when computing the attention context vector. The mathematical details pertaining to how attention mechanism works will be detailed in Chapter 6.

Apart from machine translation, attention mechanism has been found to improve the performance of text summarization (Rush et al., 2015), dialog generation (Li et al., 2017), textual entailment (Rocktäschel et al., 2015), question generation Du et al. (2017) and so on.

2.2.7 Variational Inference

In variational inference, we use machine learning and optimization to approximate probability distributions which are otherwise difficult to estimate (Blei et al., 2017). In Bayesian Inference, it is often of interest to compute posterior distributions. This usually involves solving for intractable integrals which becomes cumbersome. In general terms, we try to find an approximate distribution from a family of distributions that is similar to the posterior which we wish to estimate. In other words, we minimize the Kullback-Leibler divergence between the two distributions.

Although there are methods such as mean field approximation (Wainwright et al., 2008) for variational inference, we will focus on variational auto-encoders (VAE) in this work. In comparison to such traditional methods, VAEs leverage modern neural networks which are universal function approximators and are a more powerful density estimator. VAEs were first introduced by Kingma and Welling (2013) in the image domain, to learn latent representations for images of handwritten digits. What makes VAEs powerful is that these learnt latent representation (approximately) belong to a pre-defined distribution, such as Gaussian with a known mean and variance. This makes it possible to simply sample a vector from this known distribution and to generate the desired image. It is also possible to manipulate the latent representation to change certain characteristics of the input image. For instance, Kulkarni et al. (2015) showed that VAEs could be used as a 3D graphics rendering engine. Specifically, they could manipulate the latent representation of an input image in order to change the pose and orientation of objects within that image. Pu et al. (2016) train VAEs jointly with images and captions. They demonstrate that the same learnt intermediate representations could be used for a number of downstream supervised tasks including image classification and image captioning.

The VAEs discussed so far either use MLPs or CNNs as encoders and decoders. In NLP, the straightforward alternative choice is to use RNNs. However, VAEs that use RNNs have been found to be more difficult to train, due to issues relating to the KL divergence between the posterior and prior vanishing to zero. Bowman et al. (2015b) were able to successfully train LSTM-VAEs after implementing optimization strategies such as KL cost annealing and word dropout. In Yang et al. (2017), the authors retain an LSTM encoder, but use

a CNN decoder for generation of text. In addition, they use dilated convolutions along with residual connections in order to prevent the collapse of the KL term during training. Similar to the image domain VAEs, it is possible to sample from the latent space and generate text. The latent space also exhibits properties such as homotopy (Bowman et al., 2015b), i.e., it is possible to smoothly interpolate between points in the latent space and generate meaningful sentences.

2.2.8 Question Generation

The task of question generation is as follows: given an input sentence or paragraph, the model is required to generate a question relevant to the input. Such a task would have applications in the field of education to prepare questions relevant to a given passage within a piece of text (Heilman and Smith, 2009). Question generation could also be used to automatically generate frequently asked questions (FAQs), given product descriptions.

Question generation is a relatively new research topic. One of the first studies in this area was conducted by Heilman (2011), who defined a set of rules to transform sentences into factoid questions. Another rule-based approach proposed by Chali and Hasan (2015) makes use of named entities and semantic role labeling for automatic question generation. Neural networks were implemented for this task only recently by Du et al. (2017) and Zhou et al. (2017). The Seq2Seq GRU model by Zhou et al. (2017) generates questions for sentences from the Stanford Question Answering Dataet (SQuAD) (Rajpurkar et al., 2016). The model requires as input the word embeddings along with lexical and answer position features. The LSTM encoder-decoder model developed by Du et al. (2017) was evaluated for fluency for both sentence level and paragraph level inputs. This model was trainable completely end-to-end, without the need for any feature engineering. Question generation can also be carried out with knowledge base triples as input (Song and Zhao, 2016).

2.2.9 Dialog Systems

Dialog systems (or chatbots) that can converse like humans can be viewed as one of the characteristics of intelligent machines. One of the earlist chatbots, ELIZA was developed by Weizenbaum (1966). ELIZA would provide responses based on a set of pre-defined rules, most of which try to paraphrase the user questions or sentences. ALICE bot (Wallace, 2009) was an extension to ELIZA, which incorporated more rules and was provided with template responses from more domains (Kerly et al., 2007).

However, with the introduction of Seq2Seq models, conversational agents could now be trained end-to-end without the need for any rules (Vinyals and Le, 2015). Neural dialog systems were enhanced by making them context sensitive (Serban et al., 2016; Sordoni et al., 2015), i.e., the conversation history would be provided as input to the model to generate a response. Bordes et al. (2016) show how deep neural networks can be used to design goal-oriented domain specific dialog systems. The persona-based conversational system developed by Li et al. (2016a) encodes speaker information through a learnt embedding, so that the system is more personalized and provides responses that are consistent. Recent studies in neural dialog generation also focus on making agent responses more diverse and less generic (Li et al., 2015a). Generative Adversarial Networks (GANs) (Li et al., 2017) and deep reinforcement learning techniques (Li et al., 2016b) have also been implemented for dialog systems.

Chapter 3

Sequence to Sequence Models

We first introduce the concept of word embeddings, which are a convenient vector representation for text data. In continuation to the LSTM concepts introduced in the previous chapter, we provide a mathematical description of Seq2Seq models. To conclude this chapter, the tools used to build the word embeddings and neural network models are briefly discussed.

3.1 Word Embeddings

In order to feed textual data into machine learning models, we need to have corresponding numeric representations. There has been multiple methods proposed in the literature to address this problem (Mitra and Craswell, 2017), such as bag-of-words (BoW) and one-hot representation. However, word embeddings such as GloVe (Pennington et al., 2014) and word2vec (Mikolov et al., 2013) are the most common way of representing text for deep neural network models used in NLP.

The notion of distributional similarity was introduced by Harris (1954), who stated that "words that occur in similar contexts would have similar meaning". For example, the words *sport* and *game* occur in similar contexts in documents and hence, share similarities in their meanings. This means that the numeric or vector representations of these two words should be similar. In NLP, the cosine distance metric is typically used to measure similarity between two vectors.

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \cdot |\vec{b}|} \tag{3.1}$$

where $|\cdot|$ refers to the L2-norm of the corresponding vector. A higher value of $\cos\theta$ implies that the angle between the two vectors is small, and hence they are more similar and *vice-versa*.

A word embedding maps words to real valued vectors, W: $words \rightarrow \mathbb{R}^n$, where n is the dimension of each word vector. Mikolov et al. (2013) show that word2vec embeddings can be obtained by training a neural network with a single hidden layer, which is provided with one-hot vectors as input. Consider a context window of $m+1$ words where the centre word is called the focus word and the remaining m words are known as the context words. They propose two methods: 1) given the context words, predict the focus word (continuous bag-of-words or CBOW approach); 2) given the focus word, predict the context words (skipgram approach). In both cases, the weight matrix in the hidden layer of this shallow neural network, at the end of training, will be our word embeddings.

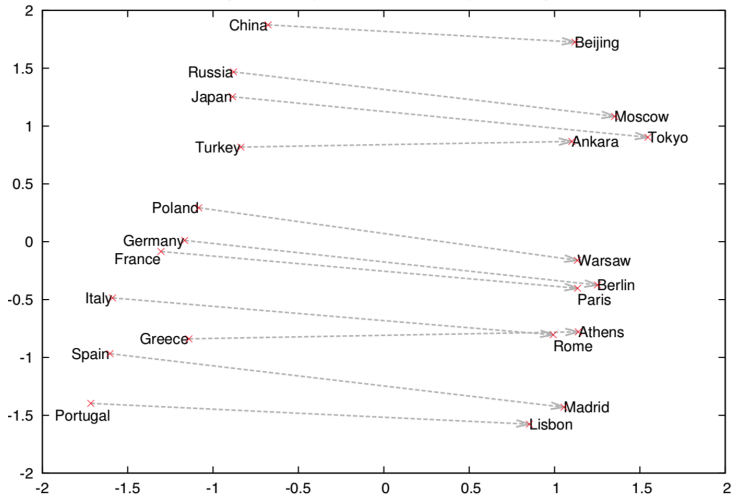

Figure 3.1: 2d PCA projection of 1000d word embeddings of countries and their capitals from Mikolov et al. (2013)

Although this model seems simple, it generates surprisingly meaningful word vectors. The embeddings represent inherent concepts and relationships between words. An example is illustrated in Figure 3.1 taken from Mikolov et al. (2013). It shows how the word vectors for countries and their capitals are organized, when projected onto a 2D surface using Principle Component Analysis (PCA). Another classic example is the following: *king - man + woman* $\approx$ *queen*. This means that if we have the vectors corresponding to *king*, *man* and *woman*, and if we carry out the arithmetic operation on the LHS, we get a vector

which is approximately equal to the vector representation of *queen*. In plain English, this makes sense because a 'king' who is not a 'man' but a 'woman' corresponds to a 'queen'.

Through this example of vector arithmetic, we realize how word embeddings represent semantic information. Word embeddings have been successfully applied to a wide range of NLP tasks due to their compactness (in comparison to one-hot representation) and ability to capture semantic concepts (without any explicit supervision). This study uses only word2vec embeddings and hence the details regarding other embedding models are skipped.

3.2 Sequence to Sequence LSTMs

Text generation is an important area in NLP. Introduced by Sutskever et al. (2014), sequence-to-sequence models have greatly benefited text generation tasks such as question answering, dialog systems and machine translation. Essentially, these models take one sequence as input and generate another sequence as output. This is in contrast to regular classification models, which output only a single class label, and not an entire sequence. Seq2Seq models are typically implemented using two recurrent neural networks, one which is referred to as the encoder and the other is called the decoder. The encoder creates a vector representation of the input sequence that is then fed into the decoder, which then generates tokens in a sequential manner. This study uses the LSTM recurrent neural networks for encoding and decoding sentences.

More concretely, let $\boldsymbol{x} = (x_1, x_2, \cdots, x_{|\boldsymbol{x}|})$ be the tokens (i.e., the corresponding word embeddings) from the source sequence and $\boldsymbol{y} = (y_1, y_2, \cdots, y_{|\boldsymbol{y}|})$ be the tokens from the target sequence. Note that $|\boldsymbol{x}|$ and $|\boldsymbol{y}|$ correspond to the number of tokens (words) in the input and output sequences, respectively. At the end of the encoding process, we will have $\boldsymbol{h}_{|\boldsymbol{x}|}^{(\text{src})}$ and $\boldsymbol{c}_{|\boldsymbol{x}|}^{(\text{src})}$, the final hidden and cell states respectively from the source sequence (see Section 2.2.3). We then set the initial states ($\boldsymbol{h}_{\boldsymbol{1}}^{(\text{tar})}$ and $\boldsymbol{c}_{\boldsymbol{1}}^{(\text{tar})}$) of the decoder LSTM to $\boldsymbol{h}_{|\boldsymbol{x}|}^{(\text{src})}$ and $\boldsymbol{c}_{|\boldsymbol{x}|}^{(\text{src})}$. This is a method to transfer information from the source side to the target side, and is known as hidden state initialization. Then, at each further timestep of the decoding process, we compute $\boldsymbol{h}_j^{(\text{tar})}$ using an input word embedding $\boldsymbol{y}_{j-1}$ (typically the groundtruth during training and the prediction from the previous timestep during testing). This is given by

$$\boldsymbol{h}_j^{(\text{tar})} = \text{LSTM}_{\boldsymbol{\theta}}(\boldsymbol{h}_{j-1}^{(\text{tar})}, \boldsymbol{y}_{j-1}) \tag{3.2}$$

where $\boldsymbol{\theta}$ refers to the weights of the LSTM network. The predicted word at timestep j is then given by a softmax layer as follows:

$$p(y_j) = \text{softmax}(W_{\text{out}} \boldsymbol{h}_j^{(\text{tar})}) \tag{3.3}$$

where W_{out} is a weight matrix and the softmax function is defined by Equation 3.4

$$\text{softmax}(y_{jk}) = \frac{\exp y_{jk}}{\sum_{k=1}^{|V|} \exp y_{jk}} \tag{3.4}$$

where y_{jk} refers to the value of the kth dimension of the output vector at timestep j. In total the output vector at each timestep has $|V|$ dimensions, where $|V|$ corresponds to the vocabulary size. The softmax layer essentially normalizes the output layer and computes a vector of probabilities. From among the $|V|$ dimensions of the output layer, we pick the dimension with the highest calculated probability and generate the word corresponding to that index. The softmax operator is demonstrated with an examples in Figure 3.2.

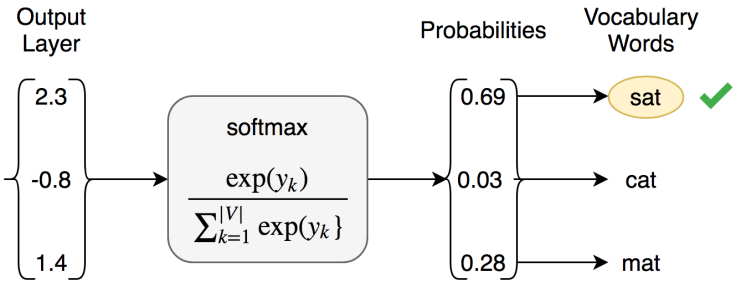

Figure 3.2: Illustration of the softmax output layer

Figure 3.3 is an illustration of the working of a sequence-to-sequence LSTM model. It has a tokenized input sequence [*where*, *do*, *you*, *live*, *?*] provided to the encoder LSTM. Note that the input at each timestep is the word embedding corresponding to the respective word in the vocabulary. The decoder LSTM generates the outputs [*i*, *reside*, *in*, *waterloo*]. Note the use of special tokens: (1) the start-of-sequence token <SOS>, which signals the decoder LSTM to start the decoding process; and (2) the end-of-sequence token <EOS>, which indicates when to stop decoding. The word embeddings for these can also be learnt either by pretraining the word2vec model or while training the Seq2Seq model.

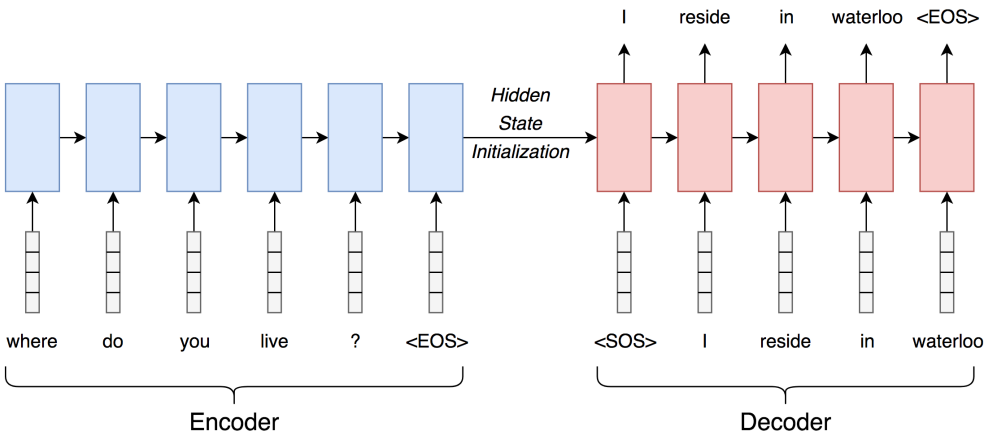

Figure 3.3: Sequence-to-sequence encoder-decoder model framework

3.3 Environment and Libraries

Python 3 was used as the programming environment for this project[1]. For building word vectors, the `gensim` package (Rehurek and Sojka, 2010) in Python was utilized. The deep learning models were implemented using `keras` (Chollet et al., 2015) and `tensorflow` Abadi et al. (2016). In particular, the `tf.contrib.seq2seq` module was used for building encoders and decoders and calculating the loss.

[1]Python Software Foundation http://www.python.org

Chapter 4

Variational Autoencoders

4.1 Introduction

In deep learning, autoencoders (Vincent et al., 2008; Bengio et al., 2014) can be used to encode high dimensional input data into lower dimensional latent code. Variational autoencoders are a very useful class of models that combine neural networks and variational inference. In Bayesian statistics, it is common to compute posterior probability distributions. Variational inference provides a method to approximate these difficult-to-compute probability distributions through optimization. A known probability distribution of the latent code makes it possible to do generative modelling, i.e., to synthesize new samples (e.g., images) similar to the original data. In this chapter, we first provide a short introduction to variational inference. Then we describe the working of variational autoencoders including the reparameterization trick. Finally, we discuss the training difficulties associated with VAEs and empirical results obtained on a natural language dataset.

4.2 Variational Inference

Consider a latent variable model (a probability distribution over two sets of variables) given by Equation 4.1

$$p(X, Z) = p(Z)p(X|Z) \tag{4.1}$$

where, X refers to the observed data and the is Z is the latent variable. In Bayesian modeling, $p(Z)$ is known as the prior distribution of the latent variable and $p(X|Z)$ is the likelihood of the observation X given the latent code Z.

The inference problem in Bayesian statistics refers to computing the posterior distribution, which refers to the conditional density of the latent variables given the data, and is given by $p(Z|X)$ (Blei et al., 2017). Mathematically, this can be written as follows:

$$p(Z|X) = \frac{p(X, Z)}{p(X)} \tag{4.2}$$

The denominator in Equation 4.2 is the marginal distribution of the data, also called the *evidence*. It is computed by marginalizing out the latent variables from the joint distribution $p(X, Z)$.

$$p(X) = \int p(X, Z)dz \tag{4.3}$$

In many cases, the evidence integral is intractable and cannot be computed in closed form (Dave, 2017).

In Variational Inference, we obtain an approximate inference solution by trying to find an approximate distribution from a family of distributions that is similar to the posterior which we wish to estimate. In other words, we minimize the Kullback-Leibler divergence between the approximation and the true posterior distribution. The KL divergence (Kullback and Leibler, 1951) is a statistical method to measure how different two probability distributions are. A lower value of divergence indicates that the distributions are more similar. It is also referred to as relative entropy. Assuming P and Q are two probability distributions, the equation for KL divergence in the discrete case is given by,

$$\mathrm{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{4.4}$$

In the continuous case, this can be written as:

$$\mathrm{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx \tag{4.5}$$

Returning to the discussion of variational inference, we need to find a distribution over the latent variables, namely $q(Z)$, from a family of distributions τ that minimize the KL divergence with respect to the exact posterior $p(Z|X)$. In equation form, this corresponds to

$$q^*(Z) = \underset{q(Z)\in\tau}{\operatorname{argmin}} \quad \mathrm{KL}(q(Z)||p(Z|X)) \tag{4.6}$$

It is to be noted that we may get a better approximation if we choose a more complex family of distributions, at the cost of a more complex optimization process. Also, it is not possible to directly optimize Equation 4.6 since it contains the term $p(Z|X)$, which was difficult to compute in the first place. Using rules of probability and logarithm, we can rewrite Equation 4.6 as follows:

$$\begin{aligned}
\mathrm{KL}(q(Z)||p(Z|X)) &= \mathbb{E}_{q(Z)}\left[\log \frac{q(Z)}{p(Z|X)}\right] \\
&= \mathbb{E}_{q(Z)}\left[\log q(Z)\right] - \mathbb{E}_{q(Z)}\left[\log p(Z|X)\right] \\
&= \mathbb{E}_{q(Z)}\left[\log q(Z)\right] - \mathbb{E}_{q(Z)}\left[\log \frac{p(Z,X)}{p(X)}\right] \\
&= \mathbb{E}_{q(Z)}\left[\log q(Z)\right] - \mathbb{E}_{q(Z)}\left[\log p(Z,X)\right] + \mathbb{E}_{q(Z)}\left[\log p(X)\right]
\end{aligned} \tag{4.7}$$

Rearranging the terms, we obtain

$$\left\{\mathbb{E}_{q(Z)}\left[\log p(Z,X)\right] - \mathbb{E}_{q(Z)}\left[\log q(Z)\right]\right\} + \mathrm{KL}(q(Z)||p(Z|X)) = \log p(X) \tag{4.8}$$

$$\mathrm{ELBO}(q) + \mathrm{KL}(q(Z)||p(Z|X)) = \log p(X) \tag{4.9}$$

The first term on the LHS of Equation 4.8 is known as the variational lower bound (Kingma, 2017) The term on the RHS, $\log p(X)$, known as log *evidence* is constant with respect to $q(Z)$. Since the variational lower bound is also a lower bound to the log *evidence*, it is also known as Evidence Lower Bound (ELBO)(Yang, 2017). That is, $\log p(X) \geq \mathrm{ELBO}(q)$ for any $q(Z)$.

The LHS, which is the sum of the variational lower bound $\mathrm{ELBO}(q)$ and the KL divergence $\mathrm{KL}(q(Z)||p(Z|X))$ adds up to a constant term on the RHS (the log probability of the observations) in Equation 4.9. Hence, the objective of minimizing KL divergence is equivalent to maximizing the variational lower bound. This is illustrated in Figure 4.1.

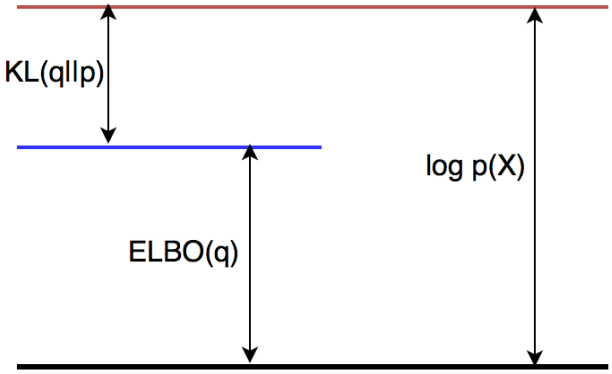

Figure 4.1: Relationship between $\mathrm{KL}(q(Z)||p(Z|X))$, $\mathrm{ELBO}(q)$ and $\log p(X)$

We can rewrite the ELBO into a simpler and more interpretable form as follows

$$\begin{aligned}
\mathrm{ELBO}(q) &= \mathbb{E}_{q(Z)}\left[\log p(Z,X)\right] - \mathbb{E}_{q(Z)}\left[\log q(Z)\right] \\
&= \mathbb{E}_{q(Z)}\left[\log p(X|Z)p(Z)\right] - \mathbb{E}_{q(Z)}\left[\log q(Z)\right] \\
&= \mathbb{E}_{q(Z)}\left[\log p(X|Z)\right] + \mathbb{E}_{q(Z)}\left[\log p(Z)\right] - \mathbb{E}_{q(Z)}\left[\log q(Z)\right] \\
&= \mathbb{E}_{q(Z)}\left[\log p(X|Z)\right] + \mathbb{E}_{q(Z)}\left[\log \frac{p(Z)}{q(Z)}\right] \\
&= \mathbb{E}_{q(Z)}\left[\log p(X|Z)\right] - \mathrm{KL}(q(Z)||p(Z))
\end{aligned} \tag{4.10}$$

The first term in Equation 4.10 corresponds to the expected log-likelihood of the data. The second term refers to the negative KL divergence between approximate posterior $q(Z)$ and the prior $p(Z)$. With the overall objective to maximize $\mathrm{ELBO}(q)$, we maximize the log-likelihood while encouraging a posterior with a density function that is close to the prior.

One way to compute the approximate posterior $q^*(Z)$ in Equation 4.6 is using mean field inference (Wainwright et al., 2008). The main assumption in the mean field variational family of distributions is that each dimension in the latent code $\mathbf{z}$ is mutually independent and is modelled by its own density function $q_i(z_i)$. The optimization is carried out using Coordinate ascent mean-field variational inference (CAVI) algorithm and readers are referred to Dave (2017) and Kuleshov and Ermon (2017) for details. This thesis explores how deep learning models can be used to compute approximate posteriors, namely with the help of variational autoencoders. In comparison to traditional methods, the VAE leverage modern neural networks and is a more powerful density estimator.

4.3 Variational Autoencoders

Introduced by Kingma and Welling (2013), variational autoencoders use neural networks to parametrize the density distributions p and q, discussed in Section 4.2. In theory, with sufficient layers used, neural networks can work as universal function approximators, i.e., they can be used to represent any function.

In the case of VAEs, neural networks can be used to represent the inference network (the encoder) and the generative network (the decoder) (Altosaar, 2017; Kuleshov and Ermon, 2017). To compute the approximate posterior q, we can design a neural network with parameters ϕ, called the encoder $q_\phi(Z|X)$. In order to reconstruct the data, we can use another neural network with parameters θ, which is represented as $p_\theta(X|Z)$, referred to as the decoder. This is illustrated with the help of Figure 4.2. As described in Section 2.2.1, these parameters correspond to the model weights and biases.

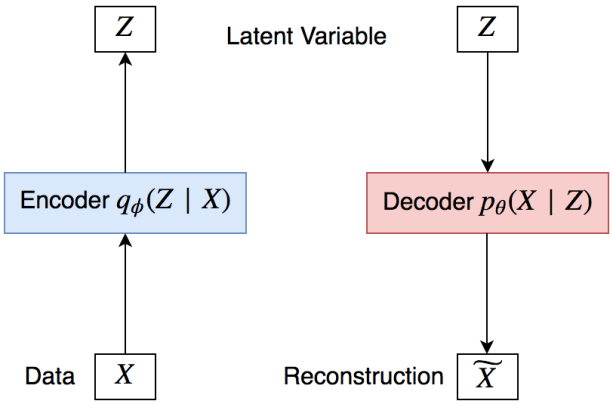

Figure 4.2: Encoder network for encoding the original data into the latent space and decoder network for reconstruction using the latent representation

Using the notation described above, we can replace the approximate posterior $q(Z)$ in Section 4.2 to $q_\phi(Z|X)$, the one parametrized by the encoder neural network. Consider a dataset $\mathcal{D} = \{x^{(n)}\}_{n=1}^N$, the likelihood of a data point (log *evidence*) and the ELBO(q) are related as follows:

$$\log p_{\boldsymbol{\theta}}(x^{(n)}) \geq \mathbb{E}_{z\sim q_{\boldsymbol{\phi}}(z|x^{(n)})}\left[\log\left\{\frac{p_{\boldsymbol{\theta}}(x^{(n)},z)}{q_{\boldsymbol{\phi}}(z|x^{(n)})}\right\}\right]$$

$$= \mathbb{E}_{z\sim q_{\boldsymbol{\phi}}(z|x^{(n)})}\left[\log p_{\boldsymbol{\theta}}(x^{(n)}|z)\right] - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(z|x^{(n)})\|p(z)\right) \tag{4.11}$$

The above equations essentially rewrite Equations 4.9 and 4.10 in the notation discussed for neural network parameterization of probability density functions. We are required to maximize the ELBO(q) shown in Equation 4.11, which is equivalent to minimizing $-$ELBO(q). The loss function for the neural network can be written as

$$\begin{aligned} J^{(n)} &= -\mathbb{E}_{z\sim q_{\boldsymbol{\phi}}(z|x^{(n)})}\left[\log p_{\boldsymbol{\theta}}(x^{(n)}|z)\right] + \mathrm{KL}\left(q_{\boldsymbol{\phi}}(z|x^{(n)})\|p(z)\right) \\ &= J_{\mathrm{rec}}(\boldsymbol{\theta}, \boldsymbol{\phi}, x^{(n)}) + \mathrm{KL}\big(q_{\boldsymbol{\phi}}(z|x^{(n)})\|p(z)\big) \end{aligned} \tag{4.12}$$

The first term, called *reconstruction loss*, is the (expected) negative log-likelihood of data, similar to traditional deterministic autoencoders. For sequence-to-sequence models, this is calculated as a summation of the categorical cross entropy of the prediction across all timesteps of the decoder. The second term refers to the KL divergence between the approximate posterior distribution $q_\phi(Z|X)$ that the encoder network maps the original data space into, and the pre-specified prior. In case of continuous latent variables, the prior is typically assumed to be Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$. In this case, the KL divergence denoted as $\mathrm{KL}(\mathcal{N}(\boldsymbol{\mu}_z^{(n)}, \boldsymbol{\sigma}_z^{(n)})\|\mathcal{N}(\mathbf{0}, \mathbf{I}))$ for input $x^{(n)}$ is given by the Equation 4.13 (assuming that only one sample is drawn for each input data point).

$$\frac{1}{2}(1 + \log((\boldsymbol{\sigma}_z^{(n)})^2) - (\boldsymbol{\mu}_z^{(n)})^2 - (\boldsymbol{\sigma}_z^{(n)})^2) \tag{4.13}$$

As described in Doersch (2016), the KL divergence term can be viewed as a regularization technique (similar to L2-regularization that is used to avoid overfitting). Without the KL term, the model boils down to a regular Deterministic Autoencoders (DAE), that encodes the data into a latent space which can be then used for reconstruction. With the KL term regularization, the latent space is forced into a pre-specified distribution. As a result, the latent space now follows a known distribution, from which one can sample and synthesize new data, such as images (Kulkarni et al., 2015) and sentences (Bowman et al., 2015b). This is in contrast to traditional DAEs, whose latent space can only be used for reconstruction and does not typically possess any such interesting properties. In other words, DAEs map input data onto arbitrary points on a high dimensional manifold. Whereas, VAEs project data onto continuous ellipsoidal regions that fill the latent space, rather than simply memorizing the arbitrary mappings for the input data (Bowman et al., 2015b). This is depicted in Figure 4.3, where a VAE is used to project the observed data X, which has an unknown distribution, into a latent code Z with a known distribution (trained to be approximately similar to that of the prior by using the KL term regularization).

Figure 4.4 provides a comparison of the architectures for a DAE versus a VAE. In the traditional DAE, after we learn a latent code $\boldsymbol{z}$, this is directly fed to the decoder. In

contrast, for VAEs, we use the encoder outputs to learn the parameters of the underlying posterior distribution. For example, if we assume Gaussian, we would want to learn its mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}$ (for simplicity we can assume a diagonal covariance matrix $\boldsymbol{\Sigma}$, which implies that the dimensions of the latent code are mutually independent). Next, we can generate a random sample $\boldsymbol{z}$ from this known Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ and feed it to the decoder. This difference in the procedure during the forward pass is demonstrated in Figure 4.4.

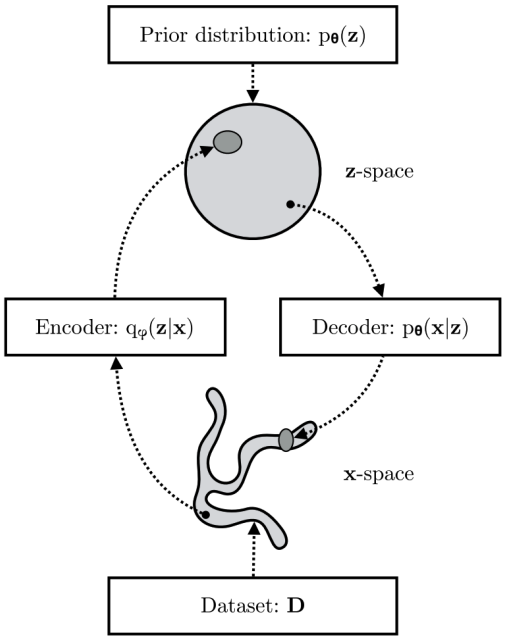

Figure 4.3: Latent space mappings learnt by a VAE. New data can be synthesized by sampling from the known prior distribution (Kingma, 2017)

4.4 Reparameterization Trick

Once we have the VAE network architecture defined, we next focus on training the model using stochastic gradient descent (SGD). However, this leads to an issue because the model in its original form has a probabilistic node in the computational graph as shown in Figure 4.5. Sampling of the latent variable $\boldsymbol{z}$ from the approximate posterior is carried out at

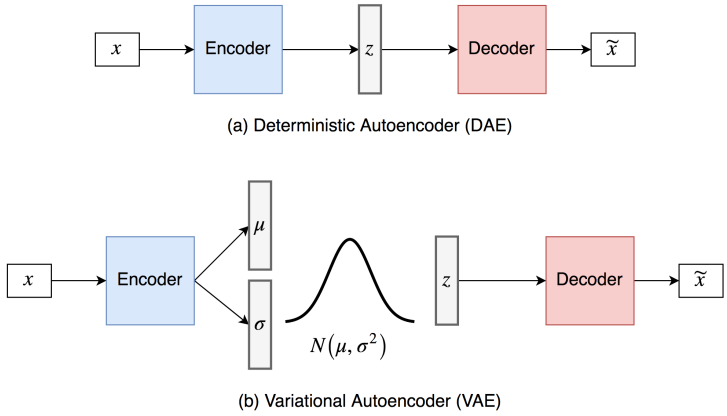

Figure 4.4: Difference in DAE and VAE architectures

the node indicated in blue on the LHS. This stochastic node results in a disconnect in the computational graph and we cannot propagate gradients back to the encoder network.

In order to circumvent this issue, Kingma and Welling (2013) proposed the *reparameterization* trick. This simple solution involves sampling from a fixed distribution, followed by a variable transformation to the original latent space.

We can consider the example of a Gaussian distribution to illustrate the reparameterization trick. Instead of directly sampling $\boldsymbol{z}$ from its posterior distribution given by $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, we can instead sample ϵ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and do a simple variable transformation as shown below:

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \otimes \epsilon \tag{4.14}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ have already been obtained by transforming the encoder output (see Figure 4.4). The effect of this reparameterization is that the gradient passes back to the encoder network (through $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$) and the model is trained end to end. There is no gradient update at the node where we do the sampling of ϵ from the fixed distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. This is exactly what we need as we do not want the fixed sampling to be influenced during gradient propagation.

4.5 Experiments

In this section, the autoencoding experiments with VAEs are detailed. We start with a description of the data and the pre-processing steps. This is followed by the training details

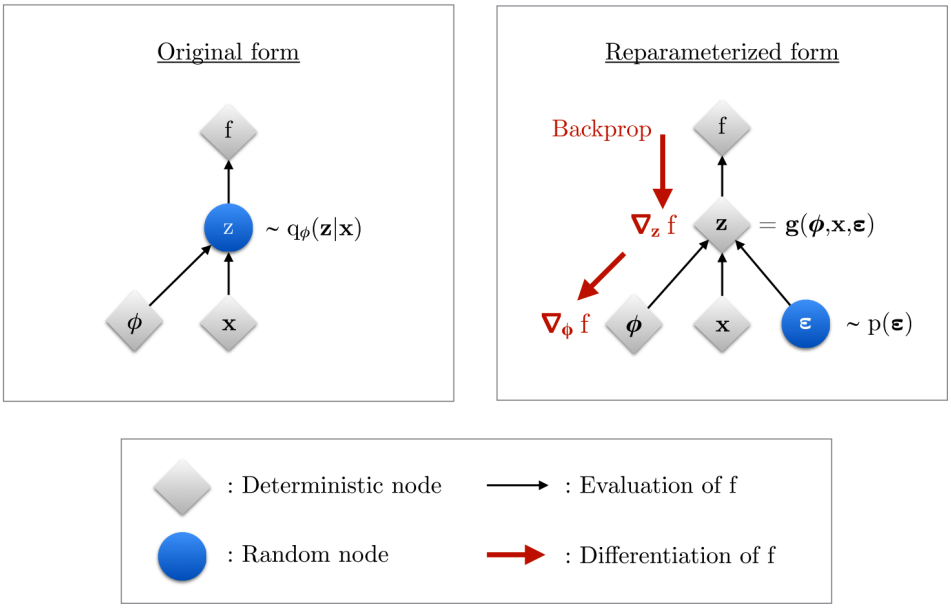

Figure 4.5: Demonstation of the Reparameterization Trick in VAEs (Kingma, 2017)

including the optimization challenges and the strategies adopted to train the network in a stable manner.

4.5.1 Dataset

For VAE training and hyperparameter tuning, we use the Stanford Natural Language Inference (SNLI) Dataset (Bowman et al., 2015a). In order to create the SNLI dataset, the authors adopted the following annotation procedure — human annotators were shown an image along with its original one-line caption; they were then asked to provide three separate sentences about the scene in the image, corresponding to the following three class labels – *entailment*, *neutral*, and *contradiction*, for the task of recognizing textual entailment (Androutsopoulos and Malakasiotis, 2010). The original corpus consists of 570k pairs of sentences and in this experiment, we use a randomly sampled subset of 80k sentences for training the VAE to carry out reconstruction.

4.5.2 Data Pre-processing

The following data pre-processing steps are adopted:

1. All sentences are converted to lowercase.
2. All punctuations except *comma* (,) and *full stop* (.) are dropped.
3. We generate `word2vec` embeddings for all the words in the subset corpus using the CBOW model (see Section 3.1). The size of the context window was set to 5 and word embedding dimension was chosen to be 300d.
4. The sentences are then shuffled and a train/validation/test split of 78k/1k/1k is created.
5. Each sentence is appended with an end of sequence token <EOS>.
6. We then decide on the vocabulary size of the model $|V|$. Only the top $|V|$ most frequently occurring words are retained in the corpus, while the rest are replaced with a special token <UNK>, referring to words *unknown* in the vocabulary.
7. Next, we tokenize sentences into a list of words using the NLTK tokenizer (Bird and Loper, 2004). Each word is then mapped into an integer index.
8. Finally, we set the maximum sequence length m (chosen as 10 for this experiment). Sentences with fewer than m words are resized to be of size m by appending a special token named <PAD>. Sentences with more than m words are trimmed off at m words.

4.6 VAE Optimization Challenges

As described in Bowman et al. (2015b), training VAEs for text generation using RNN encoder-decoder is not straightforward. Optimization challenges associated with the Kullback Leibler (KL) divergence term (between the approximate posterior and the prior) of the loss function, vanishing to zero makes the task of training VAEs notoriously difficult (Bowman et al., 2015b; Yang et al., 2017). When the KL loss is zero, this means that the approximate posterior is exactly the same as the prior. As a consequence of this, the model fails to encode any useful information into the latent space. This causes the model to have

a poor reconstruction for any given input. Hence, there is a need to balance the reconstruction term and the KL term of the loss function described in Equation 4.12. Bowman et al. (2015b) suggest two strategies to overcome this issue of KL term collapse, which are adopted in this work and are described in the following subsections.

4.6.1 KL Cost Annealing

In this approach, we introduce a coefficient to the KL term of the loss function. This coefficient, referred to as the KL weight is gradually increased (annealed) from zero to a threshold value, as the training progresses. We can rewrite Equation 4.12 as follows

$$J^{(n)} = J_{\text{rec}}(\boldsymbol{\theta}, \boldsymbol{\phi}, x^{(n)}) + \lambda \cdot \text{KL}\big(q_{\boldsymbol{\phi}}(z|x^{(n)}) \| p(z)\big) \tag{4.15}$$

where λ refers to the KL weight, whose value is set to be a function of the iteration number during training. The key idea behind this technique is that we first allow the model to learn to reconstruct the input sentences well, and then we gradually focus on mapping the sentence encodings onto a continuous latent space by making the approximate posterior to be close to the prior. Another way to think of this annealing of the KL weight is that we gradually transform the model from a completely deterministic autoencoder into a variational one. In this study, we experimentally identify new and improved KL weight annealing schedules, which are discussed in Section 4.6.4. We find that the model is sensitive to the rate at which the KL weight is increased and the details are provided in Section 4.7.

4.6.2 Word Dropout

The other method to ensure that we learn a useful latent representation is called word dropout. As mentioned in Section 3.2, we feed the ground truth tokens delayed by one timestep to the decoder during training (see Figure 3.3). In the word dropout strategy, we replace any given input token to the decoder RNN with the <UNK> token with certain probability $p \in [0, 1]$. If $p = 0$, the it means that no words are dropped out during decoding. On the other extreme, if $p = 1$ we replace each word fed to the decoder with <UNK>. The <UNK> token essentially conveys no information about the source sentence (that is to be reconstructed). Referring to Figure 4.6, an example input with $p = 0.5$ dropout can be [*i*, <*UNK*>, *in*, <*UNK*>], i.e., half of the words are replaced with the <UNK> token. Doing this weakens the decoder and encourages the model to encode more information in the latent variable $\boldsymbol{z}$ to make accurate reconstructions.

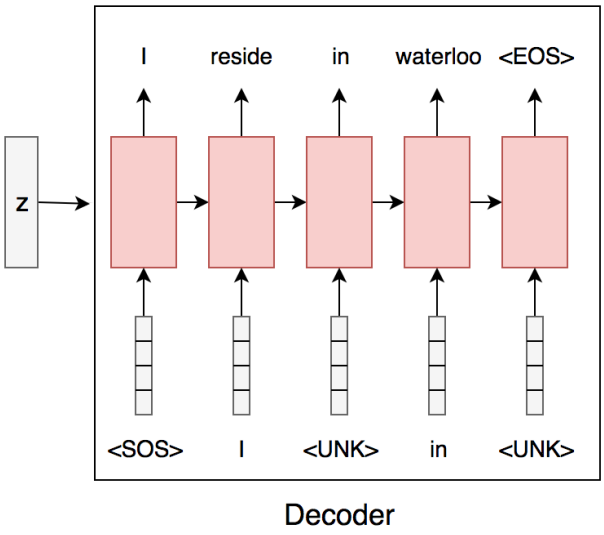

Figure 4.6: Demonstration of Word Dropout

4.6.3 Training Details

For training this sequence-to-sequence variational autoencoder model, we used LSTM units of dimension 100d for both the encoder and the decoder. The dimension of the latent vector $\boldsymbol{z}$ was also chosen to be 100d. We adopted 300d word embeddings (Mikolov et al., 2013), pretrained on the 80k subset of the SNLI dataset described in Section 4.5.1. For both the source and target sides, the maximum sequence length was set to be 10. The vocabulary was limited to the most frequent 20k tokens (i.e., $|V| = 20000$). The batch size was set to be 32. Following Kingma and Welling (2013), we choose the standard normal distribution, $\mathcal{N}(\mathbf{0}, \mathbf{I})$ to be the prior.

To learn the model weights, we experiment with both the stochastic gradient descent (SGD) algorithm (Bottou, 2010) and the Adam optimizer (Kingma and Ba, 2014). For both the optimizers, a constant learning rate of 0.001 was used throughout the training process.

The model is trained for 10 epochs. We observe that the validation set converges at around 10 epochs and hence stop training further. We also compute BLEU (Bilingual Evaluation Understudy) scores on the reconstructed sentences. Originally introduced for automatic evaluation of machine translation systems, BLEU (Papineni et al., 2002) scores can be used to assess the sentence reconstruction capability of autoencoders. BLEU-1 measures the unigram overlap between the generated sentence and a set of reference

sentences, while penalizing generated sentences that are short. In the same manner, we can determine the bigram, trigram and 4-gram overlap and report BLEU-2, BLEU-3 and BLEU-4 respectively. The automatic evaluation using BLEU scores has been reported to be correlated with human judgment (Papineni et al., 2002). For computing, BLEU-j based on the j-gram overlap (i.e., precision$_j$), we use the following equation.

$$\text{BLEU-}j = \min\left(1, \frac{\text{generated-length}}{\text{reference-length}}\right) * (\text{precision}_j) \tag{4.16}$$

In addition to the ability of a VAE to fluently reconstruct the original input, we also need to assess the quality of the latent space created (refer Figure 4.3). This is done in a qualitative manner by randomly sampling points from the latent space and generating sentences. The exact details will be discussed in Section 4.7. A well trained VAE model should be able to generate new sentences (unseen in the training set) that are both syntactically and semantically correct.

4.6.4 VAE Variants

As mentioned in Section 4.6, training variational autoencoders for probabilistic sequence generation is not very straightforward. We try out multiple settings to train the VAE. Here, we describe and compare five VAE variants which are summarized in Table 4.1.

The models ADAM-NoAnneal-1.0λ and ADAM-NoAnneal-0.001λ are trained with no annealing and no word dropout. ADAM-tanh-3000 and ADAM-linear-10000 have different annealing schedules, i.e., the rate and function based on which annealing is done. In ADAM-tanh-3000, we anneal till 3000 iterations based on Equation 4.17. At this point, the value of λ reaches 0.047, and we continue training with this constant λ till model convergence. We have a similar setting in ADAM-linear-10000, where the value of λ reaches 0.05 after 10000 iterations (based on Equation 4.18) and is then kept constant for the rest of the training. For the final 3 variants in Table 4.1, word dropout was implemented as follows - at the start of training no words are dropped out ($p = 0.0$) and at the end of every epoch, we increase the dropout rate by 0.05 until it reaches a maximum value of $p = 0.5$. The model SGD-tanh-3000 is trained with the same settings as ADAM-tanh-3000, except that we use an SGD optimizer instead of ADAM.

$$\lambda_i = \frac{\tanh\left(\frac{i-4500}{1000}\right) + 1}{2} \tag{4.17}$$

VAE Variants	
ADAM-NoAnneal-1.0λ	VAE trained with no KL cost annealing. The KL coefficient (λ) is set to a constant value of 1.0 throughout training. Optimizer used is ADAM.
ADAM-NoAnneal-0.001λ	Same setting as above, except that the λ is set to a constant value of 0.001
ADAM-tanh-3000	KL cost annealing from 0 to 3000 iterations using a rescaled tanh function (refer Eqn 4.17). Optimizer used is ADAM.
SGD-tanh-3000	Same setting as above, but with Stochastic Gradient Descent Optimizer (SGD)
ADAM-linear-10000	KL cost annealing from 0 to 10000 iterations in a linear manner (refer Eqn 4.18). Optimizer used is ADAM.

Table 4.1: Training VAE with different settings

$$\lambda_i = \frac{i}{200000} \tag{4.18}$$

where i corresponds to the iteration number.

The learning curves for the different variants described earlier are illustrated in Figure 4.7. It can be seen that when there is no annealing procedure in place, the KL loss instantly vanishes to a near zero value within the first few iterations. In contrast, when $\lambda = 0.001$, the value of $\lambda \times$ KL is low and the model has a very small effect of the KL regularizer term. As a result, such a model tends to be more *deterministic* in nature.

For the models that have annealing in place, the iteration till which annealing is done is an important hyperparameter. The method by which we decide the threshold value till which annealing is carried out is based on the $\lambda \times$ KL graph. The green line in Figure 4.7 shows that beyond 3000 iterations (approximately), the value of $\lambda \times$ KL starts to decrease after reaching a maximum value. We realize that if we do not stop annealing at this point, the KL loss steadily decreases further and collapses to zero. We determine that it is ideal to stop annealing once the value of $\lambda \times$ KL has reached its maximum value. Beyond this point, we can continue with this constant KL coefficient for the rest of the training. For ADAM-tanh-3000 and ADAM-linear-10000, these values are 0.047 and 0.05 respectively. It can be observed that during later stages of training, the graphs for these two models

tend to converge. When the optimizer is changed to SGD, a completely different pattern is observed. The reason for this unusual trend needs to be investigated further.

4.7 Results

4.7.1 Sentence Reconstruction and Random Sampling

The reconstruction performance measured in terms of BLEU scores for the different model variants are listed in Table 4.7.1. In this case, to generate the reconstructed sentence, we feed the mean vector ($\boldsymbol{\mu}$) to the decoder, rather than the sampled $\boldsymbol{z}$ (refer Equation 4.14). This is done so that we do not consider any variance in the latent space and pick the most probable value, the mean $\boldsymbol{\mu}$ (referred to as the max *a posteriori* or MAP estimate).

In order to assess the quality of the latent space, we randomly sample points from the prior distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and feed the sampled $\boldsymbol{z}$ to the decoder to generate new sentences. This is illustrated in Figure 4.8. If the learnt latent space is continuous, we can expect to generate a meaningful sentence by sampling from anywhere within the latent space. Note that in this setting, the encoder network can be discarded after completion of training. The randomly generated sentences for each VAE variant are shown in Table 4.3.

For comparison purposes, the BLEU scores and random sentence generations obtained using a deterministic autoencoder (DAE) are also presented. It is to be noted that while DAEs can give better reconstructions, there is no useful latent space learnt.

The regular VAE model trained without any optimization strategies, ADAM-NoAnneal-1.0λ, produces the same output sentence irrespective of the input. This is due to the KL divergence part of the loss function collapsing to zero which causes the model to have both (1) poor reconstruction capability and (2) poor latent space.

The observation on training the model with a small constant KL coefficient of 0.001 (ADAM-NoAnneal-0.001λ) is that the model tends to function like a deterministic autoencoder. This is expected since $\lim \lambda \rightarrow 0$, the model ignores the KL term and becomes a deterministic autoencoder. Although the model has good reconstruction performance indicated by the high BLEU scores, its latent space is not very desirable. The sentences generated are not very meaningful and also not grammatically correct in most cases.

KL weight annealing and word dropout are indeed very useful training heuristics. This can be seen from the models ADAM-tanh-3000 and ADAM-linear-10000, both of which have sentences of similar quality being generated from the latent space. The sentences are

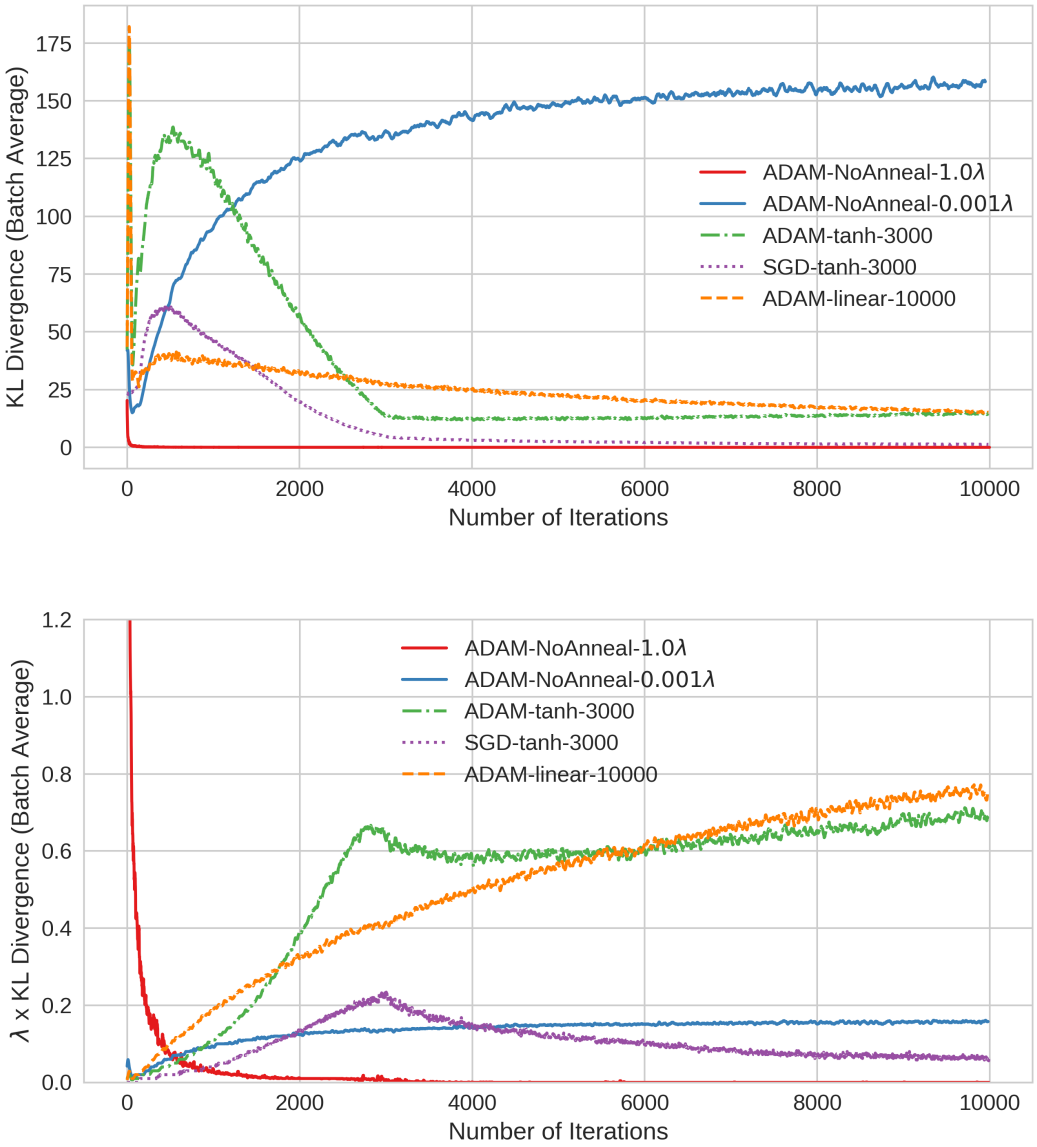

Figure 4.7: Learning curves of the VAE variants. **Top**: KL divergence, **Bottom**: $\lambda \times$ KL divergence.

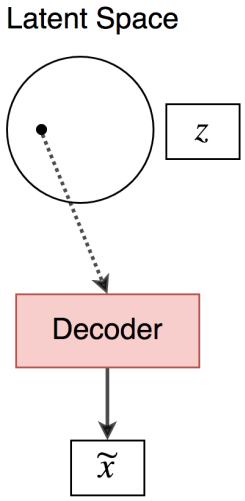

Figure 4.8: Demonstration of Random Sampling from Latent Space

usually syntactically and semantically correct. They are also diverse, i.e., truly random in the sense that they talk about different topics. In terms of BLEU scores, ADAM-tanh-3000 performs relatively better than the linear KL annealing model. However, it is to be noted that when the optimizer was changed to SGD instead of ADAM, the same model turns out to be extremely poor. The sentences generated are more or less the same, i.e., not diverse. The reconstruction capability is only slightly better than the model with the worst performance, ADAM-NoAnneal-1.0λ.

By comparing ADAM-NoAnneal-0.001λ and ADAM-tanh-3000, we realize that reconstruction performance and quality of the latent space are conflicting objectives. The model with better BLEU scores typically results in a non-continuous latent space, generating sentences of lower quality and vice-versa. If our primary objective was just sentence reconstruction, we could simply use a deterministic autoencoder, that can reconstruct sentences in a near perfect manner. From the perspective of probabilistic natural language generation by sampling from a known distribution, ADAM-tanh-3000 is a more desirable model.

The ADAM-tanh-3000 variant has a good balance between reconstruction capability and smoothness of the latent space. Hence we adopt only ADAM-tanh-3000 for further discussions and evaluations. We however compare it with the deterministic autoencoder to demonstrate other interesting properties exhibited by VAEs.

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4
Deterministic AE	89.56	83.22	78.29	73.73
ADAM-NoAnneal-1.0	26.57	10.32	4.72	2.05
ADAM-NoAnneal-0.001	88.59	81.90	76.77	72.05
ADAM-tanh-3000	66.97	53.55	44.36	36.50
SGD-tanh-3000	32.58	13.74	6.79	2.70
ADAM-linear-10000	65.55	52.03	42.98	35.29

Table 4.2: Sentence reconstruction performance for the Deterministic AE and different VAE variants

4.7.2 Linear Interpolation

The VAE objective function simultaneously minimizes the negative log likelihood of the data and the KL divergence between the approximate posterior distribution and the pre-specified prior. At the end of training, we can assume that the posterior and prior distributions are approximately close. This allows us to sample from the prior, which is $\mathcal{N}(\mathbf{0}, \mathbf{I})$ in our case, and generate new sentences from the latent space (Section 4.7.1). If the learnt latent space is continuous, then we expect the points sampled from any part of the latent space to generate valid sentences. Another method to determine the continuity of the latent space is using linear interpolation or *homotopy* (Bowman et al., 2015b).

Assume that we are give two input sentences A and B. After mapping them to the latent space, we can obtain their latent representations $\boldsymbol{z}_A$ and $\boldsymbol{z}_B$. For instance, we can linearly interpolate between A and B by manipulating the latent vector as follows:

$$\boldsymbol{z}_{\alpha_i} = \alpha_i \cdot \boldsymbol{z}_A + (1 - \alpha_i) \cdot \boldsymbol{z}_B \tag{4.19}$$

where $\alpha_i \in \left[0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1\right]$. This gives us 4 new sentences between A and B. With a good latent space, we expect the transition from A to B to happen in a smooth manner. Each linearly interpolated vector in the latent space should result in a syntactically and semantically correct sentence. The method by which linear interpolation of points in the latent space is carried out is depicted in Figure 4.9.

We carry out the above linear interpolation operation for both the deterministic AE and the variational AE. We can observe from Table 4.4 that in the case of the VAE, the

Deterministic AE	**ADAM-NoAnneal-1.0**
a men wears an umbrella waits to	*a man is sitting on a bench .*
a couple cows a monument	*a man is sitting on a bench .*
some a play mat on the gym	*a man is sitting on a bench .*
falling bricks is checking to a tree .	*a man is sitting on a bench .*
skate other women .	*a man is sitting on a bench .*
a seagull is brown to a sandbox	*a man is sitting on a bench .*
a training underwater with the jog down the mountains	*a man is sitting on a bench .*
there is sleeping and two rug .	*a man is sitting on a bench .*
a man in a pick photos	*a man is sitting on a bench .*
a boy are people at a lake escape .	*a man is sitting on a bench .*
ADAM-NoAnneal-0.001	**ADAM-tanh-3000**
two men sit past where the government entering a scene	*the dog is sleeping in the grass .*
they are excited formation to ride a castle of a	*the girls are being detained .*
their janitor is leaving the dirt wearing his suits .	*the group of people are going to begin .*
two children in it exits a	*a girl with blond-hair on a bike with a stick*
six people sitting are sorting at single radio in .	*a woman and a man are walking on a street*
the guy gets cancer .	*a brown dog barking at a small dog .*
i woman who is on watch a factory	*the people and a woman are giving a speech .*
three people are wearing overalls .	*the boy is competing with his athletes .*
an artist is giving a women in china .	*a biker is on the racetrack .*
the camel is pulling on a bull .	*a man wearing white racing outside .*
SGD-tanh-3000	**ADAM-linear-10000**
people are playing in the street .	*woman talking to the boy .*
a man is playing a blue shirt and a blue	*there are boys playing on a trampoline .*
a man is playing in the street .	*a group of people are cooking*
a man is playing in a blue shirt .	*two girls are cleaning up their beds .*
the man is in a blue shirt and a blue	*a child with bare eyes closed and being pulled by*
the man is playing in the street .	*the little girl is brushing her teeth .*
a man is wearing a blue shirt and a blue	*a few school students riding in a meadow .*
two men are playing in the street .	*the woman in the bathtub .*
a man is wearing a blue shirt and a blue	*two cyclists ride horses .*
a man is playing in a blue shirt .	*some kids are sitting .*

Table 4.3: Generation of random sentences by sampling from the latent space

transition is smooth. The sentences in between are both fluent and meaningful. Although, one may note that the input sentences A and B have not been perfectly reconstructed in case of the VAE. The deterministic AE on the other hand, has a better reconstruction of the original sentences. However, the intermediate sentences are not grammatically correct or meaningful. The transitions are also observed to be very irregular or non-continuous.

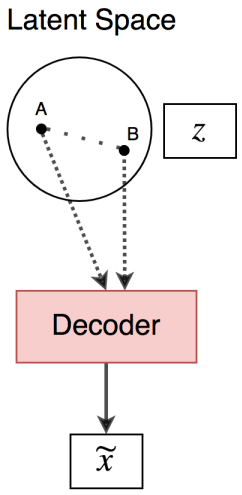

Figure 4.9: Demonstration of Linear Interpolation between points in the Latent Space

Hence, we conclude that inclusion of the KL regularization term in the loss function of the VAE gives rise to a smooth and continuous latent space.

4.7.3 Sampling From Neighborhood

In a VAE, the reparameterization trick uses the mean and standard deviation to compute the latent vector as $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \otimes \epsilon$. In other words, we sample a point in the latent space that is within one standard deviation from the mean. If we sample further away from the mean, we can expect the generated sentence to be more different from the original input which we needed to reconstruct.

In this experiment, we sample the latent vector as $\boldsymbol{z} = \boldsymbol{\mu} + 3\boldsymbol{\sigma} \otimes \epsilon$, i.e., within 3 standard deviations from the mean. This experiment of sampling from the neighborhood of a given input $\boldsymbol{x}$ is illustrated in Figure 4.10. In a VAE, the latent space is continuous, which means that there are no empty regions. Because of this, the latent vectors sampled from any region will have meaningful representations which can be decoded into sentences. Also, since we sample not too far away from the mean, we can expect the generated sentence to have some topical similarity with the input. This can be observed from the VAE examples in Table 4.5, where for a given input, we generate sentences using multiple sampled latent vectors. Each sampled $\boldsymbol{z}$ generates a different sentence, which is however topically similar to the input.

Deterministic AE	VAE ADAM-tanh-3000
Sentence A: there is a couple eating cake .	
there is a couple eating cake .	*there is a couple eating cake .*
there is a couple eating cake .	*there is a couple eating .*
there is a couple eating cake .	*there is a couple eating dinner .*
there is a group of people eating a party .	*there is a couple of people eating dinner .*
a group of men are watching a party .	*a group of people are having a conversation .*
a group of men are watching a dance party .	*a group of men are having a discussion .*
a group of men are watching a dance party .	*a group of men are watching a movie .*
a group of men are watching a dance party .	*a group of men are watching a movie theater .*
Sentence B: a group of men are watching a dance party .	
Sentence A: two boys are performing their fencing skills .	
two boys are performing their fencing skills .	*two boys are performing their martial arts .*
two boys are performing their fencing skills .	*two boys are doing their homework .*
two boys are performing their sports skills .	*two boys are doing their homework outside .*
two boys are performing chess the last .	*two men are playing video games .*
a young man is eating air inside a theater .	*a young man is playing golf at a park .*
a white man is eating alone inside a restaurant .	*a young man is eating dinner in a restaurant .*
a white man is eating alone inside a restaurant .	*a young man is eating inside a restaurant .*
a white man is eating alone inside a restaurant .	*a homeless man is eating inside a restaurant .*
Sentence B: a white man is eating alone inside a restaurant .	

Table 4.4: Linear interpolation between Sentences A and B

On the other hand, the deterministic autoencoder does not encode input information into a continous space. The latent space in this case is an arbitrary high dimensional manifold. As a result, when we sample points which are 3 standard deviations from the mean, we may end up in empty regions which have no useful latent encodings present. This causes the model to generate the exact same sentence which we would have obtained by feeding the mean vector to the decoder, even when multiple samples are drawn. In other words, the DAE maps input sentences into points on the high dimensional space that are far apart with empty regions in between.

With the results from the 3 experiments, namely random sampling, linear interpolation

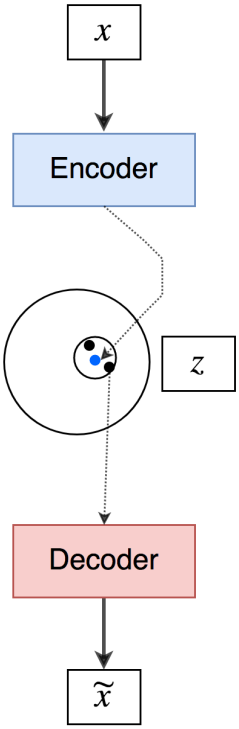

Figure 4.10: Demonstration of sampling from the neighborhood of a given input $\boldsymbol{x}$

and sampling from neighbourhood, we can conclude that the VAEs produce latent spaces that are continuous and characterized by a known distribution. This interesting property allows us to use VAEs for probabilistic generation of text, if trained in a careful manner.

Deterministic AE	VAE ADAM-tanh-3000
Input Sentence: a dog with its mouth open is running .	
a dog with its mouth is open running .	*a dog with long hair is eating .*
a dog with its mouth is open running .	*a guy and the dogs are holding hands*
a dog with its mouth is open running .	*a dog with a toy at a rodeo .*
Input Sentence: the man is wearing a black suit .	
the man is wearing a black suit .	*there are two men in a blue shirt .*
the man is wearing a black suit .	*the man is drinking from the bar .*
the man is wearing a black suit .	*the men in a black suit walking through a crosswalk .*
Input Sentence: there are people sitting on the side of the road	
there are people sitting on the side of the road	*the boy is walking down the street .*
there are people sitting on the side of the road	*there are people standing on the street outside*
there are people sitting on the side of the road	*the police are on the street corner .*

Table 4.5: Generating sentences by sampling from the neighbourhood of the mean in the latent space

Chapter 5

Bypassing Phenomenon

In this chapter, we discuss an important design aspect for variational neural network models. Specifically, we introduce a problem which we refer to as *bypassing* connection. We discuss its effects on the performance of variational autoencoders both quantitatively and qualitatively.

5.1 Problem Description

The VAE is essentially an encoder-decoder model. With the help of the learnt latent variable, the decoder can be used as a generative model $p_\theta(X|Z)$. The implementation of the VAE neural network architecture consists of sampling latent vectors and feeding them to the decoder network. This has been illustrated with the help of a diagram in Figure 5.1. We use the reparameterization trick to sample from a fixed distribution and carry out a variable transformation using the learnt mean ($\boldsymbol{\mu}$) and standard deviation ($\boldsymbol{\sigma}$).

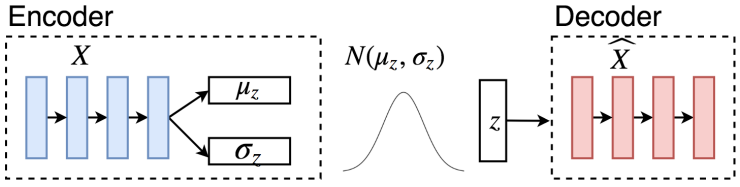

Figure 5.1: Sequence-to-sequence VAE Architechture

In regular sequence-to-sequence (Sutskever et al., 2014) models, it is common to use hidden state initialization to transfer source information to the target side. In LSTM-RNNs, this is done by setting the initial state of the decoder to the encoder's final hidden and cell states. In the Seq2Seq literature, the hidden state initialization technique has been extended to train VAEs (Serban et al., 2017; Cao and Clark, 2017). This architecture has been illustrated in Figure 5.2. However, based on our experiments with VAEs, we argue that such hidden state initializations results in *bypassing* phenomenon.

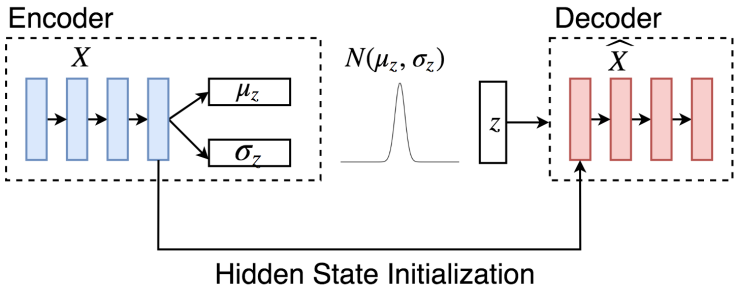

Figure 5.2: Sequence-to-sequence VAE with bypass

We observe that, if the decoder has a direct, deterministic access to the source information, the latent variables Z might not capture much information, which causes the variational space to not play a role in the training process. We call this a *bypassing phenomenon*. This renders the learnt latent space to be inadequate without exhibiting any variational properties such as linear interpolation and sampling from neighbourhood. The distribution of $\boldsymbol{z}$ corresponding to a given input tends to be more peaked with the bypass connection (see Figure 5.2). In other words, the standard deviation around the mean is very small. As a result, sampling points around the mean, we can generate only the same or very similar sentences. In contrast, VAEs without the bypass connection can generate more diverse sentences that pertain to the same topic as the input. The latent variable distribution is flatter as depicted in Figure 5.1.

This can also be explained theoretically. Assume that $\hat{X}$ is the reconstruction that needs to be generated using the latent variable Z. We can denote the decoder as $p_\theta(\hat{X}|Z)$. If the decoder is provided with a bypass connection to the source X, it can now be written as $p_\theta(\hat{X}|X, Z)$. Since the latent space is much harder to learn, the decoder in this case can choose to completely ignore Z. It can learn to reconstruct the input just using the information from X, i.e., $p_\theta(\hat{X}|X)$. In this case, the reconstruction loss from Equation 4.12 can be minimized without the effect of the KL term. In other words, the KL term fails to act as a regularizer as it can be minimized independently by fitting the posterior to

its prior. This will result in the model learning a meaningless latent space that does not encode any useful source information. Hence, a bypass connection degrades a variational Seq2Seq model to a deterministic one. We prove this empirically in Section 5.3.

5.2 Evaluation Metrics

In a variational model, we learn a continuous latent space. We had qualitatively evaluated the latent space by random sampling and linear interpolation of sentences in Chapter 4. When provided with an input, we expect a VAE to generate sentences that are similar to the input sentence but not necessarily the exact same sentence as output. Because we sample points from around the mean vector corresponding to the input, this can result in some variability in the output. As a result, the generated sentences may be diverse, although they speak about the same topic. Here, we introduce two quantitative metrics to assess the diversity of sentences generated.

5.2.1 Entropy

Assume that for a given input $\boldsymbol{x}$, we generate k outputs $\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_k$ by sampling a new latent variable each time. We can consider this set of k output sentences as our corpus and compute the unigram probability of each token. Note that the unigram probability of token w is calculated by normalizing the count of that particular token by the total number of tokens in the corpus. We can compute the entropy of this unigram probability distribution as

$$H = -\sum_{w} p(w) \log p(w) \tag{5.1}$$

A higher value of entropy corresponds to more randomness in the system. In our case, the VAE that produces more diverse sentences will have a higher entropy.

5.2.2 Distinct Scores

The *distinct* metrics were introduced by Li et al. (2015a) for evaluating the diversity of responses provided by neural conversational agents. Similar to the case of entropy

calculation, we can generate k sentences for a given input. To compute Distinct-1 and Distinct-2 scores for this set of k sentences, we can use the following equations.

$$\text{Distinct-1} = \frac{\text{Count of distinct unigrams}}{\text{Total unigram count}} \tag{5.2}$$

$$\text{Distinct-2} = \frac{\text{Count of distinct bigrams}}{\text{Total bigram count}} \tag{5.3}$$

The higher the distinct score, the more diverse the output sentences will be.

5.3 Results

We make use of the best performing VAE in Section 4.7, namely the ADAM-tanh-3000 variant for these experiments. Specifically, we train the exact same model with and without a hidden state initialization of the decoder, which we consider a bypass connection. We observe very different learning curves for the two cases. When the model is trained with a bypass connection, the KL term of the loss function gradually vanishes to zero (refer Figure 5.3). In contrast, when there is no bypass connection, the VAE can be trained in a more stable manner.

Due to the differences observed during training, we expect both models to perform differently. We employ the same technique as in Section 4.7.1 and sample points within 1 standard deviation around the mean. In this manner, we generate sentences from the latent space conditioned on a given input. We obtain diverse and topically related sentences for the VAE with no hidden state initialization. However, the bypass connection in the other model degrades it to a deterministic autoencoder. The latent space is not continuous or smooth. This can be seen from the generated sentences in Table 5.1, which have little or no diversity.

We also verify the qualitative findings with the automatic evaluation metrics described in Section 5.2. For each input sentence in the test set, we sample 10 points in the latent space and generate 10 corresponding sentences. We then compute the entropy of each set of 10 sentences using its unigram probability distribution. The average entropy of the outputs generated by the VAE with bypass is much lower than the same model without bypass connection. Note that even a decrease in entropy of 0.68 corresponds to a relatively large difference in diversity because entropy is computed using the log-scale. The same trend is observed for the other diversity metrics, namely Distinct-1 and Distinct-2. The number of distinct unigrams and bigrams produced by the VAE without hidden state initialization is much higher than the VAE with bypass.

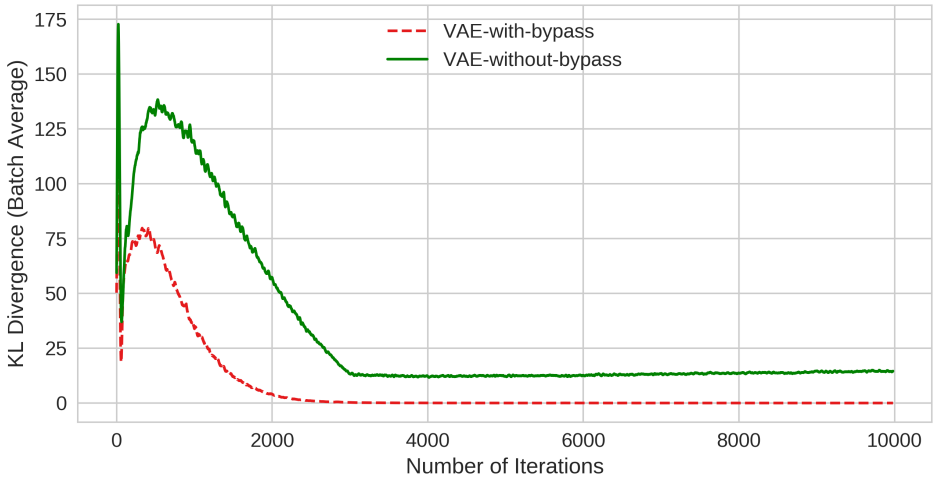

Figure 5.3: Comparison of learning curves of VAE with and without bypass

VAE with Bypass	**VAE without Bypass**
Input Sentence: the men are playing musical instruments	
the men are playing musical instruments	*the men are playing video games*
the man is playing musical instruments	*the men are playing musical instruments*
the men are playing musical instruments	*the musicians are playing musical instruments*
Input Sentence: a child holds a shovel on the beach .	
a child holds a shovel on the beach .	*a child playing with the ball on the beach .*
a child holds a shovel on the beach .	*a child holding a toy on the water .*
a child holds a shovel on the beach .	*a child holding a toy on the beach .*
Input Sentence: a group of professional football players having a game	
a group of professional football players having a game	*a group of football players celebrate a game*
a group of professional football players having a game	*a group of men watching a soccer match*
a group of professional football players having a game	*a group of football players having a good time*

Table 5.1: Generating sentences conditioned on a given input by sampling from the latent space - comparison of VAE with and without bypass connection

This observation regarding the bypassing phenomenon sheds light on the design philosophy of variational neural models. This motivates us to further explore possibilities of such bypassing connections that limits the performance of variational encoder-decoder models.

	VAE with Bypass	VAE without Bypass
Entropy	2.004	2.686
Distinct-1	0.099	0.302
Distinct-2	0.118	0.502

Table 5.2: Comparison of VAE with and without bypass connection in terms of automatic diversity metrics

We discuss this in the upcoming chapters along with ways to address the issue.

Chapter 6

Variational Attention for Seq2Seq Models

6.1 Motivation

In the previous chapters, the process of autoencoding was discussed. Given an input sequence of words, the task was to learn an intermediate representation, from which it is possible to reconstruct the input. However, in most real life applications we would need to transform a given input sequence into a different output sequence. An example is the task of machine translation, where the input sentence may be in French and we would like to obtain the corresponding translated output in English. Another example is that of a conversational system, where chatbots learn to respond to user inputs.

In such scenarios, we would need to implement encoder-decoder models (rather than autoencoders). In deep learning, recurrent neural network based sequence-to-sequence models (Sutskever et al., 2014) are the most popular models used for text generation. They have been extensively used for machine translation (Bahdanau et al., 2014), dialog systems (Vinyals and Le, 2015), text summarization (Rush et al., 2015) and so on. In most cases, these models tend to be deterministic, i.e., trained to simply maximize the log-likelihood of the data.

Variational Seq2Seq models have also been applied for encoding-decoding purposes. Serban et al. (2017) report that when dialog systems are trained with variational neural models, the output responses tend to be longer and more diverse. Zhang et al. (2016) show that by using variational encoder-decoder (VED) models for neural machine translation, they achieve higher BLEU scores than existing deterministic Seq2Seq baselines.

The introduction of attention mechanisms (Bahdanau et al., 2014; Luong et al., 2015) resulted in major performance improvements to existing Seq2Seq models. Attention mechanisms essentially align source information on the encoder side to target information on the decoder side. Due to this, the decoding process becomes more accurate by appropriately weighting the source information. Attention mechanisms in Seq2Seq models are described in detail in Section 6.3.

However, we argue that traditional attention mechanism cannot be directly applied to variational encoder-decoder models. Doing so unfortunately leads to bypassing phenomenon, similar to the one described in Chapter 5. In this case, deterministic attention provides direct access to the source information. This may cause the latent space to be ignored during training, resulting in an ineffective variational model. In this chapter we investigate the effect of such a bypassing connection caused by deterministic attention mechanism. We propose an alternative attention mechanism, which we refer to as variational attention that can circumvent this bypassing issue in variational encoder decoder models.

First, the traditional attention mechanism is introduced and then we discuss how this can be transformed to variational attention. Next, we report empirical results on two experiments - question generation and dialog systems. We prove the advantage of our proposed method by showing that it alleviates the bypassing phenomenon and increases the diversity of generated sentences while maintaining language fluency.

6.2 Variational Encoder Decoder

Encoder-decoder models are used when we wish to transform source information X into target information Y. To make the model variational, we need to learn a latent variable Z whose distribution (referred to as the posterior) is close to a pre-specified prior. In the literature, different approaches have been proposed to learn the latent space in VEDs. Zhang et al. (2016) and Cao and Clark (2017) use both X and Y as input variables to encode information into Z. In this case, $\boldsymbol{z}$ is sampled from a posterior distribution given by $q_{\boldsymbol{\phi}}(Z|X,Y)$, the encoder neural network. However, this causes a discrepancy between training and prediction because Y is not available during the prediction stage. Hence, we follow the approach mentioned in Zhou and Neubig (2017). Here, the assumption is that Y can be considered as a function of X, i.e., $Y = Y(X)$ and as a result, we have $q_{\boldsymbol{\phi}}(Z|X,Y) \triangleq q_{\boldsymbol{\phi}}(Z|X,Y(X)) \triangleq q_{\boldsymbol{\phi}}(Z|X)$. Hence, we can simply use an encoder that requires only X as input to learn the latent variable Z. This is depicted in Figure 6.1.

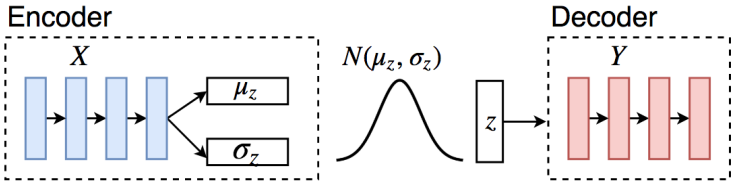

Figure 6.1: Sequence-to-sequence VED Architecture

6.3 Attention Mechanism

In deep learning, attention mechanism refers to the techniques by which neural networks are trained to focus on specific aspects of the input in order to accomplish the task at hand. In this section, the attention mechanisms popularly used in Seq2Seq models are explained. Note that this is different from visual attention in computer vision tasks such as image captioning (Xu et al., 2015a) and object detection (Borji et al., 2014), which will not be covered in this thesis.

Consider RNNs with LSTM units as both encoder and decoder. Following the same notation as in Chapter 3, let $\boldsymbol{x} = (x_1, x_2, \cdots, x_{|\boldsymbol{x}|})$ be the tokens from the source sequence and $\boldsymbol{y} = (y_1, y_2, \cdots, y_{|\boldsymbol{y}|})$ be the tokens from the target sequence. At each timestep j of the decoding process, we predict the target token by computing a softmax across the words in the vocabulary V to obtain probabilities as follows

$$p(y_j) = \text{softmax}(W_{\text{out}} \boldsymbol{h}_j^{(\text{tar})}) \tag{6.1}$$

where, W_{out} is a weight matrix, and $\boldsymbol{h}_j^{(\text{tar})}$ is the decoder LSTM output at timestep t.

$$\boldsymbol{h}_j^{(\text{tar})} = \text{LSTM}_{\boldsymbol{\theta}}(\boldsymbol{h}_{j-1}^{(\text{tar})}, \boldsymbol{y}_{j-1}, \boldsymbol{z}) \tag{6.2}$$

where $\boldsymbol{\theta}$ refers to the weights of the LSTM network, $\boldsymbol{y}_{j-1}$ is the decoder input word embedding (of the output word at the previous timestep) and $\boldsymbol{z}$ is the sampled latent representation of the input sentence $\boldsymbol{x}$

In Deterministic Encoder-Decoder (DED), some source information is passed on to the decoder via hidden state initialization (see Section 5.1). In contrast, it is to be noted that the decoder LSTM in the vanilla Seq2Seq VED does not have access to source information other than through the latent vector $\boldsymbol{z}$. Attention mechanism can serve as a way to learn additional source information in encoder-decoder models. Essentially, during each timestep j of the decoding process, the decoder is provided access to all of the encoded source tokens.

More concretely, the decoder at timestep j is provided with the encoder LSTM outputs, i.e., $\boldsymbol{h}_i^{(\text{src})}$ where $i \in \{1, 2, \cdots, |\boldsymbol{x}|\}$. Now, instead of giving equal weight to every $\boldsymbol{h}_i^{(\text{src})}$, depending on the target word to be decoded, the decoder LSTM can give higher weight to certain $\boldsymbol{h}_i^{(\text{src})}$ and lower weight to others. This means that the source outputs can be weighted differently at each decoding timestep. The idea behind this is that all source words need not contribute equally while generating the current target word.

Mathematically, attention mechanisms compute a probabilistic distribution (across the source tokens) at each decoding timestep j given by

$$\alpha_{ji} = \frac{\exp\{\widetilde{\alpha}_{ji}\}}{\sum_{i'=1}^{|\boldsymbol{x}|} \exp\{\widetilde{\alpha}_{ji'}\}} \tag{6.3}$$

where $\alpha_{ji} \in [0, 1]$ is the weight given to source output i and $\widetilde{\alpha}_{ji}$ is a pre-normalized score. In the literature of Seq2Seq models, two methods have been proposed to calculate $\widetilde{\alpha}_{ji}$.

1. **Multiplicative** (Luong et al., 2015)

$$\widetilde{\alpha}_{ji} = \boldsymbol{h}_j^{(\text{tar})} W^T \boldsymbol{h}_i^{(\text{src})} \tag{6.4}$$

2. **Additive** (Bahdanau et al., 2014)

$$\widetilde{\alpha}_{ji} = v_a^T \tanh\left(W_1 \boldsymbol{h}_j^{(\text{tar})} + W_2 \boldsymbol{h}_i^{(\text{src})}\right) \tag{6.5}$$

   where, W, W_1, W_2 and v_a^T are weights that are learnt via error backpropagation.

While both methods work equally well in practice, this thesis uses the multiplicative style attention for all the encoder-decoder experiments. The next step is to take the sum of the source outputs $\{\boldsymbol{h}_i^{(\text{src})}\}_{i=1}^{|\boldsymbol{x}|}$ weighted by α_{ji} to obtain the context vector.

$$\boldsymbol{c}_j = \sum_{i=1}^{|\boldsymbol{x}|} \alpha_{ji} \boldsymbol{h}_i^{(\text{src})} \tag{6.6}$$

Finally the attention vector can be computed using a tanh non-linear operation and learnt weights W_c as follows

$$\boldsymbol{a}_j = f(\boldsymbol{c}_j, \boldsymbol{h}_j^{(\text{tar})}) = \tanh(W_c[\boldsymbol{c}_j; \boldsymbol{h}_j^{(\text{tar})}]) \tag{6.7}$$

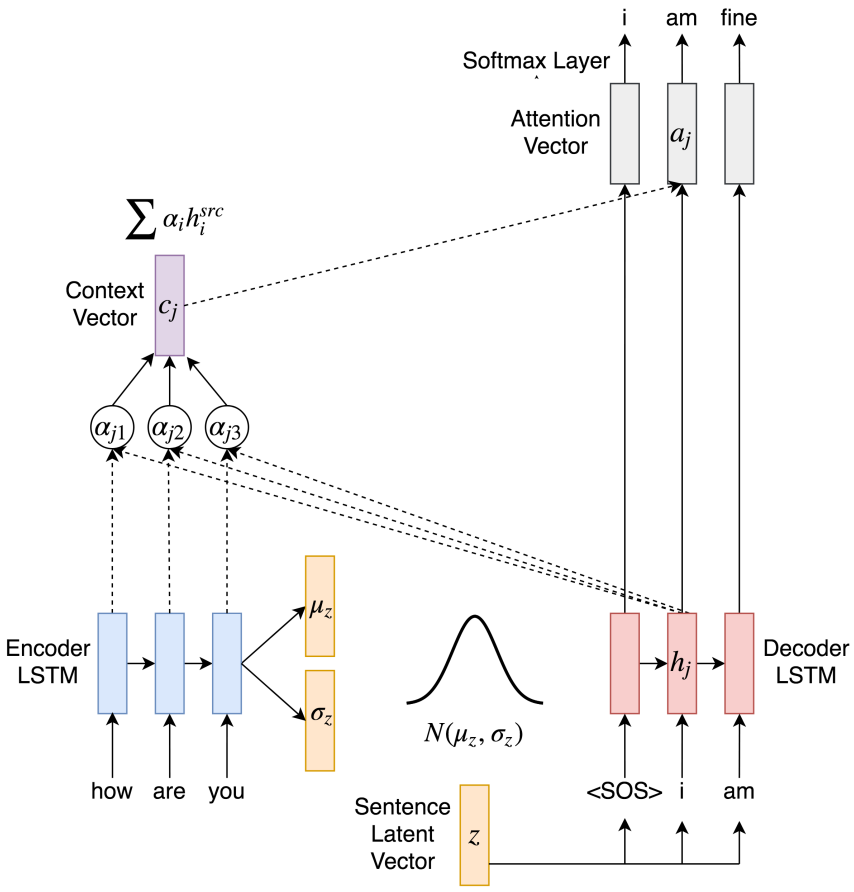

Figure 6.2: Illustration of VED with Deterministic Attention Mechanism

This attention vector can now be fed to the softmax layer at timestep j. Rewriting Equation 6.1, we obtain

$$p(y_j) = \text{softmax}(W_{\text{out}}\boldsymbol{a}_j) \tag{6.8}$$

In this manner, attention mechanisms are capable of dynamically aligning target information to the source information, during the generation process. This process has been illustrated in Figure 6.2 and we refer to this as deterministic attention.

6.4 Variational Attention

One can observe from Figure 6.2, that the decoder LSTM has direct access to source information via the context vector $\boldsymbol{c}_j$ during attention computation. Although the latent vector $\boldsymbol{z}$ is fed to the decoder at every timestep, we fear that availability of source information via $\boldsymbol{c}_j$ may cause the decoder LSTM to ignore $\boldsymbol{z}$. This may result in bypassing phenomenon due to which the VED model does not encode any useful information into the $\boldsymbol{z}$ latent space.

In this work, we propose variational attention mechanism to prevent bypassing. Specifically, the context vector is modelled as a Gaussian random variable, which can be sampled using its mean and standard deviation. This results in a stochastic node before the softmax layer as shown in Figure 6.3. That is, we now have two nodes in the computational graph at which we do sampling, one for the sentence latent representation $\boldsymbol{z}$ and the other latent space for the context vector $\boldsymbol{c}_j$ at every timestep j. The attention is not modelled in a deterministic fashion anymore. In this way, we can prevent the bypassing issue in Seq2Seq VED models with attention.

6.4.1 Derivation of Loss Function

Variational attention treats both the sentence representation $\boldsymbol{z}$ and the context vector $\boldsymbol{c}_j$ as random variables. We would need to regularize the distribution of both these latent variables in the loss function. Analogous to the case of VAE, we rewrite Equation 4.11 for evidence lower bound (ELBO) by adding the extra variable $\boldsymbol{c}_j$. The ELBO for the jth timestep of the nth data point, i.e., input $x^{(n)}$ and target $y^{(n)}$, can be expressed as follows.

$$\text{ELBO}_j^{(n)}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{z,c_j \sim q_{\boldsymbol{\phi}}(z,c_j|x^{(n)})} \left[\log p_{\boldsymbol{\theta}}(y^{(n)}|z, c_j)\right] - \text{KL}\left(q_{\boldsymbol{\phi}}(z, c_j|x^{(n)}) \| p(z, c_j)\right) \tag{6.9}$$

where $q_{\boldsymbol{\phi}}$ and $p_{\boldsymbol{\theta}}$ correspond to the encoder and decoder neural networks with respective weights $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. Given $\boldsymbol{x}$, we can assume conditional independence between $\boldsymbol{z}$ and $\boldsymbol{c}_j$. Hence, the posterior factorizes as $q_\phi(\boldsymbol{z}, \boldsymbol{c}_j|\cdot) = q_\phi^{(z)}(\boldsymbol{z}|\cdot)\, q_\phi^{(c_j)}(\boldsymbol{c}_j|\cdot)$. We also assume separate priors for $\boldsymbol{z}$ and $\boldsymbol{c}_j$. In this way, the sampling procedure can be done separately and the KL loss can also be computed independently.

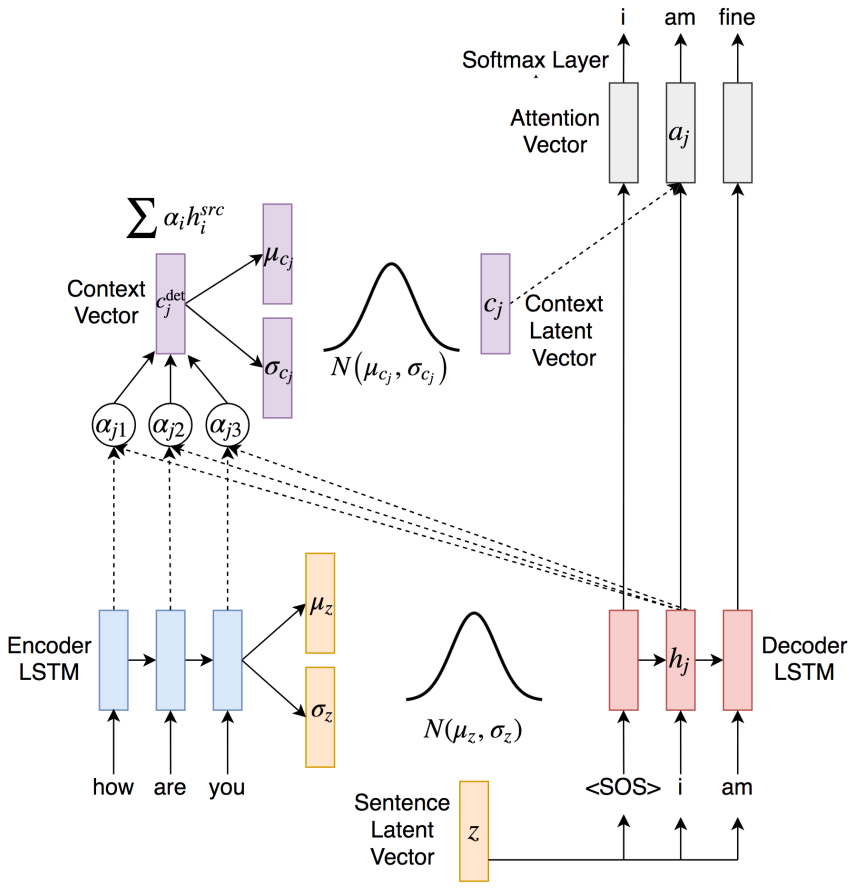

Figure 6.3: Illustration of VED with Variational Attention Mechanism

$$\begin{aligned}\text{ELBO}_j^{(n)}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{z \sim q_{\boldsymbol{\phi}}^{(z)}(z|x^{(n)}), c_j \sim q_{\boldsymbol{\phi}}^{(c_j)}(c_j|x^{(n)})} \left[\log p_{\boldsymbol{\theta}}(y^{(n)}|z, c_j)\right] \\ - \text{KL}\left(q_{\boldsymbol{\phi}}^{(z)}(z|x^{(n)}) \| p(z)\right) - \text{KL}\left(q_{\boldsymbol{\phi}}^{(c_j)}(c_j|x^{(n)}) \| p(c_j)\right)\end{aligned} \tag{6.10}$$

As mentioned in Chapter 4, we are required to maximize the evidence lower bound, which is equivalent to minimizing $-\text{ELBO}_j^{(n)}(\boldsymbol{\theta}, \boldsymbol{\phi})$. The first term then becomes the negative log-likelihood, which can be computed as the standard Seq2Seq word prediction categorical cross-entropy loss, summed across all timesteps ($J_{\text{rec}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{y}^{(n)})$). Next, we have two KL regularization terms - one for the sentence latent space $\boldsymbol{z}$ and the other for the context vector at each timestep $\boldsymbol{c}_j$. In both cases, we are required to minimize the KL divergence

between the computed posterior and the pre-specified prior. Hence, the overall training objective of the Seq2Seq VED with both variational sentence latent space $\boldsymbol{z}$ and variational attention $\boldsymbol{c}$ is to minimize the following loss function,

$$J^{(n)}(\boldsymbol{\theta}, \boldsymbol{\phi}) = J_{\text{rec}}(\boldsymbol{\theta}, \boldsymbol{\phi}, y^{(n)}) + \lambda_{\text{KL}} \Big[\text{KL} \left(q_{\phi}^{(z)}(z|x^{(n)}) \| p(z) \right) + \gamma_a \sum_{j=1}^{|y|} \text{KL} \left(q_{\phi}^{(c_j)}(c_j|x^{(n)}) \| p(c_j) \right) \Big] \tag{6.11}$$

In the above equation, λ_{KL} acts as coefficient of both the KL terms. We only anneal this hyperparameter in a manner similar to that of the VAE as discussed in Section 4.5. γ_a is the coefficient of the context vector's KL term which is kept constant throughout training. This is done because training Seq2Seq variational models can be difficult and we would like to anneal just one of the coefficients which can simultaneously influence both KL terms.

Prior

Similar to the VAE described in Chapter 4, the sentence latent code $\boldsymbol{z}$ is assumed to have a standard normal prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$. For the context vector $\boldsymbol{c}_j$, we propose two plausible priors:

1. The simplest option would be to set the prior $p(\boldsymbol{c}_j)$ to be $\mathcal{N}(\mathbf{0}, \mathbf{I})$.
2. With an understanding of attention mechanism in Seq2Seq models, one can observe that $\boldsymbol{c}_j$ is calculated as a linear combination of the source hidden states, i.e., $\boldsymbol{c}_j = \sum_{i=1}^{|\boldsymbol{x}|} \alpha_{ji} \boldsymbol{h}_i^{(\text{src})}$. Geometrically, this means that the context vector is inside the convex hull of hidden representations of the source sequence, i.e., $\boldsymbol{c}_j \in \text{conv}\{\boldsymbol{h}_i^{(\text{src})}\}$. Thus, we choose $p(\boldsymbol{c}_j) = \mathcal{N}(\bar{\boldsymbol{h}}^{(\text{src})}, \mathbf{I})$ as an alternative choice for the prior, where $\bar{\boldsymbol{h}}^{(\text{src})} = \frac{1}{|\boldsymbol{x}|} \sum_{i=1}^{|\boldsymbol{x}|} \boldsymbol{h}_i^{(\text{src})}$, the mean of the source hidden states.

Posterior

Once we have pre-specified the prior distributions of both latent variables $\boldsymbol{z}$ and $\boldsymbol{c}_j$, we now discuss how the corresponding approximate posteriors are computed. In both cases, an LSTM based encoder neural network is used to parameterize the posterior distributions. More concretely, for the sentence latent space, the encoder's final hidden state $\boldsymbol{h}_t^{(\text{src})}$ is linearly transformed to obtain $\boldsymbol{\mu}_z$ and $\boldsymbol{\sigma}_z$, the parameters of the posterior Gaussian distribution. Now the posterior of $\boldsymbol{z}$ can be defined as $q_{\phi}^{(z)}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z)$.

In an analogous manner, the posterior of the context vector at timestep j, denoted by $q_{\boldsymbol{\phi}}^{(c_j)}(\boldsymbol{c}_j|\boldsymbol{x})$ is modeled as a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_{c_j}, \boldsymbol{\sigma}_{c_j})$ using the same encoder neural network. We first compute the context vector in a deterministic manner ($\boldsymbol{c}_j^{\text{det}}$) as mentioned in Equation 6.6 and then transform it into the variational space using learnt parameters $\boldsymbol{\mu}_{c_j}$ and $\boldsymbol{\sigma}_{c_j}$. For the mean $\boldsymbol{\mu}_{c_j}$, we apply an identity transformation, i.e., $\boldsymbol{\mu}_{c_j} \equiv \boldsymbol{c}_j^{\text{det}}$. To compute $\boldsymbol{\sigma}_{c_j}$, we first transform $\boldsymbol{c}_j^{\text{det}}$ by a neural layer with tanh activation followed by another linear transformation. We can then sample the context vector $\boldsymbol{c}_j$ from its posterior $\mathcal{N}(\boldsymbol{\mu}_{c_j}, \boldsymbol{\sigma}_{c_j})$. Readers are referred to Figure 6.3 for a diagrammatic overview of the method in which the parameters of the posterior distributions are computed.

6.5 Experiments

In this section, we provide the VED training details, the datasets used and finally report the qualitative and quantitative results.

6.5.1 Datasets

We evaluate and compare the proposed variational attention on two tasks - (1) Question Generation, (2) Dialog Systems. In question generation, the task is as follows - given an input sentence or paragraph, generate a question relevant to the input. Although this may seem trivial for humans, it is more difficult for computers since the generated questions need to be semantically and syntactically correct, and maintain topical relevance to the input. Apart from these, we expect to obtain diverse but relevant questions by sampling different points from the latent space, conditioned on the same input. According to Du et al. (2017), attention mechanisms are especially critical in this task in order to generate relevant questions. We use the Stanford Question Answering Dataset (Rajpurkar et al., 2016, SQuAD) to carry out this task. The SQuAD dataset was originally curated for the task of machine comprehension, i.e., given a paragraph and a set of relevant questions, the computer is required to pick the sentences from the paragraph that are answers to the questions. The dataset has about 100k question-answer pairs, and we follow the same train-validation-test split as in Du et al. (2017).

The second task is that of developing a generative conversational agent. The goal of such a dialog system is to generate replies in response to user utterances. Here again, response fluency and relevance are critical, for which attention mechanisms can play an

important role. For this experiment, we use the Cornell Movie-Dialogs Corpus[1] (Danescu-Niculescu-Mizil and Lee, 2011) as our dataset, which contains more than 200k conversational exchanges. While 180k sentences were used for training the model, 10k was held out for validation, and the rest for reporting test set performance.

6.5.2 Training Details

We used RNNs with 100d LSTM units for both the encoder and decoder; the dimension of the latent vector $\boldsymbol{z}$ was also 100d. Pre-trained `word2vec` (Mikolov et al., 2013) word embeddings of 300d were used. For the question generation experiment, the vocabulary was set to the most frequent 40k words. The dataset in the dialog system experiment had fewer distinct unigrams and hence we set the model vocabulary size to 30k words. ADAM optimizer was used to train all the models. The convergence of the validation loss was set to be the stopping criterion. The question generation model is trained for 20 epochs. Dialog systems typically take more time for convergence and are trained for 250 epochs. The rest of the hyperparameters including the annealing schedule, word dropout rate, learning rate, etc., are set to those of the best performing VAE model described in Chapter 4, namely ADAM-tanh-3000.

Evaluation Metrics. To determine the sentence generation performance, we report BLEU scores on the test set. Diversity of generated sentences is an important aspect in our study and this is evaluated using the automatic metrics described in Chapter 5, i.e., entropy and *distinct* scores.

6.5.3 Quantitative Evaluation

For the question generation experiment, we first replicate the neural network architecture proposed by Du et al. (2017) for this task. As shown in Table 6.1, we obtain similar BLEU scores as those of the deterministic encoder-decoder (DED) described in Du et al. (2017). Incorporating deterministic attention to this vanilla Seq2Seq DED, we obtain the model referred to as DED+DAttn. As expected, the attention mechanism improves the BLEU scores in comparison to the regular DED model.

The next set of results correspond to the variational encoder-decoder models (VED). Given an input, the output sentence generation at inference time can be done either by sampling from the latent space or by using the max *a posteriori* estimate (MAP). This

[1] https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html

Model	Inference	BLEU-1	BLEU-2	BLEU-3	BLEU-4	Entropy	Dist-1	Dist-2
DED (w/o Attn) Du et al. (2017)	MAP	31.34	13.79	7.36	4.26	-	-	-
DED (w/o Attn)	MAP	29.31	12.42	6.55	3.61	-	-	-
DED+DAttn	MAP	30.24	14.33	8.26	4.96	-	-	-
VED+DAttn	MAP	**31.02**	14.57	8.49	5.02	-	-	-
	Sampling	30.87	**14.71**	**8.61**	**5.08**	2.214	0.132	0.176
VED+DAttn (2-stage training)	MAP	28.88	13.02	7.33	4.16	-	-	-
	Sampling	29.25	13.21	7.45	4.25	2.241	0.140	0.188
VED+VAttn-0	MAP	29.70	14.17	8.21	4.92	-	-	-
	Sampling	30.22	14.22	8.28	4.87	**2.320**	**0.165**	**0.231**
VED+VAttn-$\bar{h}$	MAP	30.23	14.30	8.28	4.93	-	-	-
	Sampling	30.47	14.35	8.39	4.96	2.316	0.162	0.228

Table 6.1: BLEU, entropy, and distinct scores on the question generation task. We compare the deterministic encoder-decoder (DED) and variational encoder-decoders (VEDs). For VED, we have several variates: deterministic attention (DAttn) and the proposed variational attention (VAttn). Variational models are evaluated by both max *a posteriori* (MAP) inference and sampling.

refers to the most probable output corresponding to the given input, which in the Gaussian case is the mean of the posterior distribution. In the *sampling* setting, we draw 10 samples ($\boldsymbol{z}$ and/or $\boldsymbol{a}$) from the posterior given $\boldsymbol{x}$, i.e., for each data point, and report average BLEU scores. We also compute the diversity metrics on each set of 10 sampled sentences and average it across the test data. Note that these metrics can only be calculated for the *sampling* setting.

Comparing the proposed variational attention (VED+VAttn) and the deterministic attention (VED+DAttn) models, we observe that VED+VAttn significantly outperforms VED+DAttn in terms of the diversity of generated sentences. It should be noted that entropy is a logarithmic measure, and hence the difference of 0.1 in Table 6.1 is significant; VED+VAttn also generates more distinct unigrams and bigrams than VED+DAttn. The BLEU scores obtained by the variational attention models are only slightly lower than the deterministic attention counterparts. This means that the output sentence reconstruction performance of both models is similar, while the VED+VAttn models can generate more diverse outputs. This confirms our hypothesis that deterministic attention serves as a bypassing connection that affects the learning of a good latent space. The lower diversity of DAttn models can be attributed to the learnt posterior distributions being more peaked, similar to the issue depicted in Figure 5.2.

Table 6.1 reports results for the two priors that were proposed in Section 6.4.1.

VED+VAttn-0 and VED+VAttn-$\bar{h}$ refer to the variational attention models with priors $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathcal{N}(\bar{\boldsymbol{h}}^{(\text{src})}, \mathbf{I})$ respectively. VED+VAttn-0 has slightly lower BLEU but higher diversity. The results are generally comparable, showing both priors are reasonable.

The model indicated by VED+DAttn (2-stage) refers to a heuristic based model which was implemented for the same task. In this setting, the VED is first trained without attention for 6 epochs, and then the attention mechanism is incorporated into the model for the rest of the training. The logic behind this simple heuristic is that we first allow the VED to learn a meaningful latent space during the early stages of training, and we only introduce the deterministic attention connection at a later stage. However, this still continues to influence the model performance as a result of bypassing. We conclude that such simple heuristics do not help much, and are worse than the principled variational attention mechanism in terms of all BLEU and diversity metrics.

Next we present the results on the dialog systems experiment. In general, automatic conversational systems tend to have low BLEU scores because for a given user utterance, there are multiple possible responses which are valid (Liu et al., 2016). The dataset typically only provides only one ground truth response. Hence, it is much harder to evaluate dialog systems. Nevertheless, we report BLEU scores and diversity metrics on the three main models (refer Table 6.2). The deterministic encoder-decoder model (DED) tends to have a better output reconstruction capability. This is expected since it is explicitly trained to minimize the negative log likelihood of observing the data. On the other hand, VED models have additional KL regularization terms in their loss functions. VED+VAttn-0 and VED+VAttn-$\bar{h}$ provide similar results and only the model with $\mathcal{N}(\bar{\boldsymbol{h}}^{(\text{src})}, \mathbf{I})$ prior is chosen for reporting purposes.

It can be observed that both the quality and diversity of sentences generated by VED+VAttn-$\bar{h}$ are slightly better than VED+DAttn. However, we find the improvement is not so large as in the question generation task. We conjecture the reason for this to be that in conversational systems, there is a weaker alignment between the source and target information. As a result, the attention mechanism itself is less effective.

Learning Curves

For the question generation experiment, we also studied the learning curves of the evaluation metrics (on the validation set) as training progresses. In Figure 6.4, we illustrate BLEU-2 and BLEU-4 (representative of output reconstruction quality), and entropy and Distinct-1 (representative of diversity). It can be observed that BLEU scores and diversity are conflicting objectives. In order to attain a higher BLEU score, the model needs to be

Model	Inference	BLEU-1	BLEU-2	BLEU-3	BLEU-4	Entropy	Distinct-1	Distinct-2
DED+DAttn	MAP	5.75	1.84	0.99	0.64	-	-	-
VED+DAttn	MAP	5.33	1.68	0.88	0.57	-	-	-
	Sampling	5.34	1.68	0.89	0.57	2.113	0.311	0.450
VED+VAttn-$\bar{h}$	MAP	5.48	1.78	0.97	0.64	-	-	-
	Sampling	5.55	1.79	0.97	0.64	2.167	0.324	0.467

Table 6.2: Results on the conversational systems experiment

more *deterministic* in nature, as a result of which it may lose interesting *variational* properties such as diversity. The red and green lines indicate the variational attention models, which tend to have comparable BLEU scores while maintaining high diversity. This verifies the effectiveness of our model design which circumvents the bypassing phenomenon.

Strength of Attention KL Loss

As mentioned in Equation 6.11, annealing is only done for the common KL coefficient, i.e., λ_{KL}, whereas the coefficient of the attention KL γ_a loss is fixed for each experiment. In this section, we study the influence of γ_a, which affects the strength of attention KL loss in variational attention models, specifically the VED+VAttn-$\bar{h}$ variant. This is done by running multiple experiments on the question generation task with the same model, the only difference being the value of γ_a in each run. The learning curves of the different runs are presented in Figure 6.5. It can be seen that with lower values of γ_a, the BLEU scores are higher, while the corresponding diversity metrics are lower. This is expected because a lower γ_a gives the model less incentive to optimize the attention's KL term, which then causes the model to behave more *deterministic*. On the other hand, high γ_a increases the diversity of the sentences generated by the model, at the cost of output reconstruction performance, i.e, lower BLEU scores. Based on this experiment, we chose a value of 0.1 for γ_a, as it yields a learning curve in the middle among the different γ_a values, being a good balance between quality and diversity. The results reported in Table 6.1 and Table 6.2 correspond to this setting.

6.5.4 Qualitative Evaluation

In this section, we compare the proposed variational attention to the deterministic attention counterpart in terms of qualitative samples. We generate multiple samples conditioned on the same input for both models. This was done on the task of question generation, where

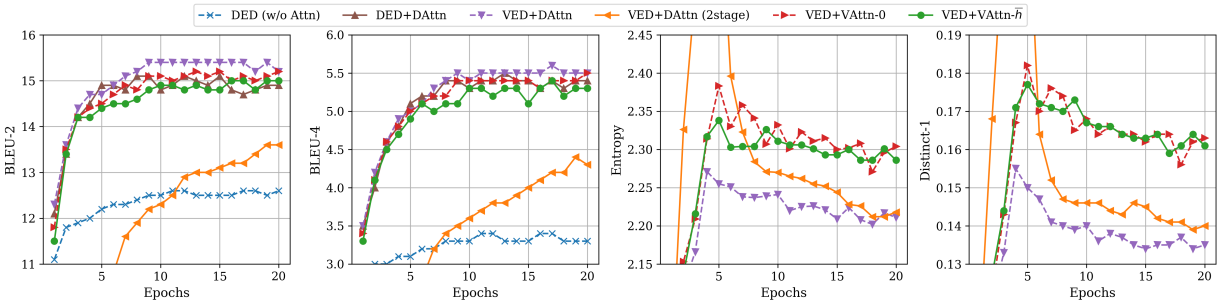

Figure 6.4: BLEU-2, BLEU-4, Entropy, and Distinct-1 calculated on the validation set as training progresses.

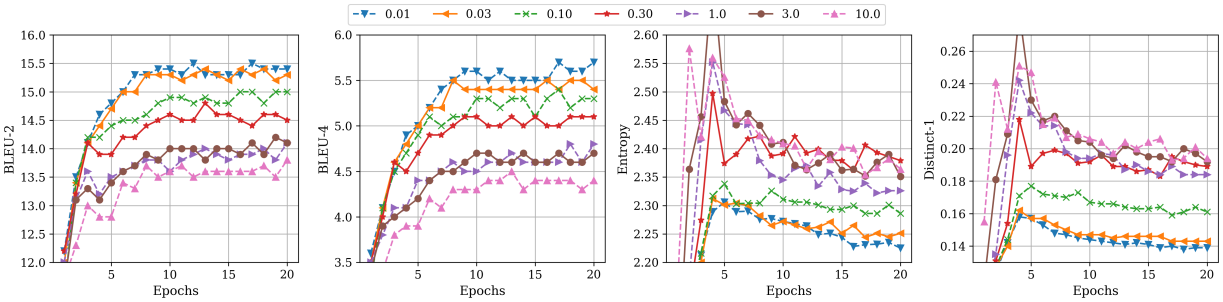

Figure 6.5: BLEU-2, BLEU-4, Entropy, and Distinct-1 for multiple runs of the same model (VED+VAttn-$\bar{h}$) with different γ_a values.

we input a sentence and attempt to generate a relevant question. It can be observed from Table 6.3 that VED+VAttn-$\bar{h}$ is capable of producing multiple diverse questions relevant to the input sentence. In contrast, the questions obtained by sampling from the latent spaces of VED+DAttn are less diverse. In many cases, the same sentence is generated multiple times, indicating that the posterior distributions corresponding to the inputs tend to be more peaked. This is caused by low standard deviations due to which the sampling always happens from just around the mean value. Thus, we are able to demonstrate the benefits of variational attention which is proposed as a method to alleviate the bypassing issue in Seq2Seq VED networks.

Human Evaluation

Although we quantitatively evaluate our model in the previous section, none of the metrics capture the language fluency of the model. It is difficult to come up with an automatic

Source	*when the british forces evacuated at the close of the war in 1783 , they transported 3,000 freedmen for resettlement in nova scotia .*
Reference	*in what year did the american revolutionary war end ?*
VED+DAttn	*how many people evacuated in newfoundland ?* *how many people evacuated in newfoundland ?* *what did the british forces seize in the war ?*
VED+Vattn-$\bar{h}$	*how many people lived in nova scotia ?* *where did the british forces retreat ?* *when did the british forces leave the war ?*
Source	*downstream , more than 200,000 people were evacuated from mianyang by june 1 in anticipation of the dam bursting .*
Reference	*how many people were evacuated downstream ?*
VED+DAttn	*how many people evacuated from the mianyang basin ?* *how many people evacuated from the mianyang basin ?* *how many people evacuated from the mianyang basin ?*
VED+VAttn-$\bar{h}$	*how many people evacuated from the tunnel ?* *how many people evacuated from the dam ?* *how many people were evacuated from fort in the dam ?*

Table 6.3: Qualitative samples of the question generation task.

evaluation metric that can compare the level of fluency of the text generated by each model. However, this is an important aspect considering the recent developments in artificial intelligence where machines are trained to be more *human-like*.

Thus, in order to assess the quality of the generated text in terms of language fluency, a human evaluation study was carried out with the text generated from the question generation task. For the two main models under comparison, VED+DAttn and VED+VAttn-$\bar{h}$, a randomly shuffled subset of 100 generated questions was selected. Six human evaluators were asked to rate the fluency of these 200 questions on a 5-point scale: 5-Flawless, 4-Good, 3-Adequate, 2-Poor, 1-Incomprehensible, following the annotation scheme described in (Stent et al., 2005). The human evaluators were fellow researchers proficient in English. No additional instructions apart from the 5-point annotation scheme were provided to the evaluators. Essentially, this gave them the freedom to decide what they thought to be a fluent question versus a poor question.

The average rating obtained for VED+DAttn was 3.99 and for VED+VAttn-$\bar{h}$ was 4.01, which shows that, on an average, the sentences generated by both models are of *good* fluency. It is to be further noted that the difference between the scores of VED+DAttn and VED+VAttn-$\bar{h}$ is not statistically significant (based on hypothesis test comparing two

means, at the 5% level). The human annotations achieved 0.61 average Spearman correlation coefficient (measuring order correlation) between any two annotators. According to Swinscow (1976), this indicates *moderate* to *strong* correlation among different annotators. Hence, we can safely conclude that variational attention does not negatively affect the fluency of sentences.

Chapter 7

Summary and Conclusions

7.1 Summary of Research Work

In this research, we present deep learning approaches to probabilistic natural language generation. In particular, we design and implement variational neural network models for text generation. Variational autoencoders are capable of mapping sentences into a continuous latent space from which it is possible to sample and generate new sentences. However, VAEs are notoriously difficult to train due to issues associated with the KL regularization term vanishing to zero, resulting in model collapse. The first part of the thesis addresses this problem with the help of various optimization strategies.

Next, we further explore VAE architectures and discover the *bypassing* phenomenon. Hidden state initialization of the decoder results in a bypassing connection which degrades the VAE into a deterministic model. We describe how this can negatively impact the learning of a meaningful latent space.

Finally, we move on to variational encoder-decoder models wherein we require to transform a given source sequence into a desired target sequence. We realize that traditional sequence-to-sequence attention mechanisms act as bypassing connections. To circumvent this problem, we propose the variational attention mechanism. We show that by treating the contect vector as a random variable, it is possible to overcome the *bypassing* issue.

7.2 Conclusions

Being able to generate meaningful textual data is an important characteristic of intelligent machines. For probabilistic generation of natural language sentences, we developed a sequence-to-sequence variational autoencoder. The VAE was successfully trained by implementing optimization heuristics, namely KL cost annealing and word dropout. By carefully engineering the annealing rate, schedule and threshold, the VAE that we trained was able to learn a meaningful continuous latent space. We demonstrate interesting properties of the latent space such as random sampling, linear interpolation and sampling from the neighbourhood, and compare it to a baseline deterministic autoencoder. We show that VAEs can learn meaningful sentence representations and also generate previously unseen sentences which are semantically and syntactically correct.

We studied the problem of bypassing phenomenon in VAEs wherein the decoder has a deterministic access to source information. We illustrate with the example of decoder hidden state initialization that such bypassing connections cause the VAE model to ignore the latent space during training. We show both quantitatively and qualitatively that bypassing results in the loss of interesting properties of the variational latent space and degrades the model into a deterministic autoencoder.

In variational encoder decoder models, we observe similar bypassing issues when traditional sequence to sequence attention is used. To prevent the decoder from having direct access to the encoder, we proposed a variational attention mechanism for VED frameworks. This technique introduces an additional stochastic node in the computational graph by modelling the context vector as a random variable with a pre-specified probability distribution. In practice, we sample the context vector at each timestep of the decoding process. An additional term, that computes the KL divergence between the context vector's posterior and prior distributions, is added to the loss function. Two plausible priors were proposed, which work equally well. With empirical results on two tasks - question generation and dialog systems, we show that variational attention yields more diversified samples while retaining high quality.

7.3 Future Work

For the VED model with variational attention, we impose a probabilistic distribution on the context vector. However, the context vector itself is computed using attention weights, α_{ji} (see Figure 6.3). Instead of the context vector $\boldsymbol{c}_j$ being modeled as a random variable, we

could assume the attention weights $\boldsymbol{\alpha}_j = \{\alpha_{ji}\}_{i=1}^{|\boldsymbol{x}|}$ as random variables which are sampled from a distribution. Note that α_j represent probability values that sum up to 1. Hence, they can be thought of as parameters of a categorical distribution which has Dirichlet distribution as its conjugate prior. To incorporate this scenario into the VED framework, we are required to sample from a Dirichlet distribution instead of sampling from a Gaussian distribution. However, this relies on a reparametrization trick to propagate error gradient back to the recognition neural network (refer Figure 4.5). In other words, we should be able to sample from a fixed distribution (i.e., without learnt parameters) and then obtain the variable in the desired latent space by doing necessary variable transformation. However, doing this is non-trivial for Dirichlet distribution. In future work, it would be interesting to investigate VEDs that model the attention weights with Dirichlet distributions.

Since the latent space learnt by VAEs is continuous and meaningful, future work can also explore how VAEs can be used for disentangling sentence level attributes. For example, if we are able to represent different attributes such as sentiment, tense, topic, etc., into different dimensions of the latent vector, this would be useful for supervised tasks later in the machine learning pipeline.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

Jaan Altosaar. Tutorial - what is a variational autoencoder? https://jaan.io/what-is-variational-autoencoder-vae-tutorial/, 2017. Accessed: 2018-06-12.

Ion Androutsopoulos and Prodromos Malakasiotis. A survey of paraphrasing and textual entailment methods. *Journal of Artificial Intelligence Research*, 38:135–187, 2010.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Hareesh Bahuleyan, Lili Mou, Kartik Vamaraju, Hao Zhou, and Olga Vechtomova. Probabilistic natural language generation with wasserstein autoencoders. *arXiv preprint arXiv:1806.08462*, 2018a.

Hareesh Bahuleyan, Lili Mou, Olga Vechtomova, and Pascal Poupart. Variational attention for sequence-to-sequence models. In *27th International Conference on Computational Linguistics (COLING)*, 2018b.

Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *International Conference on Machine Learning*, pages 226–234, 2014.

Steven Bird and Edward Loper. Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics, 2004.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Antoine Bordes, Y-Lan Boureau, and Jason Weston. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*, 2016.

Ali Borji, Dicky N Sihite, and Laurent Itti. What/where to look next? modeling top-down visual attention in complex interactive environments. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(5):523–538, 2014.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015a.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015b.

Denny Britz. Recurrent neural networks tutorial. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/, 2015. Accessed: 2018-05-25.

Kris Cao and Stephen Clark. Latent variable dialogue models and their diversity. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 182–187, 2017.

Yllias Chali and Sadid A Hasan. Towards topic-to-question generation. *Computational Linguistics*, 41(1):1–20, 2015.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

François Chollet et al. Keras, 2015.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3079–3087. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5949-semi-supervised-sequence-learning.pdf.

Cristian Danescu-Niculescu-Mizil and Lillian Lee. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*, pages 76–87, 2011.

Rahul Dave. Advance scientific computing (am207). https://am207.github.io/2017/, 2017. Accessed: 2018-06-12.

Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

Xinya Du, Junru Shao, and Claire Cardie. Learning to ask: Neural question generation for reading comprehension. *arXiv preprint arXiv:1705.00106*, 2017.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Yuchen Fan, Yao Qian, Feng-Long Xie, and Frank K Soong. Tts synthesis with bidirectional lstm based recurrent neural networks. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Kenneth R Foster, Robert Koprowski, and Joseph D Skufca. Machine learning, medical diagnosis, and biomedical engineering research-commentary. *Biomedical engineering online*, 13(1):94, 2014.

Elizabeth Gibney. Google ai algorithm masters ancient game of go. *Nature News*, 529 (7587):445, 2016.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

Kevin Gurney. *An introduction to neural networks*. CRC press, 2014.

Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Michael Heilman. *Automatic factual question generation from text*. PhD thesis, Carnegie Mellon University, 2011.

Michael Heilman and Noah A Smith. Question generation via overgenerating transformations and ranking. Technical report, Carnegie-Mellon University Language Technologies Institute, 2009.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

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

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

Alice Kerly, Phil Hall, and Susan Bull. Bringing chatbots into education: Towards natural language negotiation of open learner models. *Knowledge-Based Systems*, 20(2):177–185, 2007.

Diederik P Kingma. Variational inference & deep learning, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

Konstantina Kourou, Themis P Exarchos, Konstantinos P Exarchos, Michalis V Karamouzis, and Dimitrios I Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 13:8–17, 2015.

Volodymyr Kuleshov and Stefano Ermon. Probabilistic graphical models (cs228). https://ermongroup.github.io/cs228-notes/, 2017. Accessed: 2018-06-12.

Tejas D Kulkarni, William F. Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2539–2547. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5851-deep-convolutional-inverse-graphics-network.pdf.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Valliappa Lakshmanan, Eric Gilleland, Amy McGovern, and Martin Tingley. Machine learning and data mining approaches to climate science. In *Proceedings of the 4th International Workshop on Climate Informatics*. Springer, 2015.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*, 2015a.

Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. *arXiv preprint arXiv:1506.01057*, 2015b.

Jiwei Li, Michel Galley, Chris Brockett, Georgios P Spithourakis, Jianfeng Gao, and Bill Dolan. A persona-based neural conversation model. *arXiv preprint arXiv:1603.06155*, 2016a.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016b.

Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*, 2017.

Chia-Wei Liu, Ryan Lowe, Iulian Serban, Mike Noseworthy, Laurent Charlin, and Joelle Pineau. How not to evaluate your dialogue system: An empirical study of unsupervised

evaluation metrics for dialogue response generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2122–2132, 2016.

Xugang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori. Speech enhancement based on deep denoising autoencoder. In *Interspeech*, pages 436–440, 2013.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

John McCarthy. Artificial intelligence, logic and formalizing common sense. In *Philosophical logic and artificial intelligence*, pages 161–190. Springer, 1989.

Hongyuan Mei, Mohit Bansal, and Matthew R Walter. Coherent dialogue with attention-based language models. In *AAAI*, pages 3252–3258, 2017.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

Bhaskar Mitra and Nick Craswell. Neural text embeddings for information retrieval. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 813–814. ACM, 2017.

Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1, 2015.

Christopher Olah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. Accessed: 2018-05-25.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2012.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

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

Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Radim Rehurek and Petr Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*, 2015.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

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

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*, volume 16, pages 3776–3784, 2016.

Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Linfeng Song and Lin Zhao. Question generation from a knowledge base with web exploration. *arXiv preprint arXiv:1610.03807*, 2016.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*, 2015.

Amanda Stent, Matthew Marge, and Mohit Singhai. Evaluating evaluation methods for generation in the presence of variation. In *Proceedings of International Conference on Intelligent Text Processing and Computational Linguistics*, pages 341–351, 2005.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

TD Swinscow. Statistics at square one: Xviii-correlation. *British Medical Journal*, 2(6037): 680, 1976.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Subhashini Venugopalan, Marcus Rohrbach, Jeff Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko. Sequence to sequence-video to text. Technical report, University of Texas at Austin Austin United States, 2015.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, pages 1–305, 2008.

Richard S Wallace. The anatomy of alice. In *Parsing the Turing Test*, pages 181–210. Springer, 2009.

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

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

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

Sam Wiseman and Alexander M Rush. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*, 2016.

Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. The microsoft 2016 conversational speech recognition system. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 5255–5259. IEEE, 2017.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015a.

Yan Xu, Lili Mou, Ge Li, Yunchuan Chen, Hao Peng, and Zhi Jin. Classifying relations via long short term memory networks along shortest dependency paths. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1785–1794, 2015b.

Xitong Yang. Understanding the variational lower bound, 2017.

Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. *arXiv preprint arXiv:1702.08139*, 2017.

Kaisheng Yao and Geoffrey Zweig. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. *arXiv preprint arXiv:1506.00196*, 2015.

Kaisheng Yao, Baolin Peng, Geoffrey Zweig, and Kam-Fai Wong. An attentional neural conversation model with improved specificity. *arXiv preprint arXiv:1606.01292*, 2016.

Jun Yin, Xin Jiang, Zhengdong Lu, Lifeng Shang, Hang Li, and Xiaoming Li. Neural generative question answering. *arXiv preprint arXiv:1512.01337*, 2015.

Kun Zeng, Jun Yu, Ruxin Wang, Cuihua Li, and Dacheng Tao. Coupled deep autoencoder for single image super-resolution. *IEEE transactions on cybernetics*, 47(1):27–37, 2017.

Biao Zhang, Deyi Xiong, Hong Duan, Min Zhang, et al. Variational neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 521–530, 2016.

Chunting Zhou and Graham Neubig. Morphological inflection generation with multi-space variational encoder-decoders. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 58–65, 2017. doi: 10.18653/v1/K17-2005.

Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. Neural question generation from text: A preliminary study. In *National CCF Conference on Natural Language Processing and Chinese Computing*, pages 662–671. Springer, 2017.

APPENDICES

Appendix A: Python Code for Encoder-Decoder Models

Code is available at https://github.com/HareeshBahuleyan/tf-var-attention