Wasserstein Autoencoders with Mixture of Gaussian Priors for Stylized Text Generation

by

Amirpasha Ghabussi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Author's Declaration

This thesis includes all the material that I authored or co-authored: see *Statement of Contribution* included in this 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 Contribution

Chapter 4 is based on the following paper:

- Amirpasha Ghabussi, Lili Mou, Olga Vechtomova "Stylized Text Generation Using Wasserstein Autoencoders with a Mixture of Gaussian Prior." arXiv preprint arXiv:1911.03828 (2019)

I contributed to the implementation, experimentation, and writing the manuscript of the above paper.

Abstract

Probabilistic text generation is an important application of Natural Language Processing (NLP). Variational autoencoders and Wasserstein autoencoders are two widely used methods for text generation. New research efforts focus on improving the quality of the generated samples for these two methods. While Wasserstein autoencoders are effective for text generation, they are unable to control the topic of generated text, even when the training dataset has samples from multiple categories with different styles. We present a semi-supervised approach using Wasserstein autoencoders and a mixture of Gaussian priors for topic-aware sentence generation. Our model is trained on a multi-class dataset and generates sentences in the style/topic of a desired class. It is also capable of interpolating multiple classes. Moreover, we can train our model on relatively small datasets. While a regular WAE or VAE cannot generate diverse sentences with few training samples, our approach generates diverse sentences and preserves the style and the content of the desired classes.

Acknowledgements

I would like to thank Prof. Olga Vechtomova, without whom none of this would have been possible. Her support, guidance, and feedbacks have always been invaluable. I also want to thank Prof. Lili Mou who helped with this work from the beginning and provided constant support throughout this work.

Dedication

This thesis is dedicated to my family who always supported me throughout my life.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Stylized generative models have become an important research area during the past decade. Thanks to extensive investment in stylized image generation, today there are many models capable of creating fantastic looking pictures with a variety of desired styles. Research on stylized text generation is also progressing. While little deficiencies in a generated stylized image might not be too obvious, slightly wrong word choices or small grammatical errors are salient within text. Another interesting research area is text generation with a mixture of styles. This may have many use-cases such as generating poems or small novels with the style of multiple writers. Lastly, training deep generative models to perform the previously mentioned tasks, usually requires a big number of training samples and most models perform poorly on smaller datasets. Hence, training models that produce good results on smaller datasets is another interesting research topic.

1.2 Contributions

The main contributions of this thesis are:

- Supervised multi-class sentence generation while preserving the content and style of specified classes.
- Diverse stylized sentence generation on relatively small datasets.

1.3 Chapter Outline

This thesis is organized in the following chapters:

- Chapter 1 outlines the main contributions and the problem statement.
- Chapter 2 discusses related core concepts that are important to this work.
- Chapter 3 includes important related and previous research on stylized text generation and style transfer.
- Chapter 4 explains our approach to produce stylized text and how we manage to mix multiple styles. This chapter also includes our experiments and results.
- Chapter 5 provides a conclusion and summary of this work, followed by future work.

Chapter 2

Background

2.1 Natural Language Generation

Natural Language Generation (NLG) is a field of research that falls under the broader field of Natural Language Processing. NLG aims to generate sentences or text which resemble any form of written or spoken documents in natural human language. In early 21st century, NLG research was mostly focused on generating sequences of text based on a set of grammatical rules [40]. Other approaches were based on statistical analysis of a variety of language features such as predicting the next word based on the occurrence and co-occurrence probabilities of a word [8]. Probabilistic models such as n-gram were other early NLG approaches [9]. After the rise of neural networks in the recent years, they were widely adopted in many NLG sub-fields including, but not limited to the following:

- **Neural machine translation:** Given a sentence or document as input, neural machine translation (NMT) focuses on generating a semantically similar document in another natural human language (e.g. German to English). [20, 10, 45]
- **Dialogue generation:** Dialogue generation is the task of producing a response conditioned on a given text input. The goal of dialogue generation is to replicate a natural human to human conversation. The main application of dialogue generation is digital chat bots and to some extent, AI assistance programs [20, 42].
- **Stylized text generation:** Stylized text generation is the task of generating a sentence or document, conditioned on the style of some other documents. One example of its applications is formalizing an informal document [36].

- **Text summarization:** Text summarization's main goal is to generate a short and concise document. Given a longer document, the summary should include all the key important information contained in the original document [37, 4].

2.2 Feed Forward Neural Network

Feed forward neural networks are the most basic type of neural networks. In a feed forward neural network, information only flows in one direction, forward, hence the name feed forward neural network. This is the main factor distinguishing feed froward neural networks from their descendant, recurrent neural networks. Similar to many other neural networks, using gradient descent [41], the back-propagation of the loss updates the weights of the network.

2.3 Single Layer Perceptron

Feed forward neural networks in their simplest form are also called a single layer perceptron where the input and output layers are directly connected via a series of weights. The output of each node is the output of an activation function applied to the sum of the weights and the inputs. Single layer perceptrons are only capable of distinguishing between linearly separable inputs [35].

2.4 Multi Layer Perceptron

Multi Layer Perceptron (MLP) is the most vanilla type of neural networks widely used in NLG and machine learning. An MLP can be made from stacking two or more single layer perceptrons. Unlike a single layer perceptron, the input and the output layers of the MLP are connected via one or more hidden layers. In each hidden layer node, the output is computed as an activation function applied on the sum of the weights and the output of the previous hidden layer. One can think of each hidden layer as a function f. Assuming a neural network has n hidden layers and one output layer, the output can be written as $f_n(f_{n-1}(\dots(f_1(input))))$. Figure 2.1 includes feed forward neural network with two hidden layers. Each node is represented by a circle and each arrow represents a connection between two nodes. Note that every node in all layers is connected to all the

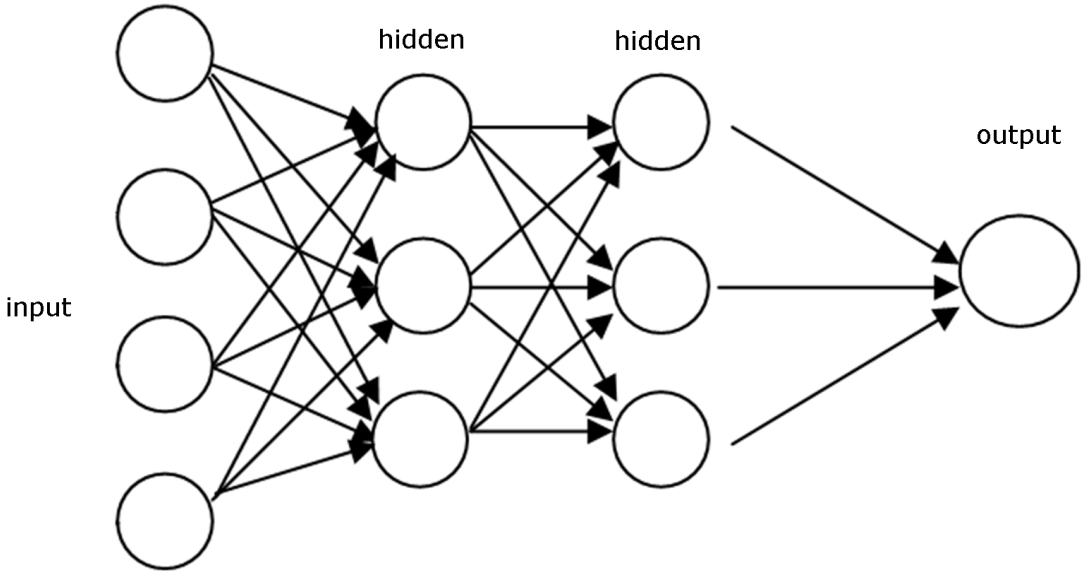

Figure 2.1: A feed forward neural network with 2 hidden layers. [Source]

nodes from the previous layer, hence, this network is also called a fully connected neural network.

The non-linear activation functions that are applied in each layer can be different; however, usually in practice all the hidden layers have a similar activation function and the final layer can have a different one. There are many popular choices for the activation function. Some of these functions are listed below:

- **ReLU:** A rectified linear unit is one of the most popular activation functions used in neural networks, due to its simple computations. A ReLU function simply returns the input if the input is bigger than one, and zero otherwise:

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ ax & \text{otherwise.} \end{cases}$$

  Moreover, the derivatives of the ReLU function are computed with very low computation power, it is 1 for values above 0 and 0 otherwise. Figure 2.2 bottom shows the ReLU function.

- **Sigmoid:** A sigmoid function clips the value of the input between zero and one and has an S-shaped curve:
$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$
Figure 2.2 top shows the sigmoid function.

- **Softmax:** When the output of a neural network is a vector and the intent is to pick the most probable component, the softmax function becomes useful. Softmax, as the name suggests, amplifies the maximum value in an array or vector and dampens the rest of the components with respect to their values:
$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } z = (z_1, \ldots, z_K) \in R^K$$

- **Tanh:** Tanh is another popular choice for activation functions in neural networks:
$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1}$$
Figure 2.2 middle shows the tanh function.

2.4.1 Regularization

In machine learning, regularization is a variety of practices and techniques that improves an approach to avoid over-fitting to the data it is being trained on and helps it to generalize easier. There are many techniques for regularization including but not limited to: dropout, batch-normalization, training knowledge augmentation, Lasso, and Ridge regularization.

2.4.2 Dropout

Dropout [46] is one of the most used regularization techniques in machine learning. The term dropout refers to a neural network hidden layer unit randomly dropping out, meaning that the output of the cell will be equal to zero and will not contribute to the input state of the next cell.

Figure 2.3 shows how dropout works on a neural network. The neural network on the left is a fully connected neural network. When dropout is added to this network, some of

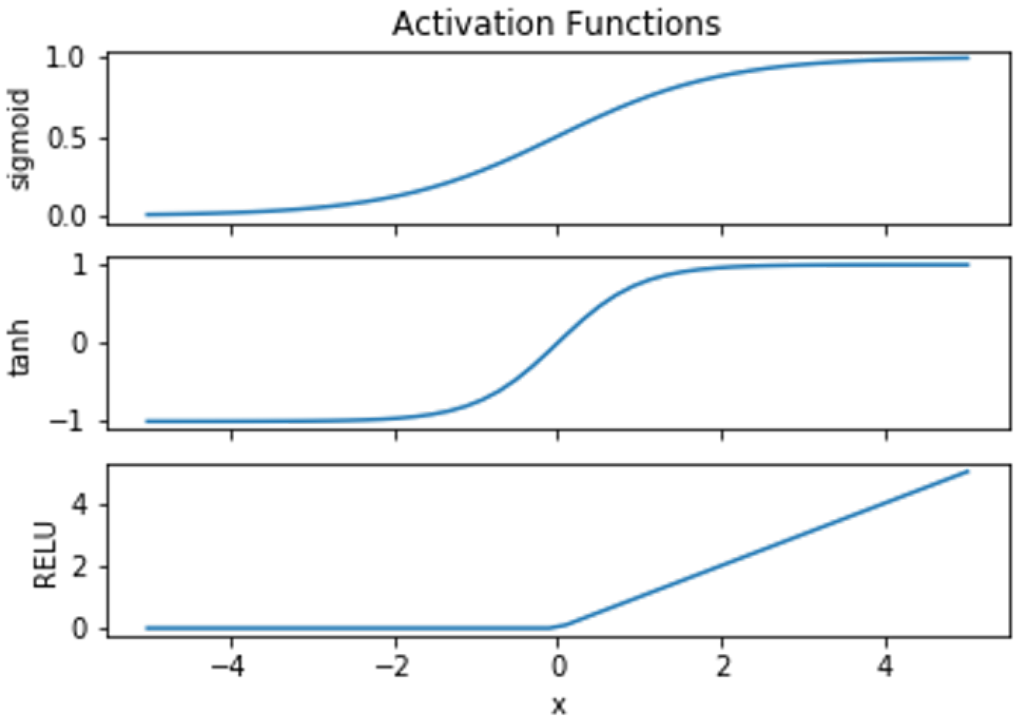

Figure 2.2: Three common activation functions. [Source]

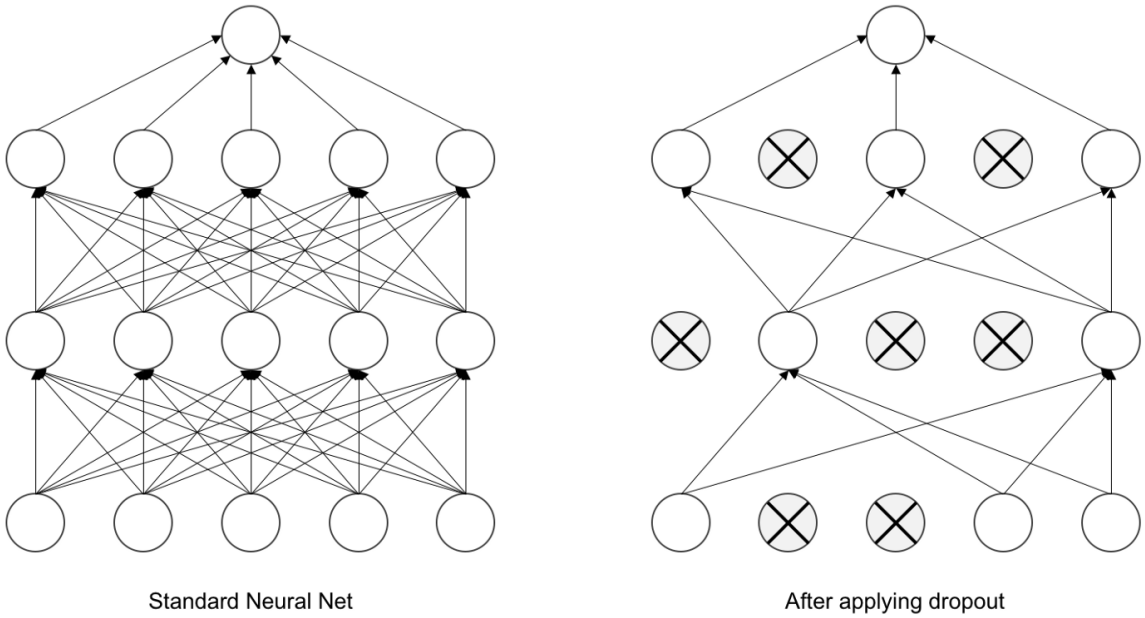

Figure 2.3: Left: a fully connected neural network. Right: the effect of dropout on the same network. [Source]

the units randomly deactivate and do not contribute to the outcome of the network. The probability of a cell being deactivated during a forward pass is as follows:

$$\hat{\mathbf{w}}_j = \begin{cases} \mathbf{w}_j, & \text{with } P(c) \\ \mathbf{0}, & \text{otherwise} \end{cases}$$

2.5 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are another type of neural networks mainly used for text sequences. They are also used in audio processing. The distinguishing factor of a RNN compared to a feed forwad neural network is that an RNN also includes the position or the time step of a sequence of data points. Another important characteristic of RNNs is that they can unfold many times until they cover the entire sequence of input data, hence the input can have arbitrary length. The reason RNNs are used instead of vanilla feed forward neural networks is that if the input is a sequence, the historical context becomes important

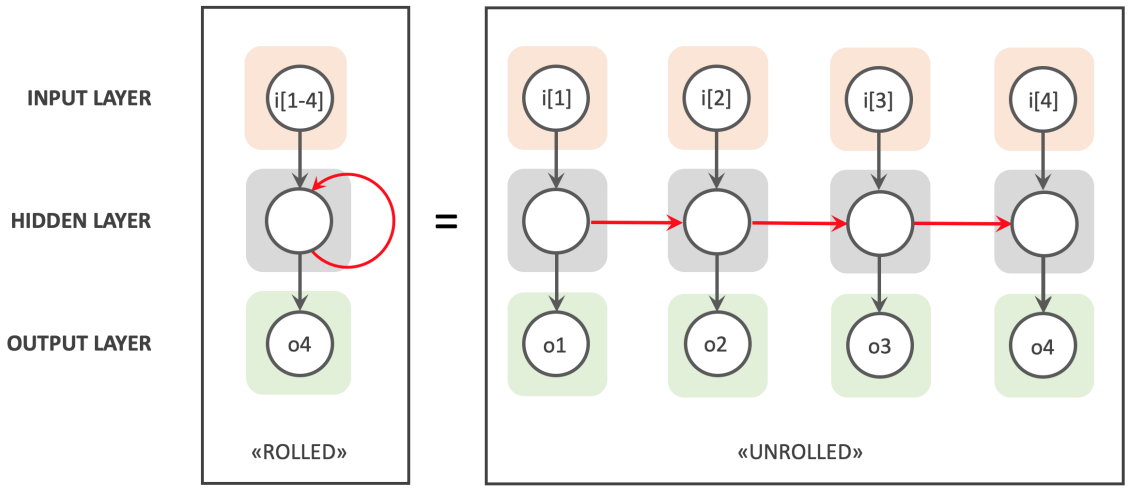

Figure 2.4: Recurrent Neural Networks. [Source]

when the network is interpreting the next components of the sequence. For example, the word "bad" by itself has a negative sentiment, but when used in a sentence, it can change the sentiment of the sentence.

There is a recurrent connection between each unfold of an RNN when it is processing a sequence of data. In other words, the output of the RNN when it is processing the data point in the t position, also depends on the features that are extracted at position $t-1$. Similarly, the output of the RNN at $t-1$ depends on both the $t-1$ data point and the RNN output at $t-2$. This can be written as equation 2.1:

$$h_t = f(W1.h_{t-1} + W2.x_t) \tag{2.1}$$

Figure 2.4 shows how a recurrent neural network can extract features from a sequence of data.

2.6 BiDirectional RNNs

Bidirectional RNNs are a special type of RNNs. Instead of processing the input sequence from the first element to the last, bidirectional RNNs process the sequence from the last element to the first one as well. This means that the output of the network at position t is

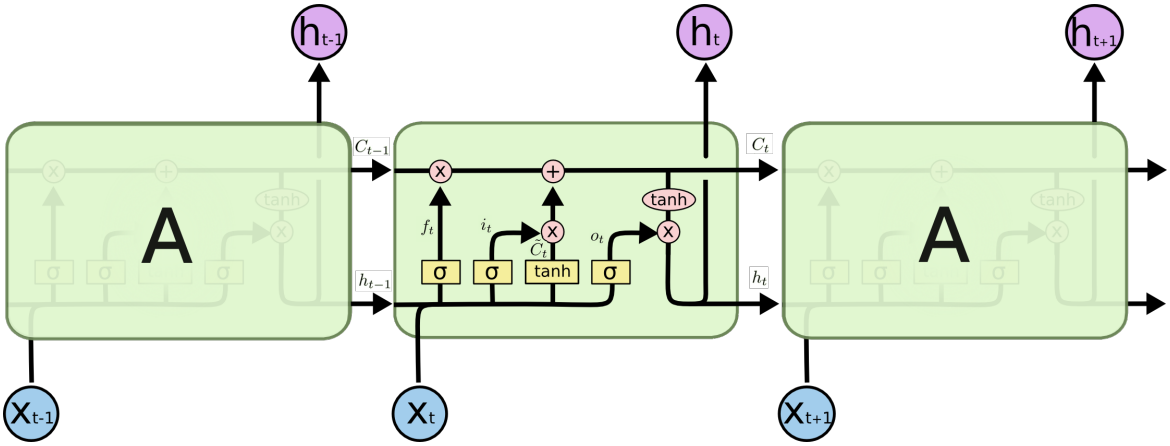

Figure 2.5: A long short term memory cell. [Source]

a concatenation of the forward and backward passes of the RNN over the input sequence. In practice, bidirectional RNNs can improve the accuracy of a network.

2.6.1 Long Short Term Memory

Vanilla RNNs suffer from two types of problems, vanishing and exploding gradients. The vanishing gradients problem is when the gradient in an RNN is multiplied by values less than one. Due to the nature of the RNNs, they usually form a deep computational graph. Therefore, the computed gradient has to flow through this deep graph. When the gradient is multiplied by values less than one, it starts to vanish rapidly and gets closer to zero. Thus, after a while the gradient becomes zero and it will not be possible to train the network anymore. Similarly, when the gradient is multiplied by values above one, the gradient starts to grow rapidly and becomes useless for training the neural network.

To overcome the abovementioned problems, a Long Short Term Memory (LSTM)[17] can be used. Figure 2.5 shows the internal architecture of andLSTM cell and 2.6 provides the notations used in the LSTM internal architecture.

The functions in figure 2.5 have the following definitions:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

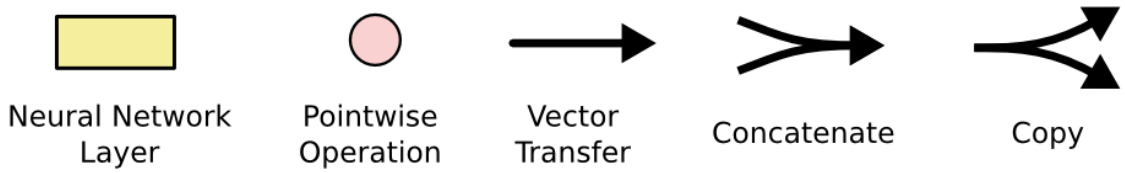

Figure 2.6: The notations used in figure 2.5. [Source]

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \sigma_h(c_t)$$

Where w_q and U_q are the weights of the input and recurrent connections respectively where the subscript $_q$ can be any of the forgot gate f, input gate i, or the output gate o of the memory cell c. Finally, h is the hidden state of the cell.

The way the LSTM works and fixes the problem of vanishing or exploding gradients is as follows: First, the forget gate f decides how much information from the previous state C_{t-1} should be preserved. Then the input gate i and the *tanh* gate determine by how much should the previous state be updated. Finally, the last step determines the output for this observation through the output gate o. This step requires both a sigmoid function and a *tanh* function. There are many variations of the LSTM cells that are adopted by many. However, the main idea behind all of them is to avoid vanishing or exploding gradients while memorizing the context of the input sequence.

2.7 Word Embeddings

There are many approaches to quantify words and documents into numerical values. Some approaches such as Bag Of Words (BOW) work on a document and sentence level. BOW represents each document as a vector with a length equal to the number of words available in the vocabulary. The value of each component in this vector corresponds to the number of times a specific word appears in the document. For example, if a vocabulary only includes three words and a sample sentence consists of the first word of the vocabulary followed by the second word of the vocabulary followed by the first word, then the BOW representation of this sentence is $[2, 1, 2]$.

Although BOW is a powerful representation for documents, it does not provide enough information on a word level. To provide a better representation for individual words, the word embeddings were proposed. A word embedding is a $m \times n$ matrix where m is the number of available words in the vocabulary and n is a fixed number. Each sentence can be represented as a sequence of vectors, each with n components. This allows individual words to have a set of characteristics defined by the value of their corresponding vector.

To find useful values that represents each word, Mikolov et al. proposed word2vec [34]. Word2vec learns the word representation from a text corpus using either the Continuous Bag of Words (CBOW) or the skip-gram approach. CBOW and skip-gram are very similar approaches. CBOW optimizes a loss function based on predicting the words included in a fixed sized window around each word in the document. The skip-gram model instead, predicts the probability distribution of the words in a window, given the word in the middle of the window. Finally, another popular word embedding is GloVe [38] which is trained based on the aggregated global word co-occurrence. GloVe and Word2vec both are very popular word embeddings widely used in machine learning.

2.8 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are another type of neural networks mainly used in image and video processing applications. As the name suggests, they involve convolutional mathematical operations. CNNs usually have an input layer, one or more hidden layers and a fully connected output layer. The dimensions of the input layer matches the height, width, and the channels of the input. Channel size is the depth of the image. For example, a RGB image has three channels, red, green, and blue.

The main intention behind CNNs was to decrease the parameter count of the neural networks and to avoid overfitting. For example, when a fully connected neural network is applied on a 50 by 50 image with a single channel, The number of weights required to construct the network is 50×50. However, a CNN can decrease the number of parameters by orders of magnitude depending on its hyper parameters.

Figure 2.7 is an overview of a 6 layer CNN. The input layer and the third layers of the networks are convolutional layers. The second and the fourth layers however, are pooling layers. Pooling is another commonly used practice in CNNs. A pooling layer simply picks the maximum or the minimum value of a matrix from the previous layer. The final two layers are fully connected layers that produce the final output of the neural network.

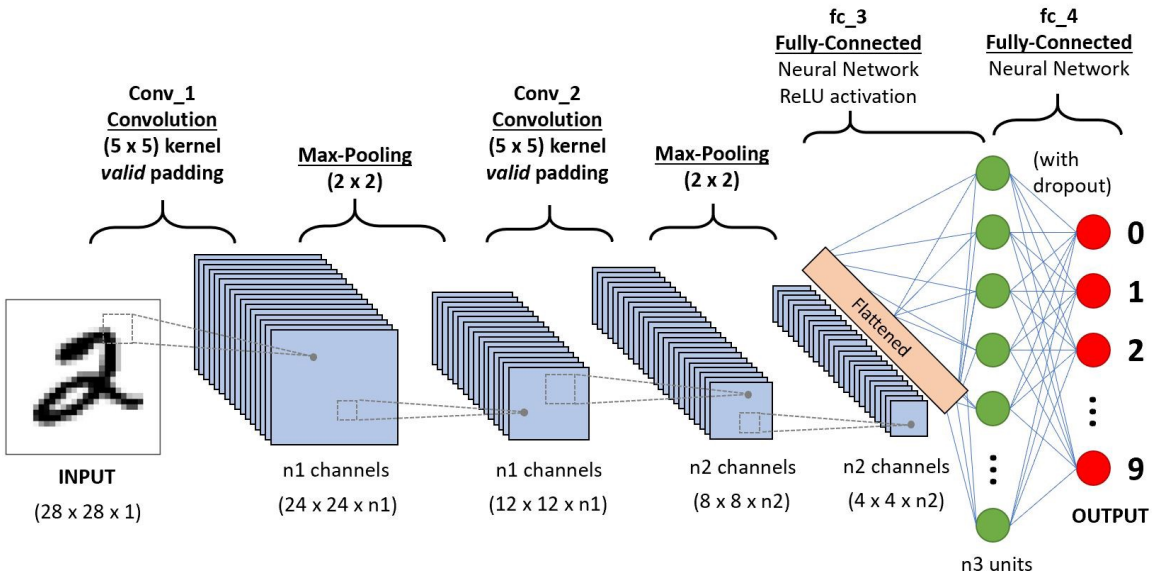

Figure 2.7: Convolutional neural networks. [Source]

Although CNNs are mostly used for image and video processing, they can also be used on text as well. They achieve this by setting the width of all layers to 1.

2.9 Sequence to Sequence Models

Sequence to Sequence (seq2seq) [47] models are widely used in natural language processing to transform an input sequence with an arbitrary length to an output sequence. Most of the seq2seq models consist of an encoder and a decoder. Both the encoder and the decoder are usually implemented with RNNs. The encoder is responsible to learn an intermediate representation of the input sequence which will be used by the decoder later on. The decoder uses this representation and generates a desired output sequence. For example, Sutskever et al. [47] uses a seq2seq model for English to French translation. Figure 2.8 provides an overview of a seq2seq network used for translation.

Other use cases of seq2seq models were used for many NLP tasks including but not limited to style transfer [23], dialogue generation [31], and auto-encoders [1].

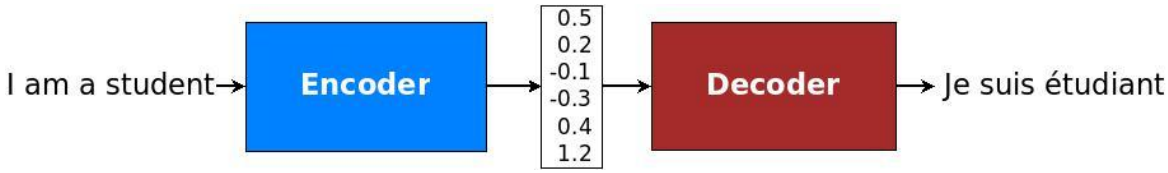

Figure 2.8: Sequence to sequence models used for translation. [Source]

Chapter 3

Related Work

This chapter outlines the related topics that are closely connected with the contributions of this work.

3.1 Autoencoders

Autoencoders [3] are models that are specialized in reconstructing the data points using three important components, the encoder, the decoder, and the latent space. The encoder is mainly implemented with CNNs for image processing tasks and with RNNs for text and audio processing. However, depending on the task, there are other types of neural networks that can be used to implement the encoder based on the input structure. For example, one choice for text processing is a feed-forward neural network when the input format is bag-of-words (BOW) [50]. It encodes the input into a vectorized representations usually with fewer dimensions.

The latent space captures the key features of the input. This latent space is one of the important components of an autoencoder. The latent representation of a data point can be used for many use cases. For example, if the size of the latent space is smaller than the size of the input, the latent representation can be stored as a compressed version of the input, while preserving most of the key important information included in the original data point.

The decoder uses the information stored in the latent space to reconstruct the input. Usually the decoder is implemented with the same component as the encoder. Simply

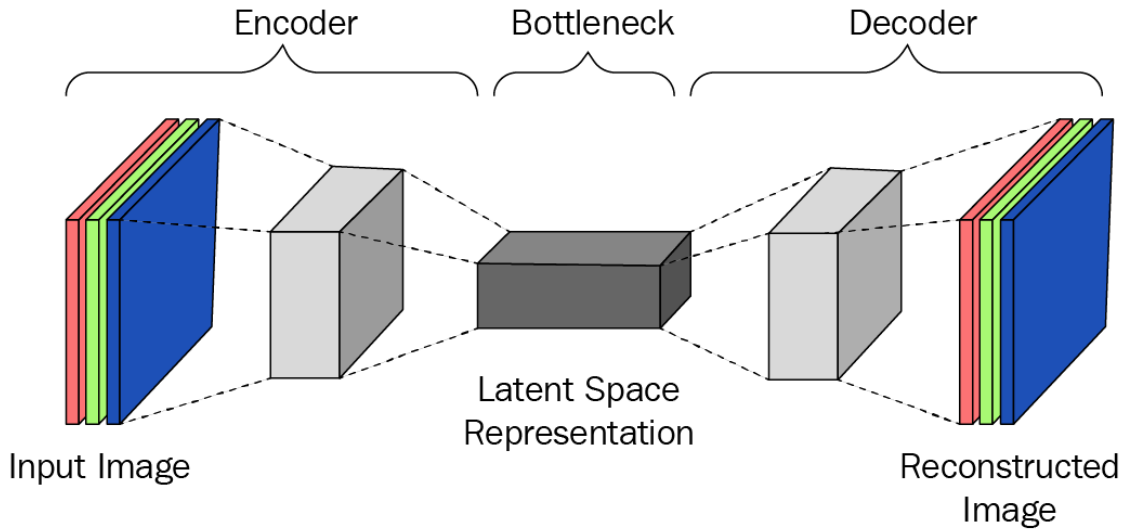

Figure 3.1: Autoencoders reconstruct the input data point. [Source]

reconstructing a data point is not a useful task on its own. But the decoder of an autoencoder along with the latent space or the latent representation of the data points can be very useful. The example scenario included in the previous paragraph cannot be achieved without a decoder, as merely storing a latent representation of the input data is not enough when there is no decoder to recreate the original inputs from their latent representation. Figure 3.1 represents an autoencoder.

Usually, the autoencoder's input has a much higher dimension than its corresponding latent representation. However, in some applications, such as noise reduction and text enhancement, the latent representation has higher dimensions [33]. Since autoencoders are trained on unlabeled data, they are relatively simple to train. The input of the network is a data point and the output should be the same data point. This makes the autoencoders very popular since they can be trained on datasets that are simply gathered from various sources on the internet. Since human labeling is not necessary on these datasets, these datasets can be very large which allows training autoencoders with great performance.

Another important application of autoencoders is style transfer [43]. Style transfer is the task of conserving the contents of an input, but changing its style. For example, if the network's input is a photograph of an apple, the output of the network can be a painted apple with a brush in the style of an artist. Style transfer is also a popular application in Natural Language Processing (NLP). In NLP, style transfer refers to regenerating a

sentence or a document and changing its style. For example, reconstructing a formal sentence to an informal one.

The autoencoders' training loss depends on the data points and the input structure. For example, the loss function of autoencoders for text can be defined as follows: Given that at time step t the decoder predicts the next token to be x_t , the training loss of the autoencoder J_{AE} is definded as:

$$J_{AE} = \sum_{i=1}^{N} \sum_{t=1}^{T} -log(p(x_t|h, x_1, x_2, ..., x_{t-1})) \tag{3.1}$$

where h is the latent vector representation, N is the number of training samples, and T is the total number of decoding steps.

3.1.1 Variational Autoencoders

Variational autoencoders (VAE) [25] are a probabilistic variant of the vanilla autoencoders. Instead of simply reconstructing the input, VAE encoder approximates the posterior distribution by encoding each input into two separate latent representations. One is used as the mean and the other one is used as the variance of the posterior latent distribution. During the training phase, the latent representation of each data point is used as the mean and the variance and a sample is generated from a normal distribution. Then the decoder reconstructs the data point. During the inference phase, the decoder generates novel data samples by sampling from the learned prior distribution. All the above can only be achieved by imposing a normal probability distribution over the latent space. To push the posterior distribution close to the prior, Kullback-Leibler (KL) [28] divergence is applied on the latent space. Finally, the total training loss of the VAE can be written as follows:

$$J_{VAE} = -E_{q_E(h|x)} log(p(x|h, x_1, x_2, ..., x_{t-1}) + \lambda_{vae} KL_{q_E(h|x)||p(h)} \tag{3.2}$$

3.1.2 Wasserstein Autoencoders

One approach to regularize the posterior is to impose a constraint that the aggregated posterior of h should be similar to its prior [48]. This constraint can be relaxed by penalizing the Wasserstein distance between $q(h)$ and $p(h)$. This can be computed as the Maximum Mean Discrepancy (MMD) between $Q(h)$ and $P(h)$:

$$MMD = \left\| \int k(h, .)dP(h) - \int k(h, .)dQ(h) \right\|_{H_k} \tag{3.3}$$

where H_k is the reproducing kernel Hilbert space defined by kernel k. We chose the inverse multi-quadratic kernel $k(x, y) = \frac{C}{C+||x-y||_2^2}$ in our experiments which is a common choice.

The MMD penalty can be estimated by empirical samples as:

$$\widehat{MMD} = \frac{1}{N(N-1)} \left(\sum_{n \neq m} k(h^{(n)}, h^{(m)}) + \sum_{n \neq m} k(\widetilde{h}^{(n)}, \widetilde{h}^{(m)}) \right) - \frac{1}{N^2} \sum_{n,m} k(h^{(n)}, \widetilde{h}^{(m)}) \tag{3.4}$$

where $\widetilde{h}^{(n)}$ is a sample from prior p and $h^{(n)}$ is a sample from the aggregated posterior q. Therefore, the training objective of a WAE is:

$$J_{\text{WAE}} = J_{\text{AE}} + \lambda_{\text{MMD}} \cdot \widehat{\text{MMD}} \tag{3.5}$$

A variation of the normal WAE is the stochastic WAE. The WAE's encoder can be either stochastic or deterministic. When the WAE is using a deterministic encoder, it has high reconstruction performance, but it usually lacks diversity when it comes to sentence generation. This stochasticity can be imposed on the encoder by using a KL-divergence term, regularizing the encoder. Bahuleyan et al. [2] show that this is a relaxed optimization of the Wasserstein distance.

It is important to note that the KL term is not what pushes the prior and the posterior distributions close to each other. The KL term is only meant as a regularization term to make the encoder stochastic. It is applied on two normal distributions, between the predicted posterior $q(z|x) = \mathcal{N}(\mu_{post}, \sigma^2_{post})$ and $\mathcal{N}(\mu_{post}, I)$. Therefore, the training objective of a stochastic wae is

$$J_{\text{Stochastic WAE}} = J_{\text{AE}} + \lambda_{\text{KL}} \cdot \text{KL}(\mathcal{N}(\mu_{post}, \sigma^2_{post}), \mathcal{N}(\mu_{post}, I)) + \lambda_{\text{MMD}} \cdot \widehat{\text{MMD}} \tag{3.6}$$

Figure 3.2 depicts how the KL terms is used. The black circles show the encoded data points. The green circles show the normal distribution with a mean equal to the encoded mean of each data point and a variance of 1. The red circle shows the normal distribution used to regularize the aggregated posterior of the input data which is represented by the dark blue ellipse.

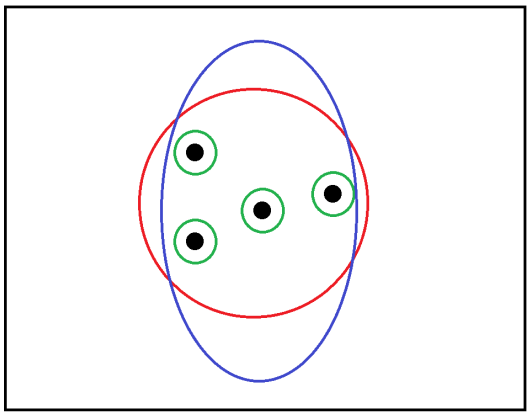

Figure 3.2: Stochastic WAE latent representation. [2]

3.2 Style Transfer

Neural style transfer was first proposed by Gatys et al. [15] for image style transfer. The original image style transfer task was to replicate an input image and reconstruct all the objects included in the original one, but change the style of the image. Style in this context was referred to as different features of the image such as brush style and color palettes. For image style transfer, the default choice of the neural network is a CNN. Figure 3.3 represents an instance of image style transfer.

Similarly in NLP, the task of style transfer refers to reconstructing a sentence or a document and preserving all the important information and details, while changing the writing style of the language. In natural language processing there is no unique definition of style. Different authors choose a variety of text characteristics as style. Sentiment, formality, genre, and authorship are common choices for representing the style of a sentence [21, 43, 13, 23]. One example of style transfer in NLP can be the following: transforming a formal document into an informal one while preserving the context, sentiment, and the important information included in the origianl sentence.

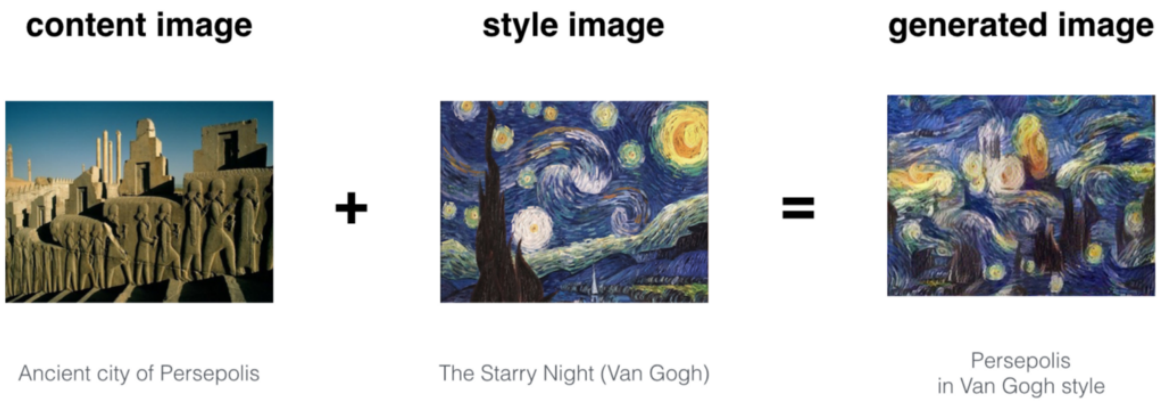

Figure 3.3: Neural image style transfer. [Source]

3.3 Stylized Text Generation

Stylized text generation is another application of NLG. Compared to style transfer where the task is to preserve the context of a document and modify its style, stylized text generation focuses on generating novel data points with a desired style. There are many approaches for stylized text generation. One approach is to use style-specific embeddings for sentence generation [13, 49]. Others focus on learning separate latent representations of style and content. Gao et al. [14] use a structured latent space with an autoencoder and a shared decoder to generate stylized dialogue responses. John et al. [23] apply adversarial and multitask losses to separate style from content. Their approach is designed for style transfer but can also be used for stylized generation.

3.4 Gaussian Mixture Models

Mixture of Gaussian prior was previously used for image clustering [5] and image generation [16]. In NLP Gaussian mixture model (GMM) priors have been used for several tasks. Shen et al. [44] use Gaussian mixtures for machine translation. Gu et al. [20] use an autoencoder network with a GMM prior to learn the latent representation of sentence-level data points, and jointly train a GAN to generate and discriminate in the same space. They use the Wasserstein distance to model dialogue responses. They only use GMM as a multi-modal representation of the latent space and do not add conditions to mixture distributions. Wang et al. [51] propose an unsupervised approach using a VAE with a GMM prior for

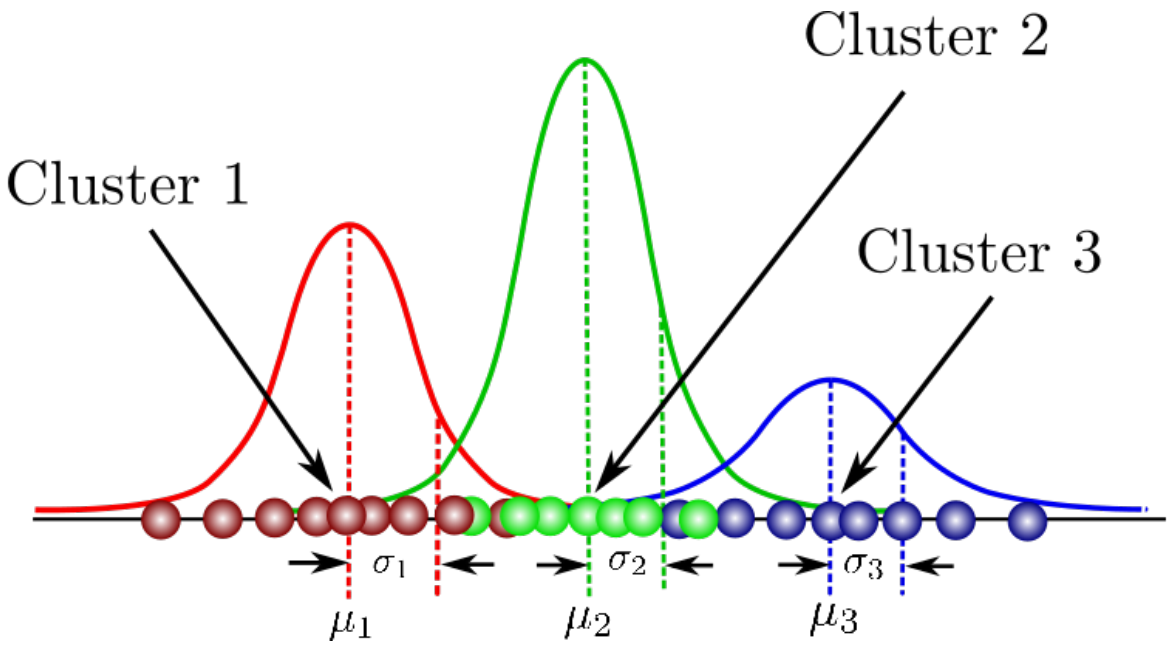

Figure 3.4: A mixture of Gaussian example. [Source]

topic modeling. They use bag-of-words for document representation, so their approach cannot be used for sentence generation. Moreover, their unsupervised approach prevents them from having any control over the styles learned by their model.

The next chapter introduces the Mixture of Gaussian prior following the equation:

$$p(\boldsymbol{\theta}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}) \tag{3.7}$$

3.5 Jensen-Shannon Divergences

Jensen-Shannon Divergence (JSD) [32] is our metric of choice to evaluate our multi-class sentence inference accuracy. We sample from two of the Gaussian distributions and average the sampled vectors with equal weights. Then, we feed this vector to the decoder. To determine the class of the inferred sentence, we use our pre-trained classifier. In the ideal situation, the output of the last layer of the classifier should be 0 for all of the classes

and 0.5 for the two sampled classes. JSD quantifies the difference between the sampling probability distribution and the classification distribution of the sampled sentences.

Let $p^{(1)} = (p_1^{(1)}, p_2^{(1)}, ..., p_n^{(1)})$ and $p^{(2)} = (p_1^{(2)}, p_2^{(2)}, ..., p_n^{(2)})$ be two probability distributions where $\sum_{i=1}^{n} p_i^{(j)} = 1$ and $0 =< p_i^{(j)} =< 1$ for all $i = 1, 2, ..., n$ and $j = 1, 2$. Then, the Jensen Shannon Divergence D between the two probability distributions $p^{(1)}$ and $p^{(2)}$ with same weights is:

$$D(p^{(1)}, p^{(2)}) = H(\frac{p^{(1)}}{2} + \frac{p^{(2)}}{2}) + \frac{H(p^{(2)}) - H(p^{(1)})}{2} \tag{3.8}$$

where

$$H(p^{(j)}) = \sum_{i=1}^{n} p_i^{(j)} * log_2(p_i^{(j)}) \tag{3.9}$$

denotes the Shannon entropy of of $p^{(j)}$.

Chapter 4

Approach

Probabilistic text generation is an important application of Natural Language Processing (NLP). Variational autoencoder [25] is a common method for sentence generation. VAE imposes a prior distribution on the latent space which is typically set to standard normal. It regularizes the latent space by Kullback-Leibler (KL) divergence [28] while reconstructing a data sample. This is equivalent to maximizing the variational lower bound of the likelihood of data. VAE is very difficult to train due to the issue of KL collapse. This can be resolved by adding word dropout or KL annealing to the training process [7]. Another approach for text generation is Generative Adversarial Networks (GAN) [18]. However, GAN loss is not differentiable for discrete data such as text, and they have difficulties generating discrete sequences [22], therefore VAE seems more appropriate for sentence generation.

Wasserstein autoencoders (WAE) [48] address the aforementioned problems. They regularize the latent space by pushing the aggregated posterior to the prior. This can be achieved by comparing empirical samples from the prior and the posterior distributions. Since unlike VAE, WAE does not push the latent posterior to be close to the prior based on any given input, this results in a better reconstruction performance. Moreover, WAE is much easier to train since it does not use KL divergence to regularize the latent space.

Regular VAE and WAE both generate a sentence by learning a distribution for the latent space. At the inference time, by sampling from this space, they can generate sentences similar to the distribution of the dataset they have been trained on. When the dataset has one class or a topic, this produces satisfactory results. Yet, since they use a standard normal distribution as their prior, they tend to over-regularize the latent space in cases where the dataset consists of multiple classes with different styles or topics. This can be a major drawback of using VAE or WAE for style-specific text generation.

To solve this problem, we propose a WAE with a Gaussian Mixture Prior (GMP) with the number of mixtures set to the number of classes in the dataset. This allows us to generate samples with the style of a specified class by only sampling from the GMP corresponding to this class. Moreover, since we share the same encoder and decoder over all of the classes, we can generate more diverse sentences by training our model on relatively small datasets. Lastly, this allows us to also generate sentences with a mixture of styles by using a sample from the weighted mixture of Gaussians from the the latent distribution.

In addition to over regularizing the latent space, most neural networks depend on big datasets and perform poorly when trained on small datasets. However, achieving good results using small data is an important real-world challenge, and in most cases it is harder than solving a big data challenge. With our proposed approach we show that we can train our model on a small number of data samples by adding data points from different topics to our data. Our experiments show that this will have a minimal effect on the style of the generated sentences.

To summarize, our main contributions are:

- Supervised single-class and multi-class sentence generation while preserving the content and style of specified classes
- Diverse sentence generation on relatively small datasets

We conduct several experiments to evaluate our approach. We use the Multi-genre Natural Language Inference (MNLI) dataset [52] to run all of our experiments. We experiment with style-conditioned and style-interpolated sentence generation. Our model produces the most diverse sentences among previous works. Moreover, we illustrate how our model can outperform others in fluency, diversity, and style accuracy even when it is trained on a small portion of the MNLI dataset.

In natural language processing there is no unique definition of style. Different authors choose a variety of text characteristics as style. Sentiment, formality, genre, and authorship are common choices for representing the style of a sentence [21, 43, 13, 23]. There are different approaches to style transfer, stylized generation, and style-specific topic modeling.

One approach to stylized text generation is using style-specific embeddings for sentence generation. Vechtomova et al. [49] use author-specific embeddings to generate stylized poetry, using multi-modal training data. By pretraining the embeddings using a CNN classifier they are able to generate novel and poetic data samples. Fu et al. [13] propose two different approaches for style transfer: style-specific embeddings and style-specific

decoder. By applying adversarial losses during training, they encourage the encoder to only include the content of the sentence in the latent space. They use sentiment as the style of a sentence.

Other works focus on learning separate latent representations of style and content for style transfer or stylized generation. Gao et al. [14] use a structured latent space to generate stylized dialogue responses. Their model uses a sequence-to-sequence module and an autoencoder with a shared decoder. John et al. [23] propose another approach and apply an adversarial loss to separate style from content. This approach is designed for style transfer, but it can be conditioned on a desired style and used for stylized generation as well.

Mixture of Gaussian prior was previously used for image clustering [5]. However, using mixture of Gaussian for text generation is different from previous works both in terms of the training objective and the model structure. There are different approaches to generate stylized sentences or style transfer. Previous work used Gaussian mixture models as the prior distribution for several NLP tasks. Shen et al. [44] uses Gaussian mixtures for machine translation. Gu et al. [20] uses an autoencoder network with a GMP to learn the latent representation of sentence-level data points and jointly trains a GAN to generate and discriminate in the same space. They use the Wasserstein distance to model dialogue responses. Wang et al. [51] propose an unsupervised approach using a VAE with Gaussian mixture prior for topic modeling. They apply a training penalty to push the Gaussian distributions further apart in the latent space. However, their choice of bag of words for data point representation does not allow them to generate coherent sentences. Moreover, mixing new data points with their dataset of choice might completely change the topics of their model.

This work is different from the previous works in that we use a semi-supervised approach with a GMM as our prior distribution and use labeled data for training. Moreover, we refer to a specific topic/class as the style of a sentence similar to Wang et al. [51], but we propose a semi-supervised approach using Wasserstein distance. This allows us to have more control over the specific styles our model will learn. Moreover, it allows us to mix these styles at the inference time. Additionally, we do not apply any penalty to push the Gaussian distributions further away in the latent space and this makes our model easier to train. Finally, we can expand our dataset and add new training samples to help our encoder to effectively learn the latent representation of our desired classes, and help our decoder to generate much more diverse sentences.

4.1 Wasserstein Autoencoder with Gaussian Mixture Prior

We use a stochastic WAE with MMD penalty with a sequence to sequence neural network [47]. Using a Gaussian mixture distribution for our prior we are able to generate single and multi-style conditioned sentences at the inference time.

In this work we use a Gaussian mixture model as the chosen distribution for our WAE prior. There are multiple benefits gained from this. First, many datasets are a combination of different styles and classes, therefore, the model structure should account for this in order to learn a good representation of these datasets. Moreover, separating the latent representation of a group of data samples allows the model to be trained on completely different data points at the same time and learn multiple latent distributions independently. The final distribution of our latent space follows the Gaussian mixture model distribution:

$$P(z) = \sum_{i=1}^{N} w_i \mathcal{N}(\mu_i, \sigma_i) \tag{4.1}$$

Where N is the number of mixture distributions, $\Sigma_{i=1}^{N} w_i = 1$, and $w_i \geq 0$. If a dataset has N classes with distinct styles, we use the same number of Gaussian distributions for our latent space and encode every sentence to its corresponding latent distribution. Then, the latent vector representation is defined as:

$$h = \sum_{i=1}^{N} w_i \times h_i \tag{4.2}$$

Where h_i denotes the sampled vector from the i_{th} Gaussian mixture distribution $h_i \sim \mathcal{N}(\mu_i, \sigma_i)$ and w_i is its corresponding weight.

4.1.1 Datasets

The Multi-Genre Natural Language Inference (MNLI) [52] is the main dataset used in this work. It has 433k annotated sentences pairs. MNLI is based on the Stanford Natural Language Inference (SNLI) corpus [6]. MNLI is different from SNLI in that it covers multiple genres of spoken and written sentences. This allows researchers to use it for

Genre	Premise	Label	Hypothesis
Fiction	The Old One always comforted Ca'daan, except today.	Neutral	Ca'daan knew the Old One very well.
Letters	Your gift is appreciated by each and every student who will benefit from your generosity.	Neutral	Hundreds of students will benefit from your generosity.
Telephone	Yes now you know if if everybody like in August when everybody's on vacation or something we can dress a little more casual or	Contradiction	August is a black out month for vacations in the company.
9/11	At the other end of Pennsylvania Avenue, people began to line up for a White House tour.	Entailment	People formed a line at the end of Pennsylvania Avenue.

Table 4.1: A few examples of the data point from the MNLI dataset

multi-genre text generation and evaluation. Table 4.1.1 includes a few samples from the MNLI dataset.

The dataset has ten genres of data points, five of which have no training samples. The other five genres are Telephone, Slate, Government, Fiction, and Travel. The government genre includes reports, letters, speeches, and other types of government related documents gathered from publicly available government domain websites. The Fiction genre includes sentences from freely available fiction works written between 1912 and 2010. Telephone covers telephone conversations held between 1990 to 1991. Travel includes sentences from travel guides published by Berlitz Publishing in early 2000s. Finally, Slate includes a variety of articles on pop culture written between 1990 and early 2000s.

In our experiments, we only use the premise sentences of the data points from genres with available training samples. The other genres which only have 2000 dev and 2000 test samples are ignored. From the rest of the genres with available training samples, we filter out the Slate genre since the sentences included in this genres overlap in content and style with other genres included in the dataset. After filtering the genres without training samples and slate from the dataset, the remaining data points are 77,348 samples

	Gov	Fic	Trav	Tel	JSD
Gov	**70.27**	03.87	19.98	05.88	0.116
Fic	00.31	**96.58**	03.03	00.08	0.012
Trav	01.76	09.94	**87.68**	00.63	0.045
Tel	05.25	03.28	02.46	**89.00**	0.040

Table 4.2: Classification acuracy and JSD values for GMM-WAE Style-conditioned sentence generation using 40960 training samples from MNLI.

from fiction, 77,350 samples from government, 83,348 samples from telephone, and 77,350 samples from the travel genre, leading to 315,396 data points.

Finally, it is important to mention that MNLI has many sentences that are incomplete, have words appearing twice in a row, or some grammatical mistakes. This is demonstrated by the examples included in table 4.1.1. Therefore, the models trained on MNLI tend to learn these types of mistakes.

4.2 Training

At the training time, each input sequence $(x_1^i, x_2^i, ..., x_n^i)$ is mapped to its corresponding mean and variance vectors depending on its class. We simultaneously learn multiple priors by learning their mean (μ_i) and variance (σ_i) vectors. These vectors are initialized with random weights before training starts. Since we use a stochastic WAE, we then sample from a normal distribution with the same encoded mean and a variance of 1. We also use an empirical estimate of the MMD penalty to regularize the latent space. Since we use a stochastic WAE, we then sample from a normal distribution with the same encoded mean and a variance of 1. We use KL-divergence to regularize the stochastic part of our model and produce more diverse sentences based on the following objective:

$$J_{\mathrm{KL}} = \sum_{i=0}^{N} \mathrm{KL}\left(\mathcal{N}(\mu_{\mathrm{post}}, \mathrm{diag}(\sigma_{\mathrm{post}})^2) || \mathcal{N}(\mu_{\mathrm{post}}, \mathbf{I})\right) \tag{4.3}$$

Where N is the number of data points. To regularize the latent space and learn the prior distribution, we use the MMD penalty following Equation 3.4. Final training loss is the weighted sum of the KL loss, MMD loss, and reconstruction loss:

$$J_{\text{WAE}} = J_{\text{AE}} + \lambda_{\text{KL}} \cdot J_{\text{KL}} + \lambda_{\text{MMD}} \cdot \sum_{j=0}^{M} \widehat{\text{MMD}}_j \tag{4.4}$$

Where M is the total number of classes in the dataset.

During the training phase, we use mini-batches where the samples are from only one input class. This is a stochastic estimation of the actual gradient descent algorithm. The gradient computed for individual batches is biased towards a certain class, but since multiple batches are sampled from all of the classes, we can correctly optimize the training objective. For a sequence with class i we set all other latent weights to zero and $w_i = 1$. This allows us to only back-propagate the reconstruction loss through the i^{th} Gaussian distribution and the MMD penalty will push μ_i and σ_i to the prior. The one-hot class vector represents the training weights for the mixture distributions and the blue arrows show the back-propagation through just one of the distributions.

Training batches include only one class at a time. This is a stochastic estimation of the actual gradient. The gradient for each batch is biased towards a certain class. Since multiple batches are sampled from different classes, GMM-WAE correctly optimizes the training objective. For a batch sampled from class i we have $w_i = 1$ and $w_j = 0$ for $j \neq i$. This allows the gradient to back-propagate only through the i_{th} Gaussian distribution and the MMD penalty pushes the i_{th} Gaussian prior to the posterior. Figure 4.1a shows an overview of the training process and the following pseudo code explains the training phase.

Algorithm 1 Training step for a mini batch from class A

$\mu_{post}, \sigma_{post} \leftarrow encoder(batch_A)$
$J_{KL} \leftarrow KL(\mathcal{N}(\mu_{post}, \sigma_{post}), \mathcal{N}(\mu_{post}, I))$
$z \sim \mathcal{N}(\mu_{prior}, \sigma_{prior})$
$J_{AE} \leftarrow reconstructionLoss(batch_a, Decoder(z))$
$J_{MMD} \leftarrow MMD(samples\ from\ prior\ A,$
$\quad samples\ from\ posterior\ A)$
$J_{\text{WAE}} = J_{\text{AE}} + \lambda_{\text{KL}} \cdot J_{\text{KL}} + \lambda_{\text{MMD}} \cdot \widehat{\text{MMD}}$
$BackPropagate$

Finally, figure 4.2 shows a latent space with four Gaussian mixtures

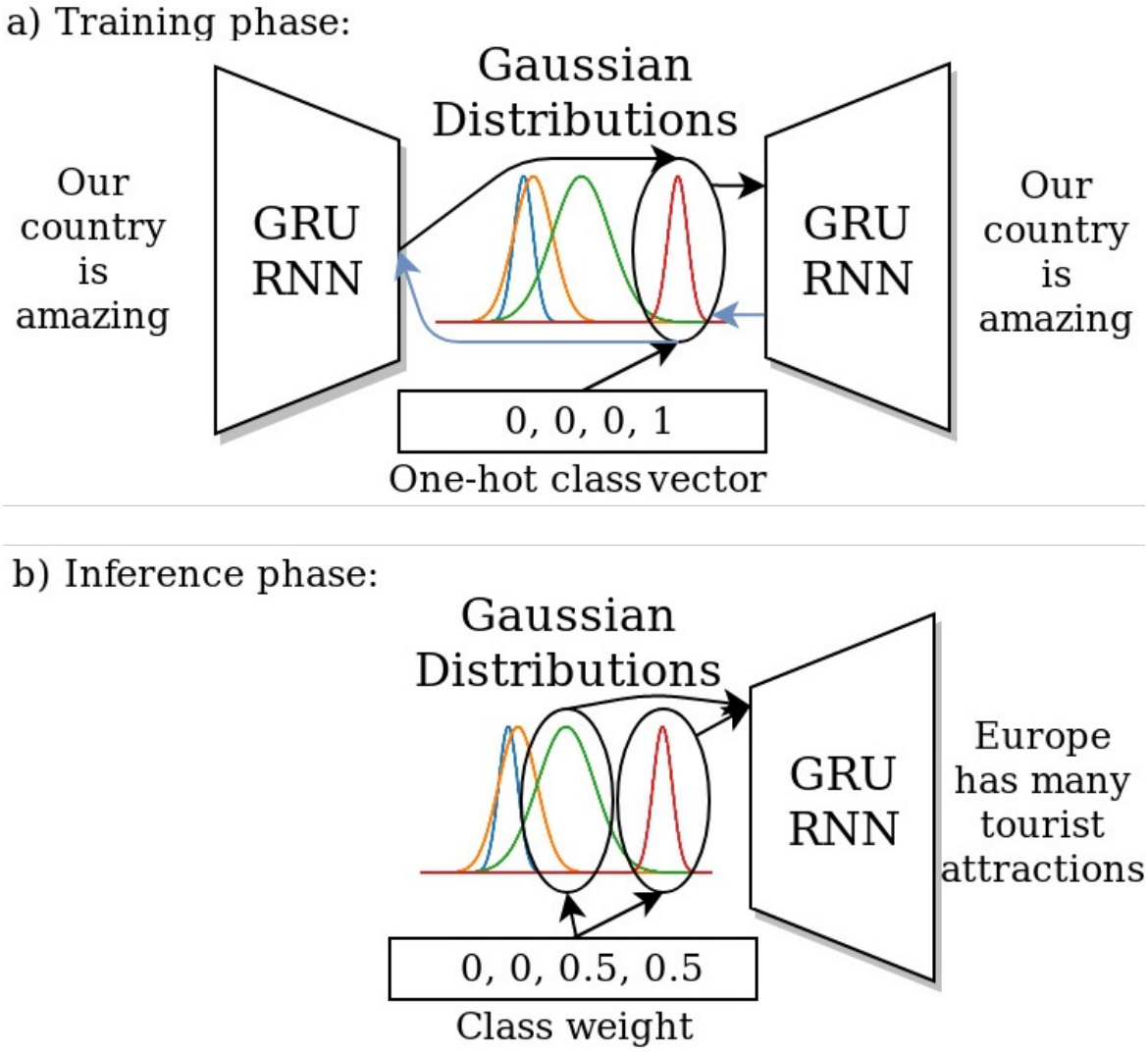

Figure 4.1: a) GMMWAE training phase. b) GMMWAE inference phase.

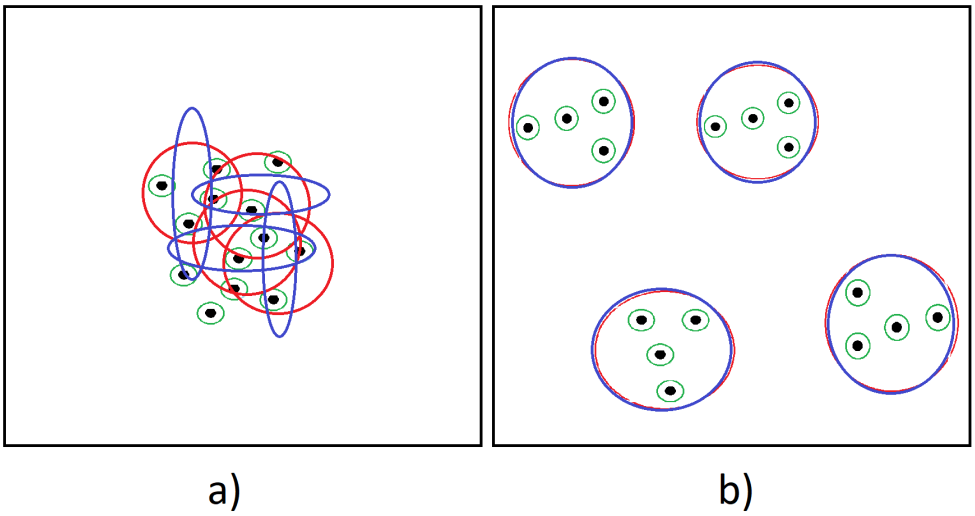

Figure 4.2: a) the latent space before training. The small green circles around each encoded data point represent the stochastic feature of the encoder. b) The latent space with four Gaussian mixtures after training. The blue circles are the latent distributions for each of the latent Gaussian distributions. They get closer to the actual distribution of each class of the data after training is finished.

4.3 Sentence Generation with GMM-WAE

Text generation with GMM-WAE is slightly different from the training process. To generate a sentence, we first have to sample from the latent space and produce the latent vector h following Equation 4.2. Then we feed this vector to the recurrent decoder as its initial state, and append it to the input at all time steps of the decoder. We use the standard inference decoder similar to Wu et al.[53]. Since we use a Gaussian mixture prior, we have the ability to decide which Gaussian prior we want to choose for sentence generation. We can also choose multiple priors and generate the sample based on a Gaussian mixture distribution. Figure 4.1b shows an overview of the inference process. The classes contributing to the style of the final sentence are the weights with non-zero values in the class vector.

Style-Conditioned Sentence Generation. In this setup, we generate sentences conditioned on one class. These sentences represent how capable GMMWAE is in terms of generating novel sentences conditioned on one of the input classes. Given the target class i we set $w_i = 1$ and $w_j = 0$ for all $j \neq i$. In other words, we set all w_i to zero except for the weight of the class, corresponding to the target class of the generated sentence. Hence the latent vector is sampled from $P(z) = \mathcal{N}(\mu_k, \sigma_k)$ where k is the target style the generated sentence is conditioned on. The sampled latent vector will only include features from the target class. This is similar to training a VAE only on the data points from the target class and gererating novel sentences. However, since our decoder is trained on many other data samples from other classes as well, it is capable of generating more diverse sentences. Additionally, in cases where the dataset is too small, a VAE will not train properly and the generated sequences might not be coherent, but since our encoder and decoder is shared among all the training classes, GMMWAE can be trained on smaller datasets where each class does not have many training samples.

Style-Interpolated Sentence Generation. In this setup, generated sentences are conditioned on two Gaussian prior distribution samples. We set w_i and w_j to be equal to 0.5 and the rest are zeroes. This means that we only use two of the latent normal distributions for our mixture of Gaussian and generate samples based on them. Hence the latent vector is sampled from a mixture of Gaussian (equation 3.7) where $K = 2$ and $\phi_1 = \phi_2 = 0.5$. This allows us to generate sentences with a mixture of styles. These types of sentences show the importance if using a prior distribution that takes into account the styles or classes the training dataset has. A vanilla VAE or WAE are not capable of doing a similar task.

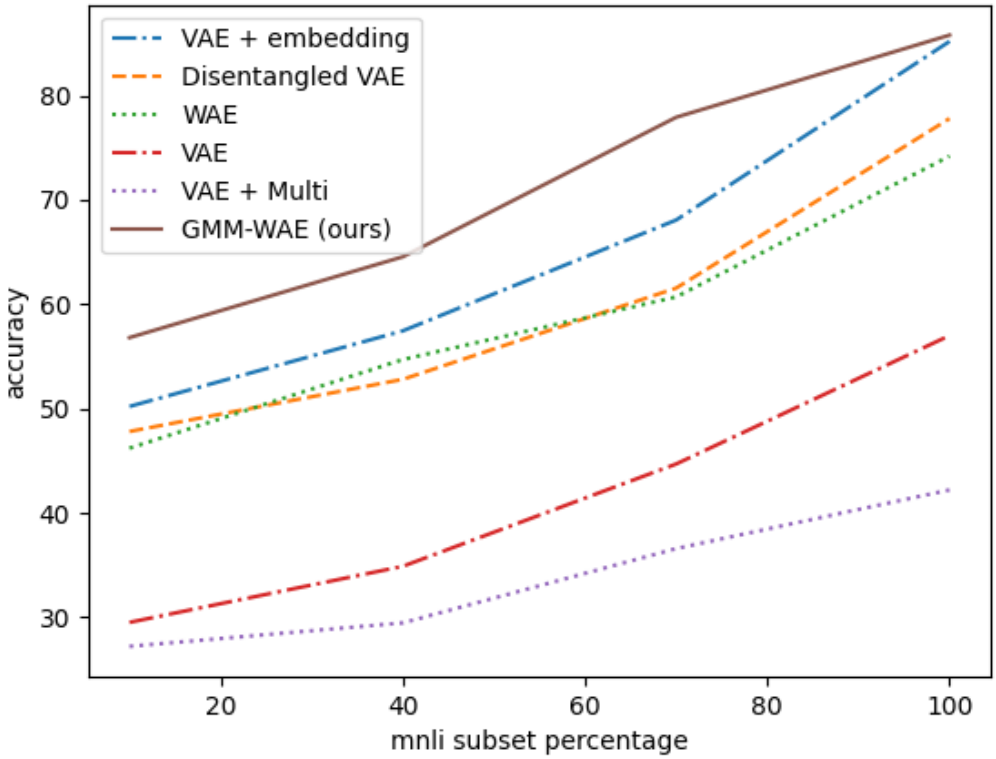

Figure 4.3: Comparison of GMM-WAE with other baselines trained on small, medium, and large datasets. The small and medium datasets are a subset of the whole MNLI dataset. the accuracy percentages provided are the average accuracy of each model for style-conditioned sentence generation.

4.4 Experiments

We use the MNLI dataset [52], containing 433k sentences from five genres: Slate, Telephone, Government, Fiction, and Travel. We ignore Slate since it covers a diverse set of topics overlapping with the other classes. We conducted all evaluations with two settings: (a) using all MNLI training data points in the four classes mentioned above; (b) using a subset of 10,240 training samples from each of these four classes to simulate a smaller dataset as illustrated in Fig. 4.3. We compare our style-conditioned experiments with the following approaches:

Separate WAE models – four separate WAE each trained on an individual class of

the MNLI dataset. The classification accuracy for separate WAE models is not a valid measure since each model is only trained on a single class. Thus the accuracy should be 100% in theory.

VAE + embedding – VAE with randomly initialized trainable class embeddings fed to the decoder [49].

Disentangled VAE – VAE with auxiliary multi-task and adversarial objectives to disentangle latent representations of style and content [23].

VAE – vanilla seq2seq VAE trained on the entire dataset. For style-conditioned generation, we sample from the distribution defined by the empirical mean and variance of training data points for each class.

WAE – vanilla WAE trained on the entire dataset. The same method was used for stylized generation as in VAE.

VAE + Multi – single VAE with multi-task loss to classify the latent representation of each training sample based on their topic.

Our model works best for generating diverse sentences and outperforms other models in most of the evaluation metrics. When the dataset is relatively small, a WAE or VAE do not capture enough features to generate diverse and fluent sentences. We use a WAE with GMP and train our model using ten percent of the data in MNLI and compare its performance with other models. Our model outperforms all of the other models in this case.

4.4.1 Evaluation

We use Jansen-Shannon Divergence to evaluate style-interpolated sentence generation classification accuracy. We also use multiple measures for sentence diversity, and finally we use perplexity to validate the coherency and fluency of the generated sentences.

Style Accuracy: We follow the approach of the previous work [21, 43, 13, 23] and separately train a CNN to classify sentences based on their classes. We use this classifier to classify sentences generated with our approach and compare our results with a separate WAE trained only on a specified class of a dataset. Table 4.1.1 provides the classification results of the style-conditioned sentences GMMWAE generates after being trained on the MNLI dataset. The classifier can easily classify the styles of the generated sentences when they are conditioned on one of the Gaussian prior distributions. The classification accuracy of the classifier over the original MNLI dataset is 98%. We used a 70-30 training-evaluation split for validating the accuracy of the CNN classifier on the original MNLI dataset.

	D-1↑	D-2↑	Entropy↑	(PPL)↓	Acc↑
Disentangled VAE	0.027	0.117	4.853	**73.81**	77.8
Separate WAE models	0.052	0.214	5.416	95.2	97.9
WAE	0.026	0.154	4.391	85.3	74.2
VAE	0.044	0.158	5.043	92.8	57.5
VAE + Multi	0.040	0.169	5.281	97.7	42.2
VAE + embedding	0.034	0.070	4.153	112.3	85.2
GMM-VAE	0.52	0.201	5.342	94.3	85.5
GMM-WAE (ours)	**0.067**	**0.465**	**5.883**	97.3	**85.8**

Table 4.3: Style-conditioned sentence generation results. All models were trained on the whole MNLI dataset. Note that the classification accuracy for separate WAE models is not a valid measure since each model is only trained on a single class. Thus the accuracy should be almost 100% in theory.

	D-1↑	D-2↑	Entropy↑	(PPL)↓	JSD↓
WAE	0.035	0.207	4.829	91.9	**0.153**
VAE	0.041	0.167	5.001	92.0	0.239
VAE + Multi	0.039	0.169	5.250	98.2	0.178
VAE + embedding	0.038	0.172	5.102	**85.4**	0.160
GMM-VAE	0.044	0.197	5.330	88.4	0.148
GMM-WAE (ours)	**0.075**	**0.490**	**5.975**	99.6	0.161

Table 4.4: Style-interpolated sentence generation results on MNLI.

We also generate style-conditioned sentences with other models and baselines to evaluate the performance of our approach. The **Acc** column of table 4.3 represents the accuracy of each model for style-conditioned generated sentences. The accuracy percentage is the average accuracy of the models when they generate style-conditioned sentences on each of the four used MNLI classes.

For style-interpolated sentence generation, we use JSD to compare the classification accuracy of our CNN classifier. Our approach does not generate the most accurate sentences among the other approaches, however, our JSD values almost match the best performing approach. The JSD column of table 4.4 compares the classification accuracy results of GMMWAE with other approaches.

Perplexity: We use the Kneser-Ney language model [26] to evaluate the fluency of

our sampled sentences. We measure the empirical distribution of trigrams in a corpus and compute the log-likelihood of a sentence. We train the language model on the original MNLI samples and evaluate the fluency of our sampled sentences. 4.3 provides the fluency results for the style-conditioned sentences generated by GMMWAE and other approaches. Our model achieves perplexity scores similar to most other models and baselines. We are sharing the same encoder and decoder among all training samples and conditioning our decoder on one or two of the prior Gaussian distributions.

Similarly, the PPL column of table 4.4 includes the perplexity results for the style-interpolated sentences from other models and GMMWAE.

Diversity: We use distinct diversity metrics by computing the percentage of distinct unigrams and bigrams following the work of Li et al. [30] and Bahuleyan et al. [1]. D-1 and D-2 represent the unigram and the bigram percentages respectively. Our model outperforms all of the baselines and models in terms of sentence diversity both in the style-conditioned and the style-interpolated setup. The diversity results are provided in the D-1 and the D-2 columns of tables 4.3 and 4.4.

We also ran an experiment on the SQuAD dataset [39] for question generation using GMMWAE. For the question generation task, since the dataset is very small, neither of the models are successful at generating diverse sentences and they tend to generate the same set of question over and over. However, GMMWAE generates twice as many diverse sentences compared to separate WAEs or VAEs trained on each class of SQuAD questions.

To evaluate and compare style-interpolated and style-conditioned sentences, we used a few baselines since there is no prior work similar to what we are proposing. Some of the prior work such as VAE + embeddding are capable of interpolating between two styles by averaging the embedding vectors for two of the MNLI classes. However, for other baselines such as WAE and VAE we had to use a different approach. We first trained these models on the MNLI dataset. After the training process was finished, we fed the training samples from only one of the MNLI classes to the encoder and averaged the latent representation of these samples. These averaged vectors represent the average mean and variance for an individual MNLI class. We then computed similar vectors for other MNLI classes. This way, we could compute an average mean and variance latent representation for all individual classes. We then used these mean and variance vectors to generate new samples for each class. For style-conditioned sentences, we simply used the averaged vectors from individual classes and generated novel sentences for the baselines. For style-interpolated sentence generation, we averaged the mean and variance vectors and then passed the averaged vectors to the decoder to generate sentences with a mixture of styles. Table 4.5 provides a few sample generated sentences by GMMWAE.

Fiction
kramenin? he drew the question the last time that 's happened? man , apparently you do n't think that the doctor 's always alone.
Government
i provide guidance in determining the requirements of state agencies also used the additional databases. there are no success of delivery in california , reducing in pm concentrations.
Travel
hong kong is now a fascinating fifth-century , walled architecture. the greatest can be sensed in dublin and its surrounding farmlands , a full of historic buildings.
Telephone
uh-huh yeah i guess you ca n't have our problem. so they had to talk about it, um oh absolutely.
Government + Travel
so we 've talked to our children to pursue little observation from the standpoint that we 're split in. one provides opportunities for bargaining delivery system to link between gagas and research is helpful.
Government + Telephone
8 time, i mean you have UNK UNK outside the new government. when the general requires a current protections of federal acquisition , he said an organization had adoptedeach retiree.
Government + Fiction
it is right we just always died as a wild country. the village who still have the american culture.
Travel + Telephone
given the book now i know like that capital or UNK egypt who came to conquer. that i guess the remaining states that now is a more easily protected.
Travel + Fiction
given the book now i know like that capital or UNK egypt who came to conquer. and some point the british army and you can be sure of nancy texas.
Telephone + Fiction
exactly when the telephone of mine yeah i do it is in your way i have rather taken on that. susan is talking like san wouldoro stabbed into her.

Table 4.5: Sentences generated by GMM-WAE.

To summarize, table 4.3 and table 4.4 provide detailed comparison between our work and other models and baselines. Our model provides the best accuracy results when it is trained on smaller datasets compared to other approaches. Although some baselines have better perplexity scores, our model generates the most diverse sentences as the D-1 and D-2 values suggest. Additionally our approach generates sentences with the largest set of unique words, where other models use a smaller set of words to generate novel sentences. Perplexity results is the only metric that our model cannot outperform the other baselines. Finally, our model achieves the highest classification score. This is a good indication that style-conditioned sentences generated from our model are preserving the style and content

of their corresponding class better than all other approaches.

Chapter 5

Summary and Conclusion

5.1 Summary

This work proposed the Wasserstein autoencoders with a Gaussian mixture prior. The Gaussian mixture prior replaces the single Gaussian prior used by default by the Wasserstein autoencoders. During the training phase, each data sample is fed to the model along with its corresponding class. The class is used to choose the right Gaussian prior for this training sample. After the sample is reconstructed, the loss is only backpropagated through the prior that the data point belonged to. Additionally, an MMD penalty is also computed using empirical samples to push the latent posterior distribution to the prior. After the training phase is complete, the model is capable of generating style-conditioned and style-interpolated novel sentences.

Style-conditioned sentences are the novel sentences generated by the model when the desired class of the sentences is conditioned on one of the training classes. This can be achieved by only using one of the latent Gaussian distributions to sample from. Then the decoder will use this sample to generate a sentence similar to one of the classes that is has used during the training phase.

Style-interpolated sentences are the other types of sentences that GMMWAE is capable of generating. These types of sentences are generated by interpolating between the style of two classes. By using a Gaussian mixture model between two of the trained latent distributions, the model can mix two styles together and generate novel sentences with a mixture of styles.

GMMWAE can generate novel sentences with a single style or a mixture of styles while having the advantage of a shared decoder. This allows GMMWAE to learn easier when

there is a data sparsity in the dataset and the number of training samples is limited. In many benchmarks, GMMWAE performs better than other models we compared our work to and in other benchmarks it performs similar to other models.

5.2 Future Work

There are some areas that we would like to explore furtherThere are some areas that we would like to explore further. First, the style-interpolated experiments we conducted were only based on two chosen styles. There is an opportunity to explore the effectiveness of GMMWAE based on more than two styles.

. First, the style-interpolated experiments we conducted were only based on two chosen styles. There is an opportunity to explore the effectiveness of GMMWAE based on more than two styles.

Additionally, it would also be interesting to improve the accuracy of the model on other benchmarks as well. For example, the accuracy of GMMWAE based on the perplexity benchmark can be improved to match other approaches.

Finally, an additional penalty can be added to the latent space to push the normal distributions further away from each other during the training phase. How this will affect the accuracy and the performance of the model is another interesting topic that can be explored.

References

[1] Hareesh Bahuleyan, Lili Mou, Olga Vechtomova, and Pascal Poupart. Variational attention for sequence-to-sequence models. *arXiv preprint arXiv:1712.08207*, 2017.

[2] Hareesh Bahuleyan, Lili Mou, Hao Zhou, and Olga Vechtomova. Stochastic wasserstein autoencoder for probabilistic sentence generation. *arXiv preprint arXiv:1806.08462*, 2018.

[3] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[4] Regina Barzilay and Michael Elhadad. Using lexical chains for text summarization. *Advances in automatic text summarization*, pages 111–121, 1999.

[5] Matan Ben-Yosef and Daphna Weinshall. Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images. *arXiv preprint arXiv:1808.10356*, 2018.

[6] 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*, 2015.

[7] 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*, 2015.

[8] Erik Cambria and Bebo White. Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, 9(2):48–57, 2014.

[9] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394, 1999.

[10] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[11] Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F Stewart, and Jimeng Sun. Doctor ai: Predicting clinical events via recurrent neural networks. In *Machine Learning for Healthcare Conference*, pages 301–318, 2016.

[12] Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. Style transfer in text: Exploration and evaluation. *arXiv preprint arXiv:1711.06861*, 2017.

[13] Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. Style transfer in text: Exploration and evaluation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[14] Xiang Gao, Yizhe Zhang, Sungjin Lee, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. Structuring latent spaces for stylized response generation. *arXiv preprint arXiv:1909.05361*, 2019.

[15] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[16] Benoit Gaujac, Ilya Feige, and David Barber. Gaussian mixture models with wasserstein distance. *arXiv preprint arXiv:1806.04465*, 2018.

[17] Felix A Gers and E Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[19] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[20] Xiaodong Gu, Kyunghyun Cho, Jung-Woo Ha, and Sunghun Kim. Dialogwae: Multimodal response generation with conditional wasserstein auto-encoder. *arXiv preprint arXiv:1805.12352*, 2018.

[21] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1587–1596. JMLR. org, 2017.

[22] Ferenc Huszár. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*, 2015.

[23] Vineet John, Lili Mou, Hareesh Bahuleyan, and Olga Vechtomova. Disentangled representation learning for non-parallel text style transfer. *arXiv preprint arXiv:1808.04339*, 2018.

[24] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[25] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[26] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184. IEEE, 1995.

[27] Donald Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

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

[29] Leslie Lamport. *LATEX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[30] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. In *Proc. of NAACL-HLT*, March 2016.

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

[32] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

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

[34] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[35] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[36] Lili Mou and Olga Vechtomova. Stylized text generation: Approaches and applications. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 19–22, 2020.

[37] Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

[38] Jeffrey Pennington, Richard Socher, and Christopher D 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.

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

[40] Ehud Reiter and Robert Dale. *Building natural language generation systems*. Cambridge university press, 2000.

[41] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[42] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. *arXiv preprint arXiv:1507.04808*, 2015.

[43] Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Style transfer from non-parallel text by cross-alignment. In *Advances in neural information processing systems*, pages 6830–6841, 2017.

[44] Tianxiao Shen, Myle Ott, Michael Auli, and Marc'Aurelio Ranzato. Mixture models for diverse machine translation: Tricks of the trade. In *ICML*, pages 5719–5728, 2019.

[45] Xiaoyu Shen, Hui Su, Shuzi Niu, and Vera Demberg. Improving variational encoder-decoders in dialogue generation. *arXiv preprint arXiv:1802.02032*, 2018.

[46] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[47] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27:3104–3112, 2014.

[48] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

[49] Olga Vechtomova, Hareesh Bahuleyan, Amirpasha Ghabussi, and Vineet John. Generating lyrics with variational autoencoder and multi-modal artist embeddings. *arXiv preprint arXiv:1812.08318*, 2018.

[50] Hanna M Wallach. Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*, pages 977–984. ACM, 2006.

[51] Wenlin Wang, Zhe Gan, Hongteng Xu, Ruiyi Zhang, Guoyin Wang, Dinghan Shen, Changyou Chen, and Lawrence Carin. Topic-guided variational autoencoders for text generation. In *NAACL-HLT*, 2019.

[52] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018.

[53] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.